
PROYECTO FINAL DE CARRERA



UNIVERSITAT
ROVIRA I VIRGILI



ESCOLA
TÈCNICA
SUPERIOR
D'ENGINYERIA

Sistema Multi-Agente para la Coordinación de Transplantes a nivel español

Jaime Bocio Sanz, jaimebocio@ctv.es
Ingeniería Informática

Directores: Antonio Moreno, URV, amoreno@etse.urv.es
Aïda Valls, URV, avalls@etse.urv.es

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV), <http://www.etse.urv.es>

Septiembre, 2002

1. INTRODUCCIÓN AL PROBLEMA	4
1.1. INTERÉS DE LA APLICACIÓN DE NUEVAS TECNOLOGÍAS EN MEDICINA	4
1.2. COORDINACIÓN DE TRANSPLANTES A NIVEL NACIONAL.....	5
2. AGENTES Y SISTEMAS MULTI-AGENTE (SMA).....	7
2.1. PROPIEDADES DE LOS AGENTES	7
2.2. ARQUITECTURAS DE AGENTES	8
2.3. TIPOS DE AGENTES	8
2.4. SISTEMAS MULTI-AGENTE (SMA)	9
2.5. VENTAJAS DEL USO DE UN SMA	9
2.6. FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (FIPA)	11
2.7. MENSAJES	12
2.8. PROTOCOLOS DE COMUNICACIÓN	15
3. JADE.....	17
3.1. PACKAGES DEL JADE	17
3.2. CARACTERÍSTICAS.....	18
3.3. CONSTRUCCIÓN DE AGENTES Y COMPORTAMIENTOS	19
3.4. MANEJO DE MENSAJES	20
3.5. PROTOCOLOS	21
3.6. ONTOLOGÍA	21
3.7. FUNCIONAMIENTO DEL JADE	21
4. DESCRIPCIÓN DEL SISTEMA MULTI-AGENTE IMPLEMENTADO	25
4.1. ARQUITECTURA	26
4.1.1. AGENTE DE HOSPITAL (H).....	27
4.1.2. AGENTE COORDINADOR AUTONÓMICO (A)	29
4.1.3. AGENTE COORDINADOR ZONAL (Z).....	30
4.1.4. AGENTE COORDINADOR NACIONAL (CN).....	30
4.1.5. AGENTE HISTÓRICO (AH).....	31
4.1.6. SMA ORDENADOR (SMAO).....	31
4.1.7. BASES DE DATOS.....	32
4.2. ONTOLOGÍA	34
4.3. PROTOCOLOS DE COMUNICACIÓN	37
4.4. SEGUIMIENTO DEL PROCESO COMPLETO	39
5. DISTRIBUCIÓN MODULAR DEL SISTEMA.....	50
5.1. HOSPITALJAVA	50
5.2. AUTONOMIA.JAVA	52
5.3. ZONA.JAVA	52
5.4. NACIONALJAVA	53
5.5. HISTORICO.JAVA	53
5.6. INTERFICIE.JAVA	53
5.7. EL PAQUETE ONTOLOGY	53
6. MANUAL DE USUARIO	56
6.1. EJECUCIÓN DEL SISTEMA.....	56
6.2. INTERFICIE GRÁFICA DE USUARIO.....	58
6.3. BASES DE DATOS	63

7. JUEGO DE PRUEBAS	64
8. CÓDIGO.....	81
8.1. HOSPITAL.JAVA	81
8.2. AUTONOMIA.JAVA	108
8.3. ZONA.JAVA	120
8.4. NACIONAL.JAVA	130
8.5. INTERFICIE.JAVA	139
8.6. ENTRADA_DATOS.JAVA	146
8.7. HISTORICO.JAVA	148
8.8. EL PAQUETE ONTOLOGY	153
8.8.1. PFC_EI/JBSONTOLOGY.JAVA	153
8.8.2. PFC_EI/LISTA_OBJETOS.JAVA.....	159
8.8.3. PFC_EI/PACIENTE.JAVA.....	161
8.8.4. PFC_EI/ORGANO.JAVA.....	164
8.8.5. PFC_EI/RESULTADO.JAVA.....	165
8.8.6. PFC_EI/GRADO.JAVA.....	166
8.8.7. PFC_EI/ELECCIÓN.JAVA.....	167
8.8.8. PFC_EI/VALOR_ATRIB.JAVA.....	168
8.8.9. PFC_EI/ORDENACIÓN_IMPOSIBLE.JAVA.....	169
8.8.10. PFC_EI/AGENTE_NO_REGISTRADO.JAVA.....	170
8.8.11. PFC_EI/NO_HOSPITALES.JAVA.....	171
8.8.12. PFC_EI/NO_PACIENTES.JAVA.....	172
9. VALORACIÓN Y CONCLUSIONES	173
10. TRABAJOS FUTUROS	174
11. BIBLIOGRAFÍA	175

1. INTRODUCCIÓN AL PROBLEMA

1.1. Interés de la aplicación de nuevas tecnologías en Medicina

Es muy interesante que la medicina esté muy ligada a las nuevas tecnologías. Gracias a ellas, los médicos pueden ser más rápidos, eficaces y precisos que con algunos métodos manuales. En cambio hay algunas cosas que continúan siendo poco eficaces debido a que no se usan las ventajas del progreso. Una de estas cosas es el problema de la logística de transplantes de órganos.

El proceso de gestión de transplantes requiere de la colaboración del personal de diversos hospitales para poder encontrar rápidamente el receptor óptimo, coordinar el transporte de órganos y gestionar los equipos médicos. En nuestro país, la Organización Nacional de Transplantes [ONT], establece un sistema que está demostrando ser de los más eficaces a nivel mundial. De hecho es el número uno en el mundo en número de donaciones por millón de personas. Este sistema se conoce como “El Modelo Español”. Para ello se divide el territorio nacional en 6 zonas (como se ve en la figura 1) cada una con una serie de comunidades autónomas, organizadas a su vez por provincias, localidades y finalmente hospitales. Cada hospital tiene un Coordinador de transplantes que se encarga de gestionar el proceso de asignación de órganos a los pacientes de su hospital y de la comunicación con el resto de coordinadores de otros hospitales.

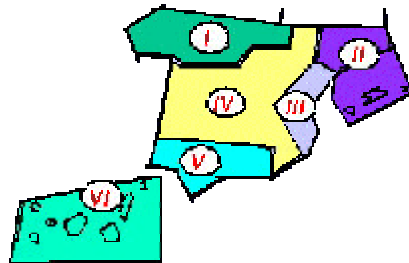


Figura 1 - Distribución del territorio nacional por zonas

El proceso de transplantes ha de ser algo rápido, y tal y como se hace actualmente no lo es demasiado, son muchas llamadas de teléfono, consultas de fichas, horarios, etc. Esto da como resultado el que se pierda un tiempo precioso, ya que un órgano para transplantar no dura eternamente, y hay que hacerlo cuanto antes. Por eso se ha pensado en la posibilidad de gestionar todo el proceso mediante un Sistema Multi-Agente (SMA), lo que nos permite incrementar aún más la rapidez, al automatizar gran parte del proceso, y el rendimiento médico, ya que una máquina puede tener en cuenta las características médicas de muchos pacientes a la vez y encontrar el que se ajusta más a las características del órgano donado.

1.2. Coordinación de transplantes a nivel nacional

Un transplante necesita de tres etapas hasta que se pueda llevar a cabo la operación. Estas tres etapas son las que forman parte de un proyecto común del GruSMA (Grupo de Sistemas Multi-Agente) [GruSMA, 00]. Cada una de ellas por separado es un proyecto individual, que se podría coordinar con el resto sin mucha dificultad a priori. Dos de estas tres partes (la segunda y tercera etapa) ya han sido implementadas en proyectos anteriores, la otra es la que nos ocupa ahora. Es importante reseñar que el sistema sigue el modelo de organización definido por la ONT (Figura 2).



Figura 2 - Jerarquía de búsqueda definido por la ONT

a) La primera etapa consiste en encontrar el paciente idóneo para transplantarle el órgano. Para ello tenemos una jerarquía de coordinadores a distintos niveles (hospitalario, autonómico, zonal y nacional), que buscan el paciente adecuado entre los hospitales de su nivel. La búsqueda también se realiza de forma jerárquica, es decir se comienza por los niveles más bajos y en caso de no encontrar solución se pasa a niveles superiores. Las distintas etapas de búsqueda se pueden ver en la figura 3.

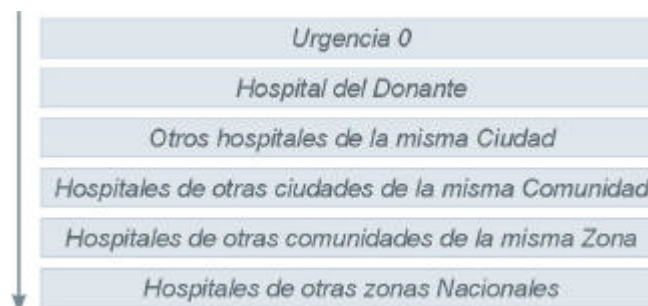


Figura 3 - Etapas jerárquicas de búsqueda de pacientes

El primer nivel corresponde a los pacientes de *Urgencia 0*. Éstos son pacientes críticos que necesitan de forma inmediata el órgano, y a bs que se da preferencia en la búsqueda. Este nivel de búsqueda no se ha incluido en nuestro proyecto, pero está en

fase de implementación como otro proyecto del GruSMA. Más adelante detallaremos como se realiza la búsqueda en cada uno de los niveles.

Para la búsqueda del paciente óptimo se hace uso de otro SMA [Sánchez, 01] también implementado como PFC, que ordena una serie de pacientes en función de unos criterios determinados. Pese a que en nuestro proyecto tratamos el SMA de David Sánchez como si fuera un bloque, tiene que quedar claro que se trata de un SMA, el cual interactúa con el nuestro mediante uno de sus agentes, en concreto el agente **Especialista**. La evaluación de estos pacientes se realiza mediante el paso de una lista con los datos de los pacientes a evaluar y del órgano donado. Estos datos son: *Hospital del paciente, localidad y provincia, tipo de sangre, peso del paciente, tamaño del órgano, valores de los antígenos B y DR, fecha de nacimiento y fecha de entrada en la lista de espera*. El resultado obtenido es una lista donde los pacientes están ordenados según la compatibilidad con el órgano y en la que cada paciente tiene un *grado de idoneidad* asignado, que va de *pésimo* a *óptimo*.

Una vez se obtiene el paciente adecuado para el transplante y se sabe el hospital donde se encuentra es cuando entran en juego las dos partes anteriormente comentadas. En este apartado sólo comentamos la idea general de este proyecto, los detalles vendrán más adelante.

b) La segunda parte [Ribes, 00] consiste en la búsqueda de una ruta para que el órgano viaje del hospital origen al destino de la forma más eficiente posible. Esto conlleva consultas a las diferentes agencias de transportes, trenes y aviones, para encontrar una combinación de transportes que pueda hacer que el órgano llegue a la hora adecuada. Si esto es posible entra en juego la tercera etapa [Bocio, 00].

c) Una vez se sabe a la hora que va a llegar un cierto órgano a un hospital y el tiempo del que se dispone para aprovecharlo, hay que preparar la operación. Para ello se necesita formar un equipo médico (médicos especialistas, enfermeras y anestesiistas) y reservar un quirófano. Dependiendo del tipo de órgano, los requerimientos en cuanto a número de personal médico son distintos. La dificultad aquí consiste en buscar los intervalos de tiempo en que podemos poner de acuerdo a todo el personal necesario, así como el quirófano, antes de que el órgano se vuelva inservible.

Como ya hemos comentado, este proceso puede llegar a ser muy costoso si se realiza de forma manual, es decir mediante llamadas telefónicas a los distintos hospitales. Esto supone un gasto enorme en tiempo y dinero. Para intentar solucionar este problema de costes, se ha pensado en la utilización de un Sistema Multi-Agente (SMA), por las grandes ventajas que conlleva y que se comentarán más adelante.

En este documento, en primer lugar se hace una introducción al proyecto, para proseguir con el porqué se ha elegido utilizar un Sistema Multi-Agente, sus ventajas y características. Se explica luego el entorno de trabajo escogido, en este caso el Jade. Después de clarificar los objetivos del proyecto, nos centramos en nuestro SMA concreto. Vemos su arquitectura, los agentes que lo componen, sus características, etc. Se expone un ejemplo completo para explicar el funcionamiento del sistema. Mostramos el manual de usuario y finalmente tenemos un juego de pruebas, y un apartado a modo de conclusión, donde además se muestran posibles trabajos futuros o mejoras a realizar sobre el tema.

2. AGENTES Y SISTEMAS MULTI-AGENTE (SMA)

Para resolver el problema de la coordinación de trasplantes se ha utilizado en este PFC la tecnología de los agentes inteligentes y los sistemas multi-agente. En este capítulo se empieza comentando qué propiedades ha de tener un programa para ser considerado un agente, y cuáles son las arquitecturas básicas usadas para implementar un agente. A continuación se hace una clasificación de diferentes tipos de agente. Un conjunto de agentes colaborativos define un Sistema Multi-agente. En la sección 2.5 se comentan las ventajas que tiene utilizar esta aproximación para la resolución del problema planteado. Al final del capítulo describimos las especificaciones que ha sugerido la FIPA para la definición de lenguajes y protocolos de comunicación entre agentes.

Como en la mayoría de los casos en que se intenta atribuir cualidades inteligentes a un proceso computacional, la definición de Agente no está demasiado clara. Se podría describir como un sistema que habita en un entorno complejo y dinámico; es capaz de sentir y actuar autónomamente en su entorno, y tiene un conjunto de objetivos o motivaciones que intentará conseguir a través de sus acciones [Weiss, 99].

2.1. Propiedades de los agentes

Las propiedades más importantes que pueden tener los Agentes son las siguientes:

- **Autonomía:** es la capacidad de operación sin la intervención directa de los humanos u otros, 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 y/o humanos 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 Capacidad de tomar Iniciativa:** han de ser capaces de exhibir algún tipo de comportamiento encaminado a conseguir un objetivo tomando la iniciativa.
- **Movilidad:** posibilidad de moverse a otros entornos a través de una red electrónica.
- **Continuidad temporal:** los agentes están continuamente ejecutando procesos.
- **Veracidad:** un agente no comunicará información falsa premeditadamente.

- **Benevolencia:** los agentes no tienen objetivos conflictivos entre si, cada agente intentará hacer aquello para lo que es requerido.
- **Racionalidad:** un agente ha de actuar para conseguir su objetivo.

2.2. Arquitecturas de agentes

Las arquitecturas utilizadas en el proceso de implantación de los agentes pueden ser de tres tipos:

- **Deliberativas:** dada una representación del mundo real, se pueden ejecutar acciones inteligentes. Siguen el modelo BDI (*Beliefs, Desires and Intentions*).
- **Reactivas:** estructura donde las acciones se llevan a cabo respondiendo a estímulos. La combinación de diversos agentes reactivos proporciona un comportamiento “inteligente”.
- **Híbrida:** estructuras que mezclan diferentes técnicas.

2.3. Tipos de agentes

Según la finalidad para la que se implementa un agente, podemos clasificarlos en diferentes tipos:

- **De información:** gestionan y manipulan datos; pueden responder a requerimientos de los usuarios o de otros agentes. Un ejemplo muy común son los buscadores de información en Internet.
- **De interficie:** sistema donde los agentes colaboran con el usuario para resolver un problema. La interacción con el individuo permite al agente desarrollar un aprendizaje basado en las acciones que se realizan.
- **De colaboración:** la característica principal es la comunicación y cooperación con otros agentes para resolver un problema común. Utilizan técnicas de IA distribuida.
- **Móviles:** su característica principal es la posibilidad de poderse mover 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 encuentran. 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 refieren a un conjunto integrado de al menos dos agentes que pertenecen a dos o más clases distintas.

2.4. Sistemas Multi-Agente (SMA)

El concepto de Sistema Multi-Agente (SMA) viene de la unión de un grupo de agentes de tipo colaborativo. Un SMA será aquel en el que un conjunto de agentes cooperan, se coordinan y se comunican para conseguir un objetivo común.

Se basan en la concurrencia de varios agentes para realizar tareas, en las que los agentes se ayudan entre sí. Con esto se consigue una mayor rapidez, paralelismo y concurrencia, ya que cada agente puede ejecutarse en una máquina distinta [Isern, 99].

Tenemos pues un conjunto de agentes en máquinas distintas, que necesitan de comunicación para resolver problemas conjuntamente. La pregunta ahora sería, ¿cómo se realiza esa comunicación? La respuesta es mediante mensajes. Los agentes son capaces de enviar y recibir mensajes, y una vez recibidos, procesarlos para actuar de una determinada manera dependiendo del contenido y el tipo de mensaje. Por ejemplo, hay mensajes para hacer peticiones, propuestas, para aceptarlas o rechazarlas, para cancelar algún proceso, etc. Estos tipos de mensajes están, en nuestro caso, reglados por la FIPA (Foundation for Intelligent Physical Agents), la cual ha definido un estándar para el Lenguaje de Comunicación de Agentes (ACL) [FIPA, 99]. Otro estándar es el KQML, pero no hablaremos de él, porque no es el que se usa en este proyecto.

Los agentes son capaces de mantener una o varias conversaciones, parecidas a las conversaciones humanas. El comportamiento que siguen es semejante a los humanos, ya que por ejemplo un agente le puede pedir a otro que haga una cosa, y este otro le puede decir que la hará o no. Luego, dependiendo de la respuesta, el primer agente actuará de una u otra forma, y tal vez le pida a otro agente que realice la tarea que le había ordenado al segundo. Se asemeja a la manera de comunicarnos nosotros, pero por el hecho de ser máquinas, con unos protocolos determinados. Estos protocolos, también están definidos por la FIPA, y más adelante se hará un repaso de ellos, así como de los tipos de mensajes y parámetros.

2.5. Ventajas del uso de un SMA

He comentado que el principal problema del planteamiento actual de la logística de transplantes es el coste, principalmente en tiempo. Esto se puede solucionar de una manera bastante eficaz con el empleo de SMA.

Pasemos ahora a las ventajas que nos aportan los SMA sobre este proyecto en concreto. Para verlo más claramente, primero explicaremos un ejemplo típico de cómo se realiza el proceso actualmente. Tenemos un coordinador general de transplantes, que recibe información de los donantes y de los órganos transplantables. Este coordinador tiene que buscar un paciente adecuado, y para ello tiene que avisar a todos los hospitales de la zona para ver quién quiere ese órgano, esto se hace mediante llamadas o mensajes. Una vez se ha encontrado el hospital, se ha de encontrar la ruta, consultando las bases de datos de agencias de transporte. Más llamadas telefónicas y más tiempo perdido,

mientras el órgano se va deteriorando. Finalmente ese órgano llegará a un hospital y se habrá de buscar el equipo médico. Esto provoca consultas de los horarios por parte de algún empleado del hospital. Una vez consultados los horarios se tendrán que intentar compatibilizar todos, una tarea bastante compleja.

Lo que nos permite un SMA es:

- Crear una jerarquía de coordinadores a distintos niveles, de manera que se puedan realizar búsquedas a determinados niveles, empezando por los más bajos y subiendo si no se encuentra el paciente adecuado.
- Ahorro de multitud de llamadas telefónicas, ya que toda la comunicación entre agentes va por red. No hace falta que sea una red especial. En principio el proyecto se ejecuta todo en la misma máquina, pero no debería constituir ningún problema su instalación en diferentes ordenadores.
- Aumento considerable de rapidez en la búsqueda, ya que no son necesarias la consulta de fichas ni las llamadas telefónicas. Todo el proceso es automático, aunque evidentemente la última palabra sobre la donación la tendrá un médico especializado que habrá de supervisar el proceso.
- El aumento de velocidad en la búsqueda provoca directamente que el tiempo de espera del órgano se reduzca, lo cual es crucial en este tipo de operaciones en las que el órgano se deteriora fácilmente.
- Se modela el esquema real expuesto por la ONT. No inventamos un nuevo modelo de coordinación de transplantes.
- El SMA aporta paralelismo y concurrencia al sistema. Cada hospital realizaría la búsqueda de pacientes por separado, ya que cada base de datos debería estar en una máquina distinta, aunque para la realización de este proyecto no haya sido así.
- Este paralelismo también hace que el sistema sea capaz de considerar muchos posibles receptores a la vez, con lo que las posibilidades de éxito de la búsqueda aumentan de manera considerable.
- La búsqueda se hará de forma jerárquica y eficiente. Es decir, primero se intentará que el órgano vaya a parar lo más cerca posible, al propio hospital. Si no hay pacientes óptimos en el hospital, se buscará en la ciudad, luego en la autonomía, y así subiendo hasta buscar en todo el país si es necesario.
- Otro punto a favor es la flexibilidad, en el sentido que se pueden añadir o eliminar agentes del sistema muy fácilmente. Esto será especialmente útil cuando haya altas o bajas de hospitales, ya que en principio el número de coordinadores no debería cambiar.
- La modularidad permite que un agente pueda realizar una tarea pequeña, y luego juntándola con las de los demás agentes que se han ejecutado en paralelo obtener una tarea mayor en un menor tiempo.

Los SMA tienen una gran cantidad de ventajas. Pero como cualquier sistema informático, están sujetos al buen funcionamiento de los ordenadores y redes. Bajadas de tensión o apagones pueden inhabilitar el sistema. Sin embargo si los agentes se ejecutan en distintas máquinas, parte del SMA puede seguir funcionando aunque fallen algunos equipos.

2.6. Foundation for Intelligent Physical Agents (FIPA)

La FIPA es una organización sin ánimo de lucro que se dedica a definir estándares sobre cómo ha de ser la arquitectura de un SMA, el lenguaje de comunicación entre agentes, los protocolos de comunicación, etc.

La administración de agentes establece el modelo lógico para la creación, registro, comunicación, establecimiento, movilidad y destrucción de agentes. La FIPA establece el siguiente modelo para este entorno de administración de agentes:

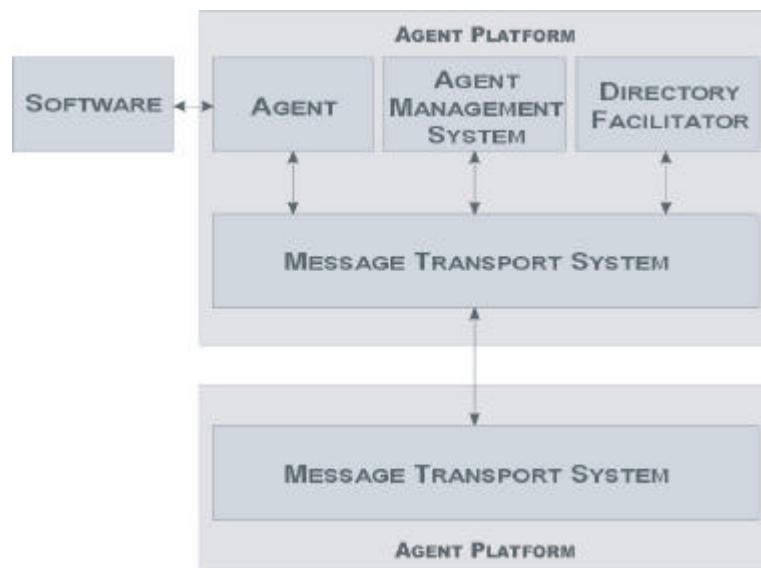


Figura 4 - Aspecto interno de un SMA

Los componentes lógicos principales que forman parte de la figura 4 son:

- **Agent:** unidad básica. Se puede describir como un programa que encapsula una serie de servicios.
- **Directory Facilitator (DF):** es un agente que proporciona un servicio de “páginas amarillas” dentro del sistema (conoce los servicios que proporcionan los diferentes agentes del SMA). Los agentes han de registrarse al 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 agentes de diversas plataformas.
- **Agent Platform (AP):** proporciona la infraestructura básica en la que pueden crearse y ejecutarse agentes.
- **Software:** cualquier programa accesible desde un agente.

2.7. Mensajes

Ya ha quedado claro que la comunicación en un SMA se basa en el paso de mensajes, lo que todavía no se ha expuesto es la composición de un mensaje y los distintos tipos que hay, eso lo haremos a continuación.

2.7.1. Composición de un Mensaje

Antes que nada, es importante decir que estas explicaciones se basan en el lenguaje de comunicación de agentes (ACL) propuesto por la FIPA, por tanto los mensajes podrían variar para otros lenguajes, pero para el caso que nos ocupa, nos basta con el ACL de la FIPA.

El aspecto de un mensaje puede ser el de la figura 5.

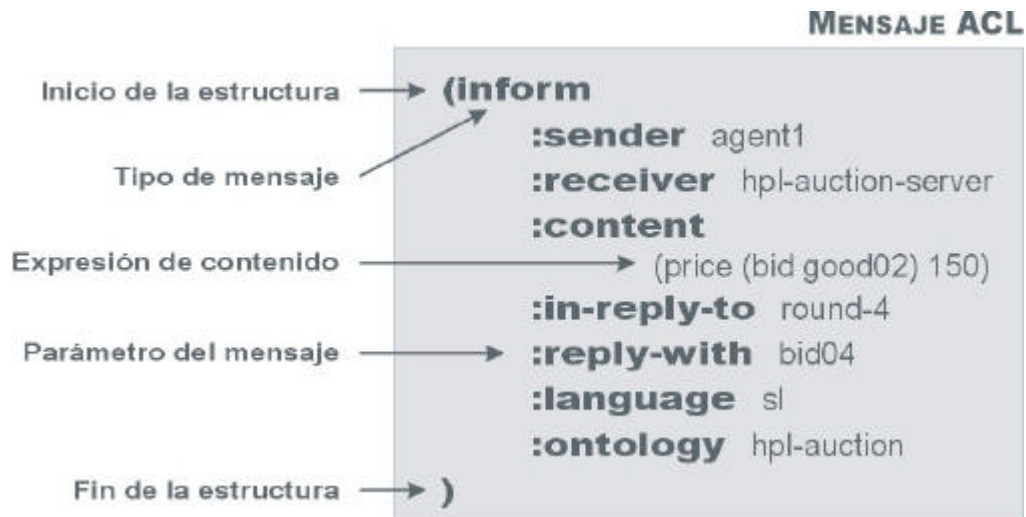


Figura 5 - Aspecto de un mensaje ACL

En un mensaje hay que indicar varias cosas, como son el tipo de mensaje y los parámetros. Del tipo de mensajes posibles nos ocuparemos más adelante, ahora nos centramos en sus parámetros.

Parámetro:	Significado:
:sender	Denota la identidad del agente que envía el mensaje.
:receiver	Indica la identidad del que recibe el mensaje. Aquí pueden haber uno o varios nombres de agente, (dependiendo de a quien se quiera hacer llegar el mensaje, puede ser individual o colectivo).
:content	Contenido del mensaje, lo que se le quiere decir a los agentes receptores.
:reply-with	Introduce una expresión que será usada por el agente que responda para identificar este mensaje. Puede ser usado, por ejemplo, para seguir conversaciones entre agentes. Si un agente envía un mensaje que contiene :reply-with query1 el receptor responderá con otro que contenga :in-reply-to query1
:in-reply-to	Hace referencia a que este mensaje es una respuesta a otro anterior.
:envelope	Sirve para poner diversos aspectos del servicio del mensaje.
:language	Denota el esquema de codificado usado en el contenido.
:ontology	Expresa la ontología usada para el significado de los distintos símbolos que aparecen en el mensaje.
:reply-by	Indica el tiempo en que el mensaje ha de ser respondido.
:protocol	Indica el protocolo usado por los mensajes. Comentaremos los distintos protocolos más adelante.
:conversation-id	Expresa un identificador de conversación. Esto es especialmente útil cuando un agente mantiene varias conversaciones a la vez.

Tabla 1 - Parámetros de los mensajes

Hay algunas normas o pautas a seguir para indicar el contenido (**:content**) de un mensaje. En este proyecto el contenido de cada mensaje se codifica mediante una cierta ontología, que permite pasar objetos que todos los agentes del sistema pueden comprender y decodificar. Para manejar los parámetros de un mensaje, **JADE** nos ofrece varios métodos en la clase **ACLMessage**, que nos permiten consultarlos o modificarlos (**getContent()**, **setContent()**, etc).

2.7.2. Tipos de Mensajes

Hay gran variedad de tipos de mensajes. Aquí explicaremos el sentido de cada uno y su posible uso, pero no nos extenderemos demasiado. Para que nos hagamos una idea general del uso de los mensajes, aquí tenemos una tabla que expone para que tipo de acto comunicativo sirve cada mensaje.

Acto Comunicativo	Paso de Información	Pedido de Información	Negociación de Acciones	Realización de Acciones	Manejo de Errores
ACCEPT-PROPOSAL			✓		
AGREE				✓	
CANCEL				✓	
CFP			✓		
CONFIRM	✓				
DISCONFIRM	✓				
FAILURE					✓
INFORM	✓				
INFORM-IF	✓				
INFORM-REF	✓				
NOT-UNDERSTOOD					✓
PROPOSE			✓		
QUERY-IF		✓			
QUERY-REF		✓			
REFUSE				✓	
REJECT-PROPOSAL			✓		

Tabla 2 - Categorías de actos comunicativos

Para ver con más detalle el uso de los tipos de mensajes, pasaremos a explicarlos un poco:

accept-proposal: Aceptación de una propuesta para realizar una acción.

agree: Para indicar que se está de acuerdo en realizar una acción.

cancel: Cancelación de alguna petición anterior que aún no ha finalizado.

cfp: (Call For Proposals) Sirve para pedir propuestas a algunos agentes para que realicen una acción.

confirm: Se le confirma al receptor que una determinada proposición es cierta, cuando el receptor no estaba seguro.

disconfirm: Al revés que en el anterior tipo, aquí se informa de que una proposición es falsa cuando en realidad se creía verdadera.

failure: Se le indica a otro agente que algo ha fallado.

inform: Simplemente se le informa al receptor de algo.

inform-if: Se le informa a un agente de si una proposición es verdadera o no.

not-understood: El agente receptor de un mensaje no entiende lo que significa y así se lo hace saber al que lo ha enviado.

propose: Hace una propuesta a un agente que se la ha pedido, dando unas ciertas precondiciones.

query-if: Se le pregunta algo a otro agente.

refuse: Se rechaza una petición para realizar una acción. Normalmente va acompañado de los motivos del rechazo.

reject-proposal: Al contrario del accept-proposal, se rechaza una propuesta durante una negociación.

request: Se le indica a un agente que realice una acción.

Request-when: Como antes, pero esta vez se quiere que realice la acción cuando se den unas determinadas condiciones.

subscribe: Sirve para mantenerse informado en todo momento del valor de una referencia.

2.8. Protocolos de Comunicación

Las conversaciones entre agentes siguen en algunas ocasiones unos patrones determinados, que se repiten en muchos casos. Aprovechando estos patrones y su repetición, existen los llamados **protocolos** de la **FIPA**. Un protocolo es un patrón que se usa para llevar por unos cauces concretos una conversación, son como conversaciones guiadas, en que cada agente sabe qué mensaje enviar y cuáles puede recibir.

Los tres protocolos básicos definidos por la FIPA, que veremos a continuación son: **FIPA-request protocol**, **FIPA-query protocol** y **FIPA-ContractNet protocol**.

2.8.1. Protocolo FIPA-request

Es el más común y utilizado, se usa cuando un agente pide a otro que realice una acción. El destinatario puede aceptar o rechazar la petición, y en caso de aceptarla, deberá realizar la acción e indicárselo al otro agente cuando finalice. En el siguiente diagrama (figura 6) se observa el flujo de los mensajes. Los grises son los que envía el “initiator” al que llamaremos Agente1, mientras que los del “responder” (Agente2) están en azul.

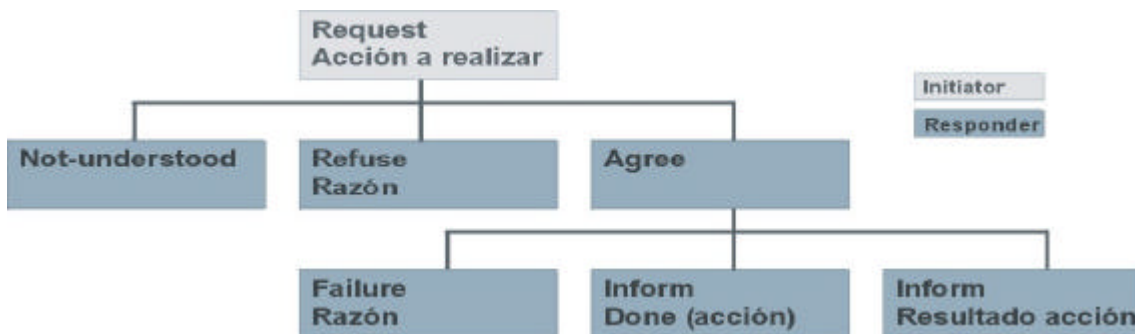


Figura 6 - Protocolo FIPA-request

Al recibir una petición, el Agente2 puede optar por aceptarla o rechazarla, indicando el motivo (o bien si no entiende el mensaje enviar un **not-understood**). Si decide aceptar, tendrá que realizar la acción. Si la realiza informa de ello al Agente1; lo mismo ocurre si tiene que pasar algún resultado o ha habido algún fallo.

2.8.2. Protocolo FIPA-query

Aquí el Agente1 pide algún tipo de información al Agente2. Este protocolo puede ser comenzado con dos tipos de mensaje **query-if** o **query-ref**. Como se ve en la figura 7, tras la petición de información, el Agente2 puede responder con la propia información, con un fallo, con un **not-understood** o con el rechazo de la petición, teniendo que alegar el motivo.

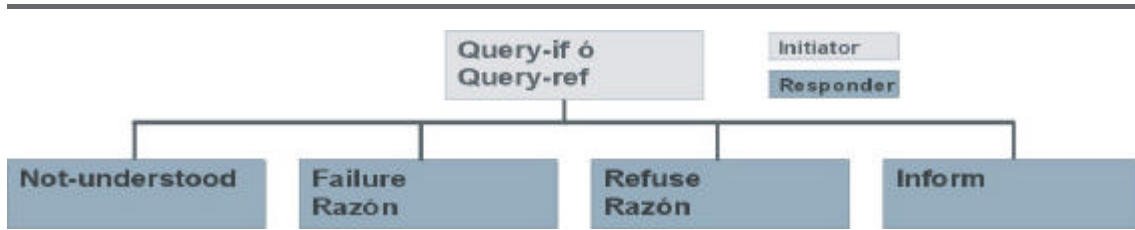


Figura 7 - Protocolo FIPA-Query

2.8.3. Protocolo FIPA-contract-net

El Agente1 quiere que uno o más agentes realicen una acción, por eso inicia una especie de encuesta, pidiendo propuestas a los distintos agentes. En la petición se especifica la acción a realizar y algunas precondiciones a la hora de hacer las propuestas. Los agentes consultados envían sus propuestas al Agente1, también indicando las condiciones de la propuesta. Éste las estudia y elige las que le convienen, rechazando el resto. El protocolo requiere que el Agente1 sepa cuando ha recibido todas las propuestas, para no dejar a ningún agente “con la palabra en la boca”. Como esto podría llevar mucho tiempo en caso de que un agente tarde más de la cuenta, se da un tiempo límite para responder, y las propuestas que lo hagan más tarde, serán rechazadas.

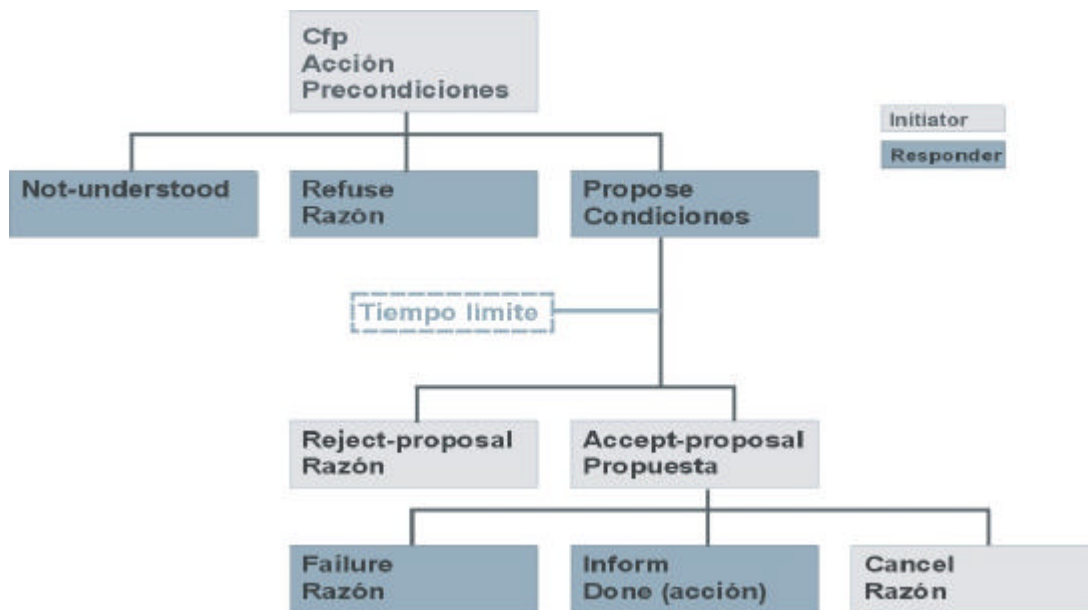


Figura 8 - Protocolo FIPA-Contract-Net

El Agente1 puede cancelar todo el proceso una vez aceptadas y rechazadas las propuestas, si es que algo cambia de la situación inicial. Cuando se acepta una propuesta el Agente2 puede responder con un **inform** indicando que se ha realizado la acción, o indicando que ha habido algún fallo con un **failure**.

Los protocolos usados en este proyecto son el FIPA-Contract-Net y el FIPA-Request. Hay que comentar que se ha tenido que hacer alguna pequeña modificación en algún protocolo para que los agentes pudieran interactuar con distintos protocolos a la vez con más de un agente. Esto quedará más claramente explicado en el apartado de funcionamiento del sistema.

3. JADE

Para la realización de este proyecto, se decidió usar el entorno de desarrollo de Sistemas Multi-Agente JADE (Java Agent Development Environment). Hay docenas de entornos de construcción de SMA, pero se escogió JADE por los siguientes motivos:

- Sigue las especificaciones de la FIPA, lo que permite seguir un estándar en el desarrollo de SMA.
- Es un entorno completamente gratuito, tanto la descarga de Internet como su posterior uso.
- Mantiene una lista de distribución activa, muy útil para la resolución de problemas.
- Elabora nuevas versiones regularmente, corrigiendo los bugs de las anteriores, mejorando algunos aspectos y adaptando otros a los estándares propuestos por la FIPA.
- Se puede hacer funcionar sobre cualquier ordenador con JAVA.
- Permite la distribución de una plataforma sobre distintas máquinas, esencial para SMA's distribuidos.
- Posee una interficie gráfica muy útil para el desarrollador, que comentaremos más adelante.

3.1. Packages del Jade

Jade está escrito en JAVA 1.2, y obliga a que los agentes también estén escritos en este lenguaje. Hace uso de varios packages [Bellifemine, 00], que son los siguientes:

- **jade.core** contiene el núcleo del sistema. Además proporciona la clase **Agent** que es básica para la programación de SMA. En **jade.core.behaviours** tenemos la jerarquía de comportamientos que se desprende de la clase **Behaviour**. La implementación de un agente se basa en los comportamientos, más adelante ya comentaré más exhaustivamente de que se tratan.
- **jade.lang** contiene sub-packages para cada lenguaje usado en JADE. En nuestro caso usamos el **jade.lang.acl**, que es el que usa la FIPA. Contiene otra de las clases más importantes para la programación de SMA, la clase **ACLMessage**.
- **jade.domain** contiene elementos básicos para la gestión de un SMA como son el **AMS**, **DF** o **ACC** (se comentan más adelante).

- **jade.proto** contiene clases para el control de protocolos definidos por la FIPA, además de clases para ayudar al usuario a crear sus propios protocolos.
- **jade.gui** contiene todo lo necesario para la implementación de interfaces gráficas que ayuden a la interacción con los agentes.
- **jade.onto** contiene el soporte para utilizar diversas ontologías en JADE. Más adelante explicaré las ontologías definidas y usadas en este proyecto.
- **jade.tools** presenta algunas herramientas útiles para el desarrollo de un SMA. Por ejemplo la interficie gráfica, que permite un mejor control del SMA. Otra herramienta importante es el agente **sniffer** que permite hacer un seguimiento de los mensajes que se intercambian los distintos agentes. Se usa una notación similar a los diagramas de secuencias UML, que ayuda muchas veces en el control y la comprensión del sistema. Otras posibilidades que ofrece la interficie gráfica es poder añadir, parar, reanudar o eliminar agentes de forma dinámica, además de permitir enviar mensajes a cualquier agente presente en el sistema.

3.2. Características

Algunas características del JADE son:

- Plataforma que cumple las características de la FIPA. Incluye el **AMS** (Agent Management System), el **DF** (Directory Facilitator) y el **ACC** (Agent Communication Channel), que son activados cuando se inicia la plataforma. Estos tres agentes sirven para realizar algunas tareas del SMA. AMS controla la creación, suspensión, reanudación y eliminación de agentes en la plataforma. ACC usa información de AMS para manipular los mensajes entre los distintos agentes de la misma u otra plataforma. Finalmente tenemos DF, que es el único con el que interactuaremos de forma visible. Este agente sabe los servicios que tiene cada agente registrado al DF. Nosotros usamos una búsqueda en el DF, para hallar hospitales y coordinadores de un determinado nivel (local, autonómico, etc).
- Plataforma de agentes distribuida, es decir que se pueden ejecutar agentes en distintas máquinas. En una plataforma pueden haber varios contenedores para agrupar agentes.
- Pueden haber varios DF's, que facilitarán la labor en caso de haber distintos grupos de agentes.
- Cumple el protocolo FIPA97 para la interconexión de diferentes plataformas de agentes.
- Transporte eficiente de mensajes ACL dentro de la misma plataforma. Se producen transformaciones de objetos JAVA a Strings y al contrario que son totalmente transparentes para el usuario.
- Librería de protocolos de interacción propuestos por la FIPA.

- Registro y desregistro automático de agentes con el **AMS**.
- Interficie gráfica de usuario para manipular múltiples contenedores de agentes.

Ya están explicadas un poco las características y los packages básicos del JADE, ahora pasaré a explicar un poco el proceso de creación de un agente y lo que son los comportamientos.

3.3. Construcción de Agentes y Comportamientos

Ya hemos visto que existe la clase **Agent**, que nos permite crear nuestros agentes. Para hacerlo, hay que hacer una subclase de esta clase y luego a la hora de iniciar el sistema, instanciarla.

Los comportamientos forman parte de la construcción de un agente, son las actividades que un agente realizará a lo largo de su vida. Hay distintos tipos de comportamientos. Los hay que sólo se ejecutan una vez, que se van ejecutando cíclicamente, que se ejecutan a partir de un evento concreto, formados por comportamientos simples, etc. Para ejecutar todos los comportamientos de un agente, existe un scheduler que hace que se vayan ejecutando todos más o menos concurrentemente, de manera que a no ser que uno de ellos bloquee el agente, todos tengan oportunidad de ejecutarse.

Los comportamientos se obtienen instanciando alguna de las clases de la jerarquía de Behaviours.

Como cualquier clase de JAVA, la clase agente tiene una serie de métodos, lo mismo ocurre en los comportamientos y en general en todas las clases de Jade.

3.3.1. Tipos y Clases de los Comportamientos

Behaviour es una clase abstracta que define el esqueleto de lo que será una tarea elemental de un agente. Posee dos métodos que hay que mencionar. Uno es **action()** que representa la tarea que habrá de ejecutar el comportamiento. El método **done()** es usado por el scheduler. Este método retorna cierto cuando el comportamiento ha finalizado y puede ser eliminado de la cola de comportamientos; si retorna falso significa que ha de seguir ejecutándose la tarea descrita en **action()**.

De esta clase **Behaviour** se desprenden tres tipos de comportamientos que son **SimpleBehaviour**, **ReceiverBehaviour** y **ComplexBehaviour**.

SimpleBehaviour se emplea para implementar acciones atómicas, es decir sin interrupción, se ejecuta toda la acción de principio a fin sin que el scheduler la pare.

ComplexBehaviour define dos métodos que le permiten formar un comportamiento complejo a partir de otros simples: **addSubBehaviour(Behaviour)** y **removeSubBehaviour(Behaviour)**. El scheduler sólo ejecuta una sub-tarea cada vez.

ReceiverBehaviour permite recibir un mensaje semejante a un patrón, el comportamiento se bloquea (sin bloquear el resto de actividades del agente) si no hay mensajes en la cola. Permite también poner un timeout o tiempo límite para la recepción de un mensaje, al final del cual, el comportamiento se reinicia.

Del **SimpleBehaviour** dependen otras dos clases: **OneShotBehaviour** y **CyclicBehaviour**. La primera crea un comportamiento que tan sólo se ejecuta una vez, para ello hace que el método **done()** siempre retorne cierto. En el comportamiento cíclico, **done()** siempre retorna falso, por lo que el comportamiento se ejecuta una y otra vez.

Del **ComplexBehaviour** se desprenden otros dos: **SequentialBehaviour** y **NonDeterministicBehaviour**. Corresponden a la manera de ejecutar los comportamientos dentro de uno complejo. La primera manera es secuencialmente, finalizando cuando el último hijo termina. Al contrario, con la forma no determinista se ejecutan de forma alternativa los hijos (con un Round Robin) y se finaliza cuando se da una cierta condición, como por ejemplo que hayan terminado cierto número de hijos.

En la clase **Agent** se nos permite añadir comportamientos en el método **setup()** de cada agente (que es como el **main()** de la clase **Agent**) gracias al método **addBehaviour(Behaviour)**. A partir de aquí, al hacer la instancia del agente en cuestión, el scheduler se encargará del resto, coordinando los distintos tipos de comportamientos que haya en cada agente.

3.4. Manejo de Mensajes

Los métodos básicos para el manejo de los mensajes son los que nos permiten enviarlos y recibirlos. Estos son:

send(ACLMessage) se usa para enviar un mensaje una vez rellenados todos los parámetros de éste. Se enviará desde el agente que invoca el método a todos los destinatarios que contiene el campo **:receiver**.

ACLMessage receive() recibe un mensaje de la cola de mensajes del agente. Se puede especificar qué tipo de mensaje se quiere recibir mediante los patrones de mensaje (**MessageTemplate**).

ACLMessage blockingReceive() bloquea el agente mientras no se reciba un mensaje. Al igual que en el método anterior, se puede especificar el tipo de mensaje a recibir, con lo que se desearía el resto, y el agente permanecería bloqueado hasta recibir el mensaje deseado.

Hay algunos métodos más para el manejo de mensajes, pero estos son los tres usados en este proyecto.

3.5. Protocolos

Para cada conversación JADE distingue dos roles, el que la inicia (Initiator) y el que la prosigue (Responder). JADE nos provee de comportamientos para estos dos roles en cada uno de los protocolos especificados por la FIPA. Un agente deberá extender estos comportamientos dependiendo del rol que adopte en la conversación. Al usar uno de estos protocolos, los mensajes enviados, tendrán en el campo **:protocol** el tipo de protocolo al que pertenece el mensaje, para que los agentes implicados sepan que se está siguiendo un protocolo.

Hay tres protocolos básicos definidos por la FIPA y que tienen comportamientos en el JADE: **FIPA-request protocol**, **FIPA-query protocol** y **FIPA-ContractNet protocol**. Ya hemos visto antes su funcionamiento.

3.6. Ontología

Una ontología define el vocabulario empleado en la comunicación entre agentes. Es muy importante definir una ontología adecuada a nuestro problema concreto para que todos los agentes puedan usarla y saber de qué están “hablando” en cada momento.

Una ontología permite definir los siguientes puntos:

- **Frames**: objetos sobre los cuales se podrá hablar y que, por tanto, estarán disponibles para rellenar el mensaje.
- **Slots**: campos de información en cada objeto o **frame**.
- **Slots** que a la vez son **frames**, para poder representar datos complejos.
- **Etiqueta** con la que se accederá y **tipos de datos** (String, long o compuesto) que contiene cada **slot**.

3.7. Funcionamiento del Jade

Como ya hemos comentado el Jade está desarrollado en JAVA, por eso una vez instalado el software de Jade, habrá que añadir algunas cosas al CLASSPATH. Estos archivos son los **.jar**. Una vez hecho esto ya estamos preparados para ejecutar una plataforma del Jade, de la siguiente manera:

```
java jade.Boot -platform [opciones] [Lista de agentes]
```

Para ver una lista de todos los argumentos disponibles, usaremos la opción **-h**. La lista de agentes es una secuencia de texto, en la que se indica, separado por dos puntos, el nombre del agente y la clase de Java que implementa el agente. En nuestro caso, por ejemplo nos encontramos con **H:pfc_ei.Hospital**, que indica que el agente **H** es de la clase **Hospital** y pertenece al package **pfc_ei**.

3.7.1. Interficie gráfica

Activando la opción **-gui**, se nos abrirá una interficie gráfica, con el aspecto de la figura 9. El agente que controla esta interficie es el RMA, que junto a ACC, AMS y DF se inician automáticamente.

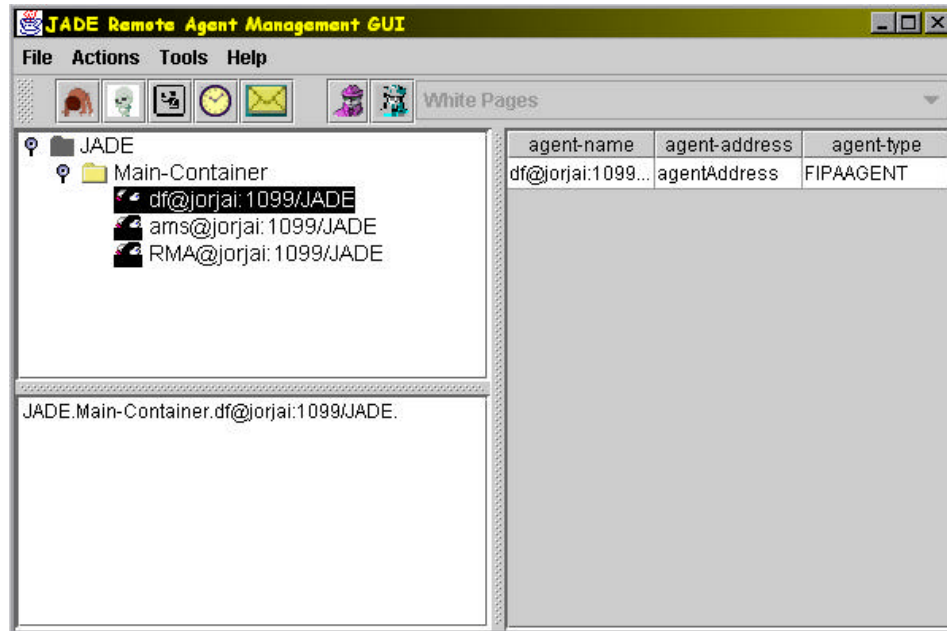
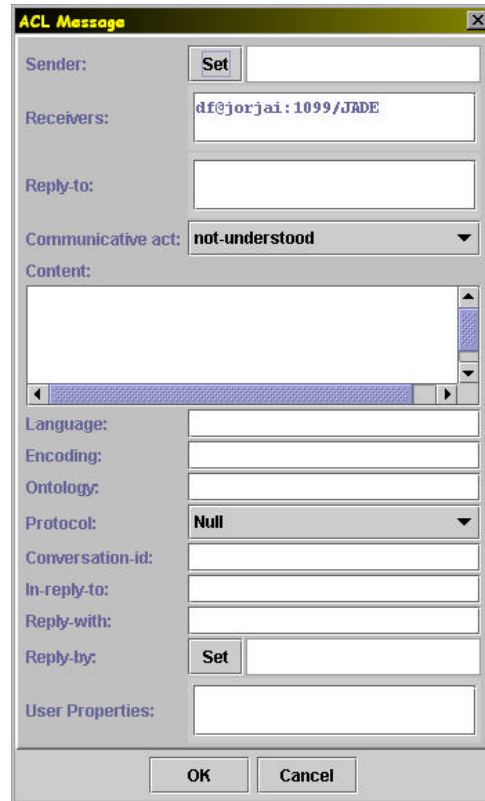


Figura 9 - Pantalla Principal del Jade

Gracias a esta interficie gráfica, se nos dan múltiples herramientas para la utilización de los agentes. Vamos a explicar un poco en que consisten estas herramientas y utilidades. En primer lugar, podemos ver en la ventana de la izquierda los agentes que tenemos en cada contenedor. En este caso no hemos ejecutado ningún agente, y tan sólo están los propios del sistema (AMS, RMA y DF). A la derecha y abajo podemos ver más información acerca del agente seleccionado.

En los botones se nos presentan las acciones más frecuentes, de izquierda a derecha son:

- Añadir un nuevo agente al sistema: Nos permite ejecutar un agente. Hemos de indicarle el nombre, la clase en que está implementado y el contenedor al que pertenece. Se nos da el Main-Container como contenedor por defecto.
- Matar los agentes seleccionados: elimina a los agentes del sistema.
- Suspender los agentes seleccionados: realiza una pausa sobre los agentes seleccionados.
- Reanudar agentes seleccionados.
- Enviar mensaje a los agentes seleccionados: nos permite rellenar un mensaje como este, para enviarlo al agente seleccionado.



The image shows a dialog box titled "ACL Message". It contains several fields for configuring a message:

- Sender:** A text field with a "Set" button next to it.
- Receivers:** A text field containing the value "df@jorjai:1099/JADE".
- Reply-to:** An empty text field.
- Communicative act:** A dropdown menu currently showing "not-understood".
- Content:** A large empty text area for the message content.
- Language:** An empty text field.
- Encoding:** An empty text field.
- Ontology:** An empty text field.
- Protocol:** A dropdown menu currently showing "Null".
- Conversation-id:** An empty text field.
- In-reply-to:** An empty text field.
- Reply-with:** An empty text field.
- Reply-by:** A text field with a "Set" button next to it.
- User Properties:** An empty text field.

At the bottom of the dialog are "OK" and "Cancel" buttons.

Figura 10 - Mensaje para rellenar y enviar

- Ejecutar el agente sniffer: Abre una nueva ventana donde aparece el agente sniffer. Es una agente que nos permite ver la evolución temporal del paso de mensajes en el sistema. Pinchando en las flechas, se nos abre una ventana parecida a la anterior que nos muestra el mensaje con todos sus parámetros. Obviamente podemos elegir los agentes de los cuales queremos ver los mensajes, y además guardar y cargar las listas de mensajes.

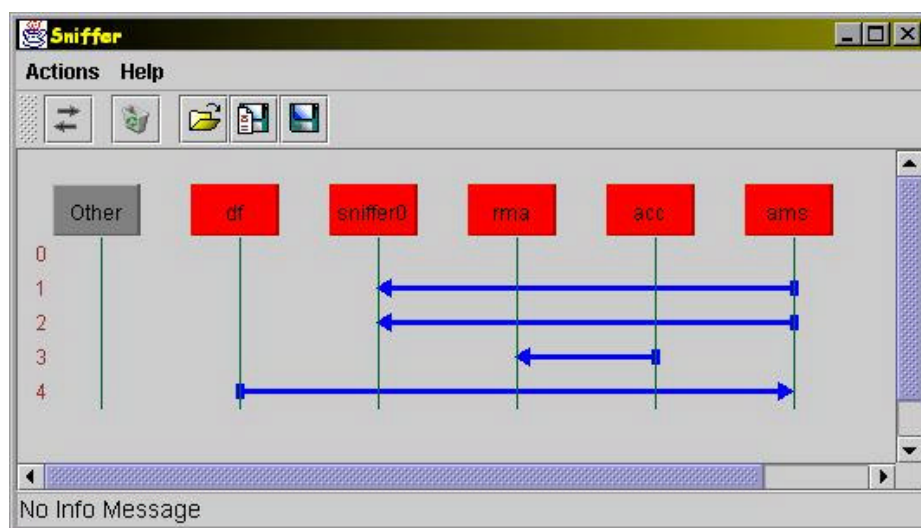


Figura 11 - Pantalla del sniffer

- Abre el Dummy-Agent. Es un agente que nos permite mandar mensajes. Es muy completo ya que nos muestra la lista de mensajes enviados y recibidos, nos permite recuperar mensajes anteriores, guardar y cargar listas de mensajes, etc.

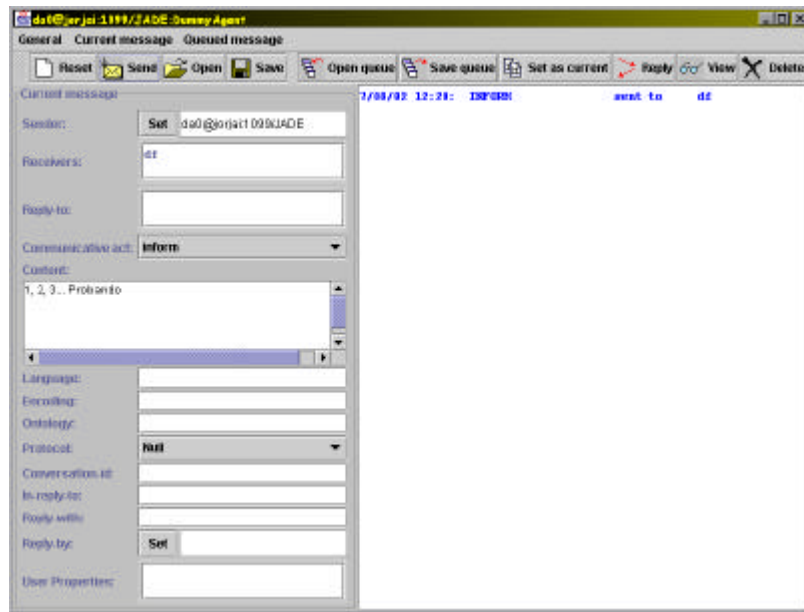


Figura 12 - Pantalla del Agente Dummy

- Aparte de esto, en el menú de herramientas, tenemos otra opción que es el activar la interficie del DF, donde se nos muestran los agentes registrados al DF, y podemos ver los servicios que nos ofrece cada uno.

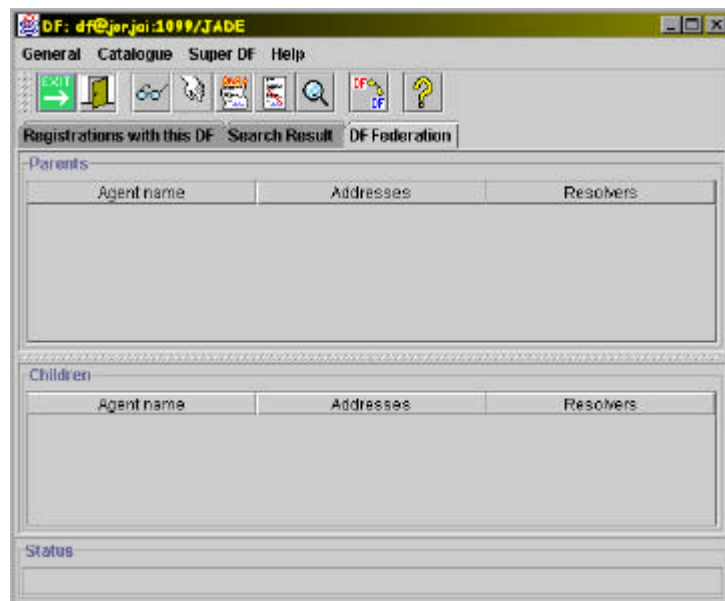


Figura 13 - Pantalla de la interficie del DF

Hay más opciones en la interficie gráfica, pero algunas de ellas están desactivadas. De hecho las más importantes las hemos comentado, y con ellas podemos manejar más que bien nuestro SMA.

4. DESCRIPCIÓN DEL SISTEMA MULTI-AGENTE IMPLEMENTADO

El objetivo del proyecto global de transplantes es presentar un prototipo general de ayuda a la coordinación del proceso de transplantes de órganos. En este proyecto final de carrera se ha diseñado y desarrollado un SMA para la gestión de la búsqueda de un receptor adecuado para un determinado órgano donado. Para ello usamos una serie de agentes que corresponden a los distintos niveles de búsqueda y que comentamos en la siguiente sección.

Se pretende hacer un prototipo de cómo sería el proceso si interviniesen los SMA. Aunque nuestro sistema se ejecuta sobre una sola máquina en este caso, JADE proporciona herramientas que facilitan la ejecución de agentes en distintas máquinas sin apenas cambios en el código implementado. En un entorno real se utilizaría esta posibilidad de distribución para permitir que cada hospital o coordinador tuviera su agente en su ordenador personal.

Entonces, podemos decir que el objetivo del proyecto es marcar las pautas para una posible implantación del sistema en el futuro. Al menos hacer una búsqueda para que luego según el resultado, un médico especialista pueda decidir sobre el paciente idóneo. Se trata pues de una especie de sistema de ayuda a la toma de decisiones.

A priori, el proyecto conjunto podría ser implantado en su globalidad, lo que sería lo mejor, o por partes. No sería difícil, una vez acabadas las tres partes del proyecto, juntarlas, ya que cada una de ellas tiene un coordinador general del sistema. La comunicación entre las tres partes sólo tendría que darse entre esos coordinadores, y los mensajes serían bastante simples.

También sería útil poder implantar alguna parte por separado. Por ejemplo, en nuestro caso, podemos hacer toda la gestión de búsqueda del paciente idóneo con nuestro sistema, y dejar el resto de etapas tal como están, bien sea manual o automáticamente.

4.1. Arquitectura

A partir de ahora me centraré en mi proyecto de coordinación de transplantes. La arquitectura del SMA, con los distintos agentes que lo componen y la interacción entre ellos es la que se puede ver en la figura 14.

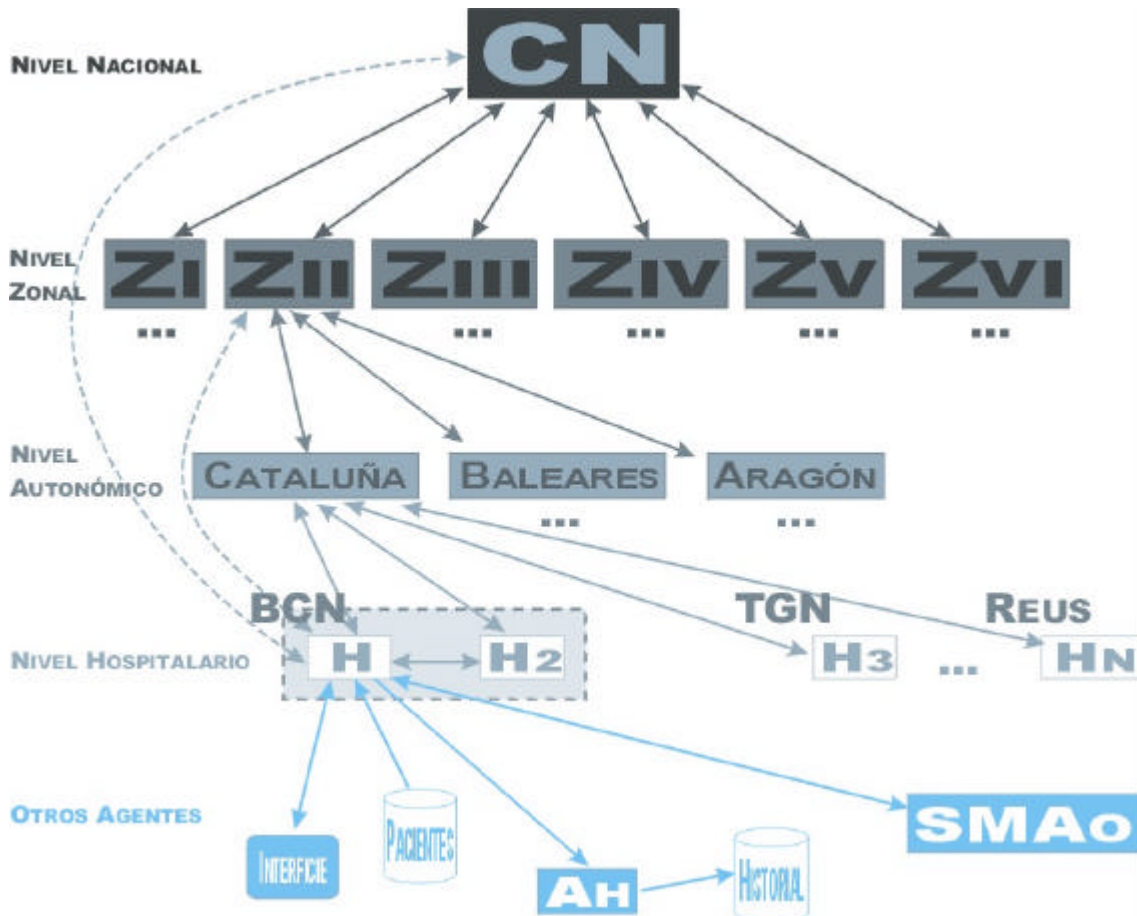


Figura 14 - Arquitectura del Sistema

La explicación de esta arquitectura dispuesta jerárquicamente en niveles, viene dada por el método actual de gestión de transplantes expuesto por la **ONT**, que ya hemos comentado anteriormente [Aldea et al., 01].

Es una estructura en forma de árbol, donde el padre de todos, el Coordinador Nacional (**CN**), no es la pieza más importante dentro de la jerarquía. Se trata de una búsqueda distribuida, donde los agentes de hospital (**H**) son los que coordinan el proceso y **CN** es una pieza que ayuda a que la búsqueda sea más rápida y equitativa, ya que procura que los órganos se repartan entre las distintas zonas cuando se llega a nivel nacional.

Quizás para entender un poco mejor esto y el porqué de la estructura haya que explicar con un poco más de detalle como funciona la búsqueda del paciente óptimo una vez tenemos un órgano disponible en nuestro propio hospital. El funcionamiento es el siguiente:

Antes que nada y para intentar agilizar los trámites, se intenta buscar un paciente adecuado en el propio hospital (**H**). Para ello se mira la base de datos de pacientes que esperan un órgano (**Pacientes**) y se intenta hallar un paciente compatible. Si se encuentra, se para la búsqueda y la donación queda registrada en la base de datos histórica (**Historial**). Si no se ha encontrado ningún paciente adecuado hemos de pasar al siguiente nivel. En este caso se trata de buscar en los hospitales de la misma ciudad, si los hay. La comunicación con éstos es directa, no trata con coordinadores intermedios, por lo que cada hospital ha de saber qué otros hospitales hay en la misma ciudad. Cuando no se encuentra ningún paciente en la ciudad se busca a través del Coordinador Autonómico (**A**) correspondiente. Lo que hace éste es buscar en todos los hospitales de su autonomía posibles pacientes. La búsqueda continúa hacia arriba si no se van encontrando pacientes, a nivel Zonal (**Z**) y finalmente se llega al nivel Nacional (**CN**).

Por lo tanto tenemos que el **CN** sólo actúa en caso de que no se halla encontrado ningún paciente en la zona donde ha sido donado el órgano (el territorio que abarca cada zona también está delimitado por la **ONT** y lo hemos visto en la figura 1). El Agente principal del SMA es el que controla cada Hospital.

Ahora pasaremos a describir cada uno de los elementos del SMA, así como las interacciones entre agentes. Primero se explican los agentes propios de nuestro sistema. Estos son el agente de Hospital (**H**), los tres tipos de coordinadores: Autonómico (**A**), Zonal (**Z**), y Nacional (**CN**) y el Agente Histórico (**AH**) (pese a haber varios agentes de tipo **A**, **Z** y **H** en el sistema, hay que remarcar que todos los agentes del mismo tipo son iguales, es decir están instanciados sobre la misma clase JAVA). Seguidamente se comenta el Sistema Multi-Agente Ordenador (**SMAO**), implementado por David Sánchez (en nuestro sistema sólo hay un **SMAO**, aunque si se ejecutara el sistema en varios ordenadores, habría uno por hospital), para finalizar con la estructura de las Bases de Datos que usa el sistema. Cada uno de estos agentes, sea del tipo que sea se registra al **DF** al iniciar su ejecución, de manera que permanece visible para el resto de agentes.

4.1.1. Agente de Hospital (H)

Este es el único agente que se comunica con el usuario del sistema. Para ello tiene una interficie que se describirá más adelante, en el apartado 6.2.

Es el agente principal del sistema, del que parten todas las peticiones y el que tiene que decidir sobre el resultado de la búsqueda. Aunque todos los hospitales se derivan de la misma clase, hemos de distinguir dos roles que puede adoptar un mismo hospital: el de hospital origen de la donación (al que llamaremos **H**) o el de hospital donde se buscan posibles receptores del órgano (**H2**).

Cuando se trata del origen de una donación, **H** recibe a través de una interficie una petición para buscarle receptor a un órgano. El primer paso ahora es buscar un paciente adecuado dentro del mismo hospital. En este momento, el hospital adopta los dos roles comentados antes, ya que es a la vez origen de la donación y posible destino del órgano (**H=H2**). Cuando es destino, se han de buscar en la base de datos del hospital

(**Pacientes**) todos los pacientes que están esperando el órgano en cuestión. Con éstos se crea una lista que se le pasa a **H**, en este nivel no hace falta enviarla porque los hospitales origen y destino son el mismo. **H** evalúa la lista, pasándosela a **SMAO** para que retorne una lista ordenada y evaluada según la compatibilidad entre el órgano y cada paciente (no profundizaremos sobre **SMAO**, aunque lo comentaremos más adelante, ya que se trata de un proyecto anterior [**Sánchez, 01**] adaptado para que pueda funcionar con este proyecto). Una vez **H** tiene la lista ordenada según la compatibilidad, ha de decidir si alguno de los pacientes es adecuado para el transplante. Esto se hace mediante un grado de idoneidad, que siguiendo la pauta marcada por **SMAO** puede ser [*óptimo, excelente, muy bueno, bueno, bastante bueno, favorable, adecuado, factible, flojo, desfavorable, bastante malo, malo, muy malo, fatal o pésimo*]. El paciente elegido, si lo hay, será el que obtenga mayor grado en la ordenación siempre que éste sea al menos igual que el grado de idoneidad elegido en la petición. Si se ha encontrado un paciente que cumple los requisitos, la búsqueda finaliza y se ha de informar al Agente Histórico (**AH**) de la donación. También se ha de modificar la **BD** de **H2** para indicar que el paciente escogido ya no está esperando el órgano.

En caso de que la búsqueda no finalice en el propio hospital, se ha de elevar el nivel de búsqueda. Se pasa a nivel de ciudad, si es que hay más hospitales a parte de **H** en la ciudad.

Para esta búsqueda aún no intervienen los coordinadores, se hace de forma directa, de hospital a hospital. Para ello **H** debe buscar todos los hospitales de la misma ciudad, para poder contactar con ellos. Esto se hace mediante el **DF**, es una herramienta de **JADE** que permite registrar a un agente con un cierto servicio y características, de modo que luego podamos buscar los agentes que hacen una u otra cosa. Es algo así como unas páginas amarillas. Con todos los hospitales obtenidos de la búsqueda en el **DF** se realiza un **Contract-Net**. Como ya hemos visto antes, el **Contract-Net** es un protocolo definido por la FIPA que nos permite comunicarnos con diversos agentes a la vez para que éstos efectúen propuestas para realizar una acción que se les pide. En este caso, los agentes de la misma ciudad (**H2, H3...**) enviarán las listas con los pacientes obtenidas de la base de datos de cada hospital. **H** realizará todo el proceso de evaluación mediante **SMAO** y finalmente avisará al hospital del que se ha escogido el paciente, si es que había alguno apto. Luego tendrá lugar la comunicación con **AH** y la modificación de la **BD** de **H2**, como antes.

Si aún no se ha encontrado paciente en los hospitales de la misma ciudad, se ha de pasar a nivel autonómico. Aquí la cosa cambia, porque entran en juego los coordinadores. A nivel de **H** la cosa es muy parecida. **H** le indica los datos del órgano al coordinador de su autonomía (**A**) y espera la lista de pacientes, que ahora contendrá todos los pacientes de la autonomía en cuestión que esperan ese tipo de órgano, excepto los de la propia ciudad (que ya han sido revisados). Con esta lista, **H** hará lo de siempre, evaluarla mediante **SMAO** y pasar el resultado a **A**, que a su vez será el encargado de pasar el resultado a los niveles inferiores.

Las búsquedas a niveles superiores se realizarán de la misma forma que con **A**, pero contactando con el coordinador al que corresponda el nivel de búsqueda (**Z** o **CN**). Con esto las tareas de **H** como hospital origen ya han quedado explicadas.

Lo que hace **H2**, es decir cuando un hospital actúa de posible destino del órgano, es siempre lo mismo a todos los niveles. Cuando se recibe el tipo de órgano donado, ya sea a través de **H** si es a nivel hospitalario, o a través de **A** si la búsqueda está en un nivel autonómico o superior, **H2** busca en su base de datos de pacientes aquellos que están esperando el órgano indicado. Envía esta lista a quien la esté esperando (**H** o **A**), y espera que le confirmen o rechacen la donación. En caso de confirmación se modifica la **BD** de **H2**, de tal modo que el paciente escogido quede marcado para que no pueda ser escogido de nuevo. Se ha tomado la decisión de marcar al paciente y no eliminarlo de la **BD** por si posteriormente hubiera complicaciones y no se pudiera completar el transplante. Entonces sólo se habría de desmarcar al paciente de la base de datos.

Tenemos pues que un hospital tiene comunicación con todos los tipos de agentes, como se veía en el diagrama: Con el resto de hospitales de la misma ciudad, con el SMA Ordenador, con el Agente Histórico y con los tres niveles de Coordinadores, Autonómico, Zonal y Nacional.

4.1.2. Agente Coordinador Autonómico (A)

Su función ya ha quedado un poco comentada al explicar **H**. De todos modos, ahora pasamos a explicar **A** en detalle.

Lo primero que hay que reseñar es que **A** no actúa nunca por cuenta propia, ni siquiera por cuenta del usuario. Se trata de un coordinador, y sólo entra en funcionamiento cuando no se encuentra ningún paciente a nivel inferior. Principalmente se comunica con hospitales, tanto del tipo **H** (que es el que activa su funcionamiento) como del tipo **H2** (donde se realiza la búsqueda), aunque también se puede comunicar con el Coordinador de Zona (**Z**) como mero intermediario.

Tenemos pues que **A** no toma decisiones, tan sólo pasa datos de un lado a otro, comunica a diversos agentes como intermediario, pero no evalúa pacientes ni decide sobre la donación. Todo eso es tarea de **H**. Cuando la búsqueda llega a nivel autonómico, **H** le envía a **A** el tipo de órgano donado. Lo que hace **A** entonces es buscar todos los hospitales de la autonomía, excepto los de la ciudad origen, en el **DF**. Una vez obtenida la lista de hospitales, **A** comienza un **Contract-Net** con todos ellos. Las listas que obtiene de todos ellos las junta en una sola para enviarla a **H**. **H** realiza la evaluación y elección del paciente adecuado y entonces **A** recibe esta elección. Ya sólo queda que **A** avise al hospital aceptado, si lo hay, de la elección.

Cuando se trata de una búsqueda a nivel superior, ya sea a nivel Zonal o Nacional, **A** recibe la petición desde el Coordinador Zonal correspondiente (**Z**). La función de **A** en este caso es similar a la anterior. Ha de buscar todos los hospitales de la autonomía, en este caso no hay excepciones porque se trata de una autonomía distinta a la de **H**, y comenzar con ellos un **Contract-Net**. El resto será igual que antes, sólo que en vez de comunicarse con **H**, ahora lo hará con **Z**.

Podemos concluir pues que la función de **A** es recolectar la lista de pacientes de los hospitales de su autonomía y pasárselas a **H** o **Z**, y una vez realizada la elección por parte de **H** comunicársela al hospital en cuestión.

4.1.3. Agente Coordinador Zonal (Z)

El funcionamiento de **Z** es muy similar al de **A**, en el sentido de que se trata de un intermediario y no toma decisión alguna. En este caso tenemos que **Z** se comunica con **H** cuando éste realiza una petición, con **A** para pasar la petición hacia abajo y con **CN** cuando la petición viene desde arriba. Veámoslo detalladamente.

Supongamos que la petición ha llegado ya al Nivel Zonal, es entonces cuando interviene **Z**. En ese momento recibe la petición desde **A**, indicándole el tipo de órgano donado. Lo que hace **Z** es muy parecido a lo que hacía **A**, es decir también realiza un **Contract-Net** con los agentes de nivel inferior, pero ahora éstos no son hospitales, sino Coordinadores Autonómicos. Se buscan todos los **A**'s pertenecientes a la zona en cuestión excepto la autonomía de la que procede el órgano y se inicia el **Contract-Net**. Como ya hemos visto **A** buscará en los hospitales de su nivel y enviará la lista de posibles pacientes a **Z**. **Z** tendrá que juntar en una todas las listas que le lleguen de las distintas autonomías y pasársela a **H**, para que evalúe y decida. Cuando se produce la elección, **H** se la indica a **Z**, que a su vez habrá de pasarla hacia abajo, indicando a la autonomía cuyo hospital sea el escogido la elección.

Si la búsqueda tampoco se satisface a Nivel Zonal, habrá que pasar al último nivel, el Nacional. La manera de actuar de **Z** es similar a la anterior, sólo que ahora la comunicación se produce con **CN** en lugar de con **H**. Es decir **CN** le pasará el tipo de órgano y esperará la lista de pacientes de toda la zona. Luego **CN** avisará de la elección a **Z** y éste tendrá que avisar al **A** correspondiente.

4.1.4. Agente Coordinador Nacional (CN)

Este es el último paso en la búsqueda, es la última alternativa que queda para encontrar un paciente adecuado. También se trata de la alternativa más lenta, ya que la búsqueda se produce en todas las zonas del país, excepto en la que proviene el órgano. Debido a esto, ahora se usa una técnica distinta. No se pregunta a todas las zonas a la vez, mediante un **Contract-Net** como veníamos haciendo hasta ahora, ya que se produciría mucha comunicación a la vez. Lo que se hace es mantener una especie de cola de prioridad de zonas, de tal manera que cuando una zona ha sido inspeccionada, ésta se pasa al final de la cola, para dar oportunidades a todas las zonas y que los órganos no vayan siempre a parar a las mismas. Detallamos ahora el funcionamiento:

Una vez hemos llegado a Nivel Nacional, **H** envía el tipo de órgano a **CN**. Éste ahora ha de buscar los pacientes que tiene por debajo excepto los de la zona de origen, pero como ya hemos dicho no lo hace con todas las zonas a la vez. Ahora **CN** saca de la lista ordenada (según búsquedas anteriores) la primera zona a explorar. Se le pasa el tipo de órgano al Coordinador de esa zona y se espera la lista con los pacientes. Ésta se le pasa a **H** y se espera la elección. En caso de que haya algún paciente apto, sólo queda avisar a **Z** y concluye la función de **CN**. Si por el contrario aún no se ha hallado paciente se ha de proseguir la búsqueda. Para ello pasamos al final de la lista ordenada la zona que ya hemos comprobado y proseguimos con la siguiente, que ahora es primera en la cola. Hacemos lo mismo que con la anterior zona y vamos examinando zonas hasta que **H** encuentre algún paciente apto. Si se han revisado todas las zonas sin encontrar pacientes, la búsqueda concluye con resultado negativo. No se ha encontrado

ningún paciente apto en todo el país. La única alternativa que tiene ahora **H** para darle una salida a su órgano es rebajar el grado de idoneidad y reanudar la búsqueda, con lo que probablemente sí que se obtengan resultados.

4.1.5. Agente Histórico (AH)

Este agente es el encargado de guardar un historial de las donaciones realizadas. Se comunica únicamente con **H**, una vez se ha concluido la búsqueda de un paciente satisfactoriamente.

Cuando acaba la búsqueda encontrando un paciente adecuado, **H** avisa al hospital de dicho paciente (**H2**), ya sea directamente o mediante la jerarquía de coordinadores. Cuando **H2** confirma la donación a **H**, éste avisa al Agente Histórico de los datos del órgano y el paciente, para que pueda incluir la donación en el historial. El funcionamiento de este Agente es muy simple, y sólo aporta funcionalidades estadísticas al sistema.

4.1.6. SMA Ordenador (SMAO)

Ya hemos comentado anteriormente que este SMA fue realizado por David Sánchez como parte de su PFC de la Ingeniería Técnica [Sánchez, 01]. No obstante, ha necesitado de alguna modificación para poder ser utilizado conjuntamente con el proyecto actual. Nos comunicamos con uno de sus agentes y el mismo nos aporta el resultado que necesitamos, en este caso la ordenación y valoración de los pacientes según los criterios requeridos.

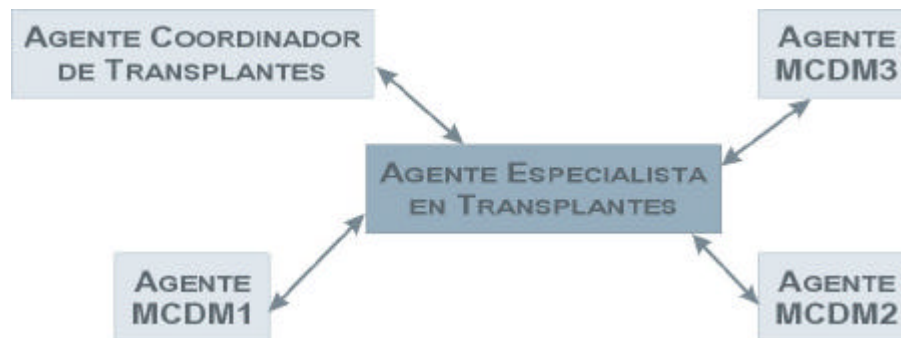


Figura 15 - Estructura interna del SMAO

Internamente el sistema tiene la estructura de la figura 15. Existe un agente central denominado **Especialista** que es el que se comunica con nuestro sistema, que en este caso correspondería al **Agente Coordinador de Transplantes** de un hospital. Este agente especialista, a parte de encargarse de la comunicación, también realiza un filtraje sobre los datos que le llegan del hospital y los transforma para que puedan ser usados por los agentes **MCDM**. El resto de agentes del **SMAO** son los que realizan la evaluación y ordenación de los pacientes. En la figura aparecen varios, aunque de momento en el sistema actual tan solo está implementado uno de ellos, el **ClusDM**. El sistema está pensado para poder añadir más agentes **MCDM** (Multi-Criteria Decisión Making) sin demasiados problemas. El agente **ClusDM** está basado en Funciones de

Negación y Clustering. Se pueden ver más detalles en la documentación del proyecto de David Sánchez [Sánchez, 01].

La parte que se ha tenido que modificar para poder integrar el **SMAO** es la que se refiere a la comunicación del SMA con el exterior, ya que antes se hacía mediante una interficie. Lo que necesitábamos en nuestro caso es que la comunicación se llevara a cabo mediante mensajes entre **H** y **SMAO**. En primer lugar, **H** le envía a **SMAO** la lista con todos los pacientes obtenidos de la base de datos y todas sus características. Usando éstas, **SMAO** ordena estos pacientes y otorga un grado de idoneidad a cada uno que va desde *óptimo* a *pésimo*. Luego **H** usará este grado de idoneidad impuesto por el usuario y escogerá el mejor paciente que tenga como mínimo la calificación exigida.

4.1.7. Bases de Datos

Hay tres tipos de bases de datos en el Sistema:

- **Base de Datos de Hospital (Pacientes):** Se trata de la BD donde se guarda la información de todos los pacientes de un hospital. A partir de ella se pueden hacer consultas para buscar los pacientes que esperan un determinado tipo de órgano. Consta de los siguientes campos:

- Identificador del paciente
- Tipo de órgano que espera el paciente
- Hospital del paciente
- Localidad del hospital
- Provincia del hospital
- Tipo de sangre del paciente
- Peso del paciente
- Tamaño del órgano
- Antígenos del paciente [B0, B1, DR0, DR1]
- Fecha de nacimiento del paciente
- Fecha en la que el paciente ingresó en la lista de espera

Además de éstos existe un último campo que indica si el paciente ha sido seleccionado para una donación o no. Es la marca que comentábamos antes cuando finaliza una búsqueda.

Existe una BD de **Pacientes** para cada Hospital registrado en el sistema, y es el propio hospital el encargado de mantener esa BD. Al registrar un Hospital se le pregunta el nombre de la BD asociada, que será con la que trabajará el Agente que represente a dicho Hospital. La única comunicación que tiene la BD de **Pacientes** es con el Hospital asociado cuando éste actúa con el rol de **H2**.

- **Base de Datos del Agente Histórico (Historial):** En este caso sólo tenemos una BD de este tipo. Aquí es donde se guarda el historial de donaciones comentado antes. Es manipulada únicamente por **AH**, que realiza altas de donaciones. La operación de dar de alta una donación es la única que se realiza de forma automática por el sistema.

Consta de los siguientes campos:

- Tipo de órgano
- Fecha de la donación
- Hospital del donante
- Localidad del donante
- Provincia del donante
- Tipo de Sangre del donante
- Peso del donante
- Tamaño del órgano donado
- Antígenos del donante
- Fecha de nacimiento del donante
- Identificador del receptor dentro del hospital
- Hospital del receptor
- Localidad del receptor
- Provincia del receptor
- Tipo de sangre del receptor
- Peso del receptor
- Tamaño del órgano del receptor
- Antígenos del receptor
- Fecha de nacimiento del receptor
- Fecha de espera del receptor

- **Base de Datos de Ciudades (BD-Ciudades):** Cada agente (hospital o coordinador) necesita acceder a una cierta información “geográfica”. Los hospitales necesitan saber a qué zona y autonomía pertenecen, las autonomías saber los hospitales que tienen por debajo, etc. Para simplificar y agilizar los agentes y evitar la introducción manual de muchos datos, hemos realizado una única BD que podrán consultar todos los agentes, en la que se define la jerarquía de zonas, autonomías y ciudades existentes en el sistema. El objetivo de esta BD es que un agente pueda saber a qué autonomía o zona pertenece un hospital, sabiendo la ciudad en la que se encuentra.

Consta de los siguientes campos:

- Ciudad
- Provincia
- Autonomía
- Zona

Esta BD no ha de modificarse, en principio es sólo para realizar consultas. Deberían estar incluidas todas las ciudades con hospitales que se dedican a donaciones de órganos, pero por simplicidad para este proyecto se han incluido sólo las capitales de provincia y alguna ciudad más.

Si finalmente el Sistema se llevara a la realidad y se ejecutara en distintas máquinas, ya no se usaría esta BD, sino que cada tipo de agente tendría internamente la información que necesita.

4.2. Ontología

Ya se ha definido anteriormente lo que es una ontología y los campos que puede tener. Ahora pasamos a describir la ontología propia de este proyecto. Hay que destacar que en la comunicación interna del SMA Ordenador se usa la ontología propia del PFC de David Sánchez, que sin embargo no detallaremos aquí.

Hay varios tipos de mensajes que contendrán objetos definidos en la ontología. La información que se intercambiará para cada tipo de mensaje es:

- **Request** de **H** a los coordinadores (**A**, **Z** y **CN**), **CFP** entre coordinadores (**Z** y **A**), **CFP** entre **A** y los hospitales de su autonomía y **CFP** entre hospitales de la misma ciudad:
 - ✓ Tipo de órgano donado
- **Request** de **H** a **SMAO** e **Inform** del resto de hospitales a **H**, pasando o no por los coordinadores (**A**, **Z** y **CN**):
 - ✓ Lista de pacientes:
 - Número de pacientes
 - Datos de los pacientes:
 - Id
 - Hospital
 - Localidad
 - Tipo de órgano
 - Tipo de Sangre
 - Provincia
 - Peso
 - Tamaño
 - Antígenos B y DR
 - Fecha de nacimiento
 - Fecha de entrada en lista de espera
 - Pesos de los atributos
- **Inform** de **SMAO** a **H** después de realizar la evaluación de pacientes:
 - ✓ Resultado de una ordenación de **SMAO**:
 - Identificador de paciente
 - Valoración del paciente
- **Inform** de **H** al resto de hospitales pasando o no por los distintos coordinadores (**A**, **Z** y **CN**):
 - ✓ Aviso del paciente escogido:
 - Identificador del paciente
 - Hospital del paciente

Además, hará falta definir una serie de objetos que indiquen situaciones de error que se puedan producir:

- Que **SMAO** no pueda evaluar una lista de pacientes, por cualquier motivo.
- Que un determinado agente (coordinador u hospital) no esté registrado en el sistema.
- Que no se encuentren pacientes esperando un cierto órgano en la BD de un hospital.
- Que no se encuentren hospitales en una determinada autonomía.

A continuación se muestra el esquema gráfico de nuestra ontología:

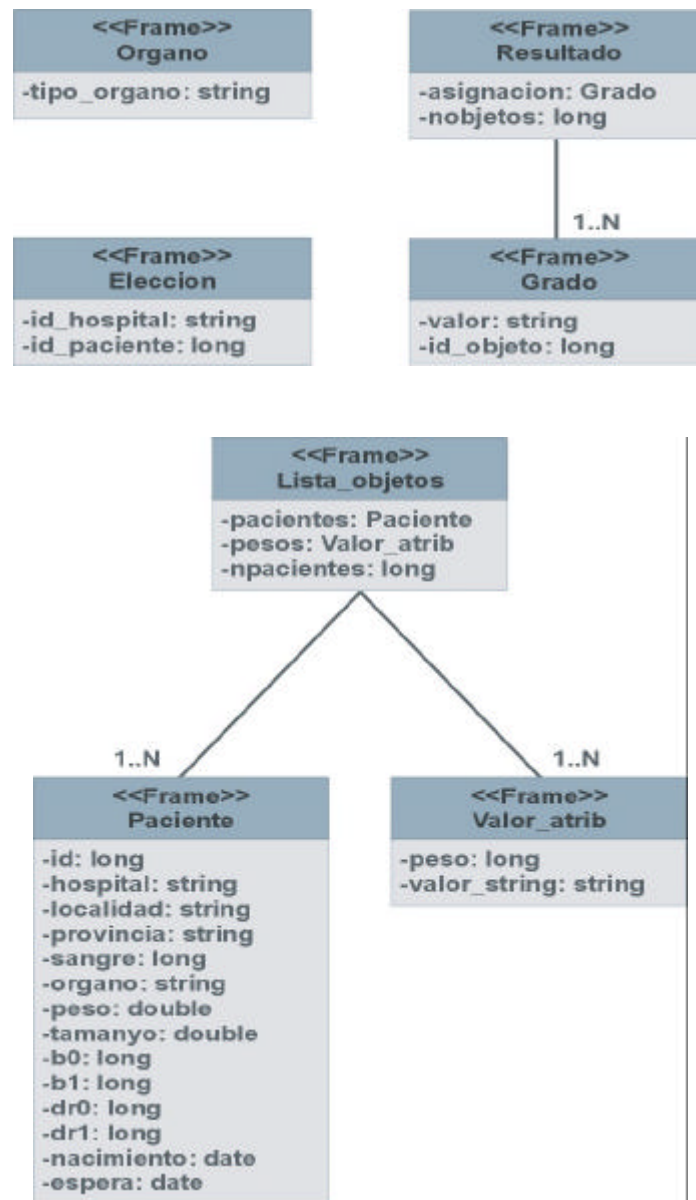


Figura 16 - JBSOntology

Por último, faltan los errores que puede haber en el sistema, también definidos en la ontología.



Figura 17 - JBSOntology (errores)

4.3. Protocolos de Comunicación

Hemos comentado de pasada los protocolos que se usan en la implementación del SMA. En concreto hacemos uso de dos de los protocolos definidos por la FIPA, como son el **FIPA-Contract-Net Protocol** y el **FIPA-Request Protocol**. Veamos ahora qué agentes y en qué momento usan estos protocolos.

- **FIPA-Contract-Net Protocol**: Lo usan diversos agentes para pedir a otros que realicen una acción. La ventaja de este protocolo es que se contacta con varios agentes a la vez, pidiendo una propuesta para que realicen una determinada acción. Una vez ellos realizan la propuesta, el agente iniciador puede elegir la que más le convenga. En nuestro caso el agente iniciador puede ser **H**, **A** o **Z**, y los agentes que realizan la acción serán **H2** o **A**.

En el primer caso, a nivel local, **H** contacta con todos los hospitales de la misma ciudad (**H2**) mediante este protocolo. Hace lo mismo **A** a nivel autonómico. A nivel zonal, **Z** contacta con todos los Coordinadores de Autonomía, del mismo modo. En estos tres casos, la acción que se pide realizar es que se devuelva una lista con los posibles pacientes. La precondition viene indicada por el tipo de órgano, que es el contenido del mensaje iniciador del protocolo. La elección del agente que realizará la acción viene dada a través de la elección del paciente adecuado por el hospital origen.

- **FIPA-Request Protocol**: Este protocolo tiene usos más diversos dentro de nuestro sistema. La característica principal es que un agente le pide a otro que realice una acción, y éste responde de manera afirmativa o negativa dependiendo de si la ha podido realizar o no. La comunicación se realiza sólo entre dos agentes, a diferencia del **Contract-Net** que incluye un mayor número.

En primer lugar es el protocolo usado para la comunicación con el Agente Histórico. Como hemos dicho se trata de una comunicación muy simple, pero siguiendo el protocolo ya establecido. **H** le indica a **AH** los datos de la donación, éste realiza la inserción en su base de datos y confirma a **H**.

Otra comunicación siguiendo este protocolo es cuando **H** quiere comunicarse con cualquiera de los tres niveles de coordinación (**A**, **Z**, o **CN**). El mensaje que envía **H** incluye el tipo de órgano, y se espera una lista con los datos de los posibles pacientes a obtener el órgano. El coordinador en cuestión realiza la búsqueda internamente como haga falta (con el **Contract-Net**) y retorna el resultado a **H** siguiendo el protocolo **Request**. Posteriormente **H** avisará al coordinador de la elección del paciente adecuado.

Finalmente tenemos la comunicación entre **CN** y **Z** cuando la búsqueda se encuentra a nivel nacional. En este caso **CN** se comunica con los coordinadores de zona uno a uno, porque la búsqueda va por turnos. Esto hace que sea mejor utilizar el **Request Protocol** que el **Contract-Net Protocol**.

Hasta aquí se han expuesto las razones del uso de uno u otro protocolo. Pero hay momentos en que un agente mantiene una conversación con dos agentes a la vez, uno de nivel superior y otro de nivel inferior. En estos momentos es preciso mezclar dos protocolos de tipo **Request** o un **Request** con un **Contract-Net**. Lo que se ha

implementado para poder hacer esto es una especie de simulación de un protocolo dentro del otro. Es decir, supongamos que **A** está manteniendo una conversación con **H** siguiendo el protocolo **Request** como hemos visto. Tenemos que **A** deberá comunicarse con los hospitales de su autonomía (**H2**) mediante un protocolo **Contract-Net**, como también hemos visto. ¿Qué sucede aquí? Que el mismo agente mantiene dos conversaciones a la vez con distintos agentes y distintos protocolos. Esto no sería problema si no se solaparan ambas conversaciones, pero en este caso se da ese solapamiento. La respuesta que **A** le dé a **H** en el **Request** depende de las que los **H2** le hayan dado a **A** en el **Contract-Net**, y aún en ese momento no habrá finalizado ninguno de los dos protocolos. La solución es continuar el **Request** con **H** simulándolo dentro del **Contract-Net**. Esto es factible porque sabemos qué tipo de mensaje espera **H** y qué tipo de mensaje devolverá a **A**, ya que se sigue el protocolo **Request**.

Lo que sucede en JADE es que al añadir un comportamiento que extiende un protocolo, éste se va ejecutando, pero no podemos pausarlo ni reanudarlo cuando nosotros queramos. Es decir, en nuestro caso tenemos un momento en que ninguno de los dos protocolos ha finalizado, pero ambos necesitan que el otro continúe porque necesitan datos el uno del otro. Una posible solución sería implementar una serie de semáforos o pausar un comportamiento, pero esto no se pudo realizar, debido a que se pausa todo el agente, no sólo el comportamiento deseado. La solución adoptada es la siguiente:

Suponemos de nuevo que tenemos una comunicación entre **H** y **A**, mediante un protocolo **Request**. **H** le envía los datos de un donante con un mensaje de tipo *request* a **A**. Ahora **A** tiene que iniciar un protocolo **Contract-Net** con los hospitales de su autonomía para buscar pacientes, pero al mismo tiempo tiene que seguir la conversación con **H** cuando tenga las listas de pacientes (obtenidas mediante el **Contract-Net**). Lo que se hace es olvidarse del **Request protocol** como tal, una vez que desde éste se inicia el **Contract-Net** (es decir, el **Request Protocol** que se extiende de la clase del JADE acaba en seco). Luego, dentro del **Contract-Net** se envían y reciben los mensajes que continúan el protocolo **Request**, cambiando los *handlers* automáticos que nos proporciona el JADE por unas funciones que reciben los mensajes siguiendo el protocolo. Durante el **Contract-Net** **A** envía el mensaje *agree* correspondiente al **Request protocol** que contiene la lista de pacientes y recibe un *inform* proveniente de **H** con el paciente escogido, si lo hay. Luego se puede proseguir con el **Contract-Net**, enviando **A** los mensajes de aceptación o rechazo a los hospitales correspondientes. Posteriormente **A** finaliza el protocolo **Request** (aún dentro del **Contract-Net**) enviando un *inform* de confirmación de la elección a **H**. Con esto conseguimos toda la estructura del protocolo **Request** de la figura 6.

Lo mismo sucede con los otros coordinadores que tienen que comunicarse con **H** y con los niveles inferiores a la vez.

4.4. Seguimiento del proceso completo

Hemos ido viendo la función de cada agente y la comunicación con el resto, pero para dejar claro el funcionamiento global del sistema lo mejor es hacer el seguimiento completo de todo el proceso.

Una vez se inicia el sistema, el usuario introduce los datos del órgano donado mediante la interficie de la figura 18. Más adelante se detallan los campos de dicha interficie.

Figura 18 - Interficie de usuario

Al pulsar el botón de **Enviar Petición**, lo que se hace es que **H** inicie la búsqueda del paciente adecuado para el órgano en cuestión. **H** obtiene los datos del órgano desde la interficie.



Figura 19 - Primer paso de la búsqueda

Una vez que **H** sabe los datos del órgano, tiene que buscar un paciente dentro del mismo hospital. Para ello el primer paso es buscar en la base de datos del hospital los pacientes que esperan un órgano del tipo del que se ha donado.



Figura 20 - Búsqueda de posibles pacientes en la BD

Las comunicaciones con la interficie y con la BD de pacientes son internas, por lo que no se intercambian mensajes entre agentes hasta ahora. Cuando **H** tiene la lista con los posibles pacientes que optan al órgano, ha de ordenarla y evaluarla. Aquí es donde interviene el SMA Ordenador, y por tanto la primera comunicación entre agentes.



Figura 21 - Ordenación y Evaluación de pacientes por SMAO

Ahora **H** ya puede decidir sobre los posibles pacientes de su hospital, porque ya los tiene todos evaluados. Supongamos que no hay ninguno con la puntuación **Óptimo**, que era el grado de idoneidad que se pedía en la interficie. Hacemos esta suposición para que la búsqueda no finalice en este nivel y podamos ver el seguimiento completo del proceso, a todos los niveles.

Tenemos pues que no se ha encontrado ningún paciente apto en el propio hospital. Esto hace que hayamos de pasar al siguiente nivel, comprobar todos los hospitales de la ciudad, si los hay. En este caso suponemos que estamos en Tarragona y hay más hospitales en dicha ciudad, concretamente **H2** y **H3**. Primero de todo, hay que proceder a buscar estos hospitales, para poder comunicarse con ellos. Esto se hace mediante una búsqueda en el **DF** de los hospitales registrados en Tarragona.



Figura 22 - Búsqueda en DF de hospitales de Tarragona

Una vez se ha encontrado la lista de hospitales de Tarragona hemos de comunicarnos con ellos para proseguir la búsqueda. Para ello se usa el protocolo **Contract-Net**. En este caso **H** actuará como *initiator* y **H2** y **H3** como *responders*. Entre medias de este Contract-Net, **H** también se tendrá que comunicar con **SMAO** para evaluar la lista de pacientes que le pasen los dos hospitales de Tarragona. Tenemos que al principio **H** manda un **CFP** a los hospitales de Tarragona indicando el tipo de órgano donado, un corazón.

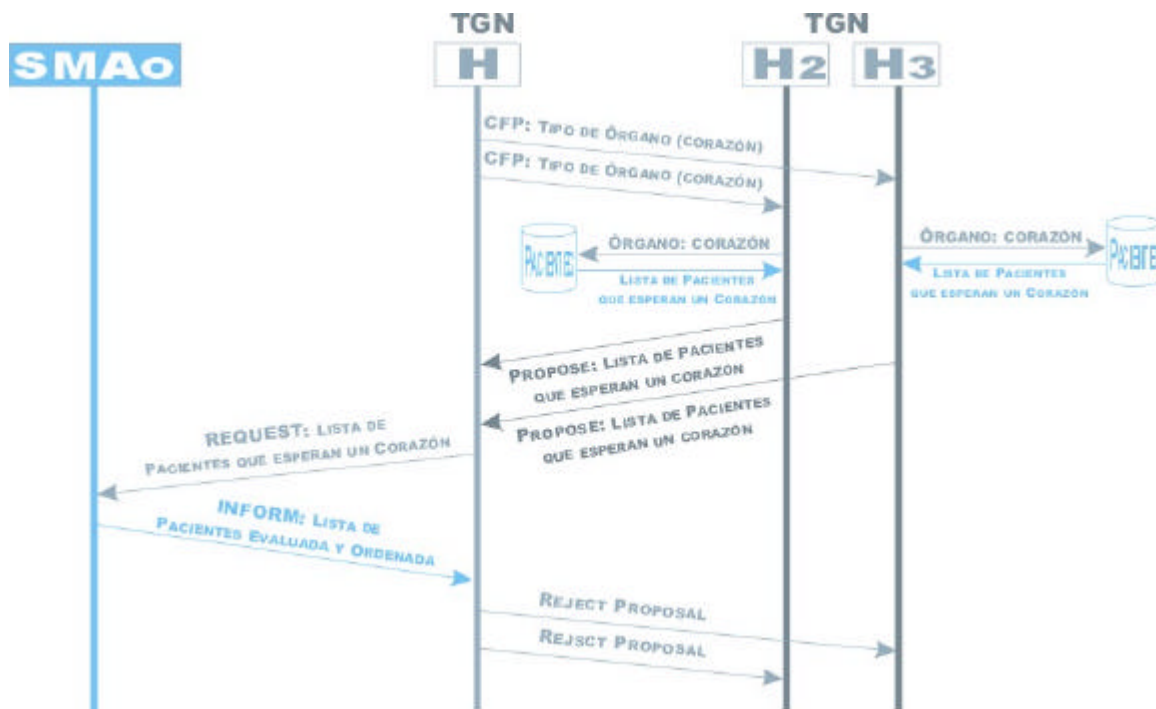


Figura 23 - Contract-Net dentro de la misma ciudad (Tarragona)

Cuando **H2** y **H3** reciben el **CFP** de parte de **H**, han de buscar en sus respectivas BD los pacientes que esperan un corazón, estas listas son la que le pasarán a **H** para que decida sobre el paciente más adecuado. Esto también se ve en la figura 23. **H** junta las listas que provienen de **H2** y **H3** y las ordena y evalúa mediante **SMAO**. Tenemos en este caso que sigue sin encontrarse un paciente *óptimo*, que era lo que buscábamos. Esto hace que **H** tenga que rechazar las propuestas de los dos hospitales, mediante un **Reject-Proposal**. A partir de ahora habremos de subir al nivel autonómico para seguir la búsqueda.

Para realizar la búsqueda a nivel autonómico, lo primero que hemos de saber es a qué autonomía pertenece nuestro hospital y qué coordinador de autonomía le pertenece. Esto se realiza mediante una búsqueda en la **BD-Ciudades** para saber la autonomía y otra en el **DF** para saber el Coordinador.



Figura 24 - Búsqueda del Coordinador de Autonomía de Tarragona

Ahora hemos de iniciar un **Request Protocol** con el coordinador de autonomía, en este caso **A1**. Para ello lo primero que tiene que hacer **H** es enviar un mensaje de tipo **request** con el tipo de órgano donado a **A1**.



Figura 25 - Inicio del Request Protocol entre H y A1

Una vez **A1** sabe el tipo de órgano, ha de buscar los hospitales que pertenecen a su autonomía, exceptuando los que sean de la ciudad origen, Tarragona. Para ello se usan una vez más la **BD-Ciudades** y el **DF**, como se muestra a continuación.

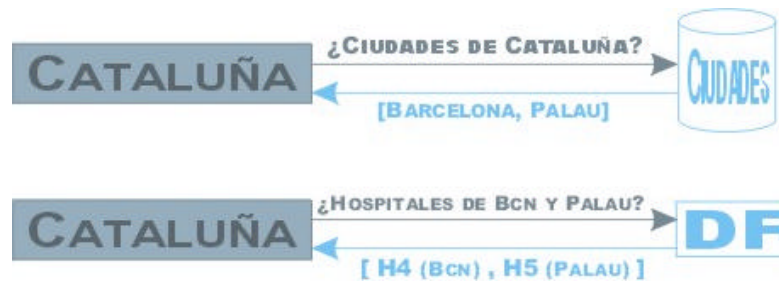


Figura 26 - Búsqueda de Hospitales de Cataluña, excepto los de Tarragona

Con la lista obtenida, **A1** ya puede empezar el **Contract-Net** con estos hospitales. El siguiente paso es mandar un mensaje de tipo **CFP** a todos estos hospitales conteniendo el tipo de órgano (corazón). Ahora los hospitales **H4** y **H5** actúan de la misma forma que lo hicieron **H2** y **H3**, respondiendo al **CFP** de **A1** con la lista de posibles pacientes recogida de la BD de su hospital. Cuando **A1** obtiene las listas de pacientes de estos hospitales, las junta y se las envía a **H**, siguiendo el protocolo **Request** iniciado antes. Todos estos pasos se ven en la figura 27.

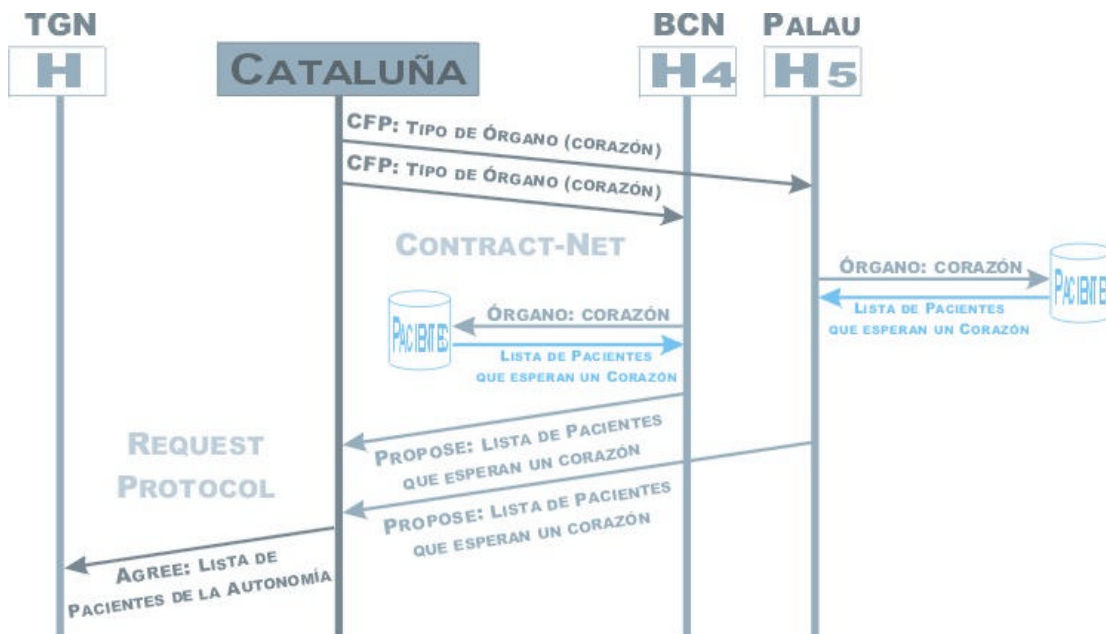


Figura 27 - Contract-Net a nivel autonómico y continuación del Request Protocol

En este momento, **A1** queda pendiente de recibir la elección del paciente adecuado (si lo hay) por parte de **H**. En este caso, seguimos sin encontrar un paciente de las características requeridas por el usuario, por lo que el mensaje de elección enviado por **H** no contiene el identificador de ningún paciente, sino que indica que no se ha encontrado ninguno adecuado.

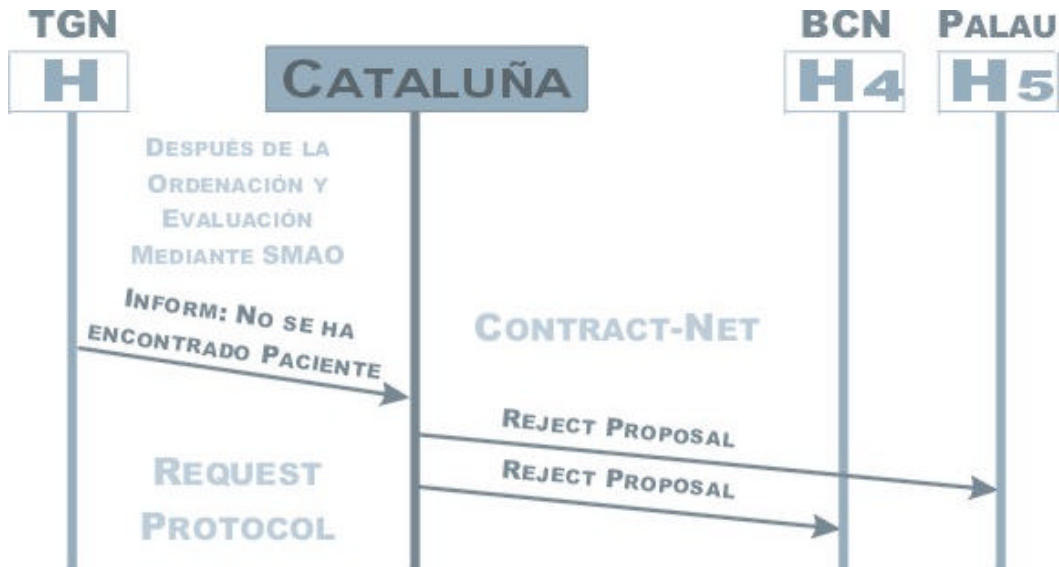


Figura 28 - Final del Contract-Net y el Request-Protocol

Al no haber ningún paciente escogido, **A1** tendrá que rechazar todas las propuestas de los hospitales de Cataluña, tal y como se muestra en la figura 28. Ahora la búsqueda vuelve a subir de nivel, esta vez buscaremos en la zona correspondiente.

Una vez más combinamos las búsquedas en **BD-Ciudades** y en el **DF** para obtener el nombre del coordinador, esta vez coordinador de zona.

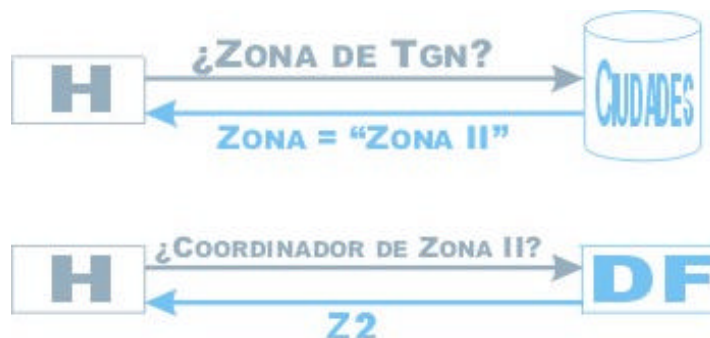


Figura 29 - Búsqueda del Coordinador de la Zona de Tarragona

Ahora **H** iniciará el **Request Protocol** con **Z2**, indicándole como siempre que el órgano donado es un corazón. Una vez hecho esto, **Z2** buscará los coordinadores de autonomía que pertenecen a su zona.

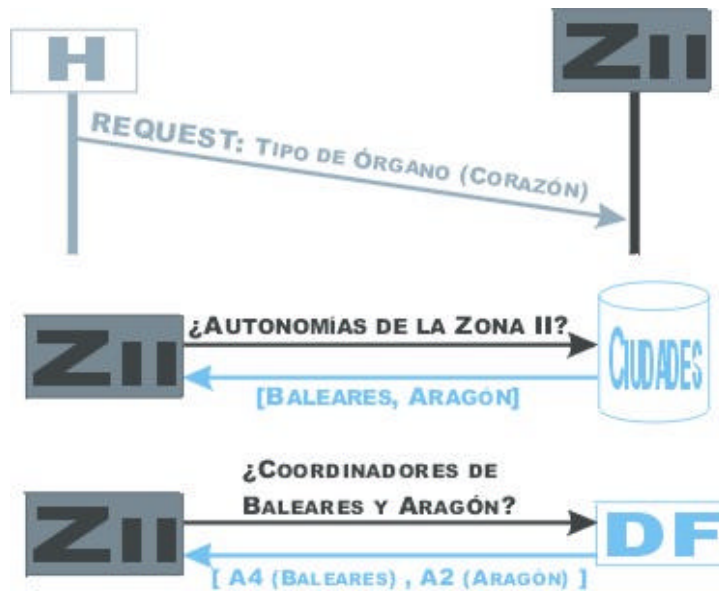


Figura 30 - Inicio del Request Protocol entre H y A2

La búsqueda de los coordinadores de autonomía se hace de la misma forma que antes, combinando búsquedas en **BD-Ciudades** y en el **DF**. En este momento **Z2** ya puede iniciar el **Contract-Net** con **A4** y **A2**, para que éstos a su vez inicien otro con los hospitales de sus autonomías. Esto queda más detallado en la figura 31.

Tenemos que cuando los coordinadores de Aragón y Baleares reciben el tipo de órgano por parte de **Z2**, buscan mediante **BD-Ciudades** y el **DF** los hospitales de la autonomía en cuestión.

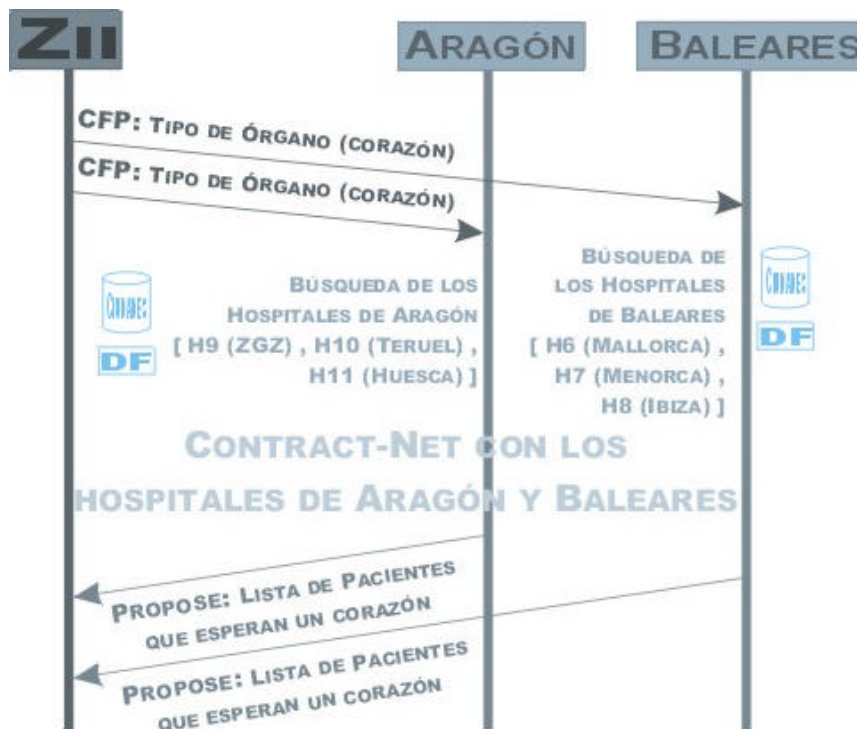


Figura 31 - Contract-Net entre Z2 y [H4,H5] a nivel zonal

Tras realizar el **Contract-Net** con ellos, se obtienen las listas de pacientes que esperan un corazón de cada ciudad. Cada Coordinador autonómico junta las listas de pacientes obtenidas de los distintos hospitales y las pasa a **Z2**. Éste a su vez vuelve a juntar las listas y las pasará al hospital origen en el siguiente paso del **Request**, que vemos en la siguiente figura.

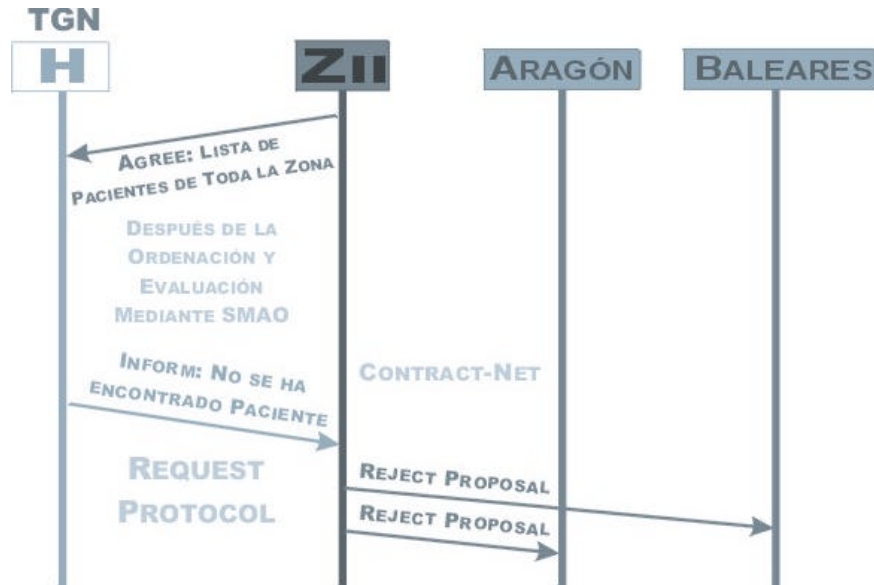


Figura 32 - Continuación del Request Protocol entre H y Z2

Como siempre, **H** evalúa los pacientes mediante **SMAO** y devuelve la elección a **Z2**. Volvemos a no encontrar ningún paciente óptimo, por lo que **Z2** tiene que rechazar todas las propuestas de los coordinadores autonómicos de su zona (**A2** y **A4**). El último paso y que no se ve en la figura 32 sería el rechazo por parte de **A2** y **A4** de todas las propuestas de los hospitales del nivel inferior mediante mensajes de tipo **Reject-Proposal**.

Una vez llegados a este paso, sólo nos queda un nivel por examinar, el nivel nacional. En este momento **H** ha de comunicarse con **CN**, también mediante un protocolo de tipo **Request** indicándole el tipo de órgano donado. Lo que ahora variará es la forma de comunicación entre **CN** y los coordinadores inferiores, ya que no se realizará mediante un protocolo **Contract-Net** con todos a la vez como estábamos acostumbrados. Como se comentó en la explicación de **CN**, esta última búsqueda se realiza zona por zona. Por tanto la comunicación entre el Coordinador Nacional y los de Zona se realiza también mediante un **Request Protocol**. Las zonas están ordenadas en una cola que actualiza **CN**. Esta cola cambia cuando se realiza la búsqueda en una zona, pasando ésta al final de la cola, y dando las mismas opciones a todas las zonas. Esto queda mejor explicado viendo la figura 34.

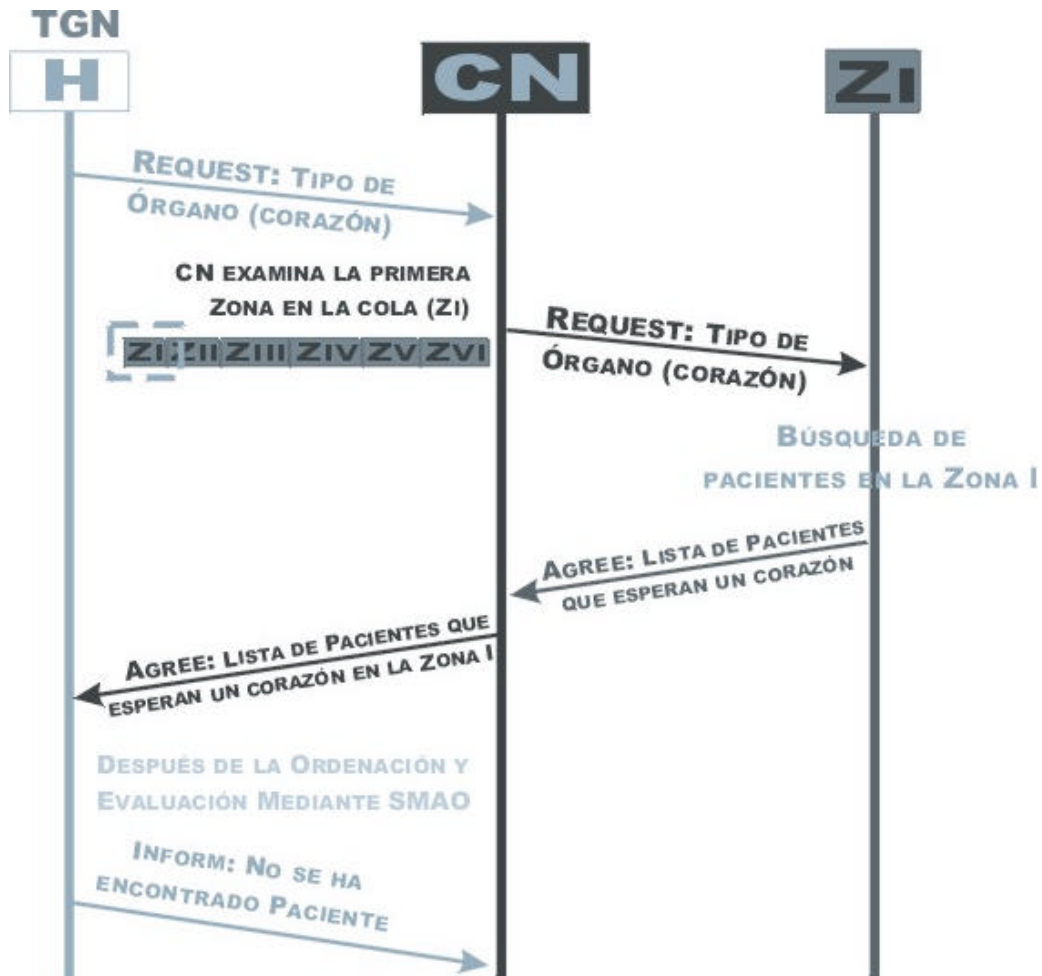


Figura 33 - Primeros pasos de la búsqueda a nivel nacional

La comunicación entre el Coordinador de Zona y los niveles inferiores se realiza de la misma forma que en anteriores búsquedas, sólo que ahora la lista de pacientes no se pasa directamente a **H**, sino que **CN** realiza de intermediario.



Figura 34 - Actualización de la cola de Zonas por parte de CN

Como se ve en la figura 33 cuando **CN** pasa la lista a **H**, éste la evalúa como siempre y retorna el resultado a **CN**. Suponemos que aún no encontramos paciente para ver otra particularidad de la búsqueda. Ahora tocaría examinar, por el orden de la cola, la Zona II, pero como es la zona origen, ésta pasa directamente al final de la cola y se procede a examinar la Zona III, donde finalmente sí encontraremos un paciente óptimo.

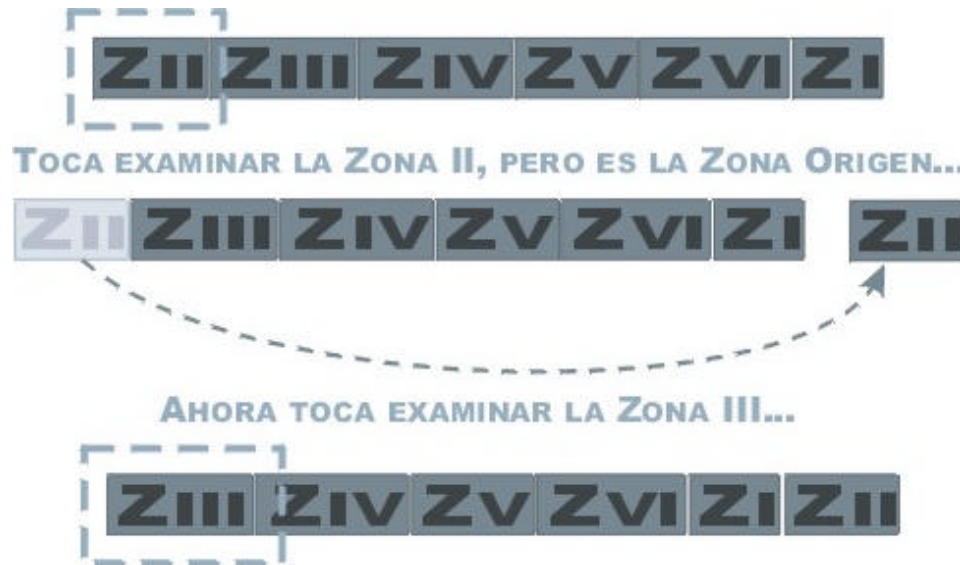


Figura 35 - La Zona II, ya examinada, se pasa por alto

Se realiza ahora el **Request** entre **CN** y **Z3**, de igual forma que antes. Finalmente obtenemos que **H** encuentra un paciente que reúne las condiciones acordadas en la Zona III. Avisa a **CN**, que a su vez avisa a **Z3**, éste al coordinador de la autonomía correspondiente, y finalmente al Hospital escogido. Cada uno de estos pasos se realiza siguiendo el protocolo de comunicación habitual para cada nivel.

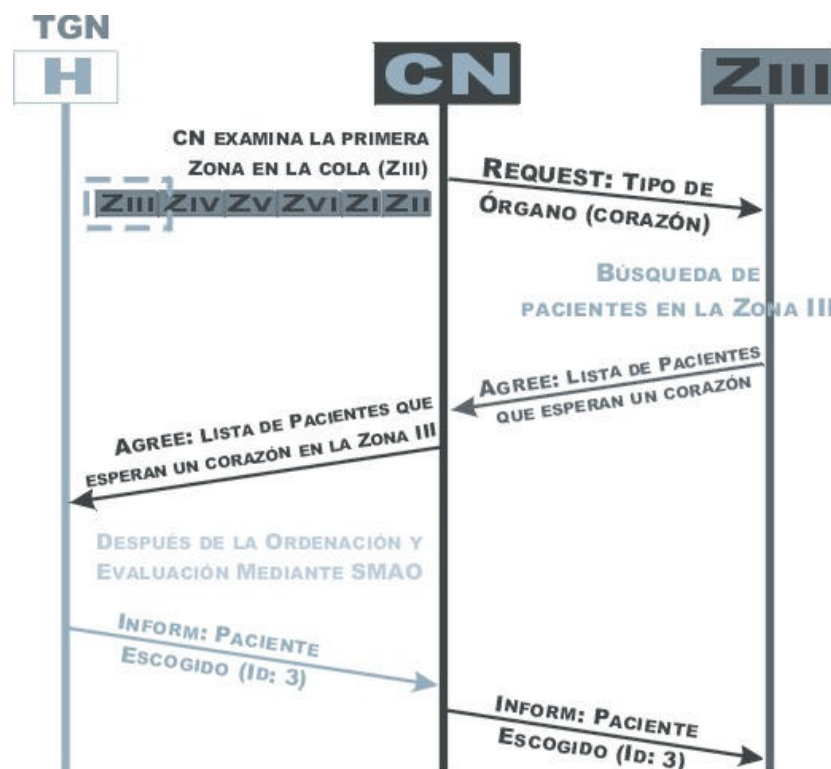


Figura 36 - Búsqueda en la Zona III

En caso de que aún no se hubiera encontrado ningún paciente apto, la búsqueda concluiría cuando **CN** hubiera examinado las cinco zonas posibles, excluyendo la zona de origen.

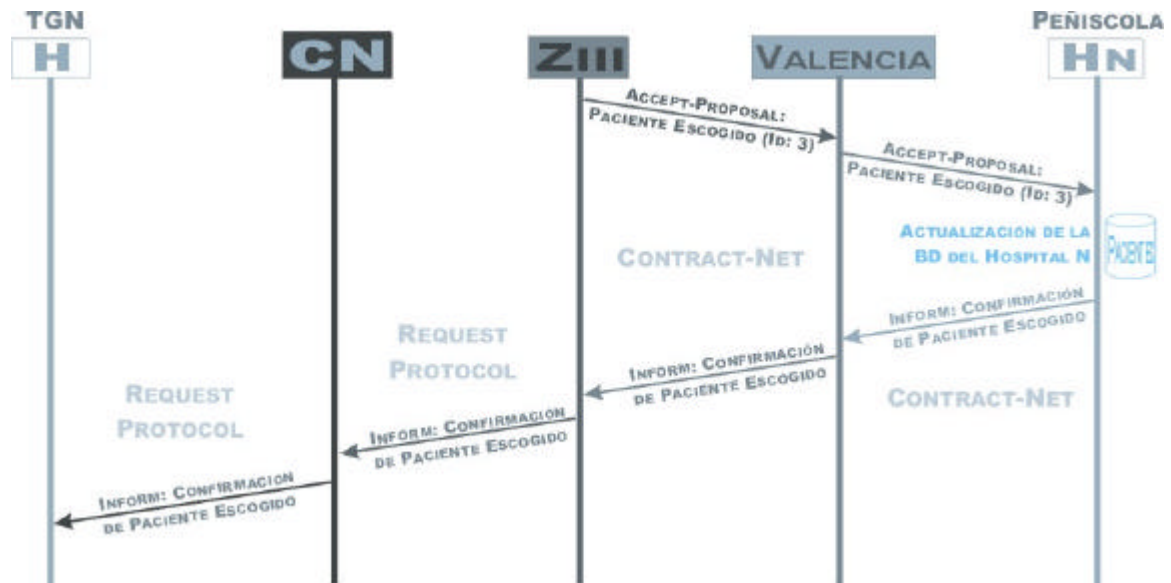


Figura 37 - Finalización de la búsqueda en la Zona III

Como se ve en las dos figuras anteriores se ha escogido al paciente número 3 del hospital N, que se encuentra en Peñíscola, en la Comunidad Valenciana. La elección del paciente se va pasando desde **H** hasta **HN**, por todos los coordinadores intermedios siguiendo el protocolo correspondiente. Cuando llega a su destino, **HN** actualiza su **BD** marcando al paciente escogido, para que no entre en la próxima búsqueda. Inmediatamente confirma a **H** la elección, pasando también por todos los niveles.

Finalmente, sólo queda que **H** se comunice con el Agente Histórico para que registre la donación en el **Historial**. Esto se realiza mediante un **Request Protocol**, como se ve a continuación.



Figura 38 - Comunicación entre H y AH

Aunque la interficie gráfica va mostrando paso a paso lo que acontece en el sistema y el paso de la búsqueda por los distintos niveles, al final de todo se muestra el resultado de la búsqueda, es decir los datos del paciente escogido, por pantalla.

En este proceso se ha intentado simular una búsqueda lo más larga posible, llegando incluso al nivel nacional. Normalmente la cosa no tiene porqué ser así y la búsqueda finalizaría en un nivel inferior. Cuando esto sucede se dan los mismos pasos que hemos visto, es decir: se avisa de la elección al hospital elegido mediante la jerarquía de coordinadores (llegando al nivel correspondiente); el Hospital escogido devuelve una confirmación, por la misma ruta; **H** se comunica con **AH** para la actualización del historial de donaciones.

La cosa cambia cuando sucede algún problema intermedio (hemos supuesto el caso ideal, sin problemas). Pueden haber distintos problemas, como: que no se encuentre un coordinador (porque no esté registrado en el sistema, por ejemplo); que no se encuentren más ciudades, autonomías, zonas u hospitales (lo que daría paso a una búsqueda a nivel superior); que la ordenación de los posibles pacientes sea imposible por parte de **SMAO** (una posible solución en el futuro sería buscar más pacientes en niveles superiores y unir las listas para ser analizadas juntas, con lo que **SMAO** tendría más posibilidades de éxito), etc. Lo que queda claro es que si surge uno de estos problemas la búsqueda no para hasta que se llega y se agota el nivel nacional.

5. DISTRIBUCIÓN MODULAR DEL SISTEMA

Como ya hemos comentado, el proyecto está implementado sobre JADE y, por tanto, escrito en JAVA. El código se encuentra en una serie de módulos o ficheros JAVA que detallaremos en este capítulo. Los más importantes son los correspondientes al código de los distintos tipos de agentes que hemos ido explicando. Además, comentaremos también los ficheros necesarios para describir la ontología con la que se comunicaran los agentes del sistema. Cabe recordar que el sistema usa otros ficheros pertenecientes al PFC de David Sánchez, que no comentaremos.

5.1. Hospital.java

Es la clase más compleja del sistema puesto que corresponde al agente **Hospital**, que es el que se comunica con el resto de agentes y el que toma las decisiones. Sus funciones y comportamientos principales son los siguientes:

- **onGuiEvent(GuiEvent ev)**: es una función asociada a la interficie que siempre se encuentra a la espera de lo que suceda en la interficie gráfica. Mientras se está realizando una búsqueda, el botón de **Enviar Petición** en la interficie está inhabilitado, por lo que no se pueden realizar dos búsquedas a la vez. Hay dos tipos de acciones en la interficie a las que se responde: La primera es de simple utilidad para las pruebas y no estaría en la versión final del producto, se trata de limpiar las BD's. Esto significa eliminar las marcas de los pacientes ya escogidos para que puedan ser examinados en las siguientes búsquedas. La segunda y principal opción es comenzar la búsqueda de pacientes. Cuando se pulsa este botón, la clase **Hospital** recoge los datos del órgano donado e inicia un nuevo comportamiento llamado **Inicia_proceso_busqueda**.
- **Inicia_proceso_busqueda (Agent myAgent, Paciente don)**: Comportamiento simple que inicia la búsqueda a nivel del propio hospital. Se ayuda de otras funciones que veremos más adelante para realizar la búsqueda siguiendo el proceso visto antes. Si no se encuentra ningún paciente a nivel local, se pasa al nivel de ciudad mediante el comportamiento **Inicia_Contract_Net_misma_ciudad**.
- **Inicia_Contract_Net_misma_ciudad(Agent a, ACLMessage msg, List responders, Organo org, Paciente don)**: Este comportamiento hace que el agente actúe como el *initiator* de un protocolo **Contract-Net**. Como tal, reaccionará a mensajes de tipo **propose** y **final**, teniendo un *handler* para cada uno de estos mensajes. Los mensajes **propose** serán los recibidos de los distintos hospitales con la lista de pacientes obtenida de su BD. Los de tipo **final** se reciben como confirmación de la elección de un paciente de un cierto hospital. En caso de que la búsqueda falle también a nivel de ciudad, hay tres funciones que inician la búsqueda en niveles superiores: **Inicia_busqueda_autonomia**, **Inicia_busqueda_zona** e **Inicia_busqueda_nacional**.

- **Inicia_búsqueda_autonomia(Paciente donante):** Busca al coordinador autonómico en cuestión e inicia la búsqueda mediante el comportamiento **Coord_Request**, pasándole el nivel de la búsqueda (autonómico) y los datos del órgano donado.
- **Inicia_búsqueda_zona(Paciente donante):** Busca al coordinador zonal correspondiente e inicia la búsqueda mediante el comportamiento **Coord_Request**, a nivel zonal.
- **Inicia_búsqueda_nacional(Paciente donante):** Busca al coordinador nacional e inicia la búsqueda mediante el comportamiento **Coord_Request**, a nivel nacional.
- **Coord_Request (Agent agent, ACLMessage requestMsg, MessageTemplate mt, Paciente don, String busq):** Inicia una conversación con un coordinador siguiendo el protocolo **Request**. Se le pasa el mensaje de tipo **request**, el nivel de la búsqueda y los datos del órgano donado. Tiene implementados *handlers* para mensajes de tipo **agree**, **inform** y **refuse**. Un **agree** se recibe con la lista de pacientes de un coordinador. El **inform** indica la confirmación de una elección. Mientras que el **refuse** indica que ha habido algún tipo de problema, como por ejemplo que un coordinador no encuentre más hospitales en su autonomía.
- **Responde_Contract_Net(Agent a):** Cuando un agente de **Hospital** actúa de posible receptor de un órgano, se usa este comportamiento para modelar el **Contract-Net** desde el punto de vista del *responder*. Este comportamiento incluye la búsqueda en la BD de pacientes y la realización de la propuesta a **H** o **A**. Si se recibe un **accept-proposal** hay que actualizar la BD, marcando al paciente escogido.
- **Resultado Consulta_AO(Lista_objetos envio):** Esta es la función para interactuar con **SMAO**. Simula un protocolo **Request**, pero como necesitamos la respuesta antes de poder seguir con el proceso, usamos el **blockingReceive** para recibir el mensaje de **SMAO**. Nos devuelve la lista de pacientes que le enviamos ordenada y evaluada.
- **AH_Request (Agent agent, ACLMessage requestMsg, MessageTemplate mt):** Realiza un **Request Protocol** con el agente histórico para que éste añada la donación al historial.

A parte de algunas otras funciones que realizan tareas auxiliares (consultar y actualizar BD, mostrar resultados por pantalla, registro y desregistro del **DF**, conversiones, etc) tenemos dos especialmente útiles para el manejo de mensajes y ontologías, que se repiten en todos los agentes. Las comentaremos a continuación:

- **ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0, Object contenido):** Esta función nos devuelve en un solo paso un mensaje del tipo *tipo_mensaje*, dirigido al receptor *receiver* y cuyo contenido es, en formato

FIPA-SL, el objeto *contenido*. Es una función muy útil, al igual que la siguiente, por la gran cantidad de mensajes que hay que crear y manipular.

- **Object Extrae_contenido(ACLMessage msg):** Realiza en un solo paso la extracción del contenido del mensaje *msg* (en formato FIPA-SL) en forma de objeto.

5.2. Autonomia.java

Es la clase implementada para el agente coordinador de autonomía. Sus principales funciones son:

- **H_Request(Agent a, ACLMessage msg):** Es el comportamiento que despierta al agente, ya que es el que recibe la petición de búsqueda por parte de **H**. Se trata del rol de *responder* de un protocolo **Request**. El agente busca las ciudades de su autonomía e inicia un protocolo **Contract-Net** con ellas.
- **Inicia_Contract_Net_misma_autonomia(Agent a, ACLMessage msg, List responders, Organo org, ACLMessage msg_del_origen, boolean busqueda_superior):** Aquí el agente ejerce el rol de *initiator* del **Contract-Net** con todos los hospitales de la autonomía. Cuando se reciben las listas de los hospitales en mensajes de tipo **propose**, se juntan y se pasan a **H**, mezclando los protocolos **Request** y **Contract-Net**.
- **Responde_Contract_Net(Agent a):** Es el rol *responder* de un **Contract-Net**, que proviene desde arriba, de un coordinador de zona. También se mezcla con el **Contract-Net** a nivel inferior para continuar la búsqueda.

5.3. Zona.java

Esta clase implementa un agente de tipo Coordinador de Zona. Las funciones principales son las que siguen:

- **H_Request(Agent a, ACLMessage msg):** Es el comportamiento que despierta al agente, ya que es el que recibe la petición de búsqueda por parte de **H** o de **CN**. Se trata del rol de *responder* de un protocolo **Request**. El agente busca las autonomías de su zona e inicia un protocolo **Contract-Net** con ellas:
- **Inicia_Contract_Net_misma_zona(Agent a, ACLMessage msg, List responders, Organo org, ACLMessage msg_usado_en_req):** Aquí el agente ejerce el rol de *initiator* del **Contract-Net** con todas las autonomías de la zona. Cuando se reciben las listas de las autonomías en mensajes de tipo **propose**, se juntan y se pasan a **H** o a **CN** (dependiendo de si es una búsqueda a nivel zonal o nacional, respectivamente), mezclando los protocolos **Request** y **Contract-Net**.

5.4. Nacional.java

Esta es la clase que implementa al agente Coordinador Nacional. Sus funciones son las siguientes:

- **H_Request(Agent a, ACLMessage msg)**: Es el comportamiento que despierta al agente, ya que es el que recibe la petición de búsqueda por parte de **H**. Se trata del rol de *responder* de un protocolo **Request**. A partir de aquí **CN** seguirá la cola de turnos para realizar la búsqueda por zonas, mediante un protocolo **Request**. Este comportamiento ejecuta la función **Inicia_búsqueda_zona** para cada una de las zonas hasta encontrar un paciente adecuado.
- **Inicia_búsqueda_zona(Organo organo, ACLMessage msg_req_de_H)**: Función que extrae de la cola ordenada la primera zona e inicia el **Request** con su coordinador.
- **Coord_Request (Agent agent, ACLMessage requestMsg, MessageTemplate mt, ACLMessage msg_req_de_H)**: Rol *initiator* de un protocolo **Request** con un coordinador zonal. La lista procedente de **Z** mediante un mensaje **agree** se ha de pasar a **H** para que la evalúe. De esta forma mezclamos los protocolos **Request** con el nivel inferior (**Z**) y con el superior (**H**).

5.5. Historico.java

Es la clase con la que se implementa el Agente Histórico. Tan sólo tiene un comportamiento significativo. Se trata de **H_Request(Agent a, ACLMessage msg)**, que hace que el agente actúe como *responder* de un protocolo **Request** con el Hospital origen. Dentro de este comportamiento se actualiza la BD del historial, añadiendo la donación realizada con todos los datos del donante y del receptor.

5.6. Interficie.java

Esta clase no implementa ningún agente, sino la interficie que interactúa con el usuario. No tiene funciones significativas, ya que casi todo es código creado automáticamente por el editor de ventanas del JBuilder. Tan solo comentar que tiene alguna función para comprobar la integridad de los datos indicados en la interficie. Comprueba que la fechas sean correctas, que todos los campos estén rellenos, etc. Finalmente tiene los *handlers* asociados a los dos botones que ya hemos comentado antes (Limpiar las BD's y Enviar Petición).

5.7. El Paquete Ontology

Como ya se ha definido anteriormente en la sección 3.6, la ontología es lo que permite a los agentes “entender” el contenido de los mensajes que se envían. La forman

un conjunto de *frames* y *slots* que se han de seguir durante la creación del mensaje para que el receptor sea capaz de extraer la información. Este es uno de los puntos que hay que haber definido antes de implementar los agentes, y está formada por dos partes principales:

Definición

En este punto es necesario especificar qué *frames*, acciones y predicados hay en la ontología. Dentro de cada uno de ellos habremos de incluir el conjunto de *slots* que los forman (indicando el tipo de datos que son) y la clase que se encargará de manipularlos (implementar los métodos de inicialización y acceso sobre los datos). El objeto donde se incluyen estas definiciones es **ontology/JBSOntology**. La estructura de la Ontología es la que hemos podido ver en la sección 4.2.

Implementación

Para cada elemento definido en la ontología, hemos de crear una clase JAVA que implemente los métodos de inicialización (SETxxx) y consulta (GETxxx) para cada uno de los *slots* que contenga. El conjunto de ficheros JAVA que implementan cada uno de los elementos definidos se encuentra en el directorio **ontology**, junto a la ontología propia del **SMAO**, pero por separado. Estos ficheros son:

- **Agente_no_registrado.java**: implementa el error producido cuando un agente coordinador no se encuentra registrado en el sistema.
- **Elección.java**: implementa un objeto de tipo elección. Sirve para notificar la elección de un determinado paciente, pudiendo indicar su identificador y el hospital de donde proviene.
- **Grado.java**: define un grado de idoneidad, de entre los catorce existentes.
- **Lista_objetos.java**: es el objeto principal de la ontología. Se trata de una lista de pacientes, que incluirán todas sus características. También tenemos la cantidad de pacientes en la lista y los pesos de las variables que podemos definir en la interficie (distancia, peso del paciente, tamaño del órgano, antígenos, edad y tiempo de espera).
- **No_hospitales.java**: implementa el error producido cuando no hay hospitales en una determinada ciudad o comunidad autónoma.
- **No_pacientes.java**: implementa el error producido cuando no se encuentran pacientes en la BD de un cierto hospital.
- **Ordenacion_imposible.java**: implementa el error producido cuando la ordenación de una lista de pacientes no es posible por parte de **SMAO**.
- **Organo.java**: implementa un objeto de tipo órgano, en el que podemos especificar el tipo del órgano donado.

- **Paciente.java:** implementa un objeto de tipo paciente, que sirve tanto para detallar las características del donante de un órgano, como del resto de pacientes que esperan ese órgano. Tiene todos los campos para incluir las características necesarias.
- **Resultado.java:** sirve para la comunicación con **SMAO** y para que éste devuelva el resultado de una ordenación. Consta de una lista de pares id-valoración, y el número de elementos que hay en la lista.
- **Valor_atrib.java:** define el valor de un atributo en forma de string, otorgando también un peso a este atributo.

6. MANUAL DE USUARIO

En este apartado se describen los aspectos fundamentales para que un usuario pueda interactuar con nuestro sistema.

6.1. Ejecución del Sistema

En principio el sistema está pensado y probado sobre una sola máquina. No debería ser muy difícil debido a las características del JADE que funcionase en varias a la vez, pero esta opción no se ha probado aún. Así que todas las pruebas y ejemplos que se mostrarán a continuación son teniendo en cuenta que se ejecuta todo en un solo ordenador.

Para la ejecución del sistema en nuestras pruebas, usamos el siguiente comando:

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent A1:pfc_ei.Autonomia A2:pfc_ei.Autonomia
A3:pfc_ei.Autonomia A4:pfc_ei.Autonomia Z1:pfc_ei.Zona Z2:pfc_ei.Zona
Z3:pfc_ei.Zona Z4:pfc_ei.Zona Z5:pfc_ei.Zona Z6:pfc_ei.Zona CN:pfc_ei.Nacional
AH:pfc_ei.Historico H2:pfc_ei.Hospital H3:pfc_ei.Hospital H4:pfc_ei.Hospital
H5:pfc_ei.Hospital H6:pfc_ei.Hospital H7:pfc_ei.Hospital H8:pfc_ei.Hospital
H9:pfc_ei.Hospital H10:pfc_ei.Hospital H11:pfc_ei.Hospital
```

Esta línea inicia, por este orden:

- Un Hospital de Tarragona, que usaremos como hospital propio (**H**).
- Los agentes necesarios para **SMAO (ClusDM y Especialista)**.
- Los coordinadores de Cataluña (**A1**), Aragón (**A2**), Comunidad Valenciana (**A3**) y Baleares (**A4**).
- Los coordinadores de las 6 zonas existentes (**Z1..Z6**).
- El Coordinador Nacional (**CN**).
- El Agente Histórico (**AH**).
- 10 agentes de tipo Hospital, correspondientes a las siguientes ciudades: Tarragona (**H2**), Tarragona (**H3**), Barcelona (**H4**), Palau (**H5**), Mallorca (**H6**), Menorca (**H7**), Ibiza (**H8**), Zaragoza (**H9**), Teruel (**H10**) y Huesca (**H11**).

Todo esto sin iniciar la interficie gráfica propia del JADE, para lo que habría que añadir:

-gui

Si iniciamos esta interficie gráfica de JADE podemos comprobar qué agentes están registrados en el DF. Son todos los que hemos iniciado. Lo vemos en la figura siguiente:

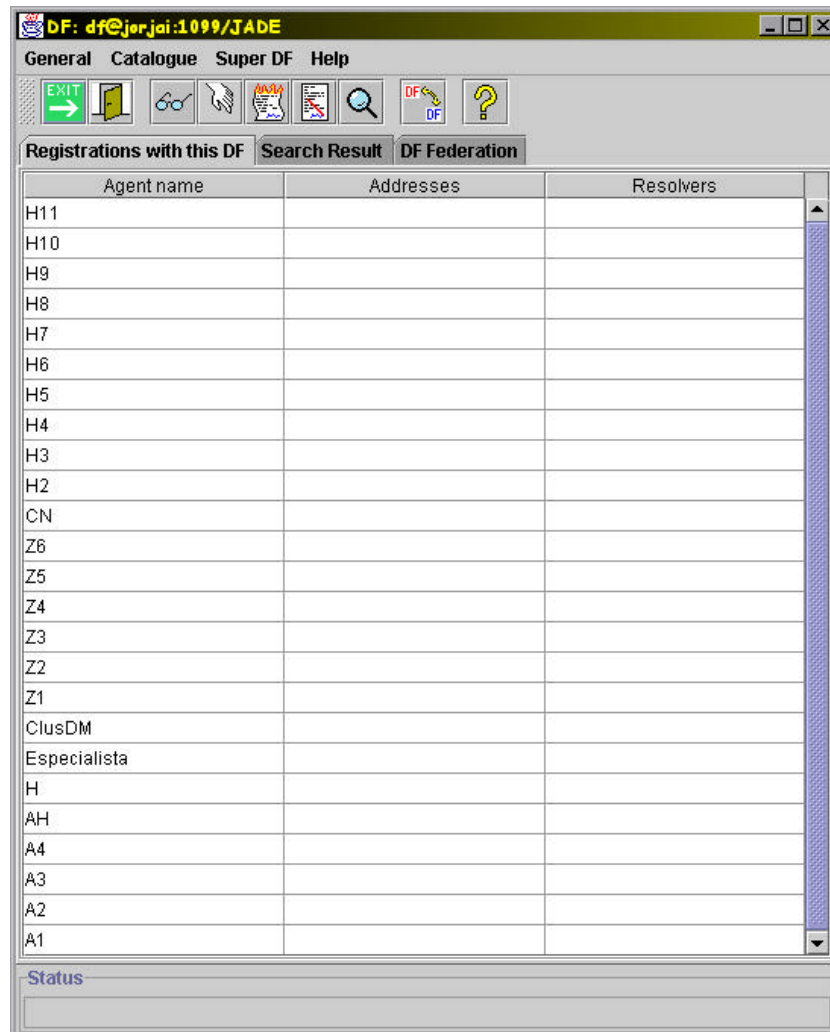


Figura 39 - Agentes registrados en el DF

6.2. Interficie Gráfica de usuario

Una vez iniciado el sistema, se abre una ventana de interficie correspondiente al Hospital **H** con el aspecto de la figura 40. Todos los campos se encuentran ya inicializados con unos valores por defecto, que son los que usamos para las pruebas.

Figura 40 - Interficie de usuario del Sistema

Otra cosa a indicar es que cada una de las clases está “preparada” para las pruebas, en el sentido que dependiendo del nombre del agente que la instancia se registran en el **DF** con unas u otras características (ciudad, autonomía o zona; nombre de la BD). Todos estos parámetros deberían preguntarse al usuario cuando se inicia un agente y por tanto hay que registrarlo al **DF**, pero para acelerar las pruebas, esto es, de momento, automático.

A continuación detallaremos los campos de la interficie.

Figura 41 - Elección del tipo de órgano donado

En este campo se puede escoger entre los distintos tipos de órgano posibles. En nuestro sistema de momento contemplamos la posibilidad de transplantes de corazón, hígado, riñón y páncreas, aunque no sería nada complicado añadir más.

Una interfaz de usuario para la entrada de la fecha de nacimiento. Incluye el texto "Fecha de Nacimiento" en azul, seguido de tres campos de entrada separados por barras inclinadas. Los campos contienen los valores "10", "10" y "1960".

Figura 42 - Fecha de Nacimiento del donante

Se introduce la fecha de nacimiento del paciente, en formato *dd/mm/aaaa*. Esta fecha se comprueba antes de iniciar la búsqueda, así por ejemplo no podemos poner el día 34, el mes 14 o años negativos.

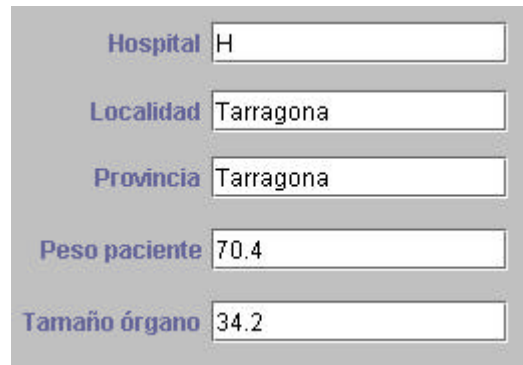
Una interfaz de usuario para la entrada de varios datos del donante. Incluye cinco campos de entrada con etiquetas en azul: "Hospital" (contiene "H"), "Localidad" (contiene "Tarragona"), "Provincia" (contiene "Tarragona"), "Peso paciente" (contiene "70.4") y "Tamaño órgano" (contiene "34.2").

Figura 43 - Datos varios del donante

Estos campos indican respectivamente: el hospital donde se encuentra el órgano donado, la localidad y provincia de dicho hospital, el peso del paciente (en Kg), y el tamaño del órgano (midiendo la longitud de su contorno en cm). Aquí la comprobación que se realiza es que todos los campos han de estar rellenados, de lo contrario se avisa de ello en la pantalla de resultados.

Dos interfaces de usuario para la entrada de antígenos y tipo de sangre. La izquierda, titulada "Antígenos", tiene cuatro opciones con casillas de verificación: "B0", "B1", "DR0" (marcada) y "DR1". La derecha, titulada "Sangre", tiene un menú desplegable con "0" seleccionado y una lista de opciones: "0", "A", "B" y "AB".

Figura 44 - Antígenos y tipo de sangre del donante

Aquí se indican los antígenos y el tipo de sangre del donante. Hay cuatro tipos de antígenos definidos (*B0*, *B1*, *DR0* y *DR1*), pudiendo indicar con un tic si se tienen o no. Para la sangre también tenemos cuatro tipos (*0*, *A*, *B* y *AB*), pudiendo escoger uno de ellos.

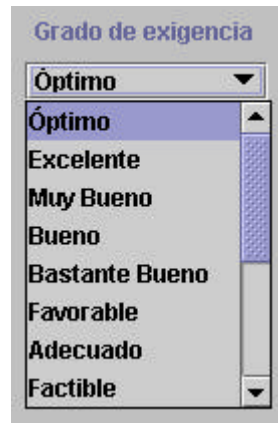


Figura 45 - Grado de idoneidad en la búsqueda

Se indica el grado de idoneidad mínimo requerido para aceptar un paciente como receptor viable. El sistema es capaz de distinguir 14 grados, que son: *óptimo, excelente, muy bueno, bueno, bastante bueno, favorable, adecuado, factible, flojo, desfavorable, bastante malo, malo, muy malo, fatal y pésimo*. Cuanto mayor sea el grado de idoneidad más difícil será encontrar un paciente adecuado, pero si se encuentra habrá más posibilidades de éxito en el transplante. Al contrario con los grados menores. Por eso una buena estrategia es iniciar la búsqueda optando a un paciente óptimo, y en caso de que no se encuentre ir disminuyendo el grado de idoneidad. Esto es factible debido a que la búsqueda es muy rápida y se puede repetir varias veces en un corto periodo de tiempo.

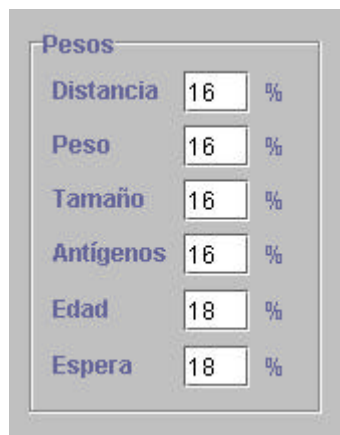


Figura 46 - Pesos de los parámetros

Esta opción permite configurar la búsqueda de manera que se le dé mayor importancia a unos criterios que a otros. El **SMAO** está implementado de tal manera que dependiendo de estos pesos y por tanto de la importancia de cada criterio, es capaz de dar valoraciones distintas para un conjunto de pacientes. Podemos controlar 6 criterios de búsqueda (*distancia geográfica del donante al receptor, peso del paciente, tamaño del órgano, compatibilidad de antígenos, edad del paciente y tiempo de espera del paciente*), que han de sumar el 100%. Esto se comprueba antes de iniciar la búsqueda y se avisa al usuario si falla.

Finalmente, tenemos los dos botones que nos permiten realizar funciones desde la interficie.



Figura 47 - Botones de acción

El primer botón es la función principal de la interficie. Lo que hace es comprobar todos los campos y parámetros de la búsqueda, y si estos son correctos, comenzar a buscar. De lo contrario, se avisa al usuario del problema surgido, para que lo solucione antes de poder empezar la búsqueda. Los botones quedan deshabilitados mientras la operación está en curso. Con esto se evita el hecho de tener dos búsquedas a la vez en el sistema, lo que podría complicar la comunicación entre los agentes y ralentizaría la búsqueda.

El segundo botón en realidad no tendría que estar en la interficie. Como ya hemos comentado antes, se trata de una funcionalidad muy útil para las pruebas, pero al mismo tiempo muy específica, ya que hace uso de unas constantes como son el número de bases de datos definidas y el número de pacientes de cada una de ellas (específicas para las pruebas). Lo que hace este botón es limpiar las BD's, es decir, eliminar las marcas de los pacientes ya escogidos, para que puedan volver a ser examinados en búsquedas posteriores. También deshabilita los dos botones cuando se pulsa.

A parte de todos los campos comentados tenemos uno que, pese a no ser interactivo como el resto, es de los más importantes. Se trata del cuadro de resultados.

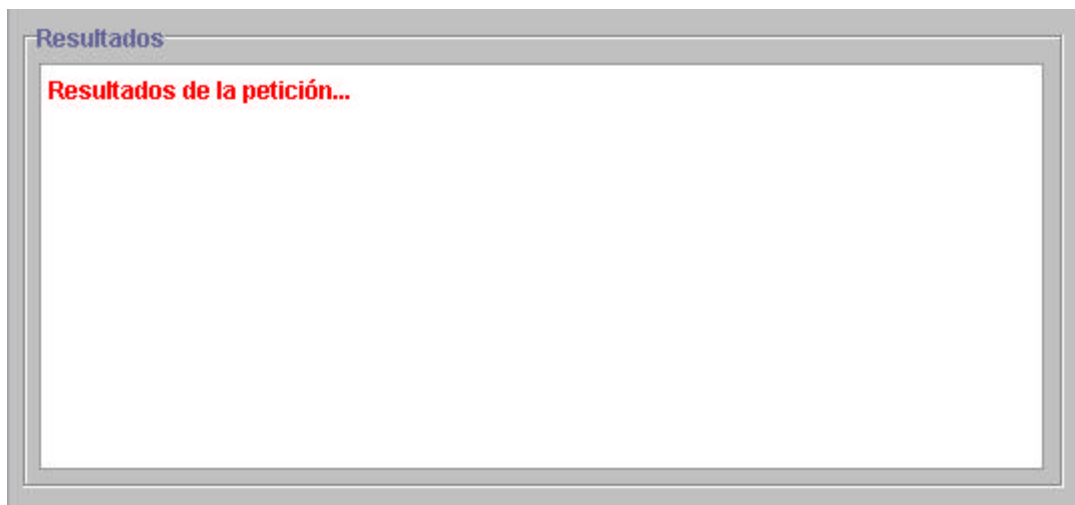


Figura 48 - Cuadro de resultados

En este cuadro aparece información de gran importancia para el usuario. En primer lugar pueden aparecer los errores comentados al rellenar los distintos campos. Éstos son como se ve en la figura 49: que falten campos por rellenar, que el usuario haya introducido incorrectamente el día, mes o año de nacimiento del donante o que la suma de los pesos de los distintos criterios no sea el 100%. De esto se avisa al usuario para que lo pueda corregir antes de iniciar la búsqueda.

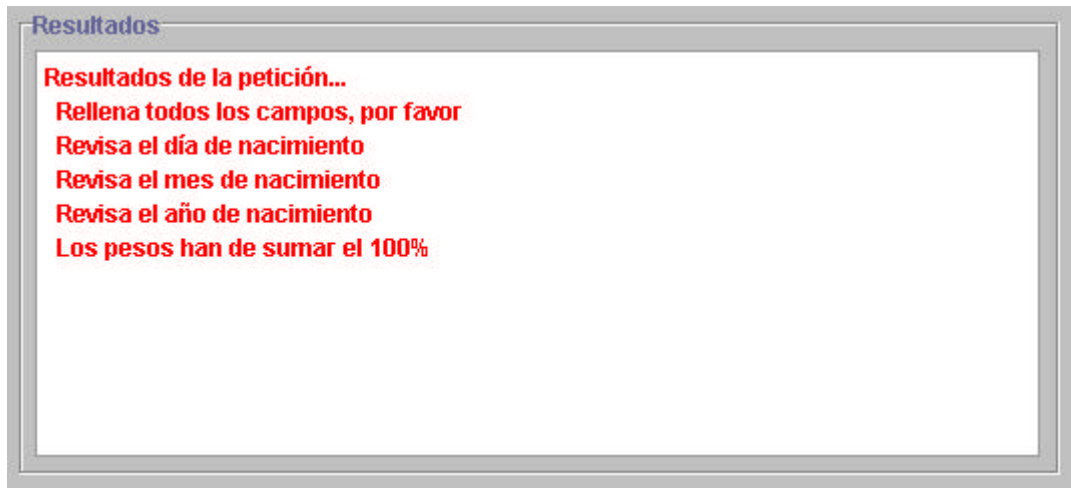


Figura 49 - Posibles errores al rellenar los campos

A parte de avisar de los errores en la introducción de los datos, el cuadro de resultados muestra información de gran interés para el usuario durante el proceso de búsqueda. Primero de todo muestra los datos del órgano donado, después el seguimiento de la búsqueda, con lo que el usuario puede ver en cada momento el nivel en que se encuentra y los diferentes pacientes que se van evaluando. Finalmente se muestra el resultado, es decir los datos del paciente escogido. En las siguientes figuras vemos algunos ejemplos:

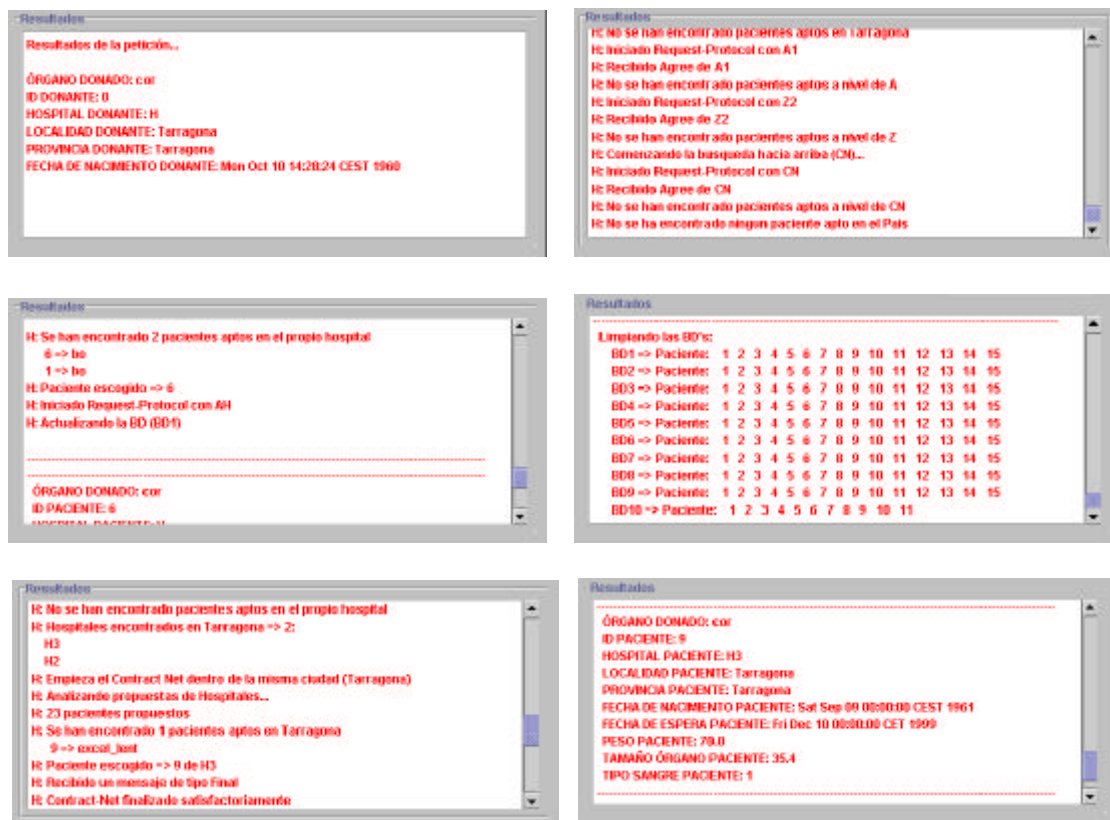


Figura 50 - Informaciones en el cuadro de resultados

6.3. Bases de Datos

Para el correcto funcionamiento del sistema son necesarias las distintas BD's del sistema, que ya comentamos en el apartado 4.1.7. Estas bases de datos se han implementado en Access, aunque podrían usarse otros sistemas siempre que sean compatibles con JADE. A continuación comentaremos qué elementos o personas interactúan con las BD's y en quien recae la responsabilidad de su mantenimiento.

- **BD de Hospital (Pacientes):** Se introducen los datos sobre los pacientes de un hospital que esperan cualquier tipo de órgano. Los campos de esta BD ya se comentaron. El usuario es el responsable de cualquier alta, baja o modificación de esta BD. El sistema sólo realiza dos acciones: la primera es de consulta, para buscar pacientes que esperan un cierto tipo de órgano y la segunda es marcar de algún modo los pacientes escogidos. Hay una tercera acción, en el modo de pruebas, que es la de desmarcar los pacientes escogidos mediante la opción **Limpiar las BD's** de la interficie gráfica.

- **BD del Agente Histórico (Historial):** Todas la actualizaciones de esta BD las realiza el sistema. Esta BD sólo habría de ser de consulta para el usuario, teóricamente, ya que se trata del historial de las distintas donaciones. La única operación que realiza el sistema es la de dar de alta operaciones.

- **BD de ciudades (BD-Ciudades):** Esta BD contiene las distintas ciudades con hospitales que tratan transplantes y su provincia, autonomía y zona. Se utiliza en este prototipo local para facilitar la inicialización del sistema y de los agentes. No contiene todas las ciudades de España (sólo unas cuantas para las pruebas), de modo que el usuario sería el encargado de añadir más datos si lo considera necesario para ampliar el campo de pruebas. La única operación que realiza el sistema sobre esta BD es la consulta de sus datos.

7. JUEGO DE PRUEBAS

Para comprobar el buen funcionamiento del sistema, se han hecho una serie de pruebas. Algunas en las que se encuentra un paciente adecuado (a distintos niveles), otras en las que no se encuentra a ningún nivel, pruebas con fallos en la búsqueda, etc. En fin, se han intentado probar todos los casos posibles.

Antes de nada hemos de comentar algunos puntos generales sobre estas pruebas:

- Las clases de hospitales y coordinadores están preparadas para que no haga falta hacer el registro manualmente. Dependiendo del nombre del agente, la clase actúa como hospital de una u otra ciudad. Lo mismo para los coordinadores, excepto el Nacional, que es único.
- Hay 11 BD's de pacientes, una para cada hospital. Cada una de estas bases de datos contiene 15 pacientes, que esperan distinto tipo de órgano. Mayoritariamente hay pacientes que esperan un corazón. El número de BD's y pacientes de cada una de ellas está fijado para que funcione la opción de limpiar las BD's de la interficie.
- Hay hospitales registrados en distintas ciudades, distintas autonomías y distintas zonas. Todo ello para que la búsqueda tenga más opciones de llevarse a cabo.
- El órgano donado, mientras no se diga lo contrario es un corazón con los parámetros por defecto al iniciar el sistema, los que se pueden ver en la figura 40. Lo mismo sucede con el grado de idoneidad, que será por defecto *óptimo*.
- Mientras no se diga lo contrario no se usará la opción de limpiar las BD's. Es decir, entre una prueba y la siguiente, se mantendrán los resultados. Si un paciente ha sido escogido en una prueba quedará marcado y no podrá ser examinado en la siguiente.

Empezamos ya con la pruebas. Las primeras pruebas serán jugando con el número de agentes ejecutados en el sistema. De esta forma se podrán dar casos en que falten hospitales, coordinadores no registrados, etc. Después ejecutaremos todos los agentes que vimos en el manual de usuario y realizaremos pruebas jugando con los distintos parámetros del órgano donado.

Primeramente ejecutamos el sistema sólo con un hospital (**H**), de Tarragona:

```
java jade.Boot -platform H:pfc_ei.Hospital
```

Lo que sucede en esta prueba es que al iniciar una búsqueda el sistema avisa de que falta el agente especialista y **H** no puede realizar la ordenación de pacientes. La búsqueda no puede finalizar, por tanto. Conclusión lógica, ya que el agente especialista es uno de los agentes imprescindibles del sistema.

Ejecutamos ahora la misma prueba que antes, pero incluyendo ahora el **SMAO**:

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent
```

El sistema realiza la búsqueda y concluye en que el paciente número 9 del propio hospital tiene valoración óptima. La operación de búsqueda finaliza, pero no se puede actualizar el historial ni la BD del hospital porque falta el Agente Histórico. Esto significa que **AH** es otro de los imprescindibles en el sistema.

Repetimos la misma ejecución anterior, añadiendo el **AH**:

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent AH:pfc_ei.Historico
```

Ejecutamos la petición y el sistema vuelve a concluir que el paciente óptimo es el número 9 del propio hospital. Esto sucede debido a que antes al no acabarse la ejecución correctamente, el paciente no quedó marcado, y por tanto puede volver a ser examinado en esta búsqueda.

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:17:38 CEST 1960
```

```
H: Se han encontrado 1 pacientes aptos en el propio hospital
  9 => optim
H: Paciente escogido => 9
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)
```

```
-----
ÓRGANO DONADO: cor
ID PACIENTE: 9
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Sat Sep 09 00:00:00 CEST 1961
FECHA DE ESPERA PACIENTE: Fri Dec 10 00:00:00 CET 1999
PESO PACIENTE: 70.0
TAMAÑO ÓRGANO PACIENTE: 35.4
TIPO SANGRE PACIENTE: 1
-----
```

Podemos ver arriba un volcado de b que se muestra en la pantalla de resultados, con el seguimiento de la petición. Como hemos comentado, primero vemos los datos del donante, luego el proceso de búsqueda y finalmente el resultado (datos del paciente receptor), si lo hay.

Repetimos otra búsqueda en la misma ejecución para ver lo que sucede una vez marcado el paciente número 9:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:21:31 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: No hay mas hospitales en Tarragona
H: Comenzando la busqueda hacia arriba (A)...
H: Coordinador de Catalunya no registrado
H: Comenzando la busqueda hacia arriba (Z)...
H: Coordinador de la ZonaII no registrado
H: Comenzando la busqueda hacia arriba (CN)...
H: Coordinador Nacional (CN) no registrado
H: No se ha encontrado ningun paciente apto en el Pais
```

La búsqueda concluye sin encontrar ningún paciente apto. Esto ocurre por varios motivos: no hay ningún paciente de grado óptimo en el propio hospital (el 9 ya está marcado), tampoco encuentra ninguno en la propia ciudad, porque no hay más hospitales. Finalmente, cuando intenta búsquedas a niveles superiores, se encuentra con que no hay ningún coordinador registrado en el sistema.

Veamos ahora que ocurre si añadimos hospitales al sistema. De momento seguimos sin ejecutar ningún coordinador, por lo que los hospitales que no sean de Tarragona no tendrán mucho sentido, ya que no se podrá acceder a ellos (**H** sólo puede comunicarse directamente con los hospitales de su misma ciudad).

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent AH:pfc_ei.Historico H2:pfc_ei.Hospital
H3:pfc_ei.Hospital H4:pfc_ei.Hospital H5:pfc_ei.Hospital H6:pfc_ei.Hospital
H7:pfc_ei.Hospital H8:pfc_ei.Hospital H9:pfc_ei.Hospital H10:pfc_ei.Hospital
H11:pfc_ei.Hospital
```

Los hospitales son los mismos que los vistos en el apartado 6.1, es decir pertenecen los tres primeros a Tarragona, luego a Barcelona, Palau, Mallorca, Menorca, Ibiza, Zaragoza, Teruel y Huesca. Veamos lo que ocurre:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:29:11 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
```

H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 23 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Coordinador de Catalunya no registrado
H: Comenzando la búsqueda hacia arriba (Z)...
H: Coordinador de la ZonaII no registrado
H: Comenzando la búsqueda hacia arriba (CN)...
H: Coordinador Nacional (CN) no registrado
H: No se ha encontrado ningun paciente apto en el Pais

Sigue sin encontrarse ningún paciente apto, a pesar de que ya se realiza la búsqueda en la misma ciudad (Tarragona). Las búsquedas a niveles superiores siguen sin poder realizarse ya que no hay coordinadores en el sistema.

Añadiremos ahora los coordinadores de autonomía al sistema. En realidad, sólo sería necesario añadir el de Cataluña, ya que el resto será inaccesible si no añadimos también el Coordinador de Zona correspondiente, como pasaba antes con los hospitales de fuera de Tarragona.

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent AH:pfc_ei.Historico A1:pfc_ei.Autonomia
A2:pfc_ei.Autonomia A3:pfc_ei.Autonomia A4:pfc_ei.Autonomia H2:pfc_ei.Hospital
H3:pfc_ei.Hospital H4:pfc_ei.Hospital H5:pfc_ei.Hospital H6:pfc_ei.Hospital
H7:pfc_ei.Hospital H8:pfc_ei.Hospital H9:pfc_ei.Hospital H10:pfc_ei.Hospital
H11:pfc_ei.Hospital
```

Hemos añadido los coordinadores de Cataluña, Aragón, Comunidad Valenciana y Baleares. El resultado de la búsqueda es el siguiente:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:36:46 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 23 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Coordinador de la ZonaII no registrado
H: Comenzando la búsqueda hacia arriba (CN)...
H: Coordinador Nacional (CN) no registrado
H: No se ha encontrado ningun paciente apto en el Pais

Siguen sin encontrarse pacientes adecuados, debido a que el grado de idoneidad es demasiado elevado. Sin embargo eso no nos importa para este tipo de pruebas, luego ya cambiaremos esos parámetros. Lo importante aquí es ver como ya se realiza la búsqueda a nivel autonómico, contactando con **A1**, que es el coordinador de Cataluña. La búsqueda a nivel zonal aún no es posible porque no está registrado el coordinador de la Zona II.

Añadimos los coordinadores de zona:

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent AH:pfc_ei.Historico A1:pfc_ei.Autonomia
A2:pfc_ei.Autonomia A3:pfc_ei.Autonomia A4:pfc_ei.Autonomia Z1:pfc_ei.Zona
Z2:pfc_ei.Zona Z3:pfc_ei.Zona Z4:pfc_ei.Zona Z5:pfc_ei.Zona Z6:pfc_ei.Zona
H2:pfc_ei.Hospital H3:pfc_ei.Hospital H4:pfc_ei.Hospital H5:pfc_ei.Hospital
H6:pfc_ei.Hospital H7:pfc_ei.Hospital H8:pfc_ei.Hospital H9:pfc_ei.Hospital
H10:pfc_ei.Hospital H11:pfc_ei.Hospital
```

Cada coordinador representa a su zona, así **Z1** es de la Zona I, **Z2** de la Zona II, etc. El resultado es el que sigue:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:43:47 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
  H3
  H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 23 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la búsqueda hacia arriba (CN)...
H: Coordinador Nacional (CN) no registrado
H: No se ha encontrado ningun paciente apto en el Pais
```

Siguen sin encontrarse pacientes debido al grado de idoneidad, pero ya se realiza la búsqueda a nivel zonal, mediante **Z2**. Ya sólo queda poder buscar a nivel nacional, lo vemos en la siguiente prueba.

Finalmente añadimos el Coordinador Nacional y ya tenemos todos los agentes en marcha en el sistema:

```
java jade.Boot -platform H:pfc_ei.Hospital ClusDM:pfc_ei.ClusDMResponderAgent
Especialista:pfc_ei.EspecialistaAgent AH:pfc_ei.Historico CN:pfc_ei.Nacional
```

A1:pfc_ei.Autonomia A2:pfc_ei.Autonomia A3:pfc_ei.Autonomia
A4:pfc_ei.Autonomia Z1:pfc_ei.Zona Z2:pfc_ei.Zona Z3:pfc_ei.Zona
Z4:pfc_ei.Zona Z5:pfc_ei.Zona Z6:pfc_ei.Zona H2:pfc_ei.Hospital
H3:pfc_ei.Hospital H4:pfc_ei.Hospital H5:pfc_ei.Hospital H6:pfc_ei.Hospital
H7:pfc_ei.Hospital H8:pfc_ei.Hospital H9:pfc_ei.Hospital H10:pfc_ei.Hospital
H11:pfc_ei.Hospital

El resultado de la búsqueda de un paciente óptimo es:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:47:36 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 23 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la búsqueda hacia arriba (CN)...
H: Iniciado Request-Protocol con CN
H: Recibido Agree de CN
H: No se han encontrado pacientes aptos a nivel de CN
H: No se ha encontrado ningun paciente apto en el País

Finalmente vemos como, aunque se siguen sin encontrar pacientes, el sistema realiza la búsqueda completa, llegando al nivel nacional y examinando las 6 zonas del país. En la ventana de MS-DOS donde se ejecuta el sistema se puede ver información complementaria a la que se ve en el cuadro de resultados, pudiendo comprobar como la búsqueda pasa por cada una de las seis zonas.

Pasamos ahora a la segunda fase de pruebas y cambiaremos algunos parámetros para ver como reacciona el sistema. Empezamos por bajar el grado de idoneidad, ya que hemos comprobado que sólo existía un paciente óptimo en todo el país (el número 9 de **H**). Realizamos la búsqueda de pacientes de grado *excelente*, sin tocar el resto de parámetros.

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 19:52:29 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2

H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 23 pacientes propuestos
H: Se han encontrado 1 pacientes aptos en Tarragona
9 => excel_lent
H: Paciente escogido => 9 de H3
H: Recibido un mensaje de tipo Final
H: Contract-Net finalizado satisfactoriamente
H: Iniciado Request-Protocol con AH

ÓRGANO DONADO: cor
ID PACIENTE: 9
HOSPITAL PACIENTE: H3
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Sat Sep 09 00:00:00 CEST 1961
FECHA DE ESPERA PACIENTE: Fri Dec 10 00:00:00 CET 1999
PESO PACIENTE: 70.0
TAMAÑO ÓRGANO PACIENTE: 35.4
TIPO SANGRE PACIENTE: 1

Vemos como ya encontramos otro nuevo paciente, se trata también del número 9, pero esta vez del hospital **H3**, también de Tarragona. La búsqueda ha concluido sin tener que llegar al nivel autonómico, encontrando un paciente en la misma ciudad.

Realizamos otra búsqueda de un paciente *excelente*, con el siguiente resultado:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:10:35 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la busqueda hacia arriba (CN)...
H: Iniciado Request-Protocol con CN
H: Recibido Agree de CN
H: No se han encontrado pacientes aptos a nivel de CN
H: No se ha encontrado ningun paciente apto en el Pais

Vemos como ya no quedan más pacientes *excelentes* en el sistema.

Probaremos disminuyendo el grado de idoneidad de nuevo y realizando varias búsquedas más. Buscamos ahora pacientes de grado *favorable*:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:14:08 CEST 1960

H: Se han encontrado 5 pacientes aptos en el propio hospital
  6 => bo
  1 => bo
  11 => bastant_bo
  5 => favorable
  2 => favorable
H: Paciente escogido => 6
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)
```

```
-----
ÓRGANO DONADO: cor
ID PACIENTE: 6
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Wed Aug 04 00:00:00 CEST 1965
FECHA DE ESPERA PACIENTE: Mon Jan 01 00:00:00 CET 2001
PESO PACIENTE: 80.0
TAMAÑO ÓRGANO PACIENTE: 39.1
TIPO SANGRE PACIENTE: 3
-----
```

Vemos como hemos encontrado un paciente adecuado en el propio hospital. De hecho hemos encontrado 5 pacientes que cumplían los requisitos. Habíamos impuesto el grado mínimo de *favorable*, pero el sistema ha encontrado dos pacientes *buenos* y ha elegido uno de ellos, en concreto el número 6 del propio hospital.

Si ahora realizamos otra búsqueda idéntica a la anterior, lo lógico sería que el paciente escogido fuera el 1 del propio hospital, ya que tenía grado de *bueno*. Veamos pues lo que ocurre:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:16:45 CEST 1960

H: Se han encontrado 4 pacientes aptos en el propio hospital
  1 => bo
  11 => bastant_bo
  5 => favorable
  2 => favorable
H: Paciente escogido => 1
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)
```

ÓRGANO DONADO: cor
ID PACIENTE: 1
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Sun Jan 01 00:00:00 CET 1956
FECHA DE ESPERA PACIENTE: Thu Nov 04 00:00:00 CET 1999
PESO PACIENTE: 60.8
TAMAÑO ÓRGANO PACIENTE: 30.0
TIPO SANGRE PACIENTE: 1

En efecto, las previsiones eran ciertas y se ha escogido al paciente número 1.

Subimos el grado ahora a *bastante bueno*. Ahora la cosa cambia, lo lógico sería que ahora el paciente escogido fuera el 11, que tenía esa misma calificación en las dos búsquedas anteriores. En cambio, veamos lo que ocurre:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:18:52 CEST 1960

H: Se han encontrado 4 pacientes aptos en el propio hospital
 11 => excel_lent
 4 => molt_bo
 5 => bo
 8 => bastant_bo
H: Paciente escogido => 11
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)

ÓRGANO DONADO: cor
ID PACIENTE: 11
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Sun Sep 17 00:00:00 CEST 1967
FECHA DE ESPERA PACIENTE: Sun Sep 17 00:00:00 CEST 2000
PESO PACIENTE: 60.8
TAMAÑO ÓRGANO PACIENTE: 30.0
TIPO SANGRE PACIENTE: 0

El paciente escogido es el 11, como era de esperar, pero ahora ha obtenido una calificación de *excelente*, en lugar del *bastante bueno* que había conseguido antes. ¿Por qué ha ocurrido esto? La respuesta radica en el **SMAO** y su manera de evaluar los pacientes. Dependiendo de varios criterios, entre ellos el número de pacientes, **SMAO** evalúa de una u otra forma, de ahí que al ir eliminando posibles pacientes, las valoraciones cambien.

Probemos ahora de aumentar de nuevo el grado a *muy bueno*:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:23:30 CEST 1960

H: Se han encontrado 1 pacientes aptos en el propio hospital
2 => molt_bo
H: Paciente escogido => 2
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)

ÓRGANO DONADO: cor
ID PACIENTE: 2
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Fri Jan 02 00:00:00 CET 1970
FECHA DE ESPERA PACIENTE: Sat May 12 00:00:00 CEST 2001
PESO PACIENTE: 65.8
TAMAÑO ÓRGANO PACIENTE: 31.2
TIPO SANGRE PACIENTE: 3

La cosa cambia radicalmente y el paciente escogido ahora es el número 2, que no había aparecido en las búsquedas anteriores. Nuevamente tiene mucho que ver la manera de evaluar de SMAO.

Repetimos de nuevo la búsqueda y vemos como esta vez se examinan más niveles a parte del propio hospital:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:25:21 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: Se han encontrado 2 pacientes aptos a nivel de A
24 => molt_bo
9 => molt_bo
H: Paciente escogido => 9 de H5
H: Recibido Inform de A1
H: Búsqueda finalizada correctamente

H: Iniciado Request-Protocol con AH

ÓRGANO DONADO: cor
ID PACIENTE: 9
HOSPITAL PACIENTE: H5
LOCALIDAD PACIENTE: Palau
PROVINCIA PACIENTE: Barcelona
FECHA DE NACIMIENTO PACIENTE: Sat Sep 09 00:00:00 CEST 1961
FECHA DE ESPERA PACIENTE: Fri Dec 10 00:00:00 CET 1999
PESO PACIENTE: 70.0
TAMAÑO ÓRGANO PACIENTE: 35.4
TIPO SANGRE PACIENTE: 1

Tras intentar la búsqueda en el mismo hospital y en la misma ciudad, entra en acción el coordinador de Cataluña (A1), que encuentra en su comunidad dos pacientes *muy buenos*, el número 24 y el número 9. Hay que destacar que ese número 24 no es en realidad el paciente 24 de ningún hospital (ya hemos dicho que cada hospital tan solo tenía 15 pacientes), sino que es producto de la nueva numeración que se da a los pacientes cuando se juntan las listas que provienen de los distintos hospitales. Se guarda en una tabla el número de paciente dentro de cada hospital, para luego poder volver a hacer la conversión a la inversa.

Finalmente se escoge al número 9 del hospital H5, que pertenece a la localidad de Palau, en la provincia de Barcelona.

Repetiremos esta búsqueda dos veces más, para agotar los pacientes *muy buenos* a nivel autonómico y pasar al nivel zonal:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:31:15 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
 H3
 H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: Se han encontrado 1 pacientes aptos a nivel de A
 9 => molt_bo
H: Paciente escogido => 9 de H4
H: Recibido Inform de A1
H: Búsqueda finalizada correctamente
H: Iniciado Request-Protocol con AH

ÓRGANO DONADO: cor

ID PACIENTE: 9
HOSPITAL PACIENTE: H4
LOCALIDAD PACIENTE: Barcelona
PROVINCIA PACIENTE: Barcelona
FECHA DE NACIMIENTO PACIENTE: Sat Sep 09 00:00:00 CEST 1961
FECHA DE ESPERA PACIENTE: Fri Dec 10 00:00:00 CET 1999
PESO PACIENTE: 70.0
TAMAÑO ÓRGANO PACIENTE: 35.4
TIPO SANGRE PACIENTE: 1

Vemos como de nuevo el paciente escogido es el número 9, pero esta vez del hospital **H4**, de Barcelona. Esto se debe a que el paciente que más se ajusta a las características por defecto del órgano donado coincide en casi todos los hospitales con el número 9, de ahí esa repetición del número de paciente escogido.

Una última búsqueda de paciente *muy bueno* nos lleva al nivel zonal:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:32:53 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
 H3
 H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: Se han encontrado 3 pacientes aptos a nivel de Z
 39 => molt_bo
 24 => molt_bo
 9 => molt_bo
H: Paciente escogido => 9 de H6
H: Recibido Inform de Z2
H: Búsqueda finalizada correctamente
H: Iniciado Request-Protocol con AH

ÓRGANO DONADO: cor
ID PACIENTE: 9
HOSPITAL PACIENTE: H6
LOCALIDAD PACIENTE: Mallorca
PROVINCIA PACIENTE: Baleares
FECHA DE NACIMIENTO PACIENTE: Sat Sep 09 00:00:00 CEST 1961
FECHA DE ESPERA PACIENTE: Fri Dec 10 00:00:00 CET 1999
PESO PACIENTE: 70.0
TAMAÑO ÓRGANO PACIENTE: 35.4
TIPO SANGRE PACIENTE: 1

Vemos como no se encuentran pacientes adecuados en Cataluña y se ha de pasar al nivel zonal. Los tres hospitales de fuera de Cataluña que pertenecen a la Zona II son los de Mallorca, Menorca e Ibiza. En ellos se realiza la búsqueda, concluyendo que el mejor paciente que cumple los requisitos es el número 9 (otra vez) del hospital **H6** en Mallorca.

Si ahora volvemos a aumentar el grado de idoneidad a *excelente* el sistema concluye que no hay ningún paciente apto en todo el país.

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 14:39:40 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
 H3
 H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la búsqueda hacia arriba (CN)...
H: Iniciado Request-Protocol con CN
H: Recibido Agree de CN
H: No se han encontrado pacientes aptos a nivel de CN
H: No se ha encontrado ningún paciente apto en el País

A partir de ahora fijaremos el grado de idoneidad en *malo* y jugaremos con otros parámetros.

Probemos de encontrar pacientes que esperan un *hígado*:

ÓRGANO DONADO: fetge
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 16:04:42 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
 H3
 H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 4 pacientes propuestos

```
H: Ordenacion-imposible
H: Comenzando la busqueda hacia arriba (A)...
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1
H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la busqueda hacia arriba (CN)...
H: Iniciado Request-Protocol con CN
H: Recibido Agree de CN
H: No se han encontrado pacientes aptos a nivel de CN
H: No se ha encontrado ningun paciente apto en el Pais
```

Vemos ahora otro tipo de problema, el de *ordenación-imposible*. Este problema surge porque sólo se han encontrado 4 pacientes que esperan un hígado en Tarragona y **SMAO** es incapaz de evaluarlos, siendo tan pocos. Ocurre lo mismo para el resto de búsquedas (aunque no se ve en el cuadro de resultados), ya que el número de pacientes de hígado en cada hospital es mucho menor que los que espera un corazón, como ya hemos comentado. Sucedería lo mismo para el resto de órganos.

Volvemos a donar un *corazón*, pero esta vez cambiamos el tipo de sangre a **AB**:

```
ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 16:10:53 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
  H3
  H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 22 pacientes propuestos
H: Se han encontrado 6 pacientes aptos en Tarragona
  6 => molt_bo
  2 => bo
  17 => bastant_bo
  8 => favorable
  4 => adequat
  9 => factible
H: Paciente escogido => 6 de H3
H: Recibido un mensaje de tipo Final
H: Contract-Net finalizado satisfactoriamente
H: Iniciado Request-Protocol con AH
```

```
-----
ÓRGANO DONADO: cor
ID PACIENTE: 6
HOSPITAL PACIENTE: H3
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Wed Aug 04 00:00:00 CEST 1965
FECHA DE ESPERA PACIENTE: Mon Jan 01 00:00:00 CET 2001
PESO PACIENTE: 80.0
TAMAÑO ÓRGANO PACIENTE: 39.1
TIPO SANGRE PACIENTE: 3
-----
```

Vemos como los resultados cambian, debido a la compatibilidad de la sangre entre donante y receptor, y ahora el paciente elegido es el número 6 del hospital **H3** de Tarragona, que ha obtenido una valoración de *muy bueno*. El motivo del fallo en la búsqueda en el propio hospital es que había pocos pacientes compatibles con la sangre **AB** y por tanto **SMAO** no podía realizar la valoración.

Seguimos con el tipo de sangre **AB** pero ahora vamos a cambiar los pesos de los criterios. Por ejemplo le pondremos la máxima importancia al tiempo de espera del paciente. Es decir ponemos un *100%* en el parámetro *Espera* y un *0%* en el resto:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 16:15:24 CEST 1960

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
 H3
 H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 21 pacientes propuestos
H: Se han encontrado 5 pacientes aptos en Tarragona
 16 => optim
 15 => excel_lent
 12 => molt_bo
 10 => bo
 3 => bastant_bo
H: Paciente escogido => 10 de H3
H: Recibido un mensaje de tipo Final
H: Contract-Net finalizado satisfactoriamente
H: Iniciado Request-Protocol con AH

ÓRGANO DONADO: cor
ID PACIENTE: 10
HOSPITAL PACIENTE: H3
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Thu Jan 02 00:00:00 CET 1975
FECHA DE ESPERA PACIENTE: Fri May 12 00:00:00 CEST 2000
PESO PACIENTE: 65.4
TAMAÑO ÓRGANO PACIENTE: 29.2
TIPO SANGRE PACIENTE: 3

Lo que ocurre en esta situación es que de entre los pacientes compatibles con el órgano donado, se escoge el número 10 de **H3** porque es el que más tiempo lleva esperando el órgano, concretamente desde el 12 de mayo de 2000.

Volvemos a poner el tipo de sangre en 0, ya que así hay más pacientes compatibles y probamos de dar el 100% de la importancia a la *edad*, en este caso:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Mon Oct 10 16:23:44 CEST 1960

H: Se han encontrado 5 pacientes aptos en el propio hospital
5 => adequat
10 => fluix
8 => desfavorable
7 => bastant_dolent
4 => dolent
H: Paciente escogido => 5
H: Iniciado Request-Protocol con AH
H: Actualizando la BD (BD1)

ÓRGANO DONADO: cor
ID PACIENTE: 5
HOSPITAL PACIENTE: H
LOCALIDAD PACIENTE: Tarragona
PROVINCIA PACIENTE: Tarragona
FECHA DE NACIMIENTO PACIENTE: Sat Sep 23 00:00:00 CEST 1950
FECHA DE ESPERA PACIENTE: Fri Dec 31 00:00:00 CET 1999
PESO PACIENTE: 77.2
TAMAÑO ÓRGANO PACIENTE: 40.3
TIPO SANGRE PACIENTE: 1

En este caso se encuentran 5 pacientes aptos en el propio hospital, escogiendo al número 5 como el mejor de ellos, debido a la compatibilidad de edad entre él y el donante.

Si ahora ponemos todos los parámetros como los teníamos al principio y cambiamos el año de nacimiento del donante, concretamente a 1990, ocurre lo siguiente:

ÓRGANO DONADO: cor
ID DONANTE: 0
HOSPITAL DONANTE: H
LOCALIDAD DONANTE: Tarragona
PROVINCIA DONANTE: Tarragona
FECHA DE NACIMIENTO DONANTE: Wed Oct 10 16:30:34 CEST 1990

H: No se han encontrado pacientes aptos en el propio hospital
H: Hospitales encontrados en Tarragona => 2:
H3
H2
H: Empieza el Contract Net dentro de la misma ciudad (Tarragona)
H: Analizando propuestas de Hospitales...
H: 20 pacientes propuestos
H: No se han encontrado pacientes aptos en Tarragona
H: No se han encontrado pacientes aptos en Tarragona
H: Iniciado Request-Protocol con A1
H: Recibido Agree de A1

H: No se han encontrado pacientes aptos a nivel de A
H: Iniciado Request-Protocol con Z2
H: Recibido Agree de Z2
H: No se han encontrado pacientes aptos a nivel de Z
H: Comenzando la búsqueda hacia arriba (CN)...
H: Iniciado Request-Protocol con CN
H: Recibido Agree de CN
H: No se han encontrado pacientes aptos a nivel de CN
H: No se ha encontrado ningún paciente apto en el País

No se encuentra ningún receptor adecuado, ya que la edad del donante (12 años) no es compatible con ninguno de los pacientes del sistema.

Ya hemos realizado suficientes pruebas para comprobar el buen funcionamiento del sistema. Veamos lo que ha ocurrido en una de las BD, por ejemplo la del hospital **H**.

Id.	Org.	Sangre	Peso	Tamaño	B0	B1	DR0	DR1	Nacimiento	Espera	Selecc.
1	cor	A	60,8	30	Sí	Sí	Sí	No	01/01/1956	04/11/1999	H
2	cor	AB	65,8	31,2	Sí	Sí	Sí	Sí	02/01/1970	12/05/2001	H
3	cor	A	75	39,64	No	No	Sí	No	04/07/1977	09/08/2000	No
4	cor	AB	65,9	37,1	Sí	Sí	Sí	No	01/02/1945	12/05/2001	No
5	cor	A	77,2	40,3	No	No	Sí	No	23/09/1950	31/12/1999	H
6	cor	AB	80	39,1	Sí	Sí	Sí	Sí	04/08/1965	01/01/2001	H
7	cor	A	78	28,3	Sí	No	Sí	No	01/01/1976	01/12/2000	No
8	cor	AB	65	37	Sí	No	Sí	Sí	02/01/1975	12/05/2001	No
9	cor	A	70	35,4	No	No	Sí	No	09/09/1961	10/12/1999	H
10	cor	AB	65,4	29,2	Sí	Sí	No	Sí	02/01/1975	12/05/2000	No
11	cor	O	60,8	30	Sí	Sí	Sí	Sí	17/09/1967	17/09/2000	H
12	cor	AB	100	40,2	No	Sí	Sí	Sí	12/08/1974	13/07/2001	No
13	cor	A	80	50	No	No	Sí	Sí	18/09/1952	19/05/2000	No
14	cor	AB	65,9	37,1	Sí	Sí	No	No	01/02/1990	22/04/2001	No
15	cor	A	64,2	35,3	Sí	No	Sí	No	23/09/1935	30/06/1999	No

(Se han omitido algunas columnas como *Hospital*, *Localidad* y *Provincia*, que son iguales para todos los pacientes con tal de que la tabla cupiera en la hoja.)

Tabla 3 - BD de datos del hospital H de Tarragona tras algunas donaciones

La columna que nos interesa ahora es la última, la que marca si un paciente ha sido *Seleccionado* o no. Vemos algunos que tienen un *No* en esa columna, eso significa que no han sido escogidos y por tanto pueden entrar aún en las siguientes búsquedas. Sin embargo, como hemos visto antes, el número de pacientes libres no es suficiente como para que **SMAO** pueda evaluarlos si sólo se miran los del propio hospital. Esto hace que debido a la mecánica interna de **SMAO**, a partir de un cierto número de pacientes mínimo, la búsqueda en el propio hospital se hace inútil, y nunca dará resultados. Por el contrario vemos algunos pacientes marcados con un *H*. Esto significa que han sido escogidos por el hospital **H**, en este caso el mismo, para ser receptores de una donación. Vemos como los pacientes marcados coinciden lógicamente con los escogidos en las pruebas anteriores: los números 1, 2, 5, 6, 9 y 11.

8. CÓDIGO

A continuación se muestra el código fuente de los distintos ficheros comentados en el capítulo 5.

8.1. Hospital.java

```
/*
*****
**
Hospital

PFC Jaime Bocio Sanz
2-4-2002

Agente que representa a un Hospital. Es el encargado de recibir del usuario
las órdenes para encontrar el paciente adecuado para el órgano. Inicia todo
el
proceso de búsqueda y toma la decisión final. También se comunica con el
Agente
Histórico para actualizar la BD de donaciones.

*****
/

package pfc_ei;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.sl.sl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.proto.*;
import jade.onto.sl.sl.*;

import jade.gui.*;

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

//Paquete con la ontología definida
import pfc_ei.ontology.*;

/*
*****
*/

public class Hospital extends GuiAgent {

    public static Interficie gui; //Interficie gráfica
    public static Entrada_Datos gui_datos; //Interficie gráfica

    // Grados de calificación de AO
    String grados[] = {"optim", "excel_lent", "molt_bo",
        "bo", "bastant_bo", "favorable", "adequat", "factible", "fluix",
        "desfavorable", "bastant_dolent", "dolent", "molt_dolent", "fatal",
        "pessim"};
```

```
/* Se piden mediante una ventana al registrar un agente de Hospital */
String ciudad="Tarragona"; //Indica la ciudad donde se encuentra el
hospital
String provincia;          //Indica la provincia

String bd = "BD";  /* Indica el nombre de la BD del H. Se indica en el
registro */

int grado_exigencia;

//Ctes para interactuar con la interficie
public final static int PETICION = 1001;
public final static int LIMPIA_BD = 1002;
public final static int INTERFICIE = 1003;
public final static int REGISTRO = 1004;

public static int distancia;
public static int peso;
public static int tamanyo;
public static int espera;
public static int antigenos;
public static int edad;

/*****
*/

/* CONSTRUCTOR DE LA CLASE Hospital */
public Hospital() {
    super();
}

/*****
*/

/* ESPECIE DE HANDLER PARA LOS EVENTOS QUE LE LLEGAN DESDE LA INTERFICIE */
protected void onGuiEvent(GuiEvent ev) {
    switch(ev.getType()) {
        //Inicia la búsqueda de un paciente
        case PETICION:
            System.out.println(getLocalName()+" : Peticion desde la interficie!");

            Paciente donante = new Paciente();
            donante = (Paciente) ev.getSource();
            grado_exigencia = gui.Grado.getSelectedIndex();

            System.out.println(getLocalName() + " : Organo donado => " +
donante.getorgano());
            addBehaviour(new Inicia_proceso_busqueda(Hospital.this, donante));
            break;

            //Limpia las BD's, "desmarcando" los pacientes seleccionados
            case LIMPIA_BD:
                gui.Resultado.append("\n\n-----
-----"
                                + "-----
-----");
                gui.Resultado.append("\n Limpiando las BD's: ");

                /**@todo Los indices i y j deberían cambiar si se modifican las BD's */

                for (int i=1;i<12;i++) {
                    gui.Resultado.append("\n          BD" + i + " => Paciente: ");
                    for (int j=1;j<16;j++) {
                        Actualiza_BD(j,"BD"+i,"No");
                        gui.Resultado.append("      " + j);
                    }
                }
            }
        }
    }
}
```

```

        gui.Resultado.append("\n-----"
        -----"
                                + "-----"
        -----\n\n");
        gui.Limpia_BD.setEnabled(true);
        gui.Ok.setEnabled(true);
        break;

//Inicia la interficie de un hospital, después del registro
case INTERFICIE:
    ciudad = gui_datos.ciudad.getText();
    provincia = gui_datos.provincia.getText();
    bd = gui_datos.bd.getText();
    gui_datos.hide();
    Registro_en_el_df(ciudad);
    addBehaviour(new Responde_Contract_Net(Hospital.this));
    gui = new Interficie(Hospital.this);
    gui.setVisible(true);
    break;

//Registra un hospital sin iniciar la interficie
case REGISTRO:
    ciudad = gui_datos.ciudad.getText();
    provincia = gui_datos.provincia.getText();
    bd = gui_datos.bd.getText();
    Registro_en_el_df(ciudad);
    addBehaviour(new Responde_Contract_Net(Hospital.this));
    gui_datos.hide();
    break;
    }
}

/*****
*/

/* FUNCIÓN QUE REGISTRA A UN AGENTE EN EL DF. SE LE PASA COMO PARÁMETRO LA
CIUDAD
DEL HOSPITAL, PARA FACILITAR LAS BÚSQUEDAS POSTERIORES EN EL DF. SE
REGISTRAN
AGENTES DE TIPO HOSPITAL, CON EL LENGUAJE Y LA ONTOLOGÍA DEFINIDAS. */
public void Registro_en_el_df(String ciudad) {
    ServiceDescription sd1=new ServiceDescription();
    //Indicamos los parametros del Agente
    sd1.setType("Hospital");
    sd1.setName(getLocalName());
    sd1.setOwnership(ciudad);
    //Obtenemos una instancia del DF
    DFAgentDescription dfd=new DFAgentDescription();
    //Nos añadimos como un nuevo servicio
    dfd.addServices(sd1);
    //Indicamos nuestro nombre
    AID aid=new AID();
    aid.setName(getLocalName());
    dfd.setName(aid);
    //Registramos el lenguaje
    registerLanguage(SLCodec.NAME, new SLCodec());
    //Registro de la ontología
    registerOntology(JBSOntology.NAME, JBSOntology.instance());

    System.out.println("Hospital (" + getLocalName() + " de " +
sd1.getOwnership() + ") on-line");

    try{
        //Nos registramos al DF
        DFServiceCommunicator.register(this,dfd);
    }catch (FIPAException e) {

```

```

        System.err.println(getLocalName()+" Error en el registro
al DF. Error: "+e.getMessage());
        doDelete();
    }
}

/*****
*/

/* FUNCIÓN QUE DESREGISTRA A UN AGENTE DEL DF. SE USA EN EL "takeDown" DEL
AGENTE. */
public void Desregistro_del_df() {
    //Obtenemos una instancia del DF
    DFAgentDescription dfd=new DFAgentDescription();
    AID aid=new AID();
    aid.setName(getLocalName());
    dfd.setName(aid);
    try{
        //Nos desregistramos
        DFServiceCommunicator.deregister(this,dfd);
        System.out.println("Agente " + getLocalName() + " eliminado
del DF");
    }catch (FIPAException e) {
        System.out.println(getLocalName()+" Error en el
desregistro del DF. Error: "+e.getMessage());
    }
}

/*****
*/

/* FUNCIÓN QUE BUSCA EN EL DF TODOS LOS NOMBRES DE LOS AGENTES QUE CUMPLEN
UN CIERTO SERVICIO. EN CONCRETO DEVUELVE UNA LISTA CON TODOS LOS AGENTES
DEL TIPO 'tipo' CUYO PROPIETARIO ES 'propietario', EXCEPTO LOS AGENTES
CUYO
NOMBRE ES 'excepto'. */
public List Busca_en_DF(String propietario, String tipo, String excepto) {
    List resultado = new ArrayList();
    DFAgentDescription dfad=new DFAgentDescription();
    ServiceDescription sd=new ServiceDescription();
    List result;

    sd.setOwnership(propietario);
    sd.setType(tipo);
    dfad.addServices(sd); // Añadimos el servicio al
descriptor de agente DF, para la búsqueda

    try { // Procedemos a buscar en df lo que queremos (dfad)
        result = jade.domain.DFServiceCommunicator.search(super,dfad);
        if (result.isEmpty())
            return resultado;

        for (int i=0;i<result.size();i++) {
            dfad = (DFAgentDescription) result.get(i);
            if (!dfad.getName().getName().equalsIgnoreCase(excepto))
                resultado.add(dfad.getName().getName()); //Con
esto obtenemos el nombre del agente
        }
    } catch (Exception exc) {
        System.out.println( exc.toString() );
    }
    return resultado;
}

/*****
*/

/* FUNCIÓN QUE REALIZA UNA CONSULTA EN LA BASE DE DATOS 'bd1'. BUSCA TODOS

```

```

    LOS PACIENTES QUE ENCUENTRA EN EL HOSPITAL DE UN CIERTO TIPO DE ÓRGANO
    'org'.
    DEVUELVE UNA LISTA DE PACIENTES, DE TIPO List. */
    public List Busca_pacientes_en_BD(String org, String bd1) {
        Connection con;
        Statement stmt;
        ResultSet rs = null;
        List lista_a_retornar = new ArrayList();

        try { // Conexión a la BD
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection("jdbc:odbc:" + bd1); //BD");
            stmt = con.createStatement();
            // Búsqueda de los pacientes del órgano "org"
            if (stmt.execute("Select * from Pacientes where Organo='" + org + "'"))
        {
            rs = stmt.getResultSet();
        }
        int i=0;
        while (rs.next()) {
            Pacient paciente_actual = new Pacient();
            paciente_actual.Num = rs.getInt("Num");
            paciente_actual.Organo = rs.getString("Organo");
            paciente_actual.Hospital = rs.getString("Hospital");
            paciente_actual.Localitat = rs.getString("Localidad");
            paciente_actual.Provincia = rs.getString("Provincia");
            String sangre = rs.getString("Sangre");
            if (sangre.equals("A"))
                paciente_actual.Tipus_sang=DadesEspecialista.A;
            else
                if (sangre.equals("O"))
                    paciente_actual.Tipus_sang=DadesEspecialista.O;
                else
                    if (sangre.equals("B"))
                        paciente_actual.Tipus_sang=DadesEspecialista.B;
                    else
                        if (sangre.equals("AB"))
                            paciente_actual.Tipus_sang=DadesEspecialista.AB;

            paciente_actual.Pes = rs.getDouble("Peso");
            paciente_actual.Tamany = rs.getDouble("Tamanyo");
            paciente_actual.Antigens[0][0] = rs.getBoolean("B0");
            paciente_actual.Antigens[0][1] = rs.getBoolean("B1");
            paciente_actual.Antigens[1][0] = rs.getBoolean("DR0");
            paciente_actual.Antigens[1][1] = rs.getBoolean("DR1");
            paciente_actual.Data_naix.setTime(rs.getDate("Nacimiento"));
            paciente_actual.Data_esp.setTime(rs.getDate("Espera"));

            if (rs.getString("Seleccionado").equals("No")) {
                lista_a_retornar.add(paciente_actual);
                i++;
            }
        }
        System.out.println(bd1 + " (" + getLocalName() + "): Pacientes
        encontrados de " + org + " => " + i);
        con.close();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
    }
    return(lista_a_retornar);
}

/*****
*/

/* FUNCIÓN QUE ACTUALIZA LA BD. ES DECIR, MARCA COMO SELECCIONADO AL
PACIENTE

```

```
CON EL ID 'id_paciente' DE LA BD 'bd1'. PARA MARCARLO, INTRODUCE EN EL
CAMPO
'Seleccionado', EL NOMBRE DEL HOSPITAL ORIGEN. RETORNA 'true' SI SE HA
REALIZADO
LA OPERACIÓN CORRECTAMENTE. */
public boolean Actualiza_BD(int id_paciente, String bd1, String hospital) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1); //BD");
        stmt = con.createStatement();
        stmt.executeUpdate("Update Pacientes set Seleccionado='"+hospital+"
where Num="+id_paciente);
        con.close();
        return true;
    }catch(Exception e){
        System.out.println(bd1 + ": " + e.toString());
        return false;
    }
}

/*****
*/

/* FUNCIÓN QUE DEVUELVE LOS DATOS DEL PACIENTE 'ind' DE LA LISTA 'lista', EN
UN OBJETO DE TIPO Paciente. */
public Paciente Obtiene_datos_paciente(Lista_objetos lista, int ind) {
    Iterator it = lista.getAllpacientes();
    Paciente pac;
    int i=0;
    while (it.hasNext()) {
        pac = (Paciente) it.next();
        if (pac.getid().intValue()==ind)
            return (pac);
        else {
            i++;
        }
    }
    return null;
}

/*****
*/

/* FUNCIÓN QUE MUESTRA LOS DATOS DEL PACIENTE 'p' ESCOGIDO EN LA PETICIÓN
POR LA
INTERFICIE GRÁFICA. */
public void Mostrar_resultado(Paciente p) {

    gui.Resultado.append("\n\n-----
-----"
                        + "-----
-----");
    gui.Resultado.append("\n-----
-----"
                        + "-----
-----");
    gui.Resultado.append("\n ÓRGANO DONADO: " + p.getorgano());
    gui.Resultado.append("\n ID PACIENTE: " + p.getid());
    gui.Resultado.append("\n HOSPITAL PACIENTE: " + p.gethospital());
    gui.Resultado.append("\n LOCALIDAD PACIENTE: " + p.getlocalidad());
    gui.Resultado.append("\n PROVINCIA PACIENTE: " + p.getprovincia());
    gui.Resultado.append("\n FECHA DE NACIMIENTO PACIENTE: " +
p.getnacimimiento());
    gui.Resultado.append("\n FECHA DE ESPERA PACIENTE: " + p.getespera());
```

```

        gui.Resultado.append("\n PESO PACIENTE: " + p.getpeso());
        gui.Resultado.append("\n TAMAÑO ÓRGANO PACIENTE: " + p.gettamanyo());
        gui.Resultado.append("\n TIPO SANGRE PACIENTE: " + p.getsangre() + "\n");
        gui.Resultado.append("-----"
+ "-----"
+ "-----\n");
        gui.Resultado.append("-----"
+ "-----\n\n");
    }

/*****
*/

/* FUNCIÓN QUE REALIZA UNA CONSULTA EN LA BASE DE DATOS. BUSCA EN LA TABLA
'tabla' DE LA BD 'bd1', LA COLUMNA 'col_return', DE LOS REGISTROS QUE
CUMPLAN
QUE 'col' = 'val', EXCEPTO LOS QUE SON 'excepto'. NO DEVUELVE VALORES
REPETIDOS. */
public List Busca_en_BD(String bd1, String tabla, String col_return, String
col, String val, String excepto) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;
    List lista_a_retornar = new ArrayList();
    String res = "";
    String aux = "";

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1);
        stmt = con.createStatement();
        // Búsqueda de los pacientes del órgano "org"
        if (stmt.execute("Select * from " + tabla + " where " + col + "=" +
val + "\"")) {
            rs = stmt.getResultSet();
        }
        int i=0;
        while (rs.next()) {
            aux = rs.getString(col_return);
            if (!aux.equals(res)) {
                res=aux;
                if (!res.equals(excepto)) {
                    lista_a_retornar.add(res);
                    i++;
                }
            }
        }
        System.out.println(bd1 + " (" + getLocalName() +"): Registros
encontrados de " + col + "=" + val + " => " + i);
        con.close();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
    }
    return(lista_a_retornar);
}

/*****
*/

/* FUNCIÓN QUE REALIZA UNA CONSULTA EN EL A0. SE LE PASA LA LISTA 'envio'
CON
LOS PACIENTES Y EL DONANTE, Y LA FUNCIÓN DEVUELVE EL RESULTADO EN FORMA DE
OBJETO Resultado. */
public Resultado Consulta_A0(Lista_objetos envio) {
    envio.clearpesos();

```

```

envio.addpesos(new Valor_atrib("Distancia",new Long(distancia)));
envio.addpesos(new Valor_atrib("Pes",new Long(peso)));
envio.addpesos(new Valor_atrib("Tamany",new Long(tamanyo)));
envio.addpesos(new Valor_atrib("Antigens",new Long(antigenos)));
envio.addpesos(new Valor_atrib("Edad",new Long(edad)));
envio.addpesos(new Valor_atrib("Espera",new Long(espera)));
ACLMessage msg_req = Prepara_msg(ACLMessage.REQUEST,new
AID("Especialista"),
    "Especialista",envio);

send(msg_req);

//Ahora hay que esperar la contestación de AO
ACLMessage msg_respuesta = new ACLMessage(ACLMessage.NOT_UNDERSTOOD);
MessageTemplate mt_respuesta = MessageTemplate.MatchSender(new
AID("Especialista"));

msg_respuesta = blockingReceive(mt_respuesta);
System.out.println(getLocalName() + ": Respuesta recibida de
Especialista");

return (Resultado) Extrae_contenido(msg_respuesta);
}

/*****
*/

/* FUNCIÓN QUE EXTRAER EL CONTENIDO DE UN MENSAJE 'msg' . FACILITA LA TAREA.
SE LE PASA EL MENSAJE DEL CUAL QUEREMOS EXTRAER EL CONTENIDO, Y SE
DEVUELVE
EL OBJETO DE LA ONTOLOGÍA QUE CONTIENE. */

public Object Extrae_contenido(ACLMessage msg) {
    List l = new ArrayList();
    SL_Action a = new SL_Action();
    Object resultado = new Object();
    try {
        l= extractContent(msg);
        a = (SL_Action) l.get(0);
        resultado = (Object) a.get_1();
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return resultado;
}

/*****
*/

/* FUNCIÓN QUE GENERA UN OBJETO DE TIPO Lista_objetos A PARTIR DE UNO DE
TIPO
Lista_pacientes. EL PACIENTE CON EL IDENTIFICADOR 0 ES EL DONANTE, ESTO
HACE
POSIBLE QUE ESTE OBJETO SIRVA TANTO PARA ENVIAR INFORMACIÓN DE UN DONANTE,
DE UNA LISTA DE PACIENTES, O DE TODO COMBINADO. */
public Lista_objetos Crea_lista_objetos(Lista_Pacientes lista) {
    // Se crea un objeto Lista_objetos con tantos pacientes como sea necesario
    Lista_objetos resultado = new Lista_objetos(new Long
(lista.numero_pacientes));
    Paciente paciente;
    Long antigens_long[][]= new Long[2][2];
    for (int i=0; i<lista.pacientes.size();i++){
        Pacient paciente_actual = (Pacient) lista.pacientes.get(i);
        //Hemos de pasar la tabla de antigens de booleanos a otra de Long
        for (int x=0;x<2;x++)
            for (int y=0;y<2;y++)
                if (paciente_actual.Antigens[x][y])
                    antigens_long[x][y]=new Long(1);
                else

```

```

        antigens_long[x][y]=new Long(0);
        //Construimos un objeto de tipo Paciente, con los campos:
        //
        Num,Organo,Hospital,Localidad,Provincia,Sangre,Peso,Tamanyo,B0,B1,DR0,DR1,Naci
        miento,Espera
        paciente = new Paciente(new
        Long(paciente_actual.Num),paciente_actual.Organ,paciente_actual.Hospital,
        paciente_actual.Localitat,paciente_actual.Provincia,new
        Long(paciente_actual.Tipus_sang),
        new Double(paciente_actual.Pes),new
        Double(paciente_actual.Tamany),

        antigens_long[0][0],antigens_long[0][1],antigens_long[1][0],antigens_long[1][1]
        ],

        paciente_actual.Data_naix.getTime(),paciente_actual.Data_esp.getTime());
        //Añadimos el paciente creado a la lista de pacientes
        resultado.addpacientes(paciente);
    }
    return (resultado);
}

/*****
*/

/* FUNCIÓN QUE DADO UN GRADO DE EXIGENCIA EN FORMA DE STRING DEVUELVE EL
NÚMERO
DE GRADO QUE ES, TENIENDO EN CUENTA QUE EL 0 ES Óptimo Y EL 14 Pésimo. */
public int Grado_a_Numero(String grado) {
    int i=0;
    while (!grado.equals(grados[i]))
        i++;
    return i;
}

/*****
*/

/* FUNCIÓN QUE DADA UNA LISTA DE RESULTADOS DE AO 'resultado' , Y UN NIVEL
DE EXIGENCIA 'grado', DEVUELVE UNA LISTA CON LOS PACIENTES, QUE SERÍAN
MÁS ADECUADOS PARA EL TRANSPLANTE, ORDENADOS DE MAYOR A MENOR. LA LISTA
CONTIENE OBJETOS DE TIPO Grado. */
public List Busca_pacientes_aptos(Resultado resultado, int grado) {
    List lista_a_retornar = new ArrayList();
    Iterator it = resultado.getAllasignacion();
    Grado g = new Grado();
    int i=1;

    while (it.hasNext()) { //Mientras haya pacientes en la lista, hay que
    mirarlos
        g = (Grado) it.next();
        if (Grado_a_Numero(g.getvalor())<=grado) //Mientras sea un grado válido
        se añaden
            lista_a_retornar.add(g);
        }
    return(lista_a_retornar);
}

/*****
*/

/* FUNCIÓN QUE ESCOGE DE ENTRE UNA LISTA DE PACIENTES, EL MÁS ÓPTIMO, ES
DECIR
EN PRINCIPIO EL PRIMERO DE LA LISTA, QUE SE SUPONE ESTÁ ORDENADA POR
GRADOS
DE EXIGENCIA. ES MUY SIMPLE, PERO COMO SE USARÁ VARIAS VECES, ES
PREFERIBLE
```

```

HACER UNA FUNCIÓN. SE LE PASA UNA LISTA DE PACIENTES CON PARES ID-GRADO Y
SE
    RETORNA EL ID DEL PACIENTE ESCOGIDO. */
public int Escoge_mejor_paciente(List lista) {
    if (!lista.isEmpty()) {
        Grado resultado = (Grado) lista.get(0);
        return(resultado.getidObjeto().intValue());
    } else
        return(0);
}

/*****
*/

/* COMPORTAMIENTO QUE INICIA UN PROTOCOLO REQUEST CON UN COORDINADOR, BIEN
DE
A, DE Z O EL CN. SE LE PASA EL DONANTE Y UN PARÁMETRO QUE INDICA EL NIVEL
AL
QUE SE ESTÁ REALIZANDO LA BÚSQUEDA. */
public class Coord_Request extends FipaRequestInitiatorBehaviour {

    AID AID_coord = new AID();
    Paciente donante;
    String busqueda;
    Paciente paciente_escogido_tipo_paciente;

    //Constructor
    public Coord_Request (Agent agent, ACLMessage requestMsg, MessageTemplate
mt, Paciente don, String busq) {
        super(agent,requestMsg,mt);
        Iterator it = requestMsg.getAllReceiver();
        AID_coord = (AID) it.next();
        gui.Resultado.append("\n" + getLocalName() + ": Iniciado Request-
Protocol con " + AID_coord);
        System.out.println(getLocalName() + ": Iniciado Request-Protocol con " +
AID_coord);
        donante = don;
        busqueda = busq;
    }

    protected void handleAgree(ACLMessage msg) {
        gui.Resultado.append("\n" + getLocalName() + ": Recibido Agree de " +
msg.getSender().getName());
        System.out.println(getLocalName() + ": Recibido Agree de " +
msg.getSender().getName());
        /* Se recibe la lista de pacientes que el coord ha recogido en el
Contract-Net */
        Lista_objetos pacientes_para_H = new Lista_objetos();
        boolean problema = false;

        pacientes_para_H = (Lista_objetos) Extrae_contenido(msg);
        /* Al juntar todos los pacientes de los distintos hospitales para hacer
        * la consulta al A0, no sabemos de qué hospital es cada paciente, y se
        * pueden repetir los identificadores, hay que modificar los
        * id's de los pacientes desde aquí, de tal manera que tengamos id's
        * secuenciales y en una lista guardemos, para cada id cambiado, el id
        * original y el hospital del que viene el paciente
        */
        int id=1;
        //Lista que contiene los verdaderos id's dentro de cada hospital de
        todos los pacientes propuestos
        List id_hosp = new ArrayList();
        Lista_objetos pacientes_para_A0 = new Lista_objetos();
        Iterator it = null;
        it = pacientes_para_H.getAllpacientes();
        Paciente paciente_actual = new Paciente();
        while (it.hasNext()) {
            //Sacamos un paciente de la lista de pacientes de un hospital

```

```

        paciente_actual = (Paciente) it.next();
        //Hemos de meter el id en una lista de pares id-hospital
        id_hosp.add(new
Eleccion(paciente_actual.gethospital(),paciente_actual.getid()));
        //Y ponerle un id correlativo
        paciente_actual.setid(new Long(id));
        //Lo metemos en la lista de pacientes total, la que se le pasará a A0
        pacientes_para_A0.addpacientes(paciente_actual);
        //Incrementamos el número de pacientes en la lista que pasaremos a A0
        pacientes_para_A0.setnpacientes(new
Long(pacientes_para_A0.getnpacientes().intValue()+1));
        id++;
    }

    /* Con la lista "pacientes_para_A0" hay que hacer la consulta a A0. */
    /* Hay que añadir el donante a la lista que se le envía a A0, y
incrementar el numPacientes */
    pacientes_para_A0.addpacientes(donante);
    pacientes_para_A0.setnpacientes(new
Long(pacientes_para_A0.getnpacientes().intValue()+1));

    /* Envío del mensaje a A0, para que evalúe los distintos pacientes
encontrados. */
    Resultado res = Consulta_A0(pacientes_para_A0);
    Eleccion paciente_escogido = new Eleccion();

    if (res.getnobjetos().intValue()!=0) { //Si la ordenación ha sido
posible
        List pacientes_aptos = Busca_pacientes_aptos(res,grado_exigencia);
        if (!pacientes_aptos.isEmpty()) { //Si hay pacientes aptos
            gui.Resultado.append("\n" + getLocalName() + ": Se han encontrado "
+ pacientes_aptos.size() + " pacientes aptos a nivel de " + busqueda);
            System.out.println(getLocalName() + ": Se han encontrado " +
pacientes_aptos.size() + " pacientes aptos a nivel de " + busqueda);
            Grado aux = new Grado();
            for (int i=0;i<pacientes_aptos.size();i++) {
                aux = (Grado) pacientes_aptos.get(i);
                gui.Resultado.append("\n          " + aux.getidObjeto() + " => " +
aux.getvalor());
                System.out.println("          " + aux.getidObjeto() + " => " +
aux.getvalor());
            }
            /* Escoger entre los distintos pacientes aptos y salida por
pantalla. */
            int id_escogido = Escoge_mejor_paciente(pacientes_aptos);

            /** Hay que meter en la variable paciente_escogido_tipo_paciente
            * todos los datos del paciente escogido, para luego pasarlos a AH
            */
            paciente_escogido_tipo_paciente =
Obtiene_datos_paciente(pacientes_para_A0,id_escogido);
            paciente_escogido = (Eleccion) id_hosp.get(id_escogido-1); //-1 pq
la 1ª posición es la 0
            //Ponemos bien el id del paciente escogido dentro del hospital
            paciente_escogido_tipo_paciente.setid(paciente_escogido.getid_paciente());

            gui.Resultado.append("\n" + getLocalName() + ": Paciente escogido =>
" + paciente_escogido.getid_paciente() + " de " +
paciente_escogido.getid_hospital());
            System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente() + " de " +
paciente_escogido.getid_hospital());
            ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM,msg.getSender(),busqueda,paciente_escogido);
            msg_inform.setConversationId(msg.getConversationId());
            send(msg_inform);
        } else //Si no hay pacientes aptos

```

```

        problema = true;
    } else //Si no se ha podido ordenar
        problema = true;
    if (problema) {
        gui.Resultado.append("\n" + getLocalName() + ": No se han encontrado
pacientes aptos a nivel de " + busqueda);
        System.out.println(getLocalName() + ": No se han encontrado pacientes
aptos a nivel de " + busqueda);
        //Le mandamos a coord la respuesta que espera, pero con una Elección
"nula"
        ACLMessage msg_inform = Prepara_msg(ACLMessage.INFORM,msg.getSender(),
busqueda,new Eleccion("No",new
Long(0)));
        msg_inform.setConversationId(msg.getConversationId());
        send(msg_inform);
        if (busqueda.equals("A")) {
            /** ZONA: Hemos de empezar la búsqueda para arriba */
            if (Inicia_busqueda_zona(donante)) {
                //Ya se ha iniciado el Coord_Request a nivel de "Z"
            } else { //Si Z no está registrado en DF o ha habido otro problema
                /** CN: Hay que empezar la búsqueda a nivel nacional */
                gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
busqueda hacia arriba (CN)...");
                System.out.println(getLocalName() + ": Comenzando la busqueda
hacia arriba (CN)...");
                if (Inicia_busqueda_nacional(donante)) {
                    //Ya se ha iniciado el Request a nivel de "CN"
                } else {
                    /** FIN: No se ha encontrado ningún paciente apto en España */
                    gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningun paciente apto en el Pais");
                    System.out.println(getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
                    gui.Ok.setEnabled(true);
                    gui.Limpia_BD.setEnabled(true);
                }
            }
        }
        if (busqueda.equals("Z")) {
            /** CN: Hay que empezar la búsqueda a nivel nacional */
            gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
busqueda hacia arriba (CN)...");
            System.out.println(getLocalName() + ": Comenzando la busqueda hacia
arriba (CN)...");
            if (Inicia_busqueda_nacional(donante)) {
                //Ya se ha iniciado el Request a nivel de "CN"
            } else {
                /** FIN: No se ha encontrado ningún paciente apto en España */
                gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningun paciente apto en el Pais");
                System.out.println(getLocalName() + ": No se ha encontrado ningun
paciente apto en el Pais");
                gui.Ok.setEnabled(true);
                gui.Limpia_BD.setEnabled(true);
            }
        }
        if (busqueda.equals("CN")) {
            gui.Resultado.append("\n" + getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
            System.out.println(getLocalName() + ": No se ha encontrado ningun
paciente apto en el Pais");
            gui.Ok.setEnabled(true);
            gui.Limpia_BD.setEnabled(true);
        }
    }
}

protected void handleInform(ACLMessage msg) {

```

```
gui.Resultado.append("\n" + getLocalName() + ": Recibido Inform de " +
msg.getSender().getName());
System.out.println(getLocalName() + ": Recibido Inform de " +
msg.getSender().getName());
gui.Resultado.append("\n" + getLocalName() + ": Busqueda finalizada
correctamente");
System.out.println(getLocalName() + ": Busqueda finalizada
correctamente");
// Recibimos la confirmación de que el coord ha finalizado correctamente
el Contract-Net
/* AH: Aquí se debe contactar con AH para indicarle los datos de la
donación. */
Lista_objetos lista_AH = new Lista_objetos(new Long(2));
lista_AH.addpacientes(donante);
lista_AH.addpacientes(paciente_escogido_tipo_paciente);
AID AID_AH = new AID((String) Busca_en_DF("AH", "Historico", "").get(0));
ACLMessage msg_req =
Prepara_msg(ACLMessage.REQUEST, AID_AH, "AH", lista_AH);
MessageTemplate mt = MessageTemplate.MatchSender(AID_AH);
addBehaviour(new AH_Request(Hospital.this, msg_req, mt));
gui.Ok.setEnabled(true);
gui.Limpia_BD.setEnabled(true);
/** GUI: Mostrar por la Interficie los datos del paciente */
Mostrar_resultado(paciente_escogido_tipo_paciente);
}

protected void handleNotUnderstood(ACLMessage msg) {
gui.Resultado.append("\n" + getLocalName() + ": Recibido Not-Understood
de " + msg.getSender().getName());
System.out.println(getLocalName() + ": Recibido Not-Understood de " +
msg.getSender().getName());
}

protected void handleFailure(ACLMessage msg) {
gui.Resultado.append("\n" + getLocalName() + ": Recibido Failure de " +
msg.getSender().getName());
System.out.println(getLocalName() + ": Recibido Failure de " +
msg.getSender().getName());
}

protected void handleRefuse(ACLMessage msg) {
gui.Resultado.append("\n" + getLocalName() + ": Recibido Refuse de " +
msg.getSender().getName());
System.out.println(getLocalName() + ": Recibido Refuse de " +
msg.getSender().getName());
if (busqueda.equals("A")) {
/** ZONA: Hemos de empezar la búsqueda para arriba */
if (Inicia_busqueda_zona(donante)) {
//Ya se ha iniciado el Coord_Request a nivel de "Z"
} else { //Si Z no está registrado en DF o ha habido otro problema
/** CN: Hay que empezar la búsqueda a nivel nacional */
gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
busqueda hacia arriba (CN)...");
System.out.println(getLocalName() + ": Comenzando la busqueda hacia
arriba (CN)...");
if (Inicia_busqueda_nacional(donante)) {
//Ya se ha iniciado el Request a nivel de "CN"
} else {
/** FIN: No se ha encontrado ningún paciente apto en España */
gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningun paciente apto en el Pais");
System.out.println(getLocalName() + ": No se ha encontrado ningun
paciente apto en el Pais");
gui.Ok.setEnabled(true);
gui.Limpia_BD.setEnabled(true);
}
```

```

    }
  }
}
if (busqueda.equals("Z")) {
  /** CN: Hay que empezar la búsqueda a nivel nacional */
  gui.Resultado.append("\n" + getLocalName() + ": Comenzando la búsqueda
hacia arriba (CN)...");
  System.out.println(getLocalName() + ": Comenzando la búsqueda hacia
arriba (CN)...");
  if (Inicia_busqueda_nacional(donante)) {
    //Ya se ha iniciado el Request a nivel de "CN"
  } else {
    /** FIN: No se ha encontrado ningún paciente apto en España */
    gui.Resultado.append("\n" + getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
    System.out.println(getLocalName() + ": No se ha encontrado ningun
paciente apto en el Pais");
    gui.Ok.setEnabled(true);
    gui.Limpia_BD.setEnabled(true);
  }
}
if (busqueda.equals("CN")) {
  /** FIN: No se ha encontrado ningún paciente apto en España */
  gui.Resultado.append("\n" + getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
  System.out.println(getLocalName() + ": No se ha encontrado ningun
paciente apto en el Pais");
  gui.Ok.setEnabled(true);
  gui.Limpia_BD.setEnabled(true);
}
}
}

/*****
*/

/* COMPORTAMIENTO QUE INICIA UN PROTOCOLO REQUEST CON AH, PARA INDICARLE
TODOS
LOS DATOS DE UNA DONACIÓN. */
public class AH_Request extends FipaRequestInitiatorBehaviour {

  AID AID_AH = new AID();

  //Constructor
  public AH_Request (Agent agent, ACLMessage requestMsg, MessageTemplate mt)
  {
    super(agent,requestMsg,mt);
    Iterator it = requestMsg.getAllReceiver();
    AID_AH = (AID) it.next();
    gui.Resultado.append("\n" + getLocalName() + ": Iniciado Request-
Protocol con " + AID_AH);
    System.out.println(getLocalName() + ": Iniciado Request-Protocol con " +
AID_AH);
  }

  protected void handleAgree(ACLMessage msg) {
    System.out.println(getLocalName() + ": Recibido Agree de " +
msg.getSender().getName());
  }

  protected void handleInform(ACLMessage msg) {
    System.out.println(getLocalName() + ": Recibido Inform de " +
msg.getSender().getName());
  }

  protected void handleNotUnderstood(ACLMessage msg) {

```

```
        System.out.println(getLocalName() + ": Recibido Not-Understood de " +
msg.getSender().getName());
    }

    protected void handleFailure(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Failure de " +
msg.getSender().getName());
    }

    protected void handleRefuse(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Refuse de " +
msg.getSender().getName());
    }
}

/*****
*/

/* FUNCIÓN QUE INICIA UNA BÚSQUEDA EN LA AUTONOMÍA. SE LE PASA EL DONANTE,
YA
QUE SE NECESITA SABER LA CIUDAD DE ÉSTE Y EL TIPO DE ÓRGANO DONADO.
DEVUELVE
UN BOOLEANO, QUE SERÁ CIERTO EN CASO DE QUE NO HAYA HABIDO PROBLEMAS. */
public boolean Inicia_búsqueda_autonomia(Paciente donante) {
    String autonomia = "";
    //Buscamos la autonomia a la que pertenece el donante
    List lista_auts = Busca_en_BD("BD-
Ciudades","Ciudades","Autonomia","Ciudad",donante.getLocalidad(),"");
    if (!lista_auts.isEmpty()) { //Si se ha encontrado la aut en la BD
        autonomia=(String) lista_auts.get(0);
        List auts = Busca_en_DF(autonomia,"Autonomia","");
        if (!auts.isEmpty()) { //Si el coord de aut está registrado en el DF
            AID AID_aut = new AID((String) auts.get(0));
            //Se inicia el Coord_Request para una búsqueda a nivel de "A"
            ACLMessage msg_req = Prepara_msg(ACLMessage.REQUEST,AID_aut,"A",new
Organo(donante.getorgano()));
            MessageTemplate mt = MessageTemplate.MatchSender(AID_aut);
            //A partir de ahora, la búsqueda continúa por el Coord_Request
            addBehaviour(new Coord_Request(Hospital.this,msg_req,mt,donante,"A"));
            return true;
        } else { //Si A no está registrado
            gui.Resultado.append("\n" + getLocalName() + ": Coordinador de " +
autonomia + " no registrado");
            System.out.println(getLocalName() + ": Coordinador de " + autonomia +
" no registrado");
            return false;
        }
    } else { //Si no se ha encontrado la autonomía en la BD
        gui.Resultado.append("\n" + getLocalName() + ": " + autonomía + " no
encontrada en la BD");
        System.out.println(getLocalName() + ": " + autonomía + " no encontrada
en la BD");
        return false;
    }
}

/*****
*/

/* FUNCIÓN QUE INICIA UNA BÚSQUEDA EN LA ZONA. SE LE PASA EL DONANTE, YA
QUE SE NECESITA SABER LA CIUDAD DE ÉSTE Y EL TIPO DE ÓRGANO DONADO.
DEVUELVE
UN BOOLEANO, QUE SERÁ CIERTO EN CASO DE QUE NO HAYA HABIDO PROBLEMAS. */
public boolean Inicia_búsqueda_zona(Paciente donante) {
    //Hay que buscar la autonomía del donante
    String zona = "";
    //Buscamos la autonomía a la que pertenece la ciudad origen
```

```

        String autonomia = (String) Busca_en_BD("BD-
Ciudades","Ciudades","Autonomía","Ciudad",donante.getlocalidad(),"").get(0);
        //Buscamos la zona a la que pertenece la autonomía
        List lista_zonas = Busca_en_BD("BD-
Ciudades","Ciudades","Zona","Autonomía",autonomia,"");
        if (!lista_zonas.isEmpty()) { //Si se ha encontrado la zona en la BD
            zona=(String) lista_zonas.get(0);
            //Buscamos al coordinador de la zona
            List zonas = Busca_en_DF(zona,"Zona","");
            if (!zonas.isEmpty()) { //Si el coord de zona está registrado en el DF
                AID AID_zona = new AID((String) zonas.get(0));
                //Comenzamos el Coord_Request a nivel de "Z"
                ACLMessage msg_req = Prepara_msg(ACLMessage.REQUEST,AID_zona,"Z",new
Organo(donante.getorgano()));
                MessageTemplate mt = MessageTemplate.MatchSender(AID_zona);
                addBehaviour(new Coord_Request(Hospital.this,msg_req,mt,donante,"Z"));
                return true;
            } else { //Si Z no está registrado
                gui.Resultado.append("\n" + getLocalName() + ": Coordinador de la
Zona" + zona + " no registrado");
                System.out.println(getLocalName() + ": Coordinador de la Zona" + zona
+ " no registrado");
                return false;
            }
        } else { //Si no se ha encontrado la zona en la BD
            gui.Resultado.append("\n" + getLocalName() + ": " + zona + " no
encontrada en la BD");
            System.out.println(getLocalName() + ": " + zona + " no encontrada en la
BD");
            return false;
        }
    }

/*****
*/
/* FUNCIÓN QUE INICIA UNA BÚSQUEDA EN EL PAÍS. SE LE PASA EL DONANTE, YA
EL TIPO DE ÓRGANO DONADO. DEVUELVE UN BOOLEANO, QUE SERÁ CIERTO EN CASO DE
QUE NO HAYA HABIDO PROBLEMAS. */
public boolean Inicia_búsqueda_nacional(Paciente donante) {
    List nacional = Busca_en_DF("CN","Nacional","");
    if (!nacional.isEmpty()) { //Si el CN está registrado en el DF
        AID AID_CN = new AID((String) nacional.get(0));
        //Comenzamos el Coord_Request a nivel de "CN"
        ACLMessage msg_req = Prepara_msg(ACLMessage.REQUEST,AID_CN,"CN",new
Organo(donante.getorgano()));
        MessageTemplate mt = MessageTemplate.MatchSender(AID_CN);
        addBehaviour(new Coord_Request(Hospital.this,msg_req,mt,donante,"CN"));
        return true;
    } else { //Si CN no está registrado
        gui.Resultado.append("\n" + getLocalName() + ": Coordinador Nacional
(CN) no registrado");
        System.out.println(getLocalName() + ": Coordinador Nacional (CN) no
registrado");
        return false;
    }
}

/*****
*/

/* COMPORTAMIENTO QUE INICIA EL CONTRACT NET CON OTROS AGENTES. SIRVE PARA
ENCONTRAR EL PACIENTE ADECUADO EN SEGÚN QUE NIVELES JERÁRQUICOS USANDO UN
PROTOCOLO DE TIPO FIPA-CONTRACT-NET */
public class Inicia_Contract_Net_misma_ciudad extends
jade.proto.FipaContractNetInitiatorBehaviour {

    Paciente donante;
    Paciente paciente_escogido_tipo_paciente;

```

```
Organo organo;
Eleccion paciente_escogido = new Eleccion();

// Constructores de la clase Inicia_Contract_Net
public Inicia_Contract_Net_misma_ciudad(Agent a, ACLMessage msg, List
responders, Organo org, Paciente don) {
    super(a,msg,responders);
    organo=org;
    donante=don;
}

public Inicia_Contract_Net_misma_ciudad(Agent a, ACLMessage msg, Organo
org, Paciente don) {
    this(a,msg,null,org,don);
    organo=org;
    donante=don;
}

// Handler para los mensajes de tipo Propose
public Vector handleProposeMessages(Vector proposals) {
    gui.Resultado.append("\n" + getLocalName() + ": Analizando propuestas de
Hospitales...");
    System.out.println(getLocalName() + ": Analizando propuestas de
Hospitales...");

    Vector respuestas = new Vector();
    /** Hemos de juntar todas las propuestas que nos llegan desde los
    * distintos hospitales. Cogemos las listas de pacientes que nos pasan
y
    * las juntamos en una. Para cada paciente tendremos el nombre del H y
el
    * id del paciente. Esta lista habrá que pasarla a A0 para que la
evalúe.
    * Después, H se encargará de aceptar la mejor propuesta y rechazar el
    * resto.
    */
    ACLMessage msg_actual = new ACLMessage(ACLMessage.NOT_UNDERSTOOD);
    Lista_objetos pacientes = new Lista_objetos();
    Lista_objetos pacientes_para_A0 = new Lista_objetos();
    Iterator it;
    Paciente paciente_actual = new Paciente();
    /** Al juntar todos los pacientes de los distintos hospitales para
hacer
    * la consulta al A0, no sabemos de qué hospital es cada paciente, y se
    * pueden repetir los identificadores, hay que modificar los
    * id's de los pacientes desde aquí, de tal manera que tengamos id's
    * secuenciales y en una lista guardemos, para cada id cambiado, el id
    * original y el hospital del que viene el paciente
    */
    int id=1;
    /**Lista que contiene los verdaderos id's dentro de cada hospital de
todos los pacientes propuestos
    List id_hosp = new ArrayList();
    for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
    //Hemos de juntar las listas recibidas de cada hospital
    msg_actual = (ACLMessage) proposals.get(i);
    pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
    it = pacientes.getAllpacientes();
    while (it.hasNext()) {
        //Sacamos un paciente de la lista de pacientes de un hospital
        paciente_actual = (Paciente) it.next();
        //Hemos de meter el id en una lista de pares id-hospital
        id_hosp.add(new
Eleccion(paciente_actual.gethospital(),paciente_actual.getid()));
        //Y ponerle un id correlativo
        paciente_actual.setid(new Long(id));
        //Lo metemos en la lista de pacientes total, la que se le pasará a
A0
```

```

        pacientes_para_A0.addpacientes(paciente_actual);
        //Incrementamos el número de pacientes en la lista que pasaremos a
A0
        pacientes_para_A0.setnpacientes(new
Long(pacientes_para_A0.getnpacientes().intValue()+1));
        id++;
    }
}
gui.Resultado.append("\n" + getLocalName() + ": " +
pacientes_para_A0.getnpacientes().intValue() + " pacientes propuestos");
System.out.println(getLocalName() + ": " +
pacientes_para_A0.getnpacientes().intValue() + " pacientes propuestos");

    /* Con la lista "pacientes_para_A0" hay que hacer la consulta a A0. */
    /* Hay que añadir el donante a la lista que se le envía a A0, y
incrementar el numPacientes */
    pacientes_para_A0.addpacientes(donante);
    pacientes_para_A0.setnpacientes(new
Long(pacientes_para_A0.getnpacientes().intValue()+1));

    /* Envío del mensaje a A0, para que evalúe los distintos pacientes
encontrados. */
    Resultado res = Consulta_A0(pacientes_para_A0);
    //En res tenemos la respuesta de A0
    if (res.getnobjetos().intValue()!=0) { //Si la ordenación ha sido
posible
        List pacientes_aptos = Busca_pacientes_aptos(res,grado_exigencia);
        if (!pacientes_aptos.isEmpty()) { //Si hay pacientes aptos
            gui.Resultado.append("\n" + getLocalName() + ": Se han encontrado "
+ pacientes_aptos.size() + " pacientes aptos en " + ciudad);
            System.out.println(getLocalName() + ": Se han encontrado " +
pacientes_aptos.size() + " pacientes aptos en " + ciudad);
            Grado aux = new Grado();
            for (int i=0;i<pacientes_aptos.size();i++) {
                aux = (Grado) pacientes_aptos.get(i);
                gui.Resultado.append("\n
                " + aux.getidObjeto() + " => " +
aux.getvalor());
                System.out.println("
                " + aux.getidObjeto() + " => " +
aux.getvalor());
            }
            /* Escoger entre los distintos pacientes aptos y salida por
pantalla. */
            int id_escogido = Escoge_mejor_paciente(pacientes_aptos);
            /** Hay que meter en la variable paciente_escogido_tipo_paciente
            * todos los datos del paciente escogido, para luego pasarlos a AH
            */
            paciente_escogido_tipo_paciente =
Obtiene_datos_paciente(pacientes_para_A0,id_escogido);
            paciente_escogido = (Eleccion) id_hosp.get(id_escogido-1); //-1 pq
la 1ª posición es la 0
            //Ponemos bien el id del paciente escogido dentro del hospital

            paciente_escogido_tipo_paciente.setid(paciente_escogido.getid_paciente());

            gui.Resultado.append("\n" + getLocalName() + ": Paciente escogido =>
" + paciente_escogido.getid_paciente() + " de " +
paciente_escogido.getid_hospital());
            System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente() + " de " +
paciente_escogido.getid_hospital());
        } else { //Si no se han encontrado pacientes aptos
            gui.Resultado.append("\n" + getLocalName() + ": No se han encontrado
pacientes
            aptos en " + ciudad);
            System.out.println(getLocalName() + ": No se han encontrado
pacientes
            aptos en " + ciudad);
            gui.Resultado.append("\n" + getLocalName() + ": No se han encontrado
pacientes
            aptos en " + ciudad);

```

```

        System.out.println(getLocalName() + ": Comenzando la búsqueda hacia
arriba (A)...");
        /** AUTONOMÍA: Hay que empezar la búsqueda con el coordinador de
autonomía */
        if (Inicia_búsqueda_autonomia(donante)) {
            //Ya se ha iniciado el Coord_Request a nivel de "A"
        } else { //Si no está registrado en el DF
            /** ZONA: Hay que empezar la búsqueda con el coordinador de Zona
*/
            gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (Z)...");
            System.out.println(getLocalName() + ": Comenzando la búsqueda
hacia arriba (Z)...");
            if (Inicia_búsqueda_zona(donante)) {
                //Ya se ha iniciado el Coord_Request a nivel de "Z"
            } else { //Si Z no está registrado en DF
                /** CN: Hay que empezar la búsqueda a nivel nacional */
                gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (CN)...");
                System.out.println(getLocalName() + ": Comenzando la búsqueda
hacia arriba (CN)...");
                if (Inicia_búsqueda_nacional(donante)) {
                    //Ya se ha iniciado el Request a nivel de "CN"
                } else {
                    /** FIN: No se ha encontrado ningún paciente apto en España */
                    gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningún paciente apto en el Pais");
                    System.out.println(getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
                    gui.Ok.setEnabled(true);
                    gui.Limpia_BD.setEnabled(true);
                }
            }
        }
    }
}
for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
//Hemos de aceptar la que viene del hospital escogido y rechazar el
resto
    msg_actual = (ACLMessage) proposals.get(i);
    pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
    it = pacientes.getAllpacientes();
    if (it.hasNext()) { //Sólo nos interesa un paciente, pq el hospital
será el mismo para todos
        //Sacamos un paciente de la lista de pacientes de un hospital
        paciente_actual = (Paciente) it.next();
        if
(paciente_actual.gethospital().equals(paciente_escogido.getid_hospital())){
            //Hay que aceptar la propuesta del hospital
            ACLMessage aceptar =
Prepara_msg(ACLMessage.ACCEPT_PROPOSAL,msg_actual.getSender(),
"Hospital",paciente_escogido);
            aceptar.setInReplyTo(msg_actual.getReplyWith());
            respuestas.addElement((ACLMessage)aceptar);

        } else { //Si el hospital del paciente no es el elegido
            //Hay que rechazar la propuesta del hospital
            ACLMessage rechazar = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
            rechazar.setSender(new AID(getLocalName()));
            rechazar.addReceiver(msg_actual.getSender());
            rechazar.setInReplyTo(msg_actual.getReplyWith());
            respuestas.addElement((ACLMessage)rechazar);
        }
    }
}
return (respuestas);
} else { //Si AO no ha podido ordenar

```

```

        gui.Resultado.append("\n" + getLocalName() + ": " +
JBSOntology.ORDENACION_IMPOSIBLE);
        System.out.println(getLocalName() + ": " +
JBSOntology.ORDENACION_IMPOSIBLE);
        gui.Resultado.append("\n" + getLocalName() + ": Comenzando la búsqueda
hacia arriba (A)...");
        System.out.println(getLocalName() + ": Comenzando la búsqueda hacia
arriba (A)...");
        /** Hay que rechazar todas las propuestas, para que finalicen el
Contract-Net */
        for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
//Hemos de rechazar todas las propuestas, pq no se ha podido ordenar
msg_actual = (ACLMessage) proposals.get(i);
ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
rechazar.setSender(new AID(getLocalName()));
rechazar.addReceiver(msg_actual.getSender());
rechazar.setInReplyTo(msg_actual.getReplyWith());
respuestas.addElement((ACLMessage)rechazar);
        }
        /** AUTONOMÍA: Hay que empezar la búsqueda con el coordinador de
autonomía */
        if (Inicia_búsqueda_autonomia(donante)) {
//Ya se ha puesto en marcha el Coord_Request a nivel de "A"
        } else { //Si no está registrado en el DF, o si ha habido otro
problema
            gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (Z)...");
            System.out.println(getLocalName() + ": Comenzando la búsqueda hacia
arriba (Z)...");
            /** ZONA: Hay que empezar la búsqueda con el coordinador de Zona */
            if (Inicia_búsqueda_zona(donante)) {
//Ya se ha iniciado el Coord_Request a nivel de "Z"
            } else { //Si Z no está registrado en DF
                /** CN: Hay que empezar la búsqueda a nivel nacional */
                gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (CN)...");
                System.out.println(getLocalName() + ": Comenzando la búsqueda
hacia arriba (CN)...");
                if (Inicia_búsqueda_nacional(donante)) {
//Ya se ha iniciado el Request a nivel de "CN"
                } else {
                    /** FIN: No se ha encontrado ningún paciente apto en España */
                    gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningun paciente apto en el Pais");
                    System.out.println(getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
                    gui.Ok.setEnabled(true);
                    gui.Limpia_BD.setEnabled(true);
                }
            }
        }
        return respuestas;
    }

    // Handler para otros tipos de mensajes
    public void handleOtherMessages(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido un mensaje de tipo " +
msg.getPerformative() + " de " + msg.getSender());
    }

    // Handler para los mensajes de tipo Final
    public Vector handleFinalMessages(Vector messages) {
        gui.Resultado.append("\n" + getLocalName() + ": Recibido un mensaje de
tipo Final");
        System.out.println(getLocalName() + ": Recibido un mensaje de tipo
Final");
    }

```

```

        gui.Resultado.append("\n" + getLocalName() + ": Contract-Net finalizado
satisfactoriamente");
        System.out.println(getLocalName() + ": Contract-Net finalizado
satisfactoriamente");
        /** AH: Aquí se debe contactar con AH para indicarle los datos de la
donación. */
        Lista_objetos lista_AH = new Lista_objetos(new Long(2));
        lista_AH.addpacientes(donante);
        lista_AH.addpacientes(paciente_escogido_tipo_paciente);
        AID_AID_AH = new AID((String) Busca_en_DF("AH", "Historico", "").get(0));
        ACLMessage msg_req =
Prepara_msg(ACLMessage.REQUEST, AID_AH, "AH", lista_AH);
        MessageTemplate mt = MessageTemplate.MatchSender(AID_AH);
        addBehaviour(new AH_Request(Hospital.this, msg_req, mt));
        gui.Ok.setEnabled(true);
        gui.Limpia_BD.setEnabled(true);
        /** GUI: Mostrar por la Interficie los datos del paciente */
        Mostrar_resultado(paciente_escogido_tipo_paciente);
        return(null);
    }

    // Devuelve el contenido del mensaje CFP
    public String createCfpContent(String cfpContent, String
receiver) {
        return(cfpContent);
    }
}

/*****
*/

/* COMPORTAMIENTO QUE CONTINUA EL CONTRACT NET EL AGENTE INICIADOR. SIRVE
PARA
ENCONTRAR EL PACIENTE ADECUADO EN SEGÚN QUE NIVELES JERÁRQUICOS USANDO UN
PROTOCOLO DE TIPO FIPA-CONTRACT-NET */
public class Responde_Contract_Net extends
jade.proto.FipaContractNetResponderBehaviour {

    Organo org = new Organo();

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

    public ACLMessage handleCfpMessage(ACLMessage cfp) {
        /** Recibimos un cfp del hospital donde se encuentra el
órgano, o de
        * algún otro coordinador intermedio. En el cfp se nos pasa el tipo de
        * órgano donado. Lo único que hay que hacer es buscar en la BD del
hospital
        * los pacientes que esperan ese tipo de órgano y meterlos en una lista
de
        * tipo Lista_objetos. Ésta es la que se envía la hospital de origen,
que
        * junta todas las propuestas, consulta al AO, y al final acepta o
rechaza
        * las distintas propuestas.
        */
        System.out.println(getLocalName() + ": Elaborando propuesta para " +
cfp.getSender().getName());

        //Extraemos el tipo de órgano donado del mensaje cfp
        org = (Organo) Extrae_contenido(cfp);
        /* Buscamos en la BD del H los pacientes que esperan ese tipo de órgano
y enviamos un mensaje al hospital origen con esa lista */
        /* 1.- Comunicación con BD */
        /* Se buscan los pacientes que optan al órgano donado en la BD del H */
        List pacientes = Busca_pacientes_en_BD(org.gettipoOrgano(), bd);

```

```
/* 2.- Rellenar el objeto Lista_Pacientes para enviarlo a AO */
Lista_Pacientes lista = new Lista_Pacientes();
lista.numero_pacientes=pacientes.size();
lista.pacientes=pacientes;

Lista_objetos envio = Crea_lista_objetos(lista);

/* Envío del mensaje al hospital origen, para que evalúe los distintos
pacientes encontrados. */
ACLMessage msg_propose =
Prepara_msg(ACLMessage.PROPOSE, cfp.getSender(), "Hospital", envio);
    return (msg_propose);
}

//Handler para mensajes ACCEPT
    public ACLMessage handleAcceptProposalMessage(ACLMessage msg) {
        System.out.println(getLocalName() + ": ACEPTADO");
        Eleccion paciente_escogido = new Eleccion();
        paciente_escogido = (Eleccion) Extrae_contenido(msg);
        System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente());
        /** BD: Modificar la BD para marcar de algún modo al paciente que ha
sido
        * escogido. Mejor no eliminarlo, pq podría ser que al final la
operación
        * no se llevase a cabo por algún motivo
        */
        System.out.println(getLocalName() + ": Actualizando la BD (" + bd +
""));

        if
(Actualiza_BD(paciente_escogido.getid_paciente().intValue(), bd, msg.getSender()
.getName()))
            System.out.println(getLocalName() + ": BD (" + bd + ") actualizada
correctamente");
        else
            System.out.println(getLocalName() + ": Problema al actualizar la BD ("
+ bd + "));

        ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM, msg.getSender(), "Hospital", paciente_escogido);

        addBehaviour(new Responde_Contract_Net(Hospital.this));

        return(msg_inform);
    }

    public void handleRejectProposalMessage(ACLMessage msg) {
        System.out.println(getLocalName() + ": RECHAZADO");

        addBehaviour(new Responde_Contract_Net(Hospital.this));
    }

    public void handleOtherMessages(ACLMessage msg) {
    }
}

/*****
*/

/* FUNCIÓN QUE PREPARA UN MENSAJE. RELLENA DE MANERA FÁCIL Y RÁPIDA LOS
CAMPOS
DE DESTINO, TIPO, Y CONTENIDO CON LOS PARÁMETROS PASADOS. LOS DE LENGUAJE
Y
ONTOLOGÍA LOS PONE POR DEFECTO, AL IGUAL QUE EL REMITENTE */
```

```

public ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0,
Object contenido) {
    ACLMessage msg = new ACLMessage(tipo_mensaje);
    msg.addReceiver(receiver);
    msg.setSender(new AID(getLocalName()));
    msg.setLanguage(SLCodec.NAME);
    msg.setOntology(JBSOntology.NAME);
    SL_Action a = new SL_Action();
    a.set_0(new AID(set0));
    a.set_1(contenido);
    List l = new ArrayList();
    l.add(a);
    try {
        fillContent(msg,l);
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return (msg);
}

/*****
*/

/* COMPORTAMIENTO QUE INICIA EL PROCESO DE BÚSQUEDA DE PACIENTES.
AL PRINCIPIO SE BUSCA EN EL PROPIO HOSPITAL, SI ESTO NO FUNCIONA SE VA
PASANDO A LOS NIVELES JERÁRQUICOS SUPERIORES. */
public class Inicia_proceso_busqueda extends OneShotBehaviour {

    Paciente donante;
    Paciente paciente_escogido_tipo_paciente;

    public Inicia_proceso_busqueda (Agent myAgent, Paciente don) {
        super(myAgent);
        donante=don;
    }

    public void action() {

        boolean solucionado=false; //Indica si se ha encontrado solución o no

        if (!solucionado) {
            /* Aquí es donde hemos de empezar la búsqueda, de momento suponemos
que
            empezamos por el propio hospital, por lo que nos tenemos que
comunicar
            con el Agente A0 para que nos diga si hay un posible paciente en
            nuestro hospital. */

            /* CUANDO SE RECIBE EL MENSAJE CON LAS CARACTERISTICAS DEL
            * ÓRGANO, HEMOS DE LLAMAR A UNA FUNCIÓN QUE CONSULTE LA BD DEL
            * HOSPITAL Y ENVIAR UN MENSAJE A A0, QUE CONTENDRÁ EL DONANTE Y LOS
            * PACIENTES QUE OPTAN AL ÓRGANO.
            */

            List pacientes = new ArrayList();

            /* 1.- Comunicación con BD */
            /* Se buscan los pacientes que optan al órgano donado en la BD del H
            */

            pacientes = Busca_pacientes_en_BD(donante.getorgano(),bd);

            /* 2.- Rellenar el objeto Lista_Pacientes para enviarlo a A0 */
            Lista_Pacientes lista = new Lista_Pacientes();
            lista.numero_pacientes=pacientes.size()+1; //+1 pq incluiremos el
donante
            lista.pacientes=pacientes;

            /* 3.- Hay que añadir el donante a la lista que se le envía a A0 */

```

```

        Lista_objetos envio=Crea_lista_objetos(lista);
        envio.addpacientes(donante);

        /* Envío del mensaje a A0, para que evalúe los distintos pacientes
encontrados. */
        Resultado res = Consulta_A0(envio);

        List pacientes_aptos = Busca_pacientes_aptos(res,grado_exigencia);
        if (!pacientes_aptos.isEmpty()) { //Si hay pacientes aptos
            solucionado=true;
            gui.Resultado.append("\n" + getLocalName() + ": Se han encontrado "
+ pacientes_aptos.size() + " pacientes aptos en el propio hospital");
            System.out.println(getLocalName() + ": Se han encontrado " +
pacientes_aptos.size() + " pacientes aptos en el propio hospital");
            Grado aux = new Grado();
            for (int i=0;i<pacientes_aptos.size();i++) {
                aux = (Grado) pacientes_aptos.get(i);
                gui.Resultado.append("\n          " + aux.getidObjeto() + " => " +
aux.getvalor());
                System.out.println("          " + aux.getidObjeto() + " => " +
aux.getvalor());
            }
            /** Si se han encontrado pacientes aptos en el mismo hospital,
            * hay que escoger entre ellos, para eso necesitamos una función
que
            * ultime los detalles del transplante, es decir, que escoja el
            * PACIENTE DEFINITIVO (OK) y que se encargue de comunicar con el
AGENTE
            * HISTÓRICO (OK) para informarle de los detalles del transplante.
            */

            /** 1- OK Escoger entre los distintos pacientes aptos y salida por
pantalla. */
            int paciente_escogido = Escoge_mejor_paciente(pacientes_aptos);

            /** Hay que meter en la variable paciente_escogido_tipo_paciente
            * todos los datos del paciente escogido, para luego pasarlos a AH
            */
            paciente_escogido_tipo_paciente =
Obtiene_datos_paciente(envio,paciente_escogido);

            gui.Resultado.append("\n" + getLocalName() + ": Paciente escogido =>
" + paciente_escogido);
            System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido);
            /** AH: Aquí se debe contactar con AH para indicarle los datos de la
            * donación.
            */
            Lista_objetos lista_AH = new Lista_objetos(new Long(2));
            lista_AH.addpacientes(donante);
            lista_AH.addpacientes(paciente_escogido_tipo_paciente);
            AID AID_AH = new AID((String)
Busca_en_DF("AH","Historico","").get(0));
            ACLMessage msg_req =
Prepara_msg(ACLMessage.REQUEST,AID_AH,"AH",lista_AH);
            MessageTemplate mt = MessageTemplate.MatchSender(AID_AH);
            addBehaviour(new AH_Request(Hospital.this,msg_req,mt));
            gui.Ok.setEnabled(true);
            gui.Limpia_BD.setEnabled(true);
            /** BD: Modificar la BD para marcar de algún modo al paciente que ha
            sido
            * escogido. Mejor no eliminarlo, pq podría ser que al final la
operación
            * no se llevase a cabo por algún motivo
            */
            gui.Resultado.append("\n" + getLocalName() + ": Actualizando la BD
(" + bd + ")");

```

```

        System.out.println(getLocalName() + ": Actualizando la BD (" + bd +
        ")");
        if (Actualiza_BD(paciente_escogido,bd,getLocalName()))
            System.out.println(getLocalName() + ": BD (" + bd + ") actualizada
correctamente");
        else
            System.out.println(getLocalName() + ": Problema al actualizar la
BD (" + bd + ")");
        /*      4- En caso de no solución: */
        /*      4.1- Buscar entre los hospitales de la misma ciudad, por
Contract-Net. */
        /*      4.2- Seguir buscando jerárquicamente hacia arriba. */
        /** GUI: Mostrar por la Interficie los datos del paciente */
        Mostrar_resultado(paciente_escogido_tipo_paciente);
    } else {
        gui.Resultado.append("\n" + getLocalName() + ": No se han
encontrado pacientes aptos en el propio hospital");
        System.out.println(getLocalName() + ": No se han encontrado
pacientes aptos en el propio hospital");
    }
}

    if (!solucionado) { //Si no se ha encontrado ningún paciente apto en el
propio hospital

        /* Hay que empezar el Contract Net con el resto de Hospitales de la
ciudad, si los hay */
        //Se buscan en el df todos los "Hospital" que hay en la "ciudad"
excepto el propio (getLocalName())
        List hospitales = Busca_en_DF(ciudad,"Hospital",getLocalName());
        if (!hospitales.isEmpty()) { //Si hay más hospitales en la ciudad, a
parte del propio
            List destinos = new ArrayList();
            gui.Resultado.append("\n" + getLocalName() + ": Hospitales
encontrados en " + ciudad + " => " + hospitales.size() + ": ");
            System.out.println(getLocalName() + ": Hospitales encontrados en " +
ciudad + " => " + hospitales.size() + ": ");

            //Añadimos los destinatarios, los hospitales hallados en la misma
ciudad
            for (int i=0;i<hospitales.size();i++) {
                String aux = (String) hospitales.get(i);
                gui.Resultado.append("\n      " + hospitales.get(i));
                System.out.println("      " + hospitales.get(i));
                //Pasamos los nombres (String) de una lista a direcciones (AID) en
otra
                destinos.add(new AID((String) hospitales.get(i)));
            }
            // Crear el mensaje cfp, y añadir el comportamiento ContractNet
ACLMessage msg_cfp = Prepara_msg(ACLMessage.CFP,null,"Hospital",new
Organo(donante.getorgano()));
            addBehaviour(new Inicia_Contract_Net_misma_ciudad(Hospital.this,
msg_cfp,destinos,new Organo(donante.getorgano()),donante));
            gui.Resultado.append("\n" + getLocalName() + ": Empieza el Contract
Net dentro de la misma ciudad (" + ciudad + ")");
            System.out.println(getLocalName() + ": Empieza el Contract Net
dentro de la misma ciudad (" + ciudad + ")");

        } else { //Si no se han encontrado más hospitales en la misma ciudad
            gui.Resultado.append("\n" + getLocalName() + ": No hay mas
hospitales en " + ciudad);
            System.out.println(getLocalName() + ": No hay mas hospitales en " +
ciudad);
            gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
busqueda hacia arriba (A)...");
            System.out.println(getLocalName() + ": Comenzando la busqueda hacia
arriba (A)...");

```

```

    /** AUTONOMÍA: Hay que empezar la búsqueda con el coordinador de
autonomía */
    if (Inicia_búsqueda_autonomia(donante)) {
        //Ya se ha iniciado el Coord_Request a nivel de "A"
    } else { //Si no está registrado en el DF
        /** ZONA: Hay que empezar la búsqueda con el coordinador de Zona
*/
        gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (Z)...");
        System.out.println(getLocalName() + ": Comenzando la búsqueda
hacia arriba (Z)...");
        if (Inicia_búsqueda_zona(donante)){
            //Ya se ha iniciado el Coord_Request a nivel de "Z"
        } else { //Si Z no está registrado en DF
            /** CN: Hay que empezar la búsqueda a nivel nacional */
            gui.Resultado.append("\n" + getLocalName() + ": Comenzando la
búsqueda hacia arriba (CN)...");
            System.out.println(getLocalName() + ": Comenzando la búsqueda
hacia arriba (CN)...");
            if (Inicia_búsqueda_nacional(donante)) {
                //Ya se ha iniciado el Request a nivel de "CN"
            } else {
                /** FIN: No se ha encontrado ningún paciente apto en España */
                gui.Resultado.append("\n" + getLocalName() + ": No se ha
encontrado ningun paciente apto en el Pais");
                System.out.println(getLocalName() + ": No se ha encontrado
ningun paciente apto en el Pais");
                gui.Ok.setEnabled(true);
                gui.Limpia_BD.setEnabled(true);
            }
        }
    }
}

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

/* SETUP DEL AGENTE */
protected void setup () {
    //Inicializaciones diferentes de cada H según su nombre

    if (getLocalName().equals("H")) {
        bd="BD1";
        ciudad="Tarragona";
    }
    if (getLocalName().equals("H2")) {
        bd="BD2";
        ciudad="Tarragona";
    }
    if (getLocalName().equals("H3")) {
        bd="BD3";
        ciudad="Tarragona";
    }
    if (getLocalName().equals("H4")) {
        bd="BD4";
        ciudad="Barcelona";
    }
    if (getLocalName().equals("H5")) {
        bd="BD5";
        ciudad="Palau";
    }
}

```

```
    }
    if (getLocalName().equals("H6")) {
        bd="BD6";
        ciudad="Mallorca";
    }
    if (getLocalName().equals("H7")) {
        bd="BD7";
        ciudad="Menorca";
    }
    if (getLocalName().equals("H8")) {
        bd="BD8";
        ciudad="Ibiza";
    }
    if (getLocalName().equals("H9")) {
        bd="BD9";
        ciudad="Zaragoza";
    }
    if (getLocalName().equals("H10")) {
        bd="BD10";
        ciudad="Teruel";
    }
    if (getLocalName().equals("H11")) {
        bd="BD11";
        ciudad="Huesca";
    }
    }

    provincia = ciudad;

    if ((getLocalName().equals("H")) || (getLocalName().equals("H"))) {
        gui_datos = new Entrada_Datos(Hospital.this);
        gui_datos.setVisible(true);
        gui_datos.ciudad.setText(ciudad);
        gui_datos.provincia.setText(provincia);
        gui_datos.bd.setText(bd);
        gui_datos.ok.setEnabled(true);
    } else {
        Registro_en_el_df(ciudad);
        addBehaviour(new Responde_Contract_Net(Hospital.this));
    }
    }

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

    /* TAKEDOWN DEL AGENTE */
    protected void takeDown(){
        Desregistro_del_df();
    }
}
```

8.2. Autonomia.java

```

/*****
**
Autonomia

PFC Jaime Bocio Sanz
2-5-2002

Agente que representa un coordinador de autonomía. Recibirá peticiones de un
Hospital de su autonomía y hará de intermediario para buscar un posible
paciente
en los hospitales de su autonomía. Contacta directamente con los hospitales
para
transmitirles los datos del órgano donado y para devolver la lista con los
pacientes ordenados al Hospital de origen

*****/

package pfc_ei;

import jade.lang.acl.*;
import jade.core.behaviours.*;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.*;
import jade.core.*;
import jade.lang.sl.sl.*;
import jade.onto.sl.sl.*;
import jade.proto.FipaRequestResponderBehaviour;

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

import jade.core.Agent;

//Paquete con la ontología definida
import pfc_ei.ontology.*;

/*****
*/

public class Autonomia extends Agent {

    String autonomia;    /** @todo ; ! Indica la autonomia de la que es
coordinador,
                                tiene que poder ser parametrizable en el registro.
*/

/*****
*/

    public Autonomia() {
        super();
    }

/*****
*/

/* FUNCIÓN QUE REGISTRA A UN AGENTE EN EL DF. SE LE PASA COMO PARÁMETRO LA
AUTONOMÍA DEL COORDINADOR, PARA FACILITAR LAS BÚSQUEDAS POSTERIORES EN EL
DF. */
    public void Registro_en_el_df(String auton) {
        ServiceDescription sd1=new ServiceDescription();
        //Indicamos los parametros del Agente

```

```

        sd1.setType("Autonomia");
        sd1.setName(getLocalName());
        sd1.setOwnership(auton);
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        //Nos añadimos como un nuevo servicio
        dfd.addServices(sd1);
        //Indicamos nuestro nombre
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);
    //Registramos el lenguaje
    registerLanguage(SLCodec.NAME, new SLCodec());
    //Registro de la ontología
    registerOntology(JBSOntology.NAME, JBSOntology.instance());
    System.out.println("Coordinador Autonomico (" + getLocalName() +
" de " + sd1.getOwnership() + ") on-line");

        try{
            //Nos registramos al DF
            DFServiceCommunicator.register(this,dfd);
        }catch (FIPAException e) {
            System.err.println(getLocalName()+": Error en el registro
al DF. Error: "+e.getMessage());
            doDelete();
        }
    }

/*****
*/

/* FUNCIÓN QUE DESREGISTRA A UN AGENTE DEL DF. SE USA EN EL "takeDown" DEL
AGENTE. */
public void Desregistro_del_df() {
    //Obtenemos una instancia del DF
    DFAgentDescription dfd=new DFAgentDescription();
    AID aid=new AID();
    aid.setName(getLocalName());
    dfd.setName(aid);

    try{
        //Nos desregistramos
        DFServiceCommunicator.deregister(this,dfd);
        System.out.println("Agente " + getLocalName() + " eliminado
del DF");
    }catch (FIPAException e) {
        System.out.println(getLocalName()+": Error en el
desregistro del DF. Error: "+e.getMessage());
    }
}

/*****
*/

/* FUNCIÓN QUE BUSCA EN EL DF TODOS LOS NOMBRES DE LOS AGENTES QUE CUMPLEN
UN CIERTO SERVICIO. EN CONCRETO DEVUELVE UNA LISTA CON TODOS LOS AGENTES
DEL TIPO tipo CUYO PROPIETARIO ES propietario, EXCEPTO LOS AGENTES CUYO
NOMBRE ES excepto. */
public List Busca_en_DF(String propietario, String tipo, String excepto) {
    List resultado = new ArrayList();
    DFAgentDescription dfad=new DFAgentDescription();
    ServiceDescription sd=new ServiceDescription();
    List result;

    sd.setOwnership(propietario);
    sd.setType(tipo);
    dfad.addServices(sd); // Añadimos el servicio al
descriptor de agente DF, para la búsqueda

```

```
        try { // Procedemos a buscar en df lo que queremos (dfad)
            result = jade.domain.DFServiceCommunicator.search(super, dfad);
            if (result.isEmpty())
                return resultado;

            for (int i=0;i<result.size();i++) {
                dfad = (DFAgentDescription) result.get(i);
                if (!dfad.getName().getName().equalsIgnoreCase(excepto))
                    resultado.add(dfad.getName().getName()); //Con
esto obtenemos el nombre del agente
            }
        } catch (Exception exc) {
            System.out.println( exc.toString() );
        }
        return resultado;
    }
}

/*****
*/

/* FUNCIÓN QUE PREPARA UN MENSAJE. RELLENA DE MANERA FÁCIL Y RÁPIDA LOS
CAMPOS
DE DESTINO, TIPO, Y CONTENIDO CON LOS PARÁMETROS PASADOS. LOS DE LENGUAJE
Y
ONTOLOGÍA LOS PONE POR DEFECTO, AL IGUAL QUE EL REMITENTE */
public ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0,
Object contenido) {
    ACLMessage msg = new ACLMessage(tipo_mensaje);
    msg.addReceiver(receiver);
    msg.setSender(new AID(getLocalName()));
    msg.setLanguage(SLCodec.NAME);
    msg.setOntology(JBSontology.NAME);
    SL_Action a = new SL_Action();
    a.set_0(new AID(set0));
    a.set_1(contenido);
    List l = new ArrayList();
    l.add(a);
    try {
        fillContent(msg,l);
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return (msg);
}

/*****
*/

/* FUNCIÓN QUE EXTRAER EL CONTENIDO DE UN MENSAJE. FACILITA LA TAREA. SE LE
PASA
EL MENSAJE DEL CUAL QUEREMOS EXTRAER EL CONTENIDO, Y SE DEVUELVE EL OBJETO
DE
LA ONTOLOGÍA QUE CONTIENE. */
public Object Extrae_contenido(ACLMessage msg) {
    List l = new ArrayList();
    SL_Action a = new SL_Action();
    Object resultado = new Object();
    try {
        l= extractContent(msg);
        a = (SL_Action) l.get(0);
        resultado = (Object) a.get_1();
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return resultado;
}
}
```

```
/* *****
*/

/* FUNCIÓN QUE REALIZA UNA CONSULTA EN LA BASE DE DATOS. BUSCA EN LA TABLA
   tabla DE LA BD bd1, LA COLUMNA col_return, DE LOS REGISTROS QUE CUMPLAN
   QUE col = val, EXCEPTO LOS QUE SON excepto*/
public List Busca_en_BD(String bd1, String tabla, String col_return, String
col, String val, String excepto) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;
    List lista_a_retornar = new ArrayList();
    String res = "";
    String aux = "";

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1);
        stmt = con.createStatement();
        // Búsqueda de los pacientes del órgano "org"
        if (stmt.execute("Select * from " + tabla + " where " + col + "='" +
val + "'")) {
            rs = stmt.getResultSet();
        }
        int i=0;
        while (rs.next()) {
            aux = rs.getString(col_return);
            if (!aux.equals(res)) {
                res=aux;
                if (!res.equals(excepto)) {
                    lista_a_retornar.add(res);
                    i++;
                }
            }
        }
        System.out.println(bd1 + " (" + getLocalName() + "): Registros
encontrados de " + col + "=" + val + " => " + i);
        con.close();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
    }
    return(lista_a_retornar);
}

/* *****
*/

/* COMPORTAMIENTO QUE RESPONDE A UN REQUEST PROCEDENTE DE UN HOSPITAL. */
public class H_Request extends FipaRequestResponderBehaviour.ActionHandler
implements FipaRequestResponderBehaviour.Factory {

    Organo organo = new Organo();

    public H_Request(Agent a, ACLMessage msg){
        super(a,msg);
    }

    public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
        return new H_Request(myAgent, msg);
    }

    public void action(){
        ACLMessage msg_req = getRequest();
        organo = (Organo) Extrae_contenido(msg_req);
        // System.out.println(msg_req.toString());

        /** 1- Encontrar los hospitales con los que hay que hacer el Contract-
Net */

```

```

        //Buscamos la ciudad a la que pertenece el órgano donado
        String ciudad = "";
        DFAgentDescription dfad=new DFAgentDescription();
        ServiceDescription sd=new ServiceDescription();
        List result = new ArrayList();

        sd.setName(msg_req.getSender().getName());
        sd.setType("Hospital");
        dfad.addServices(sd); // Añadimos el servicio al
        descriptor de agente DF, para la búsqueda

        try { // Procedemos a buscar en df lo que queremos (dfad)
            result =
jade.domain.DFServiceCommunicator.search(Autonomia.this,dfad);
        } catch (Exception exc) {
            System.out.println( exc.toString() );
        }
        dfad = (DFAgentDescription) result.get(0);
        Iterator it = dfad.getAllServices();
        sd = (ServiceDescription) it.next();
        ciudad = sd.getOwnership();

        List hospitales = new ArrayList();
        List destinos = new ArrayList();
        List ciudades = Busca_en_BD("BD-
Ciudades","Ciudades","Ciudad","Autonomia",autonomia,ciudad);
        System.out.println(getLocalName() + ": Ciudades de " + autonomia + ":");
        for (int i=0;i<ciudades.size();i++) {
            System.out.println(" " + ciudades.get(i));
            hospitales.addAll(Busca_en_DF((String)
ciudades.get(i),"Hospital",""));
        }
        if (hospitales.isEmpty()) { //No hay más Hospitales en la Autonomía
            System.out.println(getLocalName() + ": No hay mas hospitales en " +
autonomia);
            /** Hay que enviarle respuesta a H, indicando el problema */
            ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
            msg_refuse.setProtocol("fipa-request");
            msg_refuse.setConversationId(msg_req.getConversationId());
            msg_refuse.setInReplyTo(msg_req.getReplyWith());
            send(msg_refuse);
        } else { //Se han encontrado Hospitales en la autonomía, a parte del
origen
            System.out.println(getLocalName() + ": Hospitales de " + autonomia +
":");
            for (int i=0;i<hospitales.size();i++) {
                System.out.println(" " + hospitales.get(i));
                destinos.add(new AID((String) hospitales.get(i)));
            }

            /** 2- Comenzar el Contract-Net */
            ACLMessage msg_cfp =
Prepara_msg(ACLMessage.CFP,null,"Hospital",organo);
            addBehaviour(new
Inicia_Contract_Net_misma_autonomia(Autonomia.this,msg_cfp,destinos,organo,msg
_req,false));
        }
    }

    public boolean done() {
        return true;
    }

    public void reset() {
    }
}

```

```

public class miFipaRequestResponderBehaviour extends
FipaRequestResponderBehaviour {
    public miFipaRequestResponderBehaviour(Agent a) {
        super(a);
    }
    protected String getActionName(ACLMessage msg) throws
NotUnderstoodException, RefuseException {
        return JBSontology.ORGANO;
    }
}

/*****
*/

/* COMPORTAMIENTO QUE INICIA EL CONTRACT NET CON OTROS AGENTES. SIRVE PARA
ENCONTRAR EL PACIENTE ADECUADO EN SEGÚN QUE NIVELES JERÁRQUICOS USANDO UN
PROTOCOLO DE TIPO FIPA-CONTRACT-NET */
public class Inicia_Contract_Net_misma_autonomia extends
jade.proto.FipaContractNetInitiatorBehaviour {

    Organo organo;
    ACLMessage msg_origen;
    boolean busq_sup;
    ACLMessage msg_eleccion;

    // Constructores de la clase Inicia_Contract_Net
    public Inicia_Contract_Net_misma_autonomia(Agent a, ACLMessage msg, List
responders, Organo org, ACLMessage msg_del_origen, boolean busqueda_superior)
{
        super(a,msg,responders);
        organo=org;
        msg_origen = msg_del_origen;
        busq_sup = busqueda_superior;
        System.out.println(getLocalName() + ": Iniciando Contract-Net con
hospitales");
    }

    public Inicia_Contract_Net_misma_autonomia(Agent a, ACLMessage msg, Organo
org, ACLMessage msg_del_origen, boolean busqueda_superior) {
        this(a,msg,null,org,msg_del_origen, busqueda_superior);
        organo=org;
        msg_origen = msg_del_origen;
        busq_sup = busqueda_superior;
    }

    // Handler para los mensajes de tipo Propose
    public Vector handleProposeMessages(Vector proposals) {
        System.out.println(getLocalName() + ": Analizando propuestas de
Hospitales...");

        Vector respuestas = new Vector();
        /** Hemos de juntar todas las propuestas que nos llegan desde los
        * distintos hospitales. Cogemos las listas de pacientes que nos pasan
y
        * las juntamos en una. Para cada paciente tendremos el nombre del H y
el
        * id del paciente. Esta lista habrá que pasarla a H para que la
evalúe.
        * Después cuando H devuelva la eleccion, A se encargará de aceptar
        * la mejor propuesta y rechazar el resto.
        */
        ACLMessage msg_actual = new ACLMessage(ACLMessage.NOT_UNDERSTOOD);
        Lista_objetos pacientes = new Lista_objetos();
        Lista_objetos pacientes_para_H = new Lista_objetos();
        Iterator it;
        Paciente paciente_actual = new Paciente();
        for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
            //Hemos de juntar las listas recibidas de cada hospital

```

```

        msg_actual = (ACLMessage) proposals.get(i);
        pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
        it = pacientes.getAllpacientes();
        while (it.hasNext()) {
            //Sacamos un paciente de la lista de pacientes de un hospital
            paciente_actual = (Paciente) it.next();
            //Lo metemos en la lista de pacientes total, la que se le pasará a
A0
            pacientes_para_H.addpacientes(paciente_actual);
            //Incrementamos el número de pacientes en la lista que pasaremos a
A0
            pacientes_para_H.setnpacientes(new
Long(pacientes_para_H.getnpacientes().intValue()+1));
        }
        System.out.println(getLocalName() + ": " +
pacientes_para_H.getnpacientes().intValue() + " pacientes propuestos");

        if (!busq_sup) { //Búsqueda a nivel de A
            //Usamos el msg_origen para mezclar el Contract-Net con el Request de
H
            ACLMessage msg_agree =
Prepara_msg(ACLMessage.AGREE,msg_origen.getSender(),"Hospital",pacientes_para_
H);
            msg_agree.setProtocol(msg_origen.getProtocol());
            msg_agree.setConversationId(msg_origen.getConversationId());
            msg_agree.setInReplyTo(msg_origen.getReplyWith());
            send(msg_agree);

            System.out.println(getLocalName() + ": Lista de pacientes enviada a "
+ msg_origen.getSender().getName());

            ACLMessage msg_eleccion =
blockingReceive(MessageTemplate.MatchConversationId(msg_origen.getConversation
Id()));
            Eleccion paciente_escogido = (Eleccion)
Extrae_contenido(msg_eleccion);

            //Hay que comprobar si la eleccion es valida, es decir, no se trata de
un id=0

            if (paciente_escogido.getid_paciente().intValue()!=0) { //Si no es
una elección nula
                for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
                    //Hemos de aceptar la que viene del hospital escogido y rechazar
el resto
                    msg_actual = (ACLMessage) proposals.get(i);
                    pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
                    it = pacientes.getAllpacientes();
                    if (it.hasNext()) { //Sólo nos interesa un paciente, pq el
hospital será el mismo para todos
                        //Sacamos un paciente de la lista de pacientes de un hospital
                        paciente_actual = (Paciente) it.next();
                        if
(paciente_actual.gethospital().equals(paciente_escogido.getid_hospital())){
                            //Hay que aceptar la propuesta del hospital
                            ACLMessage aceptar = Prepara_msg(ACLMessage.ACCEPT_PROPOSAL,
msg_actual.getSender(),"Hospital",paciente_escogido);
                            aceptar.setInReplyTo(msg_actual.getReplyWith());
                            respuestas.addElement((ACLMessage)aceptar);

                        } else { //Si el hospital del paciente no es el elegido
                            //Hay que rechazar la propuesta del hospital
                            ACLMessage rechazar = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
                            rechazar.setSender(new AID(getLocalName()));
                            rechazar.addReceiver(msg_actual.getSender());

```

```

        rechazar.setInReplyTo(msg_actual.getReplyWith());
        respuestas.addElement((ACLMessage)rechazar);
    }
}
} else { //Si se trata de una elección nula
//Hemos de rechazar todas las propuestas
for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
//Hemos de rechazar todas las propuestas
msg_actual = (ACLMessage) proposals.get(i);
ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
rechazar.setSender(new AID(getLocalName()));
rechazar.addReceiver(msg_actual.getSender());
rechazar.setInReplyTo(msg_actual.getReplyWith());
respuestas.addElement((ACLMessage)rechazar);
}
System.out.println(getLocalName() + ": El hospital de origen no ha
encontrado pacientes aptos");
}
return (respuestas);
} else { //Búsqueda a nivel superior, por tanto hemos de pasar las
propuestas hacia arriba
//Usamos el msg_origen para mezclar los dos Contract-Net
ACLMessage msg_propose =
Prepara_msg(ACLMessage.PROPOSE,msg_origen.getSender(),"Z",pacientes_para_H);
msg_propose.setProtocol(msg_origen.getProtocol());
msg_propose.setConversationId(msg_origen.getConversationId());
msg_propose.setInReplyTo(msg_origen.getReplyWith());
msg_propose.setSender(getAID());
send(msg_propose);

System.out.println(getLocalName() + ": Lista de pacientes enviada a "
+ msg_origen.getSender().getName());

msg_eleccion =
blockingReceive(MessageTemplate.MatchConversationId(msg_origen.getConversation
Id()));
/* De alguna manera aquí se hace lo mismo que con el Contract Net, se
reciben
las respuestas a la propuesta, ACCEPT o REJECT, y se pasan hacia los
hospitales
que realmente han hecho las propuestas */
Eleccion paciente_escogido = (Eleccion)
Extrae_contenido(msg_eleccion);

//Hay que comprobar si la eleccion es valida, es decir, no se trata de
un id=0
if (paciente_escogido.getid_paciente().intValue()!=0) { //Si no es
una elección nula
System.out.println(getLocalName() + ": " +
pacientes_para_H.getnpacientes().intValue() + " pacientes propuestos");
for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
//Hemos de aceptar la que viene del hospital escogido y rechazar
el resto
msg_actual = (ACLMessage) proposals.get(i);
pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
it = pacientes.getAllpacientes();
if (it.hasNext()) { //Sólo nos interesa un paciente, pq el
hospital será el mismo para todos
//Sacamos un paciente de la lista de pacientes de un hospital
paciente_actual = (Paciente) it.next();
if
(paciente_actual.gethospital().equals(paciente_escogido.getid_hospital())){
//Hay que aceptar la propuesta del hospital
ACLMessage aceptar = Prepara_msg(ACLMessage.ACCEPT_PROPOSAL,
msg_actual.getSender(),"Hospital",paciente_escogido);
aceptar.setInReplyTo(msg_actual.getReplyWith());

```

```

        respuestas.addElement((ACLMessage)aceptar);

    } else { //Si el hospital del paciente no es el elegido
        //Hay que rechazar la propuesta del hospital
        ACLMessage rechazar = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
        rechazar.setSender(new AID(getLocalName()));
        rechazar.addReceiver(msg_actual.getSender());
        rechazar.setInReplyTo(msg_actual.getReplyWith());
        respuestas.addElement((ACLMessage)rechazar);
    }
}
} else { //Si se trata de una elección nula
    for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
        //Hemos de rechazar todas las propuestas
        msg_actual = (ACLMessage) proposals.get(i);
        ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
        rechazar.setSender(new AID(getLocalName()));
        rechazar.addReceiver(msg_actual.getSender());
        rechazar.setInReplyTo(msg_actual.getReplyWith());
        respuestas.addElement((ACLMessage)rechazar);
    }
    System.out.println(getLocalName() + ": El hospital de origen no ha
encontrado pacientes aptos");
}
return respuestas;
}

// Handler para otros tipos de mensajes
public void handleOtherMessages(ACLMessage msg) {
    System.out.println(getLocalName() + ": Recibido un mensaje de tipo " +
msg.getPerformative() + " de " + msg.getSender());
}

// Handler para los mensajes de tipo Final
public Vector handleFinalMessages(Vector messages) {
    ACLMessage aux = (ACLMessage) messages.get(0);
    System.out.println(getLocalName() + ": Recibido un mensaje de tipo Final
de " + aux.getSender());
    System.out.println(getLocalName() + ": Contract-Net finalizado
satisfactoriamente");

    Eleccion paciente_escogido = (Eleccion) Extrae_contenido(aux);

    if (!busq_sup) { //Si se trata de una búsqueda a nivel de A
        ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM,msg_origen.getSender(),"Hospital",paciente_escog
ido);
        msg_inform.setConversationId(msg_origen.getConversationId());
        msg_inform.setInReplyTo(msg_origen.getReplyWith());
        msg_inform.setProtocol("fipa-request");
        send(msg_inform);
    } else {
        System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente());
        ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM,msg_eleccion.getSender(),"Hospital",paciente_esc
ogido);
        msg_inform.setProtocol(msg_eleccion.getProtocol());
        msg_inform.setConversationId(msg_eleccion.getConversationId());
        msg_inform.setInReplyTo(msg_eleccion.getReplyWith());
        msg_inform.setSender(getAID());
        send(msg_inform);
        // System.out.println(msg_inform.toString());
    }
    return(null);
}

```

```

    }

    // Devuelve el contenido del mensaje CFP
    public String createCfpContent(String cfpContent, String
receiver) {
        return(cfpContent);
    }
}

/*****
*/

/* COMPORTAMIENTO QUE CONTINUA EL CONTRACT NET EL AGENTE INICIADOR. SIRVE
PARA
ENCONTRAR EL PACIENTE ADECUADO EN SEGÚN QUE NIVELES JERÁRQUICOS USANDO UN
PROTOCOLO DE TIPO FIPA-CONTRACT-NET */
public class Responde_Contract_Net extends
jade.proto.FipaContractNetResponderBehaviour {

    Organo org = new Organo();

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

    public ACLMessage handleCfpMessage(ACLMessage cfp) {
        /** Recibimos un cfp del coordinador superior de zona. Se
nos indida el
        * tipo de órgano que recibimos. A tiene que realizar un Contract-Net
con
        * los hospitales de su autonomía, y pasar las propuestas hacia
arriba,
        * en forma de Lista_objetos con los pacientes propuestos
        */
        System.out.println(getLocalName() + ": Elaborando propuesta para " +
cfp.getSender().getName());

        //Extraemos el tipo de órgano donado del mensaje cfp
        org = (Organo) Extrae_contenido(cfp);

        List hospitales = new ArrayList();
        List destinos = new ArrayList();
        List ciudades = Busca_en_BD("BD-
Ciudades", "Ciudades", "Ciudad", "Autonomia", autonomia, "");
        System.out.println(getLocalName() + ": Ciudades de " + autonomía + ":");
        for (int i=0;i<ciudades.size();i++) {
            System.out.println(" " + ciudades.get(i));
            hospitales.addAll(Busca_en_DF((String)
ciudades.get(i), "Hospital", ""));
        }
        if (hospitales.isEmpty()) { //No hay más Hospitales en la Autonomía
            System.out.println(getLocalName() + ": No hay mas hospitales en " +
autonomía);
            /** Hay que enviarle respuesta a Z, indicando el problema */
            ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE, cfp.getSender(), "Z", new No_hospitales());
            msg_refuse.setProtocol("FIPA-Contract-Net");
            msg_refuse.setConversationId(cfp.getConversationId());
            msg_refuse.setInReplyTo(cfp.getReplyWith());
            send(msg_refuse);
        } else { //Se han encontrado Hospitales en la autonomía, a parte del
origen
            System.out.println(getLocalName() + ": Hospitales de " + autonomía +
":");
            for (int i=0;i<hospitales.size();i++) {
                System.out.println(" " + hospitales.get(i));
                destinos.add(new AID((String) hospitales.get(i)));
            }
        }
    }
}

```

```

    }

    /** 2- Comenzar el Contract-Net */
    ACLMessage msg_cfp = Prepara_msg(ACLMessage.CFP,null,"Hospital",org);
    addBehaviour(new
Inicia_Contract_Net_misma_autonomia(Autonomia.this,msg_cfp,destinos,org,cfp,tru
ue));
    }
    /** Aquí hay que combinar de algún modo los 2 Contract-Nets que
tenemos,
    * uno con los hospitales y otro con el Z. En principio este ya no hará
nada
    * más, y el resto de la comunicación la hace el que contacta con Z
    */
    return null;
    }

    //Handler para mensajes ACCEPT
    public ACLMessage handleAcceptProposalMessage(ACLMessage msg) {
        System.out.println(getLocalName() + ": ACEPTADO");
        Eleccion paciente_escogido = new Eleccion();
        paciente_escogido = (Eleccion) Extrae_contenido(msg);
        System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente());

        ACLMessage msg_inform = Prepara_msg(ACLMessage.ACCEPT_PROPOSAL,new
AID(paciente_escogido.getid_hospital()),
                                "Hospital",paciente_escogido);

        send(msg_inform);
    //    System.out.println(msg_inform.toString());
    return null;
    }

    public void handleRejectProposalMessage(ACLMessage msg) {
        System.out.println(getLocalName() + ": RECHAZADO");
    }

    public void handleOtherMessages(ACLMessage msg) {
    }
}

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

/* SETUP DEL AGENTE */

protected void setup () {
    if (getLocalName().equals("A1"))
        autonomia="Catalunya";
    if (getLocalName().equals("A2"))
        autonomía="Aragon";
    if (getLocalName().equals("A3"))
        autonomía="Valencia";
    if (getLocalName().equals("A4"))
        autonomía="Balears";

    Registro_en_el_df(autonomia);

    FipaRequestResponderBehaviour requester = new
miFipaRequestResponderBehaviour(Autonomia.this);
    requester.registerFactory(JBSOntology.ORGANO,new
H_Request(Autonomia.this,null));
    addBehaviour(requester);
}

```

```
        addBehaviour(new Responde_Contract_Net(Autonomia.this));
    }

    /**
     *
     */
    /** TAKEDOWN DEL AGENTE */

    protected void takeDown(){
        Desregistro_del_df();
    }
}
```

8.3. Zona.java

```

/*****
**
Zona

PFC Jaime Bocio Sanz
2-5-2002

Agente que representa un coordinador de Zona. Recibirá peticiones de un
Hospital de su Zona y hará de intermediario para buscar un posible paciente
en los hospitales de su Zona. Contacta con A para que éste contacte con los
hospitales para trasmitirles los datos del órgano donado y para devolver la
lista con los pacientes ordenados al Hospital de origen

*****/

/

package pfc_ei;

import jade.lang.acl.*;
import jade.core.behaviours.*;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.*;
import jade.core.*;
import jade.lang.sl.sl.*;
import jade.onto.sl.sl.*;
import jade.proto.FipaRequestResponderBehaviour;

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

import jade.core.Agent;

//Paquete con la ontología definida
import pfc_ei.ontology.*;

/*****
*/

public class Zona extends Agent {

    String zona;    /** @todo ¡ Indica la zona de la que es coordinador,
                     tiene que poder ser parametrizable en el registro.
    */

    /*****
    */

    public Zona() {
        super();
    }

    /*****
    */

    /** FUNCIÓN QUE REGISTRA A UN AGENTE EN EL DF. SE LE PASA COMO PARÁMETRO LA
    ZONA DEL COORDINADOR, PARA FACILITAR LAS BÚSQUEDAS POSTERIORES EN EL DF. */
    public void Registro_en_el_df(String zon) {
        //Registro al DF
        ServiceDescription sd1=new ServiceDescription();
        //Indicamos los parametros del Agente
        sd1.setType("Zona");
        sd1.setName(getLocalName());
        sd1.setOwnership(zon);
    }
}

```

```

        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        //Nos añadimos como un nuevo servicio
        dfd.addServices(sd1);
        //Indicamos nuestro nombre
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);
        //Registramos el lenguaje
        registerLanguage(SLCodec.NAME, new SLCodec());
        //Registro de la ontología
        registerOntology(JBSOntology.NAME, JBSOntology.instance());
        System.out.println("Coordinador Zonal (" + getLocalName() + " de
la Zona" + sd1.getOwnership() + ") on-line");

        try{
            //Nos registramos al DF
            DFServiceCommunicator.register(this,dfd);
        }catch (FIPAException e) {
            System.err.println(getLocalName()+": Error en el registro
al DF. Error: "+e.getMessage());
            doDelete();
        }
    }

    /*****
    */

    /* FUNCIÓN QUE DESREGISTRA A UN AGENTE DEL DF. SE USA EN EL "takeDown" DEL
    AGENTE. */
    public void Desregistro_del_df() {
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);

        try{
            //Nos desregistramos
            DFServiceCommunicator.deregister(this,dfd);
            System.out.println("Agente " + getLocalName() + " eliminado
del DF");
        }catch (FIPAException e) {
            System.out.println(getLocalName()+": Error en el
desregistro del DF. Error: "+e.getMessage());
        }
    }

    /*****
    */

    /* FUNCIÓN QUE BUSCA EN EL DF TODOS LOS NOMBRES DE LOS AGENTES QUE CUMPLEN
    UN CIERTO SERVICIO. EN CONCRETO DEVUELVE UNA LISTA CON TODOS LOS AGENTES
    DEL TIPO tipo CUYO PROPIETARIO ES propietario, EXCEPTO LOS AGENTES CUYO
    PROPIETARIO ES excepto. */
    public List Busca_en_DF(String propietario, String tipo, String excepto) {
        List resultado = new ArrayList();
        DFAgentDescription dfad=new DFAgentDescription();
        ServiceDescription sd=new ServiceDescription();
        List result;

        sd.setOwnership(propietario);
        sd.setType(tipo);
        dfad.addServices(sd); // Añadimos el servicio al
descriptor de agente DF, para la búsqueda

        try { // Procedemos a buscar en df lo que queremos (dfad)
            result = jade.domain.DFServiceCommunicator.search(super,dfad);

```

```

        if (result.isEmpty())
            return resultado;

        for (int i=0;i<result.size();i++) {
            dfad = (DFAgentDescription) result.get(i);
            if (!dfad.getName().getName().equalsIgnoreCase(excepto))
                resultado.add(dfad.getName().getName());           //Con
esto obtenemos el nombre del agente
        }
    } catch (Exception exc) {
        System.out.println( exc.toString() );
    }
    return resultado;
}

/*****
*/

/* FUNCIÓN QUE PREPARA UN MENSAJE. RELLENA DE MANERA FÁCIL Y RÁPIDA LOS
CAMPOS
DE DESTINO, TIPO, Y CONTENIDO CON LOS PARÁMETROS PASADOS. LOS DE LENGUAJE
Y
ONTOLOGÍA LOS PONE POR DEFECTO, AL IGUAL QUE EL REMITENTE */
public ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0,
Object contenido) {
    ACLMessage msg = new ACLMessage(tipo_mensaje);
    msg.addReceiver(receiver);
    msg.setSender(new AID(getLocalName()));
    msg.setLanguage(SLCodec.NAME);
    msg.setOntology(JBSOntology.NAME);
    SL_Action a = new SL_Action();
    a.set_0(new AID(set0));
    a.set_1(contenido);
    List l = new ArrayList();
    l.add(a);
    try {
        fillContent(msg,l);
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return (msg);
}

/*****
*/

/* FUNCIÓN QUE EXTRAER EL CONTENIDO DE UN MENSAJE. FACILITA LA TAREA. SE LE
PASA
EL MENSAJE DEL CUAL QUEREMOS EXTRAER EL CONTENIDO, Y SE DEVUELVE EL OBJETO
DE
LA ONTOLOGÍA QUE CONTIENE. */
public Object Extrae_contenido(ACLMessage msg) {
    List l = new ArrayList();
    SL_Action a = new SL_Action();
    Object resultado = new Object();
    try {
        l= extractContent(msg);
        a = (SL_Action) l.get(0);
        resultado = (Object) a.get_1();
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return resultado;
}

/*****
*/

```

```
/* FUNCIÓN QUE REALIZA UNA CONSULTA EN LA BASE DE DATOS. BUSCA EN LA TABLA
   tabla DE LA BD bd1, LA COLUMNA col_return, DE LOS REGISTROS QUE CUMPLAN
   QUE col = val, EXCEPTO LOS QUE SON excepto*/
public List Busca_en_BD(String bd1, String tabla, String col_return, String
col, String val, String excepto) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;
    List lista_a_retornar = new ArrayList();
    String res = "";
    String aux = "";

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1);
        stmt = con.createStatement();
        // Búsqueda de los pacientes del órgano "org"
        if (stmt.execute("Select * from " + tabla + " where " + col + "=" +
val + " + """)) {
            rs = stmt.getResultSet();
        }
        int i=0;
        while (rs.next()) {
            aux = rs.getString(col_return);
            if (!aux.equals(res)) {
                res=aux;
                if (!res.equals(excepto)) {
                    lista_a_retornar.add(res);
                    i++;
                }
            }
        }
        System.out.println(bd1 + " (" + getLocalName() + "): Registros
encontrados de " + col + "=" + val + " => " + i);
        con.close();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
    }
    return(lista_a_retornar);
}

/*****
*/

/* COMPORTAMIENTO QUE RESPONDE A UN REQUEST PROCEDENTE DE UN HOSPITAL. */
public class H_Request extends FipaRequestResponderBehaviour.ActionHandler
    implements FipaRequestResponderBehaviour.Factory {

    Organo organo = new Organo();

    public H_Request(Agent a, ACLMessage msg){
        super(a,msg);
    }

    public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
        return new H_Request(myAgent, msg);
    }

    public void action(){
        ACLMessage msg_req = getRequest();
        organo = (Organo) Extrae_contenido(msg_req);
        // System.out.println(msg_req.toString());

        /* Si el mensaje viene del CN, la cosa cambia, pq hay que buscar
        * en toda la zona, ya que el órgano proviene de otra zona, y por tanto
        * el hospital origen no se encuentra en esta zona.
        */
    }
}
```

```
String autonomia = "";

System.out.println(getLocalName() + ": Recibido Request de " +
msg_req.getSender());

//Buscamos en el DF el nombre del CN, para ver si el mensaje proviene de
él o no
List nacional = new ArrayList();
nacional = Busca_en_DF("CN", "Nacional", "");
AID AID_CN = new AID();
if (!nacional.isEmpty()) { //Si CN está registrado en el DF
    AID_CN = new AID((String) nacional.get(0));
} else { //Si CN no está registrado
    System.out.println(getLocalName() + ": Coordinador Nacional (CN) no
registrado");
}

if (!msg_req.getSender().getName().equals(AID_CN.getName())) { //Si el
Request no viene de CN
    /** 1- Encontrar los A's con los que hay que hacer el Contract-Net */
    //Buscamos la ciudad a la que pertenece el órgano donado
    String ciudad = "";
    DFAgentDescription dfad=new DFAgentDescription();
    ServiceDescription sd=new ServiceDescription();
    List result = new ArrayList();

    sd.setName(msg_req.getSender().getName());
    sd.setType("Hospital");
    dfad.addServices(sd); // Añadimos el servicio al
descriptor de agente DF, para la búsqueda

    try { // Procedemos a buscar en df lo que queremos (dfad)
        result = jade.domain.DFServiceCommunicator.search(Zona.this,dfad);
    } catch (Exception exc) {
        System.out.println( exc.toString() );
    }
    dfad = (DFAgentDescription) result.get(0);
    Iterator it = dfad.getAllServices();
    sd = (ServiceDescription) it.next();
    ciudad = sd.getOwnership();
    //Buscamos la autonomía a la que pertenece la ciudad origen
    autonomia = (String) Busca_en_BD("BD-
Ciudades","Ciudades","Autonomia","Ciudad",ciudad,"").get(0);
} else { //Si viene de CN no nos interesa la procedencia del órgano
    autonomía = "";
}

//Buscamos los coord de A con los q se tiene q hacer el Contract-Net
List coords_A = new ArrayList();
List destinos = new ArrayList();
List autonomias = Busca_en_BD("BD-
Ciudades","Ciudades","Autonomia","Zona",zona,autonomia);
System.out.println(getLocalName() + ": Autonomias de la Zona" + zona +
":");
for (int i=0;i<autonomias.size();i++) {
    System.out.println(" " + autonomias.get(i));
    coords_A.addAll(Busca_en_DF((String)
autonomias.get(i),"Autonomia",""));
}
if (coords_A.isEmpty()) { //No hay más Autonomías en la Zona
    System.out.println(getLocalName() + ": No hay Coordinadores de
autonomia en la Zona" + zona);
    /** Hay que enviarle respuesta a H, indicando el problema */
    ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
    msg_refuse.setProtocol("fipa-request");
    msg_refuse.setConversationId(msg_req.getConversationId());
    msg_refuse.setInReplyTo(msg_req.getReplyWith());
    send(msg_refuse);
}
```

```

    } else { //Se han encontrado Autonomías en la zona, a parte del
origen
        System.out.println(getLocalName() + ": Coordinadores de la Zona" +
zona + ":");
        for (int i=0;i<coords_A.size();i++) {
            System.out.println("      " + coords_A.get(i));
            destinos.add(new AID((String) coords_A.get(i)));
        }

        /** 2- Comenzar el Contract-Net */
        ACLMessage msg_cfp = Prepara_msg(ACLMessage.CFP,null,"A",organo);
        addBehaviour(new
Inicia_Contract_Net_misma_zona(Zona.this,msg_cfp,destinos,organo,msg_req));
    }

    public boolean done() {
        return true;
    }

    public void reset() {
    }

    public class miFipaRequestResponderBehaviour extends
FipaRequestResponderBehaviour {
        public miFipaRequestResponderBehaviour(Agent a) {
            super(a);
        }
        protected String getActionName(ACLMessage msg) throws
NotUnderstoodException, RefuseException {
            return JBSontology.ORGANO;
        }
    }

    /*****
    */

    /* COMPORTAMIENTO QUE INICIA EL CONTRACT NET CON OTROS AGENTES. SIRVE PARA
    ENCONTRAR EL PACIENTE ADECUADO EN SEGÚN QUE NIVELES JERÁRQUICOS USANDO UN
    PROTOCOLO DE TIPO FIPA-CONTRACT-NET */
    public class Inicia_Contract_Net_misma_zona extends
jade.proto.FipaContractNetInitiatorBehaviour {

        Organo organo;
        ACLMessage msg_req;

        // Constructores de la clase Inicia_Contract_Net
        public Inicia_Contract_Net_misma_zona(Agent a, ACLMessage msg, List
responders, Organo org, ACLMessage msg_usado_en_req) {
            super(a,msg,responders);
            organo=org;
            msg_req = msg_usado_en_req;
            System.out.println(getLocalName() + ": Iniciado Contract-Net en la Zona"
+ zona);
        }

        public Inicia_Contract_Net_misma_zona(Agent a, ACLMessage msg, Organo org,
ACLMessage msg_usado_en_req) {
            this(a,msg,null,org,msg_usado_en_req);
            organo=org;
            msg_req = msg_usado_en_req;
        }

        // Handler para los mensajes de tipo Propose
        public Vector handleProposeMessages(Vector proposals) {
            System.out.println(getLocalName() + ": Analizando propuestas de
Autonomías...");

```

```

Vector respuestas = new Vector();
/** Hemos de juntar todas las propuestas que nos llegan desde los
 * distintos coordinadores. Cogemos las listas de pacientes que nos
pasan y
 * las juntamos en una. Para cada paciente tendremos el nombre del H y
el
 * id del paciente. Esta lista habrá que pasarla a H para que la
evalúe.
 * Después cuando H devuelva la eleccion, se le pasará a los A's, que
se
 * encargarán de aceptar la mejor propuesta y rechazar el resto.
 */
ACLMessage msg_actual = new ACLMessage(ACLMessage.NOT_UNDERSTOOD);
Lista_objetos pacientes = new Lista_objetos();
Lista_objetos pacientes_para_H = new Lista_objetos();
Iterator it;
Paciente paciente_actual = new Paciente();
for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
//Hemos de juntar las listas recibidas de cada autonomía
msg_actual = (ACLMessage) proposals.get(i);
pacientes = (Lista_objetos) Extrae_contenido(msg_actual);
it = pacientes.getAllpacientes();
while (it.hasNext()) {
//Sacamos un paciente de la lista de pacientes de una autonomía
paciente_actual = (Paciente) it.next();
//Lo metemos en la lista de pacientes total, la que se le pasará a
A0
    pacientes_para_H.addpacientes(paciente_actual);
//Incrementamos el número de pacientes en la lista que pasaremos a
A0
    pacientes_para_H.setnpacientes(new
Long(pacientes_para_H.getnpacientes().intValue()+1));
}
}

//Usamos el msg_req para mezclar el Contract-Net con el Request de H
ACLMessage msg_agree =
Prepara_msg(ACLMessage.AGREE,msg_req.getSender(),"Hospital",pacientes_para_H);
msg_agree.setProtocol("fipa-request");
msg_agree.setConversationId(msg_req.getConversationId());
msg_agree.setInReplyTo(msg_req.getReplyWith());
send(msg_agree);

System.out.println(getLocalName() + ": Lista de pacientes enviada a " +
msg_req.getSender().getName());

ACLMessage msg_eleccion =
blockingReceive(MessageTemplate.MatchConversationId(msg_agree.getConversationI
d()));
Eleccion paciente_escogido = (Eleccion) Extrae_contenido(msg_eleccion);
/** Hemos de pasarle la elección a los distintos coordinadores, para
 * que estos acepten o rechacen las propuestas. A su vez, Z sólo
aceptará
 * la propuesta de la autonomía cuya ciudad sea la elegida
 */

//Hay que comprobar si la eleccion es valida, es decir, no se trata de
un id=0
if (paciente_escogido.getid_paciente().intValue()!=0) { //Si no es una
elección nula
//Buscamos la ciudad a la que pertenece el paciente escogido
String ciudad = "";
DFAgentDescription dfad=new DFAgentDescription();
ServiceDescription sd=new ServiceDescription();
List result = new ArrayList();

sd.setName(paciente_escogido.getid_hospital());

```

```

        sd.setType("Hospital");
        dfad.addServices(sd); // Añadimos el servicio al
        descriptor de agente DF, para la búsqueda

        try { // Procedemos a buscar en df lo que queremos (dfad)
            result = jade.domain.DFServiceCommunicator.search(Zona.this,dfad);
        } catch (Exception exc) {
            System.out.println( exc.toString() );
        }
        dfad = (DFAgentDescription) result.get(0);
        it = dfad.getAllServices();
        sd = (ServiceDescription) it.next();
        ciudad = sd.getOwnership();
        //Buscamos la autonomía a la que pertenece la ciudad del paciente
        escogido
        String autonomia_escogida = (String) Busca_en_BD("BD-
        Ciudades","Ciudades","Autonomia","Ciudad",ciudad,"").get(0);
        AID coord_A = new AID((String)
        Busca_en_DF(autonomia_escogida,"Autonomia","").get(0));

        System.out.println(getLocalName() + ": El paciente escogido se
        encuentra en " + autonomia_escogida);
        Lista_objetos lista = new Lista_objetos();

        boolean aceptado = false; //Indica si alguna A ha sido aceptada

        ACLMessage aceptar = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);

        for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
            //Hemos de aceptar la que viene de la autonomía escogida y rechazar
        el resto
            msg_actual = (ACLMessage) proposals.get(i);
            //Buscamos la autonomía a la que pertenece el paciente actual
            lista = (Lista_objetos) Extrae_contenido(msg_actual);
            it = lista.getAllpacientes();
            paciente_actual = (Paciente) it.next();
            //Buscamos la autonomía a la que pertenece la ciudad del paciente
        actual
            String autonomia_actual = (String) Busca_en_BD("BD-
            Ciudades","Ciudades",
            "Autonomia","Ciudad",paciente_actual.getlocalidad(),"").get(0);
            if (autonomia_actual.equals(autonomia_escogida)) {
                //Hay que aceptar la propuesta del hospital
                aceptar = Prepara_msg(ACLMessage.ACCEPT_PROPOSAL,
        msg_actual.getSender(),"A",paciente_escogido);
                aceptar.setInReplyTo(msg_actual.getReplyWith());
                aceptar.setConversationId(msg_actual.getConversationId());
                send(aceptar);
                respuestas.addElement((ACLMessage)aceptar);
                aceptado = true;
            //
                System.out.println(aceptar.toString());
            } else { //Si el hospital del paciente no es el elegido
                //Hay que rechazar la propuesta del hospital
                ACLMessage rechazar = Prepara_msg(ACLMessage.REJECT_PROPOSAL,
        msg_actual.getSender(),"A",paciente_escogido);
                //
                ACLMessage rechazar = new
        ACLMessage(ACLMessage.REJECT_PROPOSAL);
                //
                rechazar.setSender(new AID(getLocalName()));
                //
                rechazar.addReceiver(msg_actual.getSender());
                rechazar.setInReplyTo(msg_actual.getReplyWith());
                rechazar.setConversationId(msg_actual.getConversationId());
                send(rechazar);
                respuestas.addElement((ACLMessage)rechazar);
            }
        }
    }
}

```

```
        System.out.println(getLocalName() + ": Enviadas las respuestas al
Contract-Net");

        if (aceptado) {
            /** Aquí hay que recibir la respuesta de la Autonomía aceptada */
            ACLMessage msg_inform =
blockingReceive(MessageTemplate.MatchConversationId(aceptar.getConversationId(
)));
            System.out.println(getLocalName() + ": Recibido Inform de " +
msg_inform.getSender());
            /** Aquí se ha de finalizar el Request con H, indicando que todo OK
*/
            Eleccion paciente = (Eleccion) Extrae_contenido(msg_inform);
            ACLMessage msg_inform_final =
Prepara_msg(ACLMessage.INFORM,msg_req.getSender(),"Hospital",paciente);
            msg_inform_final.setConversationId(msg_req.getConversationId());
            msg_inform_final.setInReplyTo(msg_req.getReplyWith());
            msg_inform_final.setProtocol("fipa-request");
            send(msg_inform_final);
        }

        } else { //Si se trata de una elección nula
            //Hay que rechazar todas las propuestas de las autonomías
            for (int i=0;i<proposals.size();i++) { //Para todas las propuestas
                msg_actual = (ACLMessage) proposals.get(i);
                ACLMessage rechazar = Prepara_msg(ACLMessage.REJECT_PROPOSAL,
msg_actual.getSender(),"A",paciente_escogido);
                // ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
                // rechazar.setSender(new AID(getLocalName()));
                // rechazar.addReceiver(msg_actual.getSender());
                rechazar.setInReplyTo(msg_actual.getReplyWith());
                rechazar.setConversationId(msg_actual.getConversationId());
                send(rechazar);
                respuestas.addElement((ACLMessage)rechazar);
            }
            System.out.println(getLocalName() + ": El hospital de origen no ha
encontrado pacientes aptos");
        }

        return null;
    }

    // Handler para otros tipos de mensajes
    public void handleOtherMessages(ACLMessage msg) {
        if (msg.getPerformative()==ACLMessage.REFUSE) {
            System.out.println(getLocalName() + ": Recibido un mensaje de tipo
Refuse de " + msg.getSender());
            /** Hay que enviarle respuesta a H, indicando el problema */
            ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
            msg_refuse.setProtocol("fipa-request");
            msg_refuse.setConversationId(msg_req.getConversationId());
            msg_refuse.setInReplyTo(msg_req.getReplyWith());
            send(msg_refuse);
            System.out.println(getLocalName() + ": Enviado Refuse a " +
msg_req.getSender());
        }
    }

    // Handler para los mensajes de tipo Final
    public Vector handleFinalMessages(Vector messages) {
        System.out.println(getLocalName() + ": Recibido un mensaje de tipo
Final");
        System.out.println(getLocalName() + ": Contract-Net finalizado
satisfactoriamente");
    }
}
```

```
        Eleccion paciente_escogido = (Eleccion) Extrae_contenido((ACLMessage)
messages.get(0));

        ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM,msg_req.getSender(),"Hospital",paciente_escogido
);
        msg_inform.setConversationId(msg_req.getConversationId());
        msg_inform.setInReplyTo(msg_req.getReplyWith());
        msg_inform.setProtocol("fipa-request");
        send(msg_inform);
        return(null);
    }

    // Devuelve el contenido del mensaje CFP
    public String createCfpContent(String cfpContent, String
receiver) {
        return(cfpContent);
    }
}

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

/* SETUP DEL AGENTE */

protected void setup () {
    if (getLocalName().equals("Z1")) {
        zona="I";
    }
    if (getLocalName().equals("Z2")) {
        zona="II";
    }
    if (getLocalName().equals("Z3")) {
        zona="III";
    }
    if (getLocalName().equals("Z4")) {
        zona="IV";
    }
    if (getLocalName().equals("Z5")) {
        zona="V";
    }
    if (getLocalName().equals("Z6")) {
        zona="VI";
    }
    Registro_en_el_df(zona);

    FipaRequestResponderBehaviour requester = new
miFipaRequestResponderBehaviour(Zona.this);
    requester.registerFactory(JBSOntology.ORGANO,new
H_Request(Zona.this,null));
    addBehaviour(requester);
}

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

/* TAKEDOWN DEL AGENTE */

protected void takeDown(){
    Desregistro_del_df();
}

}
```

8.4. Nacional.java

```

/*****
**
Nacional

PFC Jaime Bocio Sanz
2-5-2002

Agente que representa un coordinador Nacional. Recibirá peticiones de un
Hospital y hará de intermediario para buscar un posible paciente. Contacta
con
Z para que éste contacte con los hospitales para trasnmitirles los datos del
órgano donado y para devolver la lista con los pacientes ordenados al
Hospital
de origen

*****/

package pfc_ei;

import jade.lang.acl.*;
import jade.core.behaviours.*;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.*;
import jade.core.*;
import jade.lang.sl.sl.*;
import jade.onto.sl.sl.*;
import jade.proto.FipaRequestResponderBehaviour;
import jade.proto.FipaRequestInitiatorBehaviour;

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

import jade.core.Agent;

//Paquete con la ontología definida
import pfc_ei.ontology.*;

/*****
*/

public class Nacional extends Agent {

    List zonas_ordenadas = new ArrayList();
    boolean todo_comprobado = false; //Indica si se han comprobado todas las
zonas
    int cont = 0; //Indica el número de zonas comprobadas

/*****
*/

    public Nacional() {
        super();
    }

/*****
*/

/* FUNCIÓN QUE REGISTRA A CN EN EL DF. */
    public void Registro_en_el_df() {
        //Registro al DF
        ServiceDescription sd1=new ServiceDescription();
        //Indicamos los parametros del Agente
    }
}

```

```

        sd1.setType("Nacional");
        sd1.setName(getLocalName());
        sd1.setOwnership("CN");
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        //Nos añadimos como un nuevo servicio
        dfd.addServices(sd1);
        //Indicamos nuestro nombre
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);
        //Registramos el lenguaje
        registerLanguage(SLCodec.NAME, new SLCodec());
        //Registro de la ontología
        registerOntology(JBSOntology.NAME, JBSOntology.instance());
        System.out.println("Coordinador Nacional (" + getLocalName() + ")
on-line");

        try{
            //Nos registramos al DF
            DFServiceCommunicator.register(this,dfd);
        }catch (FIPAException e) {
            System.err.println(getLocalName()+": Error en el registro
al DF. Error: "+e.getMessage());
            doDelete();
        }
    }

    /*****
    */

    /* FUNCIÓN QUE DESREGISTRA A UN AGENTE DEL DF. SE USA EN EL "takeDown" DEL
    AGENTE. */
    public void Desregistro_del_df() {
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);

        try{
            //Nos desregistramos
            DFServiceCommunicator.deregister(this,dfd);
            System.out.println("Agente " + getLocalName() + " eliminado
del DF");
        }catch (FIPAException e) {
            System.out.println(getLocalName()+": Error en el
desregistro del DF. Error: "+e.getMessage());
        }
    }

    /*****
    */

    /* FUNCIÓN QUE BUSCA EN EL DF TODOS LOS NOMBRES DE LOS AGENTES QUE CUMPLEN
    UN CIERTO SERVICIO. EN CONCRETO DEVUELVE UNA LISTA CON TODOS LOS AGENTES
    DEL TIPO tipo CUYO PROPIETARIO ES propietario, EXCEPTO LOS AGENTES CUYO
    PROPIETARIO ES excepto. */
    public List Busca_en_DF(String propietario, String tipo, String excepto) {
        List resultado = new ArrayList();
        DFAgentDescription dfad=new DFAgentDescription();
        ServiceDescription sd=new ServiceDescription();
        List result;

        sd.setOwnership(propietario);
        sd.setType(tipo);
        dfad.addServices(sd); // Añadimos el servicio al
        descriptor de agente DF, para la búsqueda
    }

```

```

        try { // Procedemos a buscar en df lo que queremos (dfad)
            result = jade.domain.DFServiceCommunicator.search(super, dfad);
            if (result.isEmpty())
                return resultado;

            for (int i=0;i<result.size();i++) {
                dfad = (DFAgentDescription) result.get(i);
                if (!dfad.getName().getName().equalsIgnoreCase(excepto))
                    resultado.add(dfad.getName().getName()); //Con
            } //esto obtenemos el nombre del agente
        } catch (Exception exc) {
            System.out.println( exc.toString() );
        }
        return resultado;
    }

    /*
    /* *****
    /*
    /* FUNCIÓN QUE PREPARA UN MENSAJE. RELLENA DE MANERA FÁCIL Y RÁPIDA LOS
    CAMPOS
    DE DESTINO, TIPO, Y CONTENIDO CON LOS PARÁMETROS PASADOS. LOS DE LENGUAJE
    Y
    ONTOLOGÍA LOS PONE POR DEFECTO, AL IGUAL QUE EL REMITENTE */
    public ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0,
    Object contenido) {
        ACLMessage msg = new ACLMessage(tipo_mensaje);
        msg.addReceiver(receiver);
        msg.setSender(new AID(getLocalName()));
        msg.setLanguage(SLCodec.NAME);
        msg.setOntology(JBSontology.NAME);
        SL_Action a = new SL_Action();
        a.set_0(new AID(set0));
        a.set_1(contenido);
        List l = new ArrayList();
        l.add(a);
        try {
            fillContent(msg,l);
        } catch (Exception e) {
            System.out.println(getLocalName() + ": " + e.toString());
        }
        return (msg);
    }

    /*
    /* *****
    /*
    /* FUNCIÓN QUE EXTRAER EL CONTENIDO DE UN MENSAJE. FACILITA LA TAREA. SE LE
    PASA
    EL MENSAJE DEL CUAL QUEREMOS EXTRAER EL CONTENIDO, Y SE DEVUELVE EL OBJETO
    DE
    LA ONTOLOGÍA QUE CONTIENE. */
    public Object Extrae_contenido(ACLMessage msg) {
        List l = new ArrayList();
        SL_Action a = new SL_Action();
        Object resultado = new Object();
        try {
            l= extractContent(msg);
            a = (SL_Action) l.get(0);
            resultado = (Object) a.get_1();
        } catch (Exception e) {
            System.out.println(getLocalName() + ": " + e.toString());
        }
        return resultado;
    }
    }
    
```

```

/*****
*/

/* FUNCIÓN QUE REALIZA UNA CONSULTA EN LA BASE DE DATOS. BUSCA EN LA TABLA
   tabla DE LA BD bd1, LA COLUMNA col_return, DE LOS REGISTROS QUE CUMPLAN
   QUE col = val, EXCEPTO LOS QUE SON excepto*/
public List Busca_en_BD(String bd1, String tabla, String col_return, String
col, String val, String excepto) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;
    List lista_a_retornar = new ArrayList();
    String res = "";
    String aux = "";

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1);
        stmt = con.createStatement();
        // Búsqueda de los pacientes del órgano "org"
        if (stmt.execute("Select * from " + tabla + " where " + col + "='" +
val + "'")) {
            rs = stmt.getResultSet();
        }
        int i=0;
        while (rs.next()) {
            aux = rs.getString(col_return);
            if (!aux.equals(res)) {
                res=aux;
                if (!res.equals(excepto)) {
                    lista_a_retornar.add(res);
                    i++;
                }
            }
        }
        System.out.println(bd1 + " (" + getLocalName() + "): Registros
encontrados de " + col + "=" + val + " => " + i);
        con.close();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
    }
    return(lista_a_retornar);
}

/*****
*/

public boolean Inicia_búsqueda_zona(Organo organo, ACLMessage msg_req_de_H)
{
    //Buscamos la zona de la que viene el órgano, pq no volveremos a buscar en
    ella
    //Buscamos la ciudad a la que pertenece el órgano donado
    DFAgentDescription dfad=new DFAgentDescription();
    ServiceDescription sd=new ServiceDescription();
    List result = new ArrayList();

    sd.setName(msg_req_de_H.getSender().getName());
    sd.setType("Hospital");
    dfad.addServices(sd); // Añadimos el servicio al descriptor
    de agente DF, para la búsqueda

    try { // Procedemos a buscar en df lo que queremos (dfad)
        result = jade.domain.DFServiceCommunicator.search(Nacional.this,dfad);
    } catch (Exception exc) {
        System.out.println( exc.toString() );
    }
    dfad = (DFAgentDescription) result.get(0);
    Iterator it = dfad.getAllServices();
}

```

```

        sd = (ServiceDescription) it.next();
        String ciudad = sd.getOwnership();
        //Buscamos la zona a la que pertenece la ciudad origen
        String zona = (String) Busca_en_BD("BD-
Ciudades","Ciudades","Zona","Ciudad",ciudad,"").get(0);

        //Buscamos en el DF el coordinador de la 1ª zona de la lista
        List zonas = Busca_en_DF((String) zonas_ordenadas.get(0),"Zona","");
        //La zona en la que queremos buscar es esta
        String zona_actual = (String) zonas_ordenadas.get(0);

        //Comprobaciones
        for(int i=0;i<zonas_ordenadas.size();i++)
            System.out.print(" " + zonas_ordenadas.get(i));
        System.out.println(" A");

        System.out.println(zona_actual + " " + zona);

        if (zona_actual.equals(zona)) { //Si la zona es la origen
            //Hemos de buscar la siguiente
            zonas_ordenadas.add(zonas_ordenadas.size()-1,(String)
zonas_ordenadas.remove(0));
        //        zonas = Busca_en_DF((String) zonas_ordenadas.get(0),"Zona","");

            //Comprobaciones
            for(int i=0;i<zonas_ordenadas.size();i++)
                System.out.print(" " + zonas_ordenadas.get(i));
            System.out.println(" C");

            System.out.println(getLocalName() + ": Coordinador de la Zona" +
zona_actual + " ya comprobado (Origen)");
            return false;
        } else {
            //Ponemos esta zona al final, para dar oportunidades a todas
            zonas_ordenadas.add(zonas_ordenadas.size()-1,(String)
zonas_ordenadas.remove(0));
        /*
        }

        //Comprobaciones
        for(int i=0;i<zonas_ordenadas.size();i++)
            System.out.print(" " + zonas_ordenadas.get(i));
        System.out.println(" B");
    */
    if (!zonas.isEmpty()) { //Si Z está registrado en el DF
        AID AID_Z = new AID((String) zonas.get(0));
        //Comenzamos el Coord_Request con Z
        ACLMessage msg_req_Z =
Prepara_msg(ACLMessage.REQUEST,AID_Z,"Z",(Organo)Extrae_contenido(msg_req_de_H
));
        MessageTemplate mt = MessageTemplate.MatchSender(AID_Z);
        //        reset(msg_req,mt);
        addBehaviour(new
Coord_Request(Nacional.this,msg_req_Z,mt,msg_req_de_H));
        return true;
    } else { //Si Z no está registrado
        System.out.println(getLocalName() + ": Coordinador de la Zona" +
zona_actual + " no registrado");
        return false;
    }
}
}

/*****
*/

/* COMPORTAMIENTO QUE RESPONDE A UN REQUEST PROCEDENTE DE UN HOSPITAL. */
public class H_Request extends FipaRequestResponderBehaviour.ActionHandler
    implements FipaRequestResponderBehaviour.Factory {

```

```
Organo organo = new Organo();

public H_Request(Agent a, ACLMessage msg){
    super(a,msg);
}

public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
    return new H_Request(myAgent, msg);
}

public void action(){
    ACLMessage msg_req = getRequest();
    todo_comprobado = false;
    cont = 0;
    organo = (Organo) Extrae_contenido(msg_req);
//    System.out.println(msg_req.toString());
    /** Aquí la cosa cambia, no se hace Contract-Net con los Z's, sino
     *   que se hace Request. El CN tiene una lista ordenada de las
     *   Zonas según las últimas donaciones que han llegado a nivel nacional.
     */

    /** Hemos de coger al primer coordinador de Zona al que le toque,
     *   teniendo en cuenta que no sea el Coord de la Zona de Origen.
     *
     *   La búsqueda tendría que seguir hasta que se encontrara paciente en
     *   alguna de las zonas
     */

    while (!Inicia_búsqueda_zona((Organo)Extrae_contenido(msg_req),msg_req))
    {
        //Mientras no esté registrado el Z en cuestión, vamos probado con otro
        cont++;
        if (cont==6) {
            todo_comprobado = true;
            System.out.println(getLocalName() + ": Se han comprobado ya todas
las zonas");
            // Hay que avisar a H con un Refuse
            ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
            msg_refuse.setProtocol("fipa-request");
            msg_refuse.setConversationId(msg_req.getConversationId());
            msg_refuse.setInReplyTo(msg_req.getReplyWith());
            send(msg_refuse);
            break;
        }
    }
}

public boolean done() {
    return true;
}

public void reset() {
}

}

public class miFipaRequestResponderBehaviour extends
FipaRequestResponderBehaviour {
    public miFipaRequestResponderBehaviour(Agent a) {
        super(a);
    }
    protected String getActionName(ACLMessage msg) throws
NotUnderstoodException, RefuseException {
        return JBSontology.ORGANO;
    }
}
```

```
/*
*****
*/

/* COMPORTAMIENTO QUE INICIA UN PROTOCOLO REQUEST CON UN COORDINADOR
AUTONÓMICO.
SE LE PASA EL DONANTE. */
public class Coord_Request extends FipaRequestInitiatorBehaviour {

    AID AID_coord = new AID();
    ACLMessage msg_req;

    //Constructor
    public Coord_Request (Agent agent, ACLMessage requestMsg, MessageTemplate
mt, ACLMessage msg_req_de_H) {
        super(agent, requestMsg, mt);
        Iterator it = requestMsg.getAllReceiver();
        AID_coord = (AID) it.next();
        msg_req = msg_req_de_H;
        System.out.println(getLocalName() + ": Iniciado Request-Protocol con " +
AID_coord);
    }

    protected void handleAgree(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Agree de " +
msg.getSender().getName());

        /** Hay que pasarle los pacientes a H y esperar la respuesta */
        ACLMessage msg_agree = Prepara_msg(ACLMessage.AGREE, msg_req.getSender(),
"H", (Lista_objetos)
Extrae_contenido(msg));
        msg_agree.setProtocol("fipa-request");
        msg_agree.setConversationId(msg_req.getConversationId());
        msg_agree.setInReplyTo(msg_req.getReplyWith());
        send(msg_agree);

        System.out.println(getLocalName() + ": Lista de pacientes enviada a " +
msg_req.getSender().getName());
        System.out.println(getLocalName() + ": Esperando respuesta de " +
msg_req.getSender());

        ACLMessage msg_eleccion =
blockingReceive(MessageTemplate.MatchConversationId(msg_req.getConversationId(
)));
        Eleccion paciente_escogido = (Eleccion) Extrae_contenido(msg_eleccion);
        //Hay que comprobar si la eleccion es valida, es decir, no se trata de
un id=0
        if (paciente_escogido.getid_paciente().intValue()!=0) { //Si no es una
elección nula
            System.out.println(getLocalName() + ": Paciente escogido => " +
paciente_escogido.getid_paciente()
+ " de " + paciente_escogido.getid_hospital());
            /** Hemos de indicarle la elección al Z */
            ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM, msg.getSender(), "Z", paciente_escogido);
            msg_inform.setConversationId(msg.getConversationId());
            send(msg_inform);
        } else { //Si se trata de una elección nula
            /** Hemos de indicarle la elección al Z */
            ACLMessage msg_inform =
Prepara_msg(ACLMessage.INFORM, msg.getSender(), "Z", paciente_escogido);
            msg_inform.setConversationId(msg.getConversationId());
            send(msg_inform);
            System.out.println(getLocalName() + ": El hospital de origen no ha
encontrado pacientes aptos");
        }
    }

    protected void handleInform(ACLMessage msg) {
```

```

        System.out.println(getLocalName() + ": Recibido Inform de " +
msg.getSender().getName());
        Eleccion paciente = (Eleccion) Extrae_contenido(msg);
        /** Hay que pasarle el Inform a H */
        ACLMessage msg_inform_final =
Prepara_msg(ACLMessage.INFORM,msg_req.getSender(),"Hospital",paciente);
        msg_inform_final.setConversationId(msg_req.getConversationId());
        msg_inform_final.setInReplyTo(msg_req.getReplyWith());
        msg_inform_final.setProtocol("fipa-request");
        send(msg_inform_final);
    }

    protected void handleNotUnderstood(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Not-Understood de " +
msg.getSender().getName());
    }

    protected void handleFailure(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Failure de " +
msg.getSender().getName());
    }

    protected void handleRefuse(ACLMessage msg) {
        System.out.println(getLocalName() + ": Recibido Refuse de " +
msg.getSender().getName());

        cont++;
        if (cont>=6)
            todo_comprobado = true;
        if (todo_comprobado) { //Si se han comprobado todas las zonas =>
Acabamos
            System.out.println(getLocalName() + ": Se han comprobado todas las
zonas");
            ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
            msg_refuse.setProtocol("fipa-request");
            msg_refuse.setConversationId(msg_req.getConversationId());
            msg_refuse.setInReplyTo(msg_req.getReplyWith());
            send(msg_refuse);
        } else { //Si faltan zonas por comprobar => Se comprueban
            while
(!Inicia_búsqueda_zona((Organo)Extrae_contenido(msg_req),msg_req)) {
                //Mientras no esté registrado el Z en cuestión, vamos probado con
otro
                cont++;
                if (cont==6) {
                    todo_comprobado = true;
                    System.out.println(getLocalName() + ": Se han comprobado ya todas
las zonas");
                    // Hay que avisar a H con un Refuse
                    ACLMessage msg_refuse =
Prepara_msg(ACLMessage.REFUSE,msg_req.getSender(),"H",new No_hospitales());
                    msg_refuse.setProtocol("fipa-request");
                    msg_refuse.setConversationId(msg_req.getConversationId());
                    msg_refuse.setInReplyTo(msg_req.getReplyWith());
                    send(msg_refuse);
                    break;
                }
            }
        }
    }
}

/*****
*/

```

```

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

/* SETUP DEL AGENTE */

protected void setup () {
    zonas_ordenadas.add(0,"I");
    zonas_ordenadas.add(1,"II");
    zonas_ordenadas.add(2,"III");
    zonas_ordenadas.add(3,"IV");
    zonas_ordenadas.add(4,"V");
    zonas_ordenadas.add(5,"VI");

    Registro_en_el_df();

    FipaRequestResponderBehaviour requester = new
    miFipaRequestResponderBehaviour(Nacional.this);
    requester.registerFactory(JBSOntology.ORGANO,new
    H_Request(Nacional.this,null));
    addBehaviour(requester);

}

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

/* TAKEDOWN DEL AGENTE */

protected void takeDown(){
    Desregistro_del_df();
}

}
```

8.5. Interficie.java

```

/*****
**
Interficie

PFC Jaime Bocio Sanz
25-5-2002

Clase que implementa la interficie gráfica que servirá para que un usuario
interactúe con un agente Hospital.

*****/

package pfc_ei;

import java.awt.*;
import java.awt.event.*;
import com.borland.jbcl.layout.*;
import javax.swing.*;
import javax.swing.border.*;

import pfc_ei.ontology.*;

public class Interficie extends JFrame implements ActionListener {

    private Hospital hospital;
    Paciente donante = new Paciente();

    XYLayout xYLayout1 = new XYLayout();
    JComboBox Tipo_organos = new JComboBox();
    JCheckBox b0 = new JCheckBox();
    JCheckBox b1 = new JCheckBox();
    JCheckBox dr0 = new JCheckBox();
    JCheckBox dr1 = new JCheckBox();
    JTextField Texto_hospital = new JTextField();
    JTextField Texto_ciudad = new JTextField();
    JTextField Texto_provincia = new JTextField();
    JTextField Texto_peso = new JTextField();
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JComboBox Tipo_sangre = new JComboBox();
    JTextField Texto_tamano = new JTextField();
    JLabel jLabel4 = new JLabel();
    JLabel jLabel5 = new JLabel();
    JButton Ok = new JButton();
    JLabel jLabel7 = new JLabel();
    JLabel jLabel8 = new JLabel();
    JTextField Dia = new JTextField();
    JTextField Mes = new JTextField();
    JTextField Anyo = new JTextField();
    JLabel jLabel9 = new JLabel();
    JLabel jLabel10 = new JLabel();
    JLabel jLabel11 = new JLabel();
    JComboBox Grado = new JComboBox();
    JLabel jLabel12 = new JLabel();
    JTextArea Resultado = new JTextArea();
    JTextField Peso_distancia = new JTextField();
    JLabel jLabel14 = new JLabel();
    JLabel jLabel15 = new JLabel();
    JLabel jLabel16 = new JLabel();
    JTextField Peso_peso = new JTextField();
    JLabel jLabel17 = new JLabel();
    JLabel jLabel18 = new JLabel();

```

```
JTextField Peso_tamanyo = new JTextField();
JLabel jLabel19 = new JLabel();
JLabel jLabel110 = new JLabel();
JTextField Peso_antigenos = new JTextField();
JLabel jLabel111 = new JLabel();
JLabel jLabel112 = new JLabel();
JLabel jLabel113 = new JLabel();
JLabel jLabel114 = new JLabel();
JTextField Peso_edad = new JTextField();
JLabel jLabel115 = new JLabel();
JTextField Peso_espera = new JTextField();
JPanel jPanel1 = new JPanel();
TitledBorder titledBorder1;
JPanel jPanel2 = new JPanel();
JPanel jPanel3 = new JPanel();
TitledBorder titledBorder2;
TitledBorder titledBorder3;
XYLayout xYLayout2 = new XYLayout();
JScrollPane jScrollPane1 = new JScrollPane();
JButton Limpia_BD = new JButton();

public Interficie(Hospital h) {
    this.hospital = h;
    try {
        jbInit();
        this.setSize(543,560);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception {

    titledBorder1 = new TitledBorder("");
    titledBorder2 = new TitledBorder("");
    titledBorder3 = new TitledBorder("");
    jLabel12.setText("Grado de exigencia");
    Grado.setNextFocusableComponent(Ok);
    Grado.setToolTipText("Grado de exigencia para el A0");
    this.getContentPane().setBackground(Color.lightGray);
    this.setDefaultCloseOperation(3);
    this.setForeground(Color.black);
    this.setResizable(false);
    this.setTitle("Datos del donante");
    this.getContentPane().setLayout(xYLayout1);
    b0.setBackground(Color.lightGray);
    b0.setNextFocusableComponent(b1);
    b0.setText("B0");
    b1.setBackground(Color.lightGray);
    b1.setNextFocusableComponent(dr0);
    b1.setText("B1");
    dr0.setBackground(Color.lightGray);
    dr0.setNextFocusableComponent(dr1);
    dr0.setText("DR0");
    dr1.setBackground(Color.lightGray);
    dr1.setNextFocusableComponent(Tipo_sangre);
    dr1.setText("DR1");
    jLabel1.setText("Hospital");
    jLabel2.setText("Localidad");
    jLabel3.setText("Provincia");
    jLabel4.setText("Peso paciente");
    jLabel5.setText("Tamaño órgano");
    Ok.setNextFocusableComponent(Limpia_BD);
    Ok.setToolTipText("Procesa la petición e indica los resultados");
    Ok.setText("Enviar Petición");
    Ok.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            Ok_mouseClicked(e);
        }
    });
}
```

```
    }
  });
  jLabel7.setText("Órgano");
  jLabel8.setText("Sangre");
  jLabel9.setText("/");
  jLabel10.setText("/");
  jLabel11.setText("Fecha deNacimiento");
  Tipo_organo.setNextFocusableComponent(Dia);
  Tipo_organo.setToolTipText("Tipo de órgano donado");
  Resultado.setFont(new java.awt.Font("Dialog", 1, 12));
  Resultado.setForeground(Color.red);
  Resultado.setBorder(titledBorder3);
  Resultado.setToolTipText("Seguimiento y resultados de la petición");
  Resultado.setEditable(false);
  Resultado.setText("Resultados de la petición...");
  xYLayout1.setWidth(533);
  xYLayout1.setHeight(530);
  jLabel14.setText("Distancia");
  jLabel15.setText("%");
  jLabel16.setText("%");
  jLabel17.setText("Peso");
  jLabel18.setText("%");
  jLabel19.setText("Tamaño");
  jLabel110.setText("%");
  jLabel111.setText("Antígenos");
  jLabel112.setText("Espera");
  jLabel113.setText("%");
  jLabel114.setText("%");
  jLabel115.setText("Edad");
  jPanel1.setBackground(Color.lightGray);
  jPanel1.setBorder(new javax.swing.border.TitledBorder (
    new javax.swing.border.EtchedBorder (), "Pesos"));
  jPanel1.setToolTipText("Pesos para el A0");
  jPanel2.setBackground(Color.lightGray);
  jPanel2.setBorder(new javax.swing.border.TitledBorder (
    new javax.swing.border.EtchedBorder (), "Antígenos"));
  jPanel2.setToolTipText("Antígenos");
  jPanel3.setBackground(Color.lightGray);
  jPanel3.setBorder(new javax.swing.border.TitledBorder (
    new javax.swing.border.EtchedBorder (), "Resultados"));
  jPanel3.setLayout(xYLayout2);
  Peso_distancia.setNextFocusableComponent(Peso_peso);
  Peso_distancia.setText("16");
  Peso_peso.setNextFocusableComponent(Peso_tamanyo);
  Peso_peso.setText("16");
  Peso_tamanyo.setNextFocusableComponent(Peso_antigenos);
  Peso_tamanyo.setText("16");
  Peso_antigenos.setNextFocusableComponent(Peso_edad);
  Peso_antigenos.setText("16");
  Peso_edad.setNextFocusableComponent(Peso_espera);
  Peso_edad.setToolTipText("");
  Peso_edad.setText("18");
  Peso_espera.setNextFocusableComponent(Grado);
  Peso_espera.setText("18");
  jScrollPane1.setAutoscrolls(true);
  jScrollPane1.setBorder(null);
  Dia.setNextFocusableComponent(Mes);
  Dia.setToolTipText("Día");
  Mes.setNextFocusableComponent(Anyo);
  Mes.setToolTipText("Mes");
  Anyo.setNextFocusableComponent(Texto_hospital);
  Anyo.setToolTipText("Año");
  Texto_hospital.setNextFocusableComponent(Texto_ciudad);
  Texto_hospital.setToolTipText("Hospital del donante");
  Texto_ciudad.setNextFocusableComponent(Texto_provincia);
  Texto_ciudad.setToolTipText("Localidad del donante");
  Texto_provincia.setNextFocusableComponent(Texto_peso);
  Texto_provincia.setToolTipText("Provincia del donante");
```

```

        Texto_peso.setNextFocusableComponent(Texto_tamanyo);
        Texto_peso.setToolTipText("Peso del paciente (Kg)");
        Texto_tamanyo.setNextFocusableComponent(b0);
        Texto_tamanyo.setToolTipText("Tamaño del órgano");
        Tipo_sangre.setNextFocusableComponent(Peso_distancia);
        Tipo_sangre.setToolTipText("Tipo de sangre");
        Limpia_BD.setNextFocusableComponent(Tipo_organo);
        Limpia_BD.setText("Limpiar las BD's");
        Limpia_BD.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                Limpia_BD_mouseClicked(e);
            }
        });
        this.getContentPane().add(Tipo_organo, new XYConstraints(102, 19, 102,
19));
        this.getContentPane().add(jLabel7, new XYConstraints(57, 19, -1, -1));
        this.getContentPane().add(Texto_hospital, new XYConstraints(102, 55, 148,
-1));
        this.getContentPane().add(jLabel1, new XYConstraints(51, 57, -1, -1));
        this.getContentPane().add(Texto_ciudad, new XYConstraints(102, 89, 148, -
1));
        this.getContentPane().add(Texto_provincia, new XYConstraints(102, 122,
148, -1));
        this.getContentPane().add(jLabel2, new XYConstraints(42, 91, -1, -1));
        this.getContentPane().add(jLabel3, new XYConstraints(44, 124, -1, -1));
        this.getContentPane().add(Texto_peso, new XYConstraints(102, 158, 148, -
1));
        this.getContentPane().add(Texto_tamanyo, new XYConstraints(102, 194, 148,
-1));
        this.getContentPane().add(jLabel11, new XYConstraints(222, 20, -1, -1));
        this.getContentPane().add(jLabel5, new XYConstraints(8, 196, -1, -1));
        this.getContentPane().add(jLabel4, new XYConstraints(17, 160, -1, -1));
        this.getContentPane().add(jLabel12, new XYConstraints(49, 233, -1, -1));
        this.getContentPane().add(Ok, new XYConstraints(193, 251, -1, -1));
        this.getContentPane().add(Anyo, new XYConstraints(431, 19, 40, -1));
        this.getContentPane().add(Mes, new XYConstraints(386, 19, 31, -1));
        this.getContentPane().add(jLabel9, new XYConstraints(377, 20, -1, -1));
        this.getContentPane().add(Dia, new XYConstraints(347, 19, 26, -1));
        this.getContentPane().add(jLabel10, new XYConstraints(422, 21, -1, -1));
        this.getContentPane().add(dr1, new XYConstraints(275, 146, 57, -1));
        this.getContentPane().add(dr0, new XYConstraints(275, 121, 57, -1));
        this.getContentPane().add(b1, new XYConstraints(275, 97, 57, -1));
        this.getContentPane().add(b0, new XYConstraints(275, 73, 57, -1));
        this.getContentPane().add(jLabel14, new XYConstraints(378, 87, -1, -1));
        this.getContentPane().add(Peso_distancia, new XYConstraints(443, 86, 33, -
1));
        this.getContentPane().add(jLabel15, new XYConstraints(484, 88, -1, -1));
        this.getContentPane().add(Peso_peso, new XYConstraints(443, 113, 33, -1));
        this.getContentPane().add(jLabel16, new XYConstraints(484, 115, -1, -1));
        this.getContentPane().add(jLabel17, new XYConstraints(378, 114, -1, -1));
        this.getContentPane().add(Peso_tamanyo, new XYConstraints(443, 140, 33, -
1));
        this.getContentPane().add(jLabel19, new XYConstraints(378, 141, -1, -1));
        this.getContentPane().add(jLabel18, new XYConstraints(484, 142, -1, -1));
        this.getContentPane().add(Peso_antigenos, new XYConstraints(443, 167, 33,
-1));
        this.getContentPane().add(jLabel111, new XYConstraints(378, 168, -1, -1));
        this.getContentPane().add(jLabel110, new XYConstraints(484, 169, -1, -1));
        this.getContentPane().add(Peso_edad, new XYConstraints(443, 195, 33, -1));
        this.getContentPane().add(jLabel115, new XYConstraints(378, 196, -1, -1));
        this.getContentPane().add(jLabel114, new XYConstraints(484, 197, -1, -1));
        this.getContentPane().add(Peso_espera, new XYConstraints(443, 223, 33, -
1));
        this.getContentPane().add(jLabel112, new XYConstraints(378, 224, -1, -1));
        this.getContentPane().add(jLabel113, new XYConstraints(484, 225, -1, -1));
        this.getContentPane().add(Grado, new XYConstraints(44, 257, 121, 19));
        this.getContentPane().add(Tipo_sangre, new XYConstraints(271, 212, 50,
19));
    
```

```

        this.getContentPane().add(jLabel8, new XYConstraints(275, 191, -1, -1));
        this.getContentPane().add(jPanel1, new XYConstraints(364, 63, 150, 196));
        this.getContentPane().add(jPanel2, new XYConstraints(263, 54, 76, 120));
        this.getContentPane().add(jPanel3, new XYConstraints(5, 285, 524, 236));
        jPanel3.add(jScrollPane1, new XYConstraints(4, 0, 502, 203));
        this.getContentPane().add(Limpia_BD, new XYConstraints(366, 264, 146,
22));
        jScrollPane1.setViewportViewView (Resultado);

        Tipo_organo.addItem("Corazón");
        Tipo_organo.addItem("Higado");
        Tipo_organo.addItem("Riñón");
        Tipo_organo.addItem("Pancreas");

        Tipo_sangre.addItem("0");
        Tipo_sangre.addItem("A");
        Tipo_sangre.addItem("B");
        Tipo_sangre.addItem("AB");

/* String grados[] = {"pessim", "fatal", "molt_dolent", "dolent",
"bastant_dolent",
"desfavorable", "fluix", "factible", "adequat", "favorable", "bastant_bo",
"bo",
"molt_bo", "excel_lent", "optim"}; */

        Grado.addItem("Óptimo");
        Grado.addItem("Excelente");
        Grado.addItem("Muy Bueno");
        Grado.addItem("Bueno");
        Grado.addItem("Bastante Bueno");
        Grado.addItem("Favorable");
        Grado.addItem("Adecuado");
        Grado.addItem("Factible");
        Grado.addItem("Flojo");
        Grado.addItem("Desfavorable");
        Grado.addItem("Bastante Malo");
        Grado.addItem("Malo");
        Grado.addItem("Muy Malo");
        Grado.addItem("Fatal");
        Grado.addItem("Pésimo");

        Texto_hospital.setText(hospital.getLocalName());
        Texto_ciudad.setText(hospital.ciudad);
        Texto_provincia.setText(hospital.provincia);
        Texto_peso.setText("70.4");
        Texto_tamanyo.setText("34.2");

        dr0.setSelected(true);

        Dia.setText("10");
        Mes.setText("10");
        Anyo.setText("1960");

        Resultado.setAutoscrolls(true);
    }

    public void actionPerformed(ActionEvent e) {
    }

    public boolean datos_correctos() {
        if ((Texto_hospital.getText().equals(""))
            || (Texto_hospital.getText().equals(""))
            || (Texto_ciudad.getText().equals(""))
            || (Texto_provincia.getText().equals(""))
            || (Texto_peso.getText().equals(""))
            || (Texto_tamanyo.getText().equals(""))
            || (Peso_antigenos.getText().equals(""))
            || (Peso_distancia.getText().equals(""))

```

```

        || (Peso_edad.getText().equals(""))
        || (Peso_espera.getText().equals(""))
        || (Peso_peso.getText().equals(""))
        || (Peso_tamanyo.getText().equals(""))
        || (Anyo.getText().equals(""))
        || (Dia.getText().equals(""))
        || (Mes.getText().equals("")) ) {
    Resultado.append("\n Rellena todos los campos, por favor");
    return false;
}
if ((new Integer(Anyo.getText()).intValue()>2002) || (new
Integer(Anyo.getText()).intValue()<1900)) {
    Resultado.append("\n Revisa el año de nacimiento");
    return false;
}
if ((new Integer(Dia.getText()).intValue()>31) || (new
Integer(Dia.getText()).intValue()<1)) {
    Resultado.append("\n Revisa el día de nacimiento");
    return false;
}
if ((new Integer(Mes.getText()).intValue()>12) || (new
Integer(Mes.getText()).intValue()<1)){
    Resultado.append("\n Revisa el mes de nacimiento");
    return false;
}
if (new Integer(Peso_distancia.getText()).intValue()
    + new Integer(Peso_antigenos.getText()).intValue()
    + new Integer(Peso_edad.getText()).intValue()
    + new Integer(Peso_peso.getText()).intValue()
    + new Integer(Peso_tamanyo.getText()).intValue()
    + new Integer(Peso_espera.getText()).intValue() != 100) {
    Resultado.append("\n Los pesos han de sumar el 100%");
    return false;
}
else
    return true;
}

void Ok_mouseClicked(MouseEvent e) {
    if (Ok.isEnabled()) {
        if (datos_correctos()) {
            Ok.setEnabled(false);
            Limpia_BD.setEnabled(false);
            java.util.Calendar fecha = java.util.Calendar.getInstance();
            fecha.set(new Integer(Anyo.getText()).intValue(),new
Integer(Mes.getText()).intValue()-1,
                new Integer(Dia.getText()).intValue());
            Long a0 = new Long(0);
            Long a1 = new Long(0);
            Long a2 = new Long(0);
            Long a3 = new Long(0);

            if (b0.isSelected())
                a0 = new Long(1);
            if (b1.isSelected())
                a1 = new Long(1);
            if (dr0.isSelected())
                a2 = new Long(1);
            if (dr1.isSelected())
                a3 = new Long(1);

            String organo = "cor";
            if (Tipo_organos.getSelectedIndex()==1)
                organo="fetge";
            if (Tipo_organos.getSelectedIndex()==2)
                organo="ronyo";
            if (Tipo_organos.getSelectedIndex()==3)
                organo="pancrees";
        }
    }
}

```

```
Paciente donante = new Paciente(new Long(0),organo,
Texto_hospital.getText(),Texto_ciudad.getText(),
                                Texto_provincia.getText(),new
Long(Tipo_sangre.getSelectedIndex()),
                                new Double(Texto_peso.getText()),
                                new Double(Texto_tamanyo.getText()),
                                a0,a1,a2,a3,fecha.getTime(),fecha.getTime());

Resultado.append("\n\nÓRGANO DONADO: " + donante.getorgano());
Resultado.append("\nID DONANTE: " + donante.getid());
Resultado.append("\nHOSPITAL DONANTE: " + donante.gethospital());
Resultado.append("\nLOCALIDAD DONANTE: " + donante.getlocalidad());
Resultado.append("\nPROVINCIA DONANTE: " + donante.getprovincia());
Resultado.append("\nFECHA DE NACIMIENTO DONANTE: " +
donante.getnacimiento() + "\n");

hospital.distancia = new Integer(Peso_distancia.getText()).intValue();
hospital.antigenos = new Integer(Peso_antigenos.getText()).intValue();
hospital.edad = new Integer(Peso_edad.getText()).intValue();
hospital.espera = new Integer(Peso_espera.getText()).intValue();
hospital.peso = new Integer(Peso_peso.getText()).intValue();
hospital.tamanyo = new Integer(Peso_tamanyo.getText()).intValue();

hospital.postGuiEvent(new
jade.gui.GuiEvent(donante,hospital.PETICION));
    }
}

void Limpia_BD_mouseClicked(MouseEvent e) {
    if (Limpia_BD.isEnabled()) {
        Limpia_BD.setEnabled(false);
        Ok.setEnabled(false);
        hospital.postGuiEvent(new
jade.gui.GuiEvent(Interficie.this,hospital.LIMPIA_BD));
    }
}
```

8.6. Entrada_Datos.java

```

/*****
**
Entrada_Datos

PFC Jaime Bocio Sanz
28-8-2002

Clase que implementa la interficie gráfica que servirá para que un usuario
introduzca los datos necesarios de un Hospital o Coordinador.

*****/

/

package pfc_ei;

import java.awt.*;
import com.borland.jbcl.layout.*;
import javax.swing.*;
import java.awt.event.*;

public class Entrada_Datos extends JFrame {

    private Hospital hospital;

    XYLayout xYLayout1 = new XYLayout();
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JTextField ciudad = new JTextField();
    JTextField bd = new JTextField();
    JButton ok = new JButton();
    JButton cancel = new JButton();
    JLabel jLabel5 = new JLabel();
    JTextField provincia = new JTextField();
    JLabel jLabel6 = new JLabel();
    JButton registro = new JButton();

    public Entrada_Datos(Hospital h) {
        this.hospital = h;
        try {
            jbInit();
            this.setSize(440,300);
            this.setLocation(300,200);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        ok.setEnabled(false);
        ok.setText("Abrir la Interficie");
        ok.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                ok_mouseClicked(e);
            }
        });
        jLabel1.setText("Ciudad del Hospital");
        this.getContentPane().setBackground(Color.lightGray);
        this.setDefaultCloseOperation(3);
        this.setResizable(false);
        this.setTitle("Inicialización");
        this.getContentPane().setLayout(xYLayout1);
        jLabel2.setText("Nombre de la BD");
        cancel.setText("Cancelar");
        cancel.addMouseListener(new java.awt.event.MouseAdapter() {

```

```
        public void mouseClicked(MouseEvent e) {
            cancel_mouseClicked(e);
        }
    });
    jLabel5.setForeground(Color.gray);
    jLabel5.setText("Para no iniciar la Interfice pulsa el botón de
Registrarse");
    XYLayout1.setWidth(440);
    XYLayout1.setHeight(274);
    jLabel6.setText("Provincia del Hospital");
    registro.setText("Registrarse");
    registro.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            registro_mouseClicked(e);
        }
    });
    this.getContentPane().add(jLabel1, new XYConstraints(62, 37, -1, -1));
    this.getContentPane().add(provincia, new XYConstraints(181, 74, 197, -1));
    this.getContentPane().add(ciudad, new XYConstraints(181, 35, 197, -1));
    this.getContentPane().add(jLabel2, new XYConstraints(76, 115, -1, -1));
    this.getContentPane().add(bd, new XYConstraints(181, 113, 197, -1));
    this.getContentPane().add(jLabel6, new XYConstraints(52, 76, -1, -1));
    this.getContentPane().add(cancel, new XYConstraints(231, 153, 150, 23));
    this.getContentPane().add(ok, new XYConstraints(103, 220, 227, 23));
    this.getContentPane().add(registro, new XYConstraints(56, 153,
151, 23));
    this.getContentPane().add(jLabel5, new XYConstraints(64, 192, 310, -1));

    }

    void cancel_mouseClicked(MouseEvent e) {
        java.lang.System.exit(0);
    }

    void ok_mouseClicked(MouseEvent e) {
        hospital.postGuiEvent(new
jade.gui.GuiEvent(Entrada_Datos.this,hospital.INTERFICIE));
    }

    void registro_mouseClicked(MouseEvent e) {
        hospital.postGuiEvent(new
jade.gui.GuiEvent(Entrada_Datos.this,hospital.REGISTRO));
    }
}
```

8.7. Historico.java

```

/*****
**
Historico

PFC Jaime Bocio Sanz
2-5-2002

Agente que representa al Agente Histórico. Es el encargado de recibir los
datos
de una donación (paciente, donante, fecha...) de parte del hospital de origen
y meterlos en una base de datos para que se tenga constancia de ello.

*****/
/

package pfc_ei;

import jade.lang.acl.*;
import jade.core.behaviours.*;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.*;
import jade.core.*;
import jade.lang.sl.sl.*;
import jade.onto.sl.sl.*;
import jade.proto.FipaRequestResponderBehaviour;

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

import jade.core.Agent;

//Paquete con la ontología definida
import pfc_ei.ontology.*;

/*****
*/

public class Historico extends Agent {

    String bd = "BD-AH"; //BD donde AH guardará los datos de las donaciones

/*****
*/

    public Historico() {
        super();
    }

/*****
*/

/* FUNCIÓN QUE REGISTRA A AH EN EL DF. */
    public void Registro_en_el_df() {
        //Registro al DF
        ServiceDescription sd1=new ServiceDescription();
        //Indicamos los parametros del Agente
        sd1.setType("Historico");
        sd1.setName(getLocalName());
        sd1.setOwnership("AH");
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        //Nos añadimos como un nuevo servicio
        dfd.addServices(sd1);
    }
}

```

```

        //Indicamos nuestro nombre
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);
//Registramos el lenguaje
registerLanguage(SLCodec.NAME, new SLCodec());
//Registro de la ontología
registerOntology(JBSOntology.NAME, JBSOntology.instance());
        System.out.println("Agente Historico (" + getLocalName() + ") on-
line");

        try{
            //Nos registramos al DF
            DFServiceCommunicator.register(this,dfd);
        }catch (FIPAException e) {
            System.err.println(getLocalName()+" Error en el registro
al DF. Error: "+e.getMessage());
            doDelete();
        }
    }

    /*****
    */

    /* FUNCIÓN QUE DESREGISTRA A UN AGENTE DEL DF. SE USA EN EL "takeDown" DEL
    AGENTE. */
    public void Desregistro_del_df() {
        //Obtenemos una instancia del DF
        DFAgentDescription dfd=new DFAgentDescription();
        AID aid=new AID();
        aid.setName(getLocalName());
        dfd.setName(aid);

        try{
            //Nos desregistramos
            DFServiceCommunicator.deregister(this,dfd);
            System.out.println("Agente " + getLocalName() + " eliminado
del DF");
        }catch (FIPAException e) {
            System.out.println(getLocalName()+" Error en el
desregistro del DF. Error: "+e.getMessage());
        }
    }

    /*****
    */

    /* FUNCIÓN QUE PREPARA UN MENSAJE. RELLENA DE MANERA FÁCIL Y RÁPIDA LOS
    CAMPOS
    DE DESTINO, TIPO, Y CONTENIDO CON LOS PARÁMETROS PASADOS. LOS DE LENGUAJE
    Y
    ONTOLOGÍA LOS PONE POR DEFECTO, AL IGUAL QUE EL REMITENTE */
    public ACLMessage Prepara_msg(int tipo_mensaje, AID receiver, String set0,
    Object contenido) {
        ACLMessage msg = new ACLMessage(tipo_mensaje);
        msg.addReceiver(receiver);
        msg.setSender(new AID(getLocalName()));
        msg.setLanguage(SLCodec.NAME);
        msg.setOntology(JBSOntology.NAME);
        SL_Action a = new SL_Action();
        a.set_0(new AID(set0));
        a.set_1(contenido);
        List l = new ArrayList();
        l.add(a);
        try {
            fillContent(msg,l);
        } catch (Exception e) {
            System.out.println(getLocalName() + ": " + e.toString());

```

```

    }
    return (msg);
}

/*****
*/

/* FUNCIÓN QUE EXTRAER EL CONTENIDO DE UN MENSAJE. FACILITA LA TAREA. SE LE
PASA
EL MENSAJE DEL CUAL QUEREMOS EXTRAER EL CONTENIDO, Y SE DEVUELVE EL OBJETO
DE
LA ONTOLOGÍA QUE CONTIENE. */
public Object Extrae_contenido(ACLMessage msg) {
    List l = new ArrayList();
    SL_Action a = new SL_Action();
    Object resultado = new Object();
    try {
        l= extractContent(msg);
        a = (SL_Action) l.get(0);
        resultado = (Object) a.get_1();
    } catch (Exception e) {
        System.out.println(getLocalName() + ": " + e.toString());
    }
    return resultado;
}

/*****
*/

/* FUNCIÓN QUE AÑADE UN REGISTRO A LA BD DEL AH. SE INCLUYEN LOS DATOS DEL
DONANTE, DEL PACIENTE Y LA FECHA DE LA DONACIÓN. SE RETORNA true EN CASO
DE
QUE TODO HAYA IDO BIEN. */
public boolean Inserta_en_BD(Paciente donante, Paciente paciente, String
bd1) {
    Connection con;
    Statement stmt;
    ResultSet rs = null;

    java.util.Date fecha = new java.util.Date();
    java.sql.Date fecha_sql = new java.sql.Date(fecha.getTime());

    try { // Conexión a la BD
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:" + bd1);
        stmt = con.createStatement();
    } catch (Exception e){
        System.out.println(bd1 + ": " + e.toString());
        return false;
    }

    try {
        String insercion = "Insert into Donaciones Values ('" +
donante.getorgano()
                                + "','" + fecha_sql + "','" + donante.gethospital()
                                + "','" + donante.getlocalidad() + "','" +
donante.getprovincia()
                                + "','" + donante.getsangre().toString() + "','" +
donante.getpeso()
                                + "','" + donante.gettamanyo() + "','" +
donante.getb0().toString()
                                + "','" + donante.getb1().toString() + "','" +
donante.getdr0().toString()
                                + "','" + donante.getdr1().toString()
                                + "','" + new
java.sql.Date(donante.getnacimiento().getTime())
                                + "','" + paciente.getid().intValue() + "','" +
paciente.gethospital()

```

```

        + "','\" + paciente.getlocalidad() + "','\" +
paciente.getprovincia()
        + "','\" + paciente.getsangre().toString() + "','\" +
paciente.getpeso()
        + "','\" + paciente.gettamanyo() + "','\" +
paciente.getb0().toString()
        + "','\" + paciente.getb1().toString() + "','\" +
paciente.getdr0().toString()
        + "','\" + paciente.getdr1().toString()
        + "','\" + new
java.sql.Date(paciente.getnacimiento().getTime())
        + "','\" + new
java.sql.Date(paciente.getespera().getTime()) + "\"");

        stmt.executeUpdate(insertcion);
        con.close();
        stmt.close();
        return true;
    }catch (SQLException e){
        System.out.println(bd1 + ": \" + e.toString());
        return false;
    }
}

/*****
*/

/* COMPORTAMIENTO QUE RESPONDE A UN REQUEST PROCEDENTE DE UN HOSPITAL. */
public class H_Request extends FipaRequestResponderBehaviour.ActionHandler
    implements FipaRequestResponderBehaviour.Factory {

    Lista_objetos datos = new Lista_objetos();

    public H_Request(Agent a, ACLMessage msg){
        super(a,msg);
    }

    public FipaRequestResponderBehaviour.ActionHandler create(ACLMessage msg){
        return new H_Request(myAgent, msg);
    }

    public void action(){
        ACLMessage msg_req = getRequest();
        System.out.println(getLocalName() + ": Iniciado Request con \" +
msg_req.getSender());
        datos = (Lista_objetos) Extrae_contenido(msg_req);

        /** Se han recibido los datos de la donación. Los tenemos en la
        * variable "datos". Tenemos los datos del donante en la posición 0 y
los
        * del paciente escogido en la posición 1. La fecha de la donación la
        * cogeremos desde aquí, y no creo que se necesiten más detalles
        */

        Iterator it = datos.getAllpacientes();
        Paciente donante = (Paciente) it.next();
        Paciente paciente = (Paciente) it.next();
        if (Inserta_en_BD(donante,paciente,bd))
            System.out.println(getLocalName() + ": BD actualizada correctamente");
        else
            System.out.println(getLocalName() + ": No se ha podido actualizar la
BD");

    }

    public boolean done() {
        return true;
    }
}

```

```
        public void reset() {
        }
    }

    public class miFipaRequestResponderBehaviour extends
    FipaRequestResponderBehaviour {
        public miFipaRequestResponderBehaviour(Agent a) {
            super(a);
        }
        protected String getActionName(ACLMessage msg) throws
        NotUnderstoodException, RefuseException {
            return JBSontology.LISTA_OBJETOS;
        }
    }

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

    /* SETUP DEL AGENTE */

    protected void setup () {
        bd = "BD-AH";

        Registro_en_el_df();

        FipaRequestResponderBehaviour requester = new
        miFipaRequestResponderBehaviour(Historico.this);
        requester.registerFactory(JBSontology.LISTA_OBJETOS, new
        H_Request(Historico.this, null));
        addBehaviour(requester);
    }

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

    /* TAKEDOWN DEL AGENTE */

    protected void takeDown(){
        Desregistro_del_df();
    }
}
```

8.8. El Paquete Ontology

8.8.1. pfc_ei/JBSOntology.java

```
/*
ontology.JBSOntology

PFC Jaime Bocio Sanz
10-5-2002

Definición de la ontología: datos que han de recibir y enviar los agentes
*****
/

package pfc_ei.ontology;

import java.io.*;
import jade.onto.*;

import jade.onto.sl.sl.*;
import jade.onto.extensions.*;

//Definición de la ontología: datos que han de recibir y enviar los agentes
public class JBSOntology
{

//Nombre de la ontología
public static final String NAME="JBS-ontology";

//Conceptos
public static final String ORGANO="organo";
public static final String LISTA_OBJETOS="lista-objetos";
public static final String ATRIB_PACIENTES="atrib-pacientes";
public static final String VALOR_ATRIB="valor-atrib";
public static final String RESULTADO="resultado";
public static final String GRADO="grado";
public static final String ELECCION="eleccion";
public static final String PACIENTE="paciente";

//Acciones

//Errores
public static final String AGENTE_NO_REGISTRADO="Agente-no-registrado";
public static final String ORDENACION_IMPOSIBLE="Ordenacion-imposible";
public static final String NO_HOSPITALES="No-hospitales";
public static final String NO_PACIENTES="No-pacientes";

//Predicados

//Instancia de la ontología para referenciar desde el agente
private static Ontology theInstance = new SL_DefaultOntologyImp();
public static OntologyHelper onto_helper = new OntologyHelper();

static{
    initInstance();
}

//Para consultar los conceptos de la ontología
public static Ontology instance()
{
    return theInstance;
}
```

```
//Constructor
public JBSOntology(){
}

//Inicializa la ontología
private static void initInstance(){
    try{

        //Objeto Organo: contiene el tipo de órgano
        theInstance.joinOntology(SL_Ontology.instance());
        theInstance.addRole(
            ORGANO,
            new SlotDescriptor[]{
                new SlotDescriptor("tipo-
organo",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontology.M),
            },
            new RoleEntityFactory(){
                public Object create (Frame f){
                    return new Organo();
                }
                public Class getClassForRole(){
                    return Organo.class;
                }
            }
        );

        //Objeto Lista_Pacientes: contiene una lista de pacientes y el número
        theInstance.addRole(
            LISTA_OBJETOS,
            new SlotDescriptor[]{
                //Slots que pertenecen al objeto
                new
                SlotDescriptor("pacientes",Ontology.SET_SLOT,TRIB_PACIENTES,Ontology.M),
                new
                SlotDescriptor("pacientes",Ontology.SET_SLOT,PACIENTE,Ontology.M),
                new SlotDescriptor("pesos",Ontology.SET_SLOT,VALOR_trib,Ontology.M),
                new
                SlotDescriptor("npacientes",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontolog
y.M),
            },
            new RoleEntityFactory(){
                //Constructor de l'objecte
                public Object create (Frame f){
                    return new Lista_objetos();
                }
                //Classe associada
                public Class getClassForRole(){
                    return Lista_objetos.class;
                }
            }
        );

        //Objeto Trib_Pacientes: contiene una lista de pacientes, con un id para
        cada uno
        theInstance.addRole(
            TRIB_PACIENTES,
            new SlotDescriptor[]{
                //Slots que pertenecen al objeto
                new
                SlotDescriptor("valor_pac",Ontology.SET_SLOT,VALOR_trib,Ontology.M),
                new
                SlotDescriptor("pac",Ontology.SET_SLOT,PACIENTE,Ontology.M),
                new
                SlotDescriptor("id_pac",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M)
            },
            new RoleEntityFactory(){
                //Constructor de l'objecte
```

```
        public Object create (Frame f){
            return new Atrib_pacientes();
        }
        //Classe associada
        public Class getClassForRole(){
            return Atrib_pacientes.class;
        }
    }
};

//Objeto Paciente: contiene todos los datos de un paciente
theInstance.addRole(
    PACIENTE,
    new SlotDescriptor[]{
        new
SlotDescriptor("id",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
        new
SlotDescriptor("hospital",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontolog
y.M),
        new
SlotDescriptor("localidad",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontolo
gy.M),
        new
SlotDescriptor("provincia",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontolo
gy.M),
        new
SlotDescriptor("sangre",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M)
,
        new
SlotDescriptor("organo",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontology.
M),
        new
SlotDescriptor("peso",Ontology.PRIMITIVE_SLOT,Ontology.DOUBLE_TYPE,Ontology.M)
,
        new
SlotDescriptor("tamanyo",Ontology.PRIMITIVE_SLOT,Ontology.DOUBLE_TYPE,Ontology
.M),
        new
SlotDescriptor("b0",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
        new
SlotDescriptor("b1",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
        new
SlotDescriptor("dr0",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
        new
SlotDescriptor("dr1",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
        new
SlotDescriptor("nacimiento",Ontology.PRIMITIVE_SLOT,Ontology.DATE_TYPE,Ontolog
y.M),
        new
SlotDescriptor("espera",Ontology.PRIMITIVE_SLOT,Ontology.DATE_TYPE,Ontology.M)
,
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Paciente();
        }
        public Class getClassForRole(){
            return Paciente.class;
        }
    }
);

//Objeto Valor_atrib: contiene un valor y un peso para cada
atributo
theInstance.addRole(
    VALOR_ATRIB,
    new SlotDescriptor[]{
```

```
new
SlotDescriptor("peso",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
new
SlotDescriptor("valor_string",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontology.M),
/* new
SlotDescriptor("valor_int",Ontology.PRIMITIVE_SLOT,Ontology.ANY_TYPE,Ontology.M),
new
SlotDescriptor("valor_double",Ontology.PRIMITIVE_SLOT,Ontology.DOUBLE_TYPE,Ontology.M),
*/
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Valor_atrib();
        }
        public Class getClassForRole(){
            return Valor_atrib.class;
        }
    }
);

//Objeto Resultado: contiene una lista de objetos y el número de
ellos
theInstance.addRole(
    RESULTADO,
    new SlotDescriptor[]{
        new
SlotDescriptor("asignacion",Ontology.SET_SLOT,GRADO,Ontology.M),
        new
SlotDescriptor("nobjetos",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Resultado();
        }
        public Class getClassForRole(){
            return Resultado.class;
        }
    }
);

//Objeto Grado (de adecuación al modelo): contiene un
identificador de paciente y su valor
theInstance.addRole(
    GRADO,
    new SlotDescriptor[]{
        new
SlotDescriptor("valor",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Ontology.M),
        new SlotDescriptor("id-
objeto",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontology.M),
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Grado();
        }
        public Class getClassForRole(){
            return Grado.class;
        }
    }
);

//acciones
//Objeto Eleccion: contiene un id de paciente y un id de hospital
theInstance.addRole(
    ELECCION,
```

```
        new SlotDescriptor[]{
            new
SlotDescriptor("id_hospital",Ontology.PRIMITIVE_SLOT,Ontology.STRING_TYPE,Onto
logy.M),
            new
SlotDescriptor("id_paciente",Ontology.PRIMITIVE_SLOT,Ontology.LONG_TYPE,Ontolo
gy.M),
        },
        new RoleEntityFactory(){
            public Object create (Frame f){
                return new Eleccion();
            }
            public Class getClassForRole(){
                return Eleccion.class;
            }
        }
    );

//Objeto que indica un error porque un coordinador no está
registrado
theInstance.addRole(
    AGENTE_NO_REGISTRADO,
    new SlotDescriptor[]{
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Agente_no_registrado();
        }
        public Class getClassForRole(){
            return Agente_no_registrado.class;
        }
    }
);

//Objeto que indica un error porque la ordenación no es posible,
por cualquier motivo
theInstance.addRole(
    ORDENACION_IMPOSIBLE,
    new SlotDescriptor[]{
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new Ordenacion_imposible();
        }
        public Class getClassForRole(){
            return Ordenacion_imposible.class;
        }
    }
);

//Objeto que indica un error porque faltan hospitales
theInstance.addRole(
    NO_HOSPITALES,
    new SlotDescriptor[]{
    },
    new RoleEntityFactory(){
        public Object create (Frame f){
            return new No_hospitales();
        }
        public Class getClassForRole(){
            return No_hospitales.class;
        }
    }
);

//Objeto que indica un error porque faltan pacientes
theInstance.addRole(
    NO_PACIENTES,
```

```
        new SlotDescriptor[]{
        },
        new RoleEntityFactory(){
            public Object create (Frame f){
                return new No_pacientes();
            }
            public Class getClassForRole(){
                return No_pacientes.class;
            }
        }
    );

} catch (OntologyException e){
    System.err.println("Error generant la Ontologia");
    e.printStackTrace();
}

}

}
```

8.8.2. pfc_ei/Lista_objetos.java

```

/*****
**
ontology.Lista_objetos

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Lista_objetos definido en la ontología
*****/

/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;
import java.util.*;

public class Lista_objetos extends Domain_OntologyObj
{
    //Variables
    private List pacientes = new ArrayList();
    private List pesos = new ArrayList();
    private Long npacientes;

    //Construtor
    public Lista_objetos() {
        super("Lista-objetos");
        npacientes=new Long(0);
        addpesos(new Valor_atrib("Distancia",new Long(16)));
        addpesos(new Valor_atrib("Pes",new Long(16)));
        addpesos(new Valor_atrib("Tamany",new Long(16)));
        addpesos(new Valor_atrib("Antigens",new Long(16)));
        addpesos(new Valor_atrib("Edad",new Long(18)));
        addpesos(new Valor_atrib("Espera",new Long(18)));
    }

    public Lista_objetos(Long n) {
        super("Lista-objetos");
        npacientes=n;
        addpesos(new Valor_atrib("Distancia",new Long(16)));
        addpesos(new Valor_atrib("Pes",new Long(16)));
        addpesos(new Valor_atrib("Tamany",new Long(16)));
        addpesos(new Valor_atrib("Antigens",new Long(16)));
        addpesos(new Valor_atrib("Edad",new Long(18)));
        addpesos(new Valor_atrib("Espera",new Long(18)));
    }

    //Métodos SET
    public void addpacientes (Object atrib)
    {
        (this.pacientes).add(atrib);
    }
    public void clearpacientes ()
    {
        pacientes=new ArrayList();
    }
    public void addpesos (Object atrib)
    {
        (this.pesos).add(atrib);
    }
    public void clearpesos()
    {
        pesos=new ArrayList();
    }
}

```

```
    public void setnpacientes (Long n)
    {
        this.npacientes=n;
    }

    //Métodos GET
    public Iterator getAllpacientes()
    {
        return(this.pacientes).iterator();
    }
    public Iterator getAllpesos()
    {
        return(this.pesos).iterator();
    }
    public Long getnpacientes()
    {
        return this.npacientes;
    }
}
```

8.8.3. pfc_ei/Paciente.java

```

/*****
**
ontology.Paciente

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Paciente definido en la ontología
*****/
/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;
import java.util.Date;

public class Paciente extends Domain_OntologyObj
{
    //Variables
    private Long id;
    private String organo=new String("");
    private String hospital=new String("");
    private String localidad=new String("");
    private String provincia=new String("");
    private Long sangre;
    private Double peso;
    private Double tamanyo;
    private Long b0;
    private Long b1;
    private Long dr0;
    private Long dr1;
    private Date nacimiento;
    private Date espera;

    //Constructor
    public Paciente() {
        super("Paciente");
        id=new Long(0);
        organo="no_inicializado";
        hospital="no_inicializado";
        localidad="no_inicializado";
        provincia="no_inicializado";
        sangre=new Long(0);
        peso=new Double(0.0);
        tamanyo=new Double(0.0);
        b0=new Long(0);
        b1=new Long(0);
        dr0=new Long(0);
        dr1=new Long(0);
        nacimiento=new Date();
        espera=new Date();
    }

    public Paciente(Long num, String org, String hosp, String loc, String prov,
Long sang, Double pes, Double tam, Long a0, Long a1, Long a2, Long a3, Date
nac, Date esp) {
        super("Paciente");
        id=num;
        organo=org;
        hospital=hosp;
        localidad=loc;
        provincia=prov;
        sangre=sang;
        peso=pes;
    }
}

```

```
tamanyo=tam;
b0=a0;
b1=a1;
dr0=a2;
dr1=a3;
nacimiento=nac;
espera=esp;
}

//Metodos SET
public void setid (Long s) {
    this.id=s;
}
public void setorgano (String s) {
    this.organo=s;
}
public void sethospital (String s) {
    this.hospital=s;
}
public void setlocalidad (String s) {
    this.localidad=s;
}
public void setprovincia (String s) {
    this.provincia=s;
}
public void setsangre (Long s) {
    this.sangre=s;
}
public void setpeso (Double s) {
    this.peso=s;
}
public void settamanyo (Double s) {
    this.tamanyo=s;
}
public void setb0 (Long s) {
    this.b0=s;
}
public void setb1 (Long s) {
    this.b1=s;
}
public void setdr0 (Long s) {
    this.dr0=s;
}
public void setdr1 (Long s) {
    this.dr1=s;
}
public void setnacimiento (Date s) {
    this.nacimiento=s;
}
public void setespera (Date s) {
    this.espera=s;
}
//Metodos GET
public Long getid() {
    return this.id;
}
public String getorgano() {
    return this.organo;
}
public String gethospital() {
    return this.hospital;
}
public String getlocalidad() {
    return this.localidad;
}
public String getprovincia() {
    return this.provincia;
}
}
```

```
public Long getsangre() {
    return this.sangre;
}
public Double getpeso() {
    return this.peso;
}
public Double gettamanyo() {
    return this.tamanyo;
}
public Long getb0() {
    return this.b0;
}
public Long getb1() {
    return this.b1;
}
public Long getdr0() {
    return this.dr0;
}
public Long getdr1() {
    return this.dr1;
}
public Date getnacimiento() {
    return this.nacimiento;
}
public Date getespera() {
    return this.espera;
}
}
```

8.8.4. pfc_ei/Organo.java

```

/*****
**
ontology.Organo

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Organo definido en la ontología
*****/
/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class Organo extends Domain_OntologyObj
{
    //Variables
    private String tipo_organo=new String("");

    //Constructor
    public Organo() {
        super("Organo");
    }

    public Organo(String tipo) {
        super("Organo");
        tipo_organo=tipo;
    }

    //Metodos SET
    public void settipoOrgano (String s)
    {
        this.tipo_organo=s;
    }

    //Metodos GET
    public String gettipoOrgano()
    {
        return this.tipo_organo;
    }
}

```

8.8.5. pfc_ei/Resultado.java

```

/*****
**
ontology.Resultado

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Resultado definido en la ontología
*****/
/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;
import java.util.*;

public class Resultado extends Domain_OntologyObj
{
    //Variables
    private List asignacion=new ArrayList();
    private Long nobjetos;

    //Construtor
    public Resultado() {
        super("Resultado");
        nobjetos=new Long(0);
    }

    public Resultado(Long n) {
        super("Resultado");
        nobjetos=n;
    }

    //Métodos SET
    public void addasignacion (Object grado)
    {
        (this.asignacion).add(grado);
    }
    public void clearasignacion ()
    {
        asignacion=new ArrayList();
    }
    public void setnobjetos (Long n)
    {
        this.nobjetos=n;
    }

    //Métodos GET
    public Iterator getAllasignacion()
    {
        return(this.asignacion).iterator();
    }
    public Long getnobjetos()
    {
        return this.nobjetos;
    }
}

```

8.8.6. pfc_ei/Grado.java

```

/*****
**
ontology.Grado

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Grado definido en la ontología
*****/
/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class Grado extends Domain_OntologyObj
{
    //Variables
    private String valor=new String("");
    private Long id_objeto;

    //Constructor
    public Grado() {
        super("Grado");
        valor="No_inicializado";
        id_objeto=new Long(0);
    }

    public Grado(String s, Long n) {
        super("Grado");
        valor=s;
        id_objeto=n;
    }

    //Metodos SET
    public void setvalor (String s)
    {
        this.valor=s;
    }

    public void setidObjeto (Long id)
    {
        this.id_objeto=id;
    }

    //Metodos GET
    public String getvalor()
    {
        return this.valor;
    }

    public Long getidObjeto()
    {
        return this.id_objeto;
    }
}

```

8.8.7. pfc_ei/Elección.java

```

/*****
**
ontology.Eleccion

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Eleccion definido en la ontología
*****/
/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class Eleccion extends Domain_OntologyObj
{
    //Variables
    private String id_hospital=new String("");
    private Long id_paciente;

    //Constructor
    public Eleccion() {
        super("Eleccion");
        id_paciente=new Long(0);
        id_hospital="No_inicializado";
    }

    public Eleccion(String s, Long n) {
        super("Eleccion");
        id_paciente=n;
        id_hospital=s;
    }

    //Metodos SET
    public void setid_hospital (String s)
    {
        this.id_hospital=s;
    }

    public void setid_paciente (Long id)
    {
        this.id_paciente=id;
    }

    //Metodos GET
    public String getid_hospital()
    {
        return this.id_hospital;
    }

    public Long getid_paciente()
    {
        return this.id_paciente;
    }
}

```

8.8.8. pfc_ei/Valor_atrib.java

```

/*****
**
ontology.Valor_atrib

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Valor_atrib definido en la ontología
*****/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class Valor_atrib extends Domain_OntologyObj
{
    //Variables
    private String valor_string= new String("");
    private Long peso; //De momento no usaremos los pesos

    //Constructor
    public Valor_atrib() {
        super("Valor-atrib");
        valor_string = "-";
        peso = new Long(1);
    }

    public Valor_atrib(String val, Long pes) {
        super("Valor-atrib");
        valor_string=val;
        peso=pes;
    }

    public Valor_atrib(String val) {
        super("Valor-atrib");
        valor_string=val;
        peso=new Long(1);
    }

    //Metodos SET
    public void setvalor_string (String s)
    {
        this.valor_string=s;
    }

    public void setpeso (Long id)
    {
        this.peso=id;
    }

    //Metodos GET
    public String getvalor_string()
    {
        return this.valor_string;
    }

    public Long getpeso()
    {
        return this.peso;
    }
}

```

8.8.9. pfc_ei/Ordenación_imposible.java

```

/*****
**
ontology.Ordenacion_imposible

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Ordenacion_imposible (error) definido en la
ontología

*****/

package pfc_ei.ontology;
import jade.onto.sl.sl.*;
public class Ordenacion_imposible extends Domain_OntologyObj
{
    //Constructor
    public Ordenacion_imposible()
    {
        super ("Ordenacion-imposible");
    }
}

```

8.8.10. pfc_ei/Agente_no_registrado.java

```

/*****
**
ontology.Agente_no_registrado

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto Agente_no_registrado (error) definido en la
ontología

*****/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class Agente_no_registrado extends Domain_OntologyObj
{
    //Constructor
    public Agente_no_registrado()
    {
        super ("Agente-no-registrado");
    }
}

```

8.8.11. pfc_ei/No_hospitales.java

```

/*****
**
ontology.No_hospitales

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto No_hospitales (error) definido en la ontología
*****/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class No_hospitales extends Domain_OntologyObj
{
    //Constructor
    public No_hospitales()
    {
        super ("No-hospitales");
    }
}

```

8.8.12. pfc_ei/No_pacientes.java

```

/*****
**
ontology.No_pacientes

PFC Jaime Bocio Sanz
10-5-2002

Implementación del objeto No_pacientes (error) definido en la ontología
*****/

package pfc_ei.ontology;

import jade.onto.sl.sl.*;

public class No_pacientes extends Domain_OntologyObj
{
    //Constructor
    public No_pacientes()
    {
        super ("No-pacientes");
    }
}

```

9. VALORACIÓN Y CONCLUSIONES

El mundo de los agentes y en particular el de los Sistemas Multi-Agente es uno de los terrenos de la Inteligencia Artificial que aún está en fases de expansión, aunque ha crecido bastante en los últimos años. Al menos yo he notado la evolución desde hace dos años cuando realicé el Proyecto Final de Carrera de la Ingeniería Técnica. Se han implantado nuevos estándares, mejoras en las herramientas de desarrollo, etc.

Uno de los objetivos pretendidos con este trabajo era continuar de alguna manera con el proyecto global iniciado por el GruSMA y del cual yo ya había desarrollado una parte. Este objetivo está en parte cumplido, pero aún faltaría implementar algunas partes, como la de Gestión de pacientes de Urgencia 0 (que está en fase de desarrollo), y sobretodo la unión de todo el proyecto global.

Como hemos ido comentando a lo largo del documento, con este proyecto no se pretendía realizar un sistema que pudiera implantarse de inmediato para suplir el sistema actual de gestión de transplantes. Sólo es un prototipo de cómo se podría realizar mediante un SMA. Pese a esto sería muy interesante que en un futuro se pudieran unir todas las partes realizadas por los distintos miembros del GruSMA para conseguir un sistema global de gestión de transplantes a nivel nacional.

A nivel personal, este proyecto ha supuesto también una evolución en el mundo de los SMA. Cuando realicé el PFC de la Ingeniería Técnica trabajaba de una manera muy distinta, mucho más simple, sin usar apenas los distintos tipos de mensajes que nos proporciona la FIPA, sin definir ontologías (lo que obligaba, por ejemplo, a parsear strings para entender el contenido de los mensajes), etc. En este proyecto he usado muchas más técnicas y herramientas, lo que ha permitido crear un sistema más complejo que el anterior.

En definitiva, me ha sido útil la realización de este proyecto y espero que en un futuro pueda serlo también al GruSMA, para implantar el Sistema Global.

Por último dar las gracias a todos los miembros del GruSMA, en especial a mis tutores Toni Moreno y Aïda Valls, y a David Sánchez por la aportación de su PFC.

10. TRABAJOS FUTUROS

Pese a que el sistema funciona y tiene bastantes opciones, hay algunas cosas nuevas que se podrían hacer y otras que se podrían mejorar:

- Mejorar la eficiencia de la búsqueda. Por ejemplo si en un cierto nivel no se encuentra paciente porque **SMAO** no ha podido ordenar los posibles receptores (porque son pocos), éstos se podrían juntar con los del nivel siguiente y evaluarlos todos juntos. Así se evitaría pasar por alto un paciente del primer nivel mejor que uno del segundo que no ha sido escogido sólo por el hecho de que **SMAO** no los podía ordenar solos.

- Añadir nuevas opciones en la búsqueda. Por ejemplo ampliar el número de tipos de órganos, poner topes en el nivel de búsqueda, etc.

- Mejorar la comprobación de la congruencia de los datos. Ahora se hacen varias comprobaciones en la interficie, sin embargo se admite por ejemplo cosas como haber nacido el 30 de febrero o que un paciente pese 2Kg.

- La arquitectura del sistema se podría adaptar a otro campo de la medicina o incluso fuera de ella, aprovechando la jerarquía de coordinadores y la forma de comunicarse entre ellos.

- Adaptar el sistema para que se pudiera incluir sin demasiados cambios en el proyecto global de gestión de transplantes del GruSMA. Se tendría que implementar una especie de coordinador intermedio capaz de comunicarse con las otras partes.

- Permitir parametrizar los coordinadores a la hora del registro inicial. Ahora se hace automáticamente porque se suponen fijos y nos ahorramos trabajo para las pruebas, pero en la versión final los coordinadores habrían de indicar su autonomía y zona. Aunque no está incluido en la documentación, en la última versión del sistema ya es posible parametrizar un hospital, indicando la ciudad, provincia y nombre de la BD de pacientes que usa.

En definitiva, hay muchos aspectos que se pueden mejorar, y esto seguirá así a medida que crezca la tecnología y permita realizar nuevas cosas.

11. BIBLIOGRAFÍA

- [Aldea et al., 01] Aldea, A., López, B., Moreno, A., Riaño, D., Valls, A., *A multi-agent system for orhan transplant coordination*, 2001.
- [Bocio, 00] Bocio, J., *Diseño e Implementación de un Sistema Multi-Agente para la gestión de equipos médicos*, Proyecto Final de Carrera de Ingeniería Técnica en Informática de Sistemas, URV, 2000,
URL: <http://www.etse.urv.es>
- [Brenner, 98] Brenner, W., Zarnekow, R., Wittig, H., *Intelligent Software Agents*, Ed. Springer-Verlag, 1998.
- [FIPA, 99] Foundation for Intelligent Physical Agents, *Specification of FIPA*, 1999,
URL: <http://www.fipa.org/spec/fipa99spec.htm>
- [Isern, 99] Isern, D., *Avaluació d'entorns de desenvolupament de Sistemes Multiagent*, Proyecto Final de Carrera de Ingeniería Técnica en Informática de Sistemas, URV, 1999,
URL: <http://www.etse.urv.es>
- [Bellifemine, 00] Bellifemine, F., *Jade Programmer's Guide 1.12*, 2000.
URL: <http://sharon.cselt.it/projects/jade/>
- [GruSMA, 00] Moreno, A., Valls, A., *Web del GruSMA*, 2000,
URL : <http://www.etse.urv.es/recerca/banzai/toni/MAS/>
- [FIPA 1, 98] Foundation for Intelligent Physical Agents, *FIPA 98 Specification Part 1 Agent Management*, 1998,
URL: <http://www.fipa.org>
- [FIPA 2, 98] Foundation for Intelligent Physical Agents, *FIPA 97 Specification Part 2 Agent Communication Language*, 1998,
URL: <http://www.fipa.org>
- [FIPA 13, 98] Foundation for Intelligent Physical Agents, *FIPA 98 Specification Part 13 FIPA97 Developers Guide*, 1998,
URL: <http://www.fipa.org>
- [FIPA, 01] Foundation for Intelligent Physical Agents, *FIPA Device Ontology*, 2001,
URL: <http://www.fipa.org>
- [ONT] Organización Nacional de Transplantes, Ministerio de Sanidad y Consumo,
URL: <http://www.msc.es/ont/esp/home.htm>

- [Ribes, 00] Ribes, A., *SMA d'ajuda a la gestió dels transplants d'òrgans entre hospitals. Logística del transport dels òrgans*, Proyecto Final de Carrera de Ingeniería Técnica en Informática de Sistemas, URV, 2000,
URL: <http://www.etse.urv.es>
- [Sánchez, 01] Sánchez, D., *Un Sistema Multi-Agent d'ajuda a l'assignació d'òrgans en transplantaments. El mètode ClusDM amb criteris lingüístics*, Proyecto Final de Carrera de Ingeniería Técnica en Informática de Sistemas, URV, 2001,
URL: <http://www.etse.urv.es>
- [Valls et al., 02] Valls, A., Moreno, A., Sánchez, D., *A multi-criteria decision aid agent applied to the selection of the best receiver in a transplant*, 4th. International Conference on Enterprise Information Systems, pp. 431-438, Ciudad Real, Spain, 2002.
- [Weiss, 99] Weiss, G., *Multiagent Systems. A modern approach to distributed artificial intelligence*, MIT Press, 1999.