PROYECTO FINAL DE CARRERA

UNIVERSITAT ROVIRA I VIRGILI

Diseño e Implementación de un Sistema Multi-Agente para la Gestión de Equipos Médicos

Jaime Bocio Sanz, jaimebocio@ctv.es
Ingeniería Técnica en Informática de Sistemas

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, 2000

1. INTRODUCCIÓN AL PROBLEMA

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 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.

Un transplante necesita de tres partes 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 podrían coordinar sin mucha dificultad a priori. La primera parte consiste en buscar el paciente ideal para transplantarle un órgano. Es decir, hay un encargado, un Coordinador General de todos los hospitales de una zona, que recibe noticias de todos los órganos donados, dispuestos para un transplante. Éste busca entonces al paciente adecuado entre una serie de hospitales, que le envían sus respuestas para que las evalúe y encuentre la mejor. En esta decisión pueden intervenir muchos factores como la distancia entre hospitales, la duración del órgano en buen estado, la necesidad del paciente, etc. Una vez se ha decidido a qué hospital llevar ese órgano, entra en juego el segundo proyecto, buscarle una ruta al órgano para que llegue a su destino en el tiempo previsto. Aquí, el que recibe la petición del transplante del órgano a un determinado lugar, tiene que preguntar a las agencias de viajes, trenes, autobuses y ambulancias, para encontrar una combinación de transportes que pueda hacer que el órgano llegue a la hora adecuada. Si esto es posible, viene la tercera parte del proyecto, que es la que nos ocupa.

En esta tercera parte, hay un coordinador de hospital, que recibe la petición del Coordinador General de transplantes, en la que se le dice la hora de llegada del órgano, la hora en que el órgano dejará de estar en buen estado, y la duración estimada de la operación, a parte de lógicamente decir de qué órgano se trata. Lo que ha de hacer el coordinador del hospital es buscar al equipo médico necesario (médicos, enfermeras y anestesistas) y un quirófano para que se pueda llevar a cabo la operación en el intervalo fijado. Los detalles se comentarán más adelante, aquí tan sólo se pretende comentar la idea general del proyecto.

Como he dicho antes, este proceso puede llegar a ser muy costoso, tanto económicamente como en tiempo, si es realizado por humanos. Se han de consultar muchos horarios distintos de personal y quirófano, coordinar del mejor modo posible estos horarios y encontrar el personal idóneo para cada operación. Debido a esto, se ha decidido que todo el proceso se puede realizar mediante Sistemas Multi-Agente (SMA), por las grandes ventajas que esto supone, y que se comentan más adelante.

Otro aspecto importante de este proyecto es que, aunque en un principio esté destinado a coordinar los preparativos para un transplante en un hospital, su uso puede ser mucho más global. Puede servir para organizar cualquier tipo de operación en un hospital, sin necesidad de tratarse de un transplante. También podría servir para coordinar equipos humanos o físicos, que necesitan compartir una serie de recursos con restricciones temporales. El alcance del proyecto, pues, va más allá del mero mundo hospitalario, aunque nos centremos en este tema debido a que forma parte del proyecto global de logística de transplantes.

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 algún ejemplo para explicar el funcionamiento del sistema. 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. CARACTERÍSTICAS DE LOS SISTEMAS MULTI-AGENTE

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.

Un Sistema Multi-Agente, como su propio nombre indica, es un sistema formado por varios agentes. No existe una definición clara de lo que es un agente, en este sentido, cada uno aporta su definición, pero una más o menos acertada podría ser que un agente es un programa que realiza tareas para un usuario, de forma autónoma y con un comportamiento inteligente **[Brenner, 98]**. Visto así, un agente es un programa normal y corriente, pero lo bueno que tienen los agentes es su interacción con otros agentes, que es cuando se forma un SMA. Un SMA se basa 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 obviamente 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.

Pasemos ahora a las ventajas que nos aportan los SMA sobre este proyecto en concreto. Para verlo más claramente, primero voy a explicar 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 nuestro hospital y habremos de buscar el equipo médico. Esto provoca consultas de los horarios por

parte de algún empleado del hospital. Estos horarios habrán de estar en la misma máquina para que la consulta sea más rápida y eficaz, lo cual representa, que para acceder a los horarios, ya sea para consultar o modificar, el poseedor del horario en cuestión ha de acceder al ordenador central del hospital, lo que puede suponer colapsos, retardos o problemas de seguridad. Una vez consultados los horarios se tendrán que intentar compatibilizar todos, una tarea bastante compleja.

Lo que nos permite el SMA es:

- Que cada trabajador tenga su horario en su ordenador propio, o al menos que no tengan que estar todos centralizados en un mismo ordenador. De esta manera, cuando se haga una consulta a un horario, no se colapsará el ordenador central.

- Se ahorran multitud de llamadas telefónicas, ya que todo va por red. Los distintos agentes, que estarán en máquinas diferentes se comunicarán con el resto por la red del hospital. En principio puede parecer un coste adicional la instalación de la red para el sistema, pero la mayoría de los hospitales ya tienen red interna, y a la larga el coste se rentabiliza.

- La rapidez se ve altamente incrementada, ya no hay consulta de fichas ni horarios por parte de humanos, todo lo hacen los agentes, lo que puede traducir vaios minutos de búsqueda en apenas medio segundo. La parte de compatibilizar horarios también es automática, por lo que no habrá fallos ni pérdidas de tiempo. No hace falta más personal para este trabajo y se pueden dedicar a otras tareas.

- El propio sistema avisará a los trabajadores si tienen que hacer una operación. No necesitan estar pendientes del horario, éste se modificará automáticamente y les avisará.

- El tiempo de espera se decrementa muchísimo, y en este tipo de operaciones es crucial, debido a la caducidad de los órganos. Lo que antes podía suponer mucho tiempo en buscar horarios, juntarlos y responder al coordinador de transplantes si la operación era posible, ahora se puede hacer en cuestión de segundos. Esto es especialmente beneficioso si el resultado de la búsqueda es negativo, ya que además se pueden realizar búsquedas en paralelo en distintos hospitales por si acaso.

- El paralelismo es también muy importante. Se pueden consultar todos los horarios a la vez, ya que cada uno estará controlado por un agente, que se ejecutará independientemente en un ordenador, tan sólo necesitará la comunicación para el paso de peticiones y resultados.

- Cuando no se pueda realizar la operación, se sabrá rápidamente, porque faltará algún elemento. Esto permite que la búsqueda se pueda cancelar de manera inmediata de modo que no haga falta buscar más y se pueda dar la respuesta negativa cuanto antes.

- Otro punto a favor es la flexibilidad, en el sentido de que se pueden añadir o eliminar agentes del sistema muy fácilmente. Esto será especialmente útil cuando haya altas o bajas del personal.

- La modularidad permite que un agente pueda hacer una tarea pequeña, y luego juntándola con las de los demás agentes que se han ejecutado en paralelo se obtenga una tarea mayor en un menor tiempo.

Como hemos visto 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.

3. ENTORNO DE TRABAJO

Para la realización de este proyecto, se decidió usar el entorno de desarrollo de Sistemas Multi-Agente JADE (Java Agent Development Environment). Este es un entorno que utiliza los estándares propuestos por la FIPA.

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.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 agentes con un determinado servicio (médicos, quirófanos, 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.

3.4. 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.

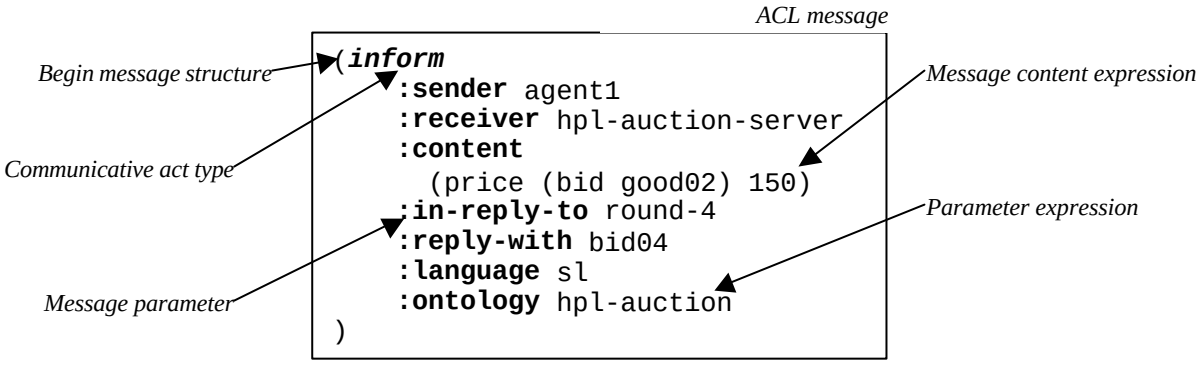
3.5. Mensajes

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.

3.5.1. 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 en otros lenguajes, pero para el caso que nos ocupa, nos basta con el ACL de la FIPA.

El aspecto de un mensaje puede ser este:



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 los parámetros.

Tabla 1 — Parámetros de los mensajes

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.*

Hay algunas normas o pautas a seguir para indicar el contenido (**:content**) de un mensaje. En este proyecto, no se ha seguido ninguna, ya que casi todos lo que se pasan los agentes son simples strings que son interpretadas por el receptor de una manera en concreto. Para manejar estos parámetros, el **JADE** nos ofrece varios métodos en la clase ACLMessage, que nos permiten consultarlos o modificarlos (getContent(), setContent(), etc).

3.5.2. 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.

Tabla 2 — Categorías de actos comunicativos

Acto Comunicativo	*Paso de Información*	*Pedido de Información*	*Negociación*	*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			✓		

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.

3.5.3. 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 desecharía 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.

3.6. Protocolos

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 cuales puede recibir.

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**.

3.6.1. 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 se observa el flujo de los mensajes. Los blancos son los que envía el "initiator" al que llamaremos Agente1, mientras que los del "responder"(Agente2) están en gris.

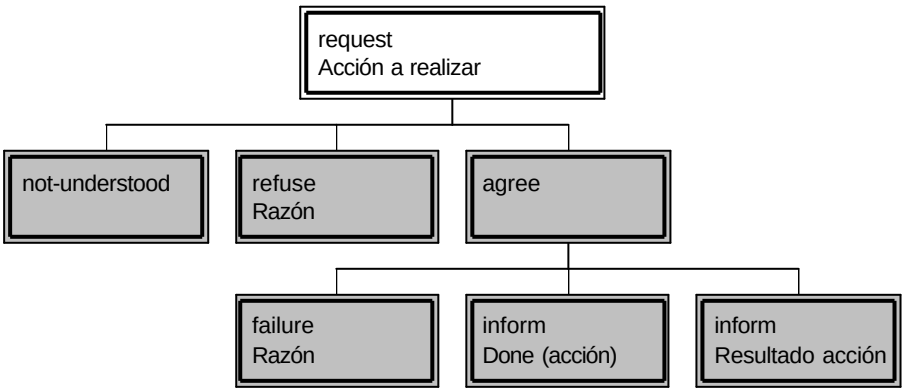


Figura 1 - Protocolo FIPA-request

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.

3.6.2. 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, 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.

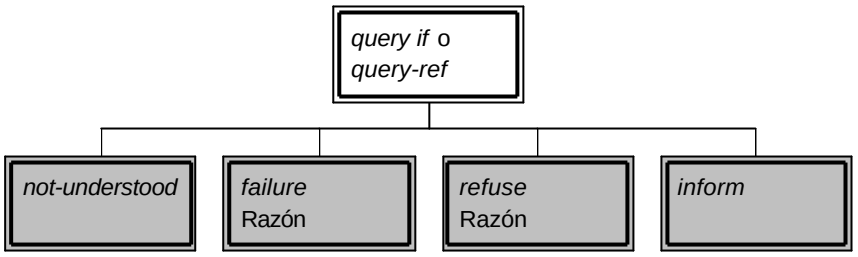


Figura 2 - Protocolo FIPA-Query

3.6.3. Protocolo FIPA-contract-net

El Agente1 quiere que uno o más agentes realicen una acción, por eso realiza 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.

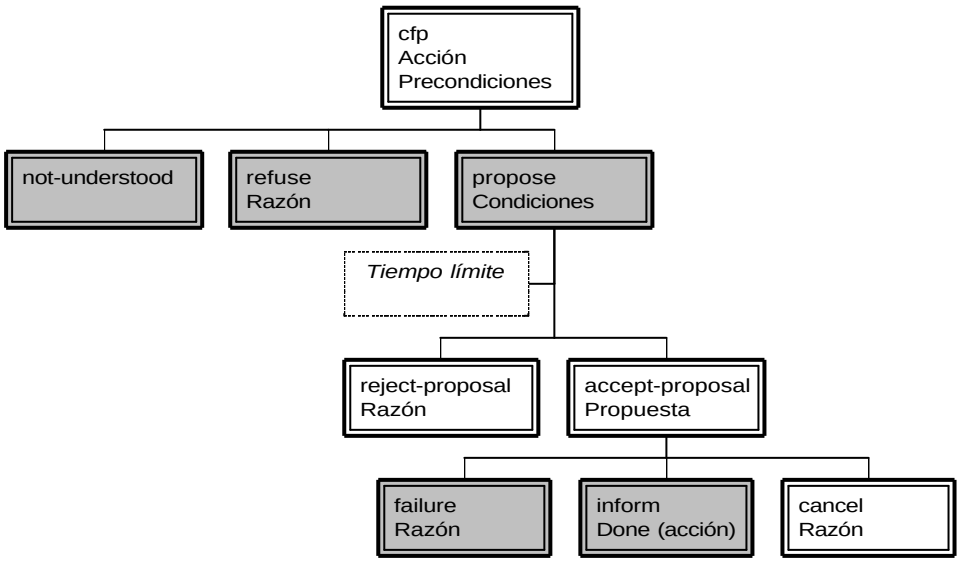


Figura 3 - Protocolo FIPA-Contract-Net

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**.

De estos protocolos se derivan otros, pero no nos interesan demasiado, de hecho para el proyecto sólo se ha usado este último, el ContractNet.

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 todas 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 **Ce:Equipo**, que indica que el agente **Ce** es de la clase **Equipo**.

3.7.1. Interficie gráfica

Activando la opción **-gui**, se nos abrirá una interficie gráfica, con el aspecto de la figura 4. El agente que controla esta interficie es el RMA, que junto a ACC, AMS y DF se inican automáticamente.

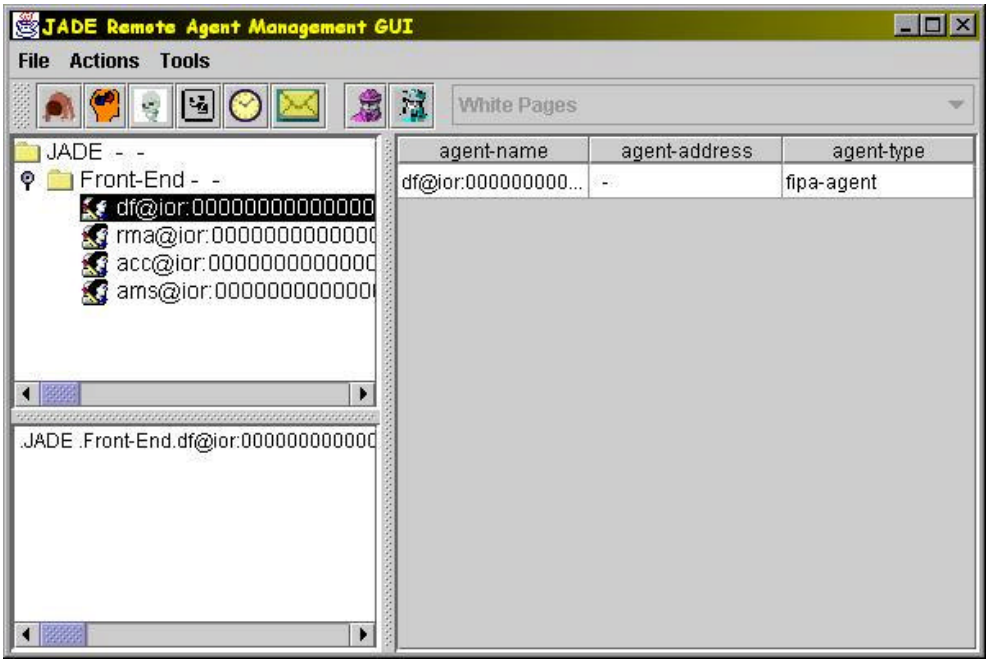


Figura 4 - Pantalla Principal del Jade

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, ACC 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 Front-End como contenedor por defecto.

- Ejecutar los agentes seleccionados.

- 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.

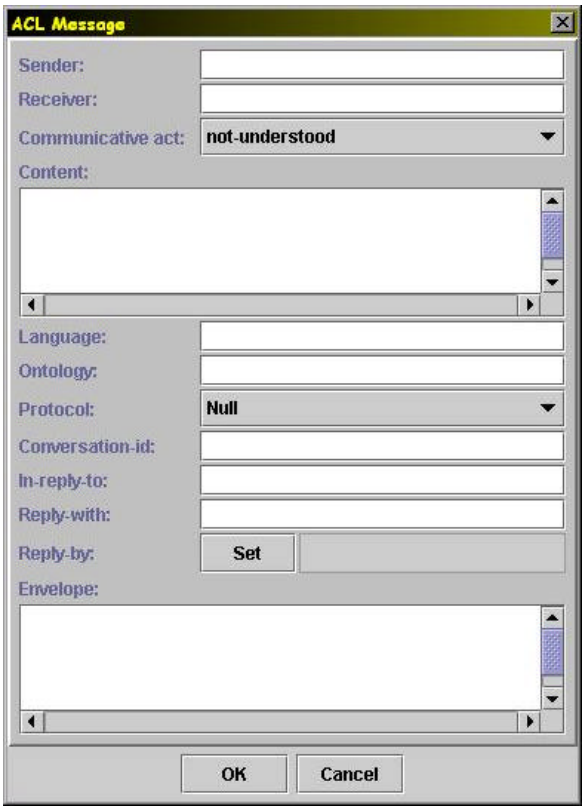


Figura 5 - Mensaje para rellenar y enviar

- 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.

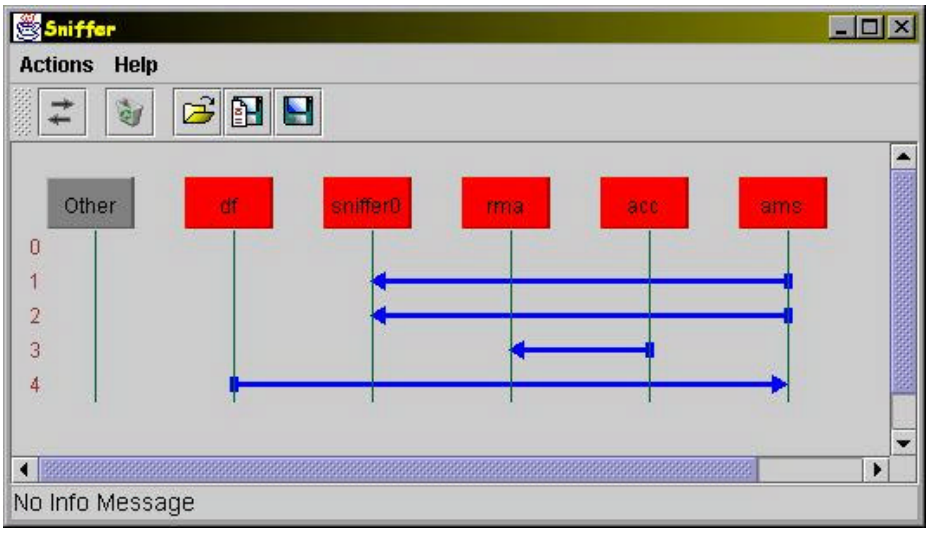


Figura 6 - Pantalla del sniffer

- 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.

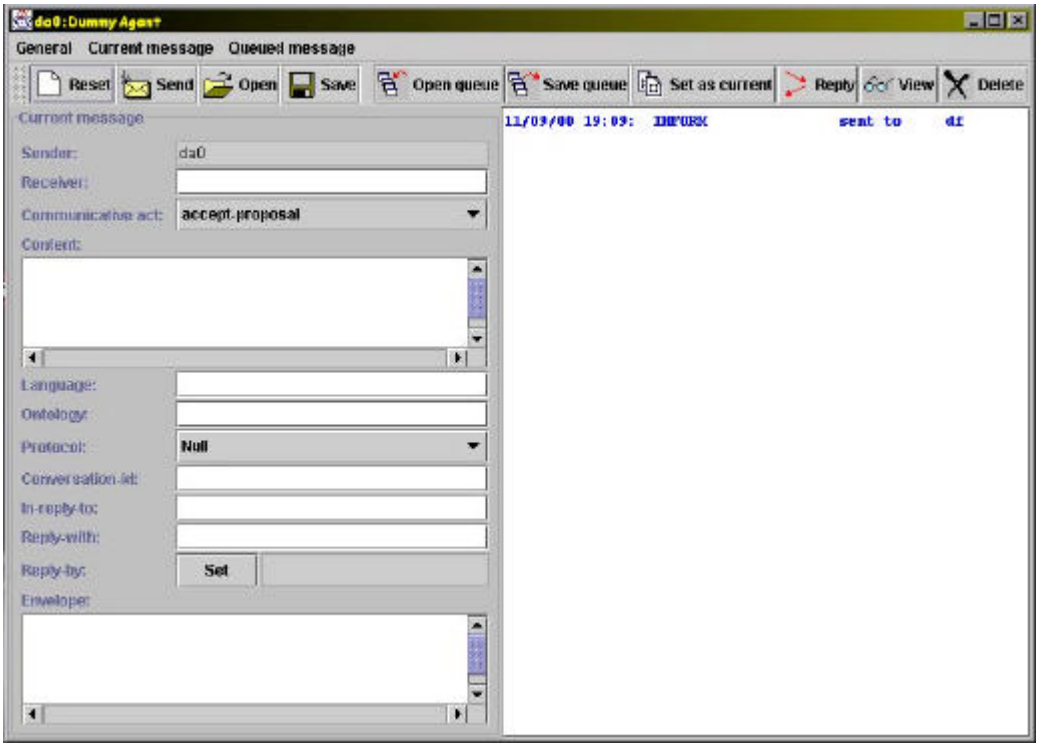


Figura 7 - Pantalla del Agente Dummy

- 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.

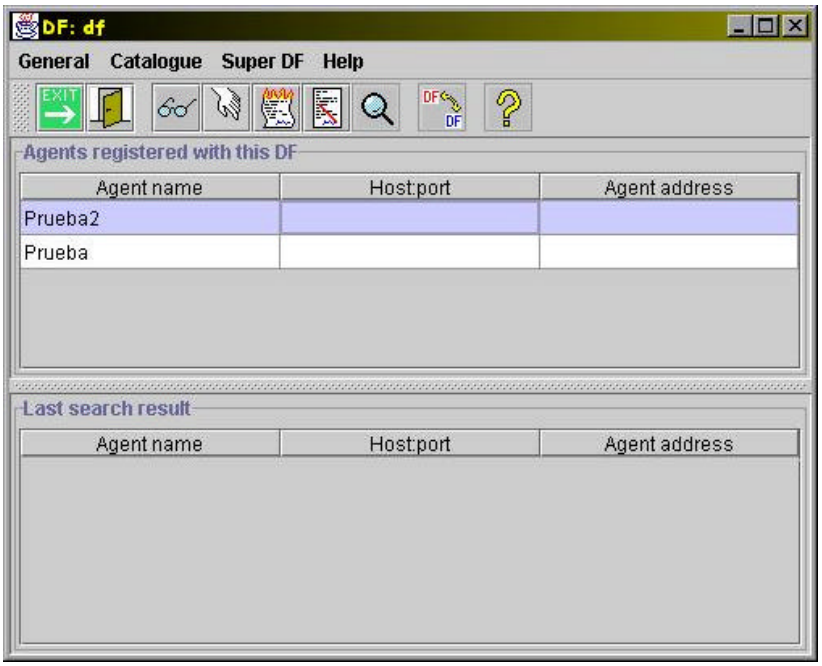


Figura 8 - Pantalla de la interficie del DF

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

4.1. Objetivos y Motivación

Como ya hemos dicho, el objetivo del proyecto global es agilizar y facilitar el proceso de los transplantes. Pero obviamente, aquí no se pretende hacer un sistema que se pueda implantar en un hospital, al menos directamente, sin cambios. Se pretende hacer una simulación de cómo sería el proceso si interviniesen los SMA. Aunque esta simulación se haya hecho 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 médico, enfermera, anestesista o gestor de quirófanos pueda tener 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.

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, si pusiéramos este sistema en un hospital, cada vez que el coordinador global de hospitales mandara un órgano a nuestro hospital, suponiendo que se hace por teléfono, el coordinador de nuestro hospital tan sólo tendría que poner en marcha el sistema simulando un mensaje con la hora de llegada, de caducidad, duración y el tipo de órgano.

4.2. Arquitectura

A partir de ahora me centraré en mi proyecto de gestión de equipos médicos. La arquitectura del SMA, con los distintos agentes que lo componen y la interacción entre ellos se puede ver en la figura 9.

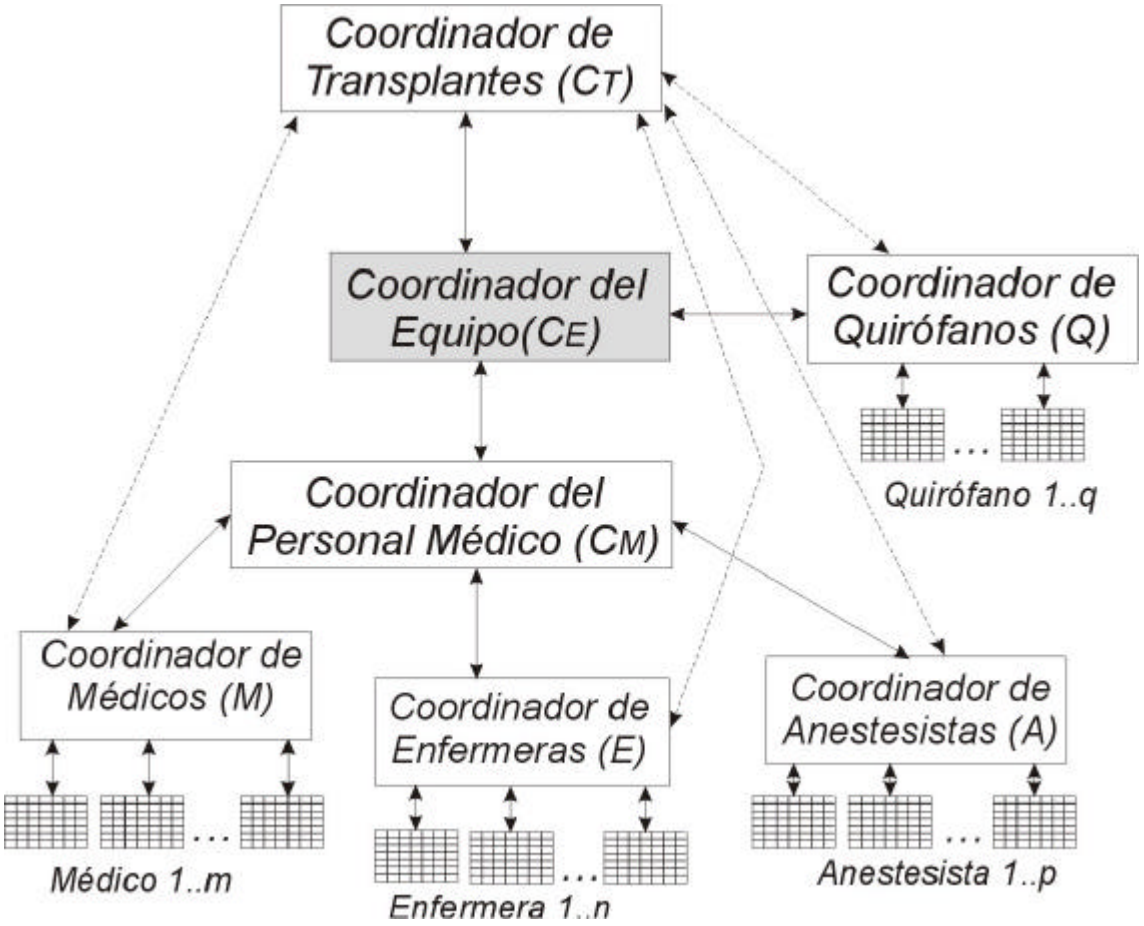


Figura 9 - Arquitectura del Sistema

Los agentes principales del sistema son siete:

- **Coordinador de Transplantes (CT)**: Es el agente encargado de comunicarse con el exterior. Es el único que tendrá contacto con agentes fuera del hospital. Su tarea es sólo esa, se comunica con el Coordinador de Transplantes General, para recibir las peticiones de transplante. También recibe las posibles cancelaciones que puedan surgir una vez ya está programada la operación. Estas peticiones y cancelaciones las guarda en una cola, para tratarlas de una en una.

Recibe las peticiones del coordinador central, como ya hemos dicho, y las pasa hacia abajo, es decir, informa al Coordinador del Equipo (CE) de que va a llegar un órgano nuevo al hospital, y que deberá preparar la infraestructura necesaria para realizar el transplante. Una vez CE haya hecho esto, CT recibirá la respuesta. Puede ser afirmativa o negativa. En el primer caso, CT informará de todos los pormenores

de la operación: quirófano, personal médico, hora y órgano. En caso de que la operación no se pueda realizar, CT informará de los motivos por los cuales no es posible. Estos motivos pueden ser: que no haya quirófano, que falte un determinado tipo de personal, que los horarios del personal no sean compatibles entre ellos o con los de quirófanos, etc. Cuando CE está listo para recibir una nueva petición, avisará a CT, para que le envíe otra de las que tiene guardadas en la cola.

A la hora de hacer las cancelaciones, CT no se comunica con CE, sino directamente con los agentes coordinadores de médicos, anestesistas, enfermeras y quirófanos. Les indicará la hora de la operación que tienen que cancelar, y estos indicarán a las personas involucradas que han de modificar los horarios para cancelar la operación. Si todo ha sido correcto, CT recibirá confirmación de todos y lo indicará por pantalla. El motivo de que CT no se comunique ahora con CE, es simplemente porque no es necesario poner de acuerdo al personal y quirófano, que es la tarea principal de CE. Tan sólo se necesita que los coordinadores de cada tipo de personal y de quirófanos, avisen a sus hijos que la operación se tiene que cancelar. Pero CE no tiene porqué enterarse, porque además él no guarda las operaciones programadas, cosa que si hacen M, A, E y Q.

- **Coordinador del Equipo (CE)**: Este es el agente principal de todo el sistema. Es el nexo de unión entre todos los elementos y el que toma las decisiones más importantes. Recibe las peticiones de CT y las sigue pasando hacia abajo, a los agentes del nivel inferior. Cuando recibe los posibles horarios de cada persona y de cada quirófano, se encarga de coordinarlos para obtener alguna hora libre para la operación. Si lo consigue, avisa a los agentes inferiores para que actualicen los horarios indicando la reserva, y una vez recibidas todas las confirmaciones, informa a CT del resultado de la búsqueda.

Este agente no se comunica con el exterior, tan sólo lo hace con otros tres agentes principales del sistema. Ya hemos visto su comunicación con CT. Además también lo hace con el Coordinador de Quirófanos (Q) y el Coordinador de Personal Médico (CM). Más adelante explicaremos más a fondo cómo trabaja y cómo es la comunicación con estos dos agentes.

- **Coordinador de Quirófanos (Q)**: Este agente trata directamente con los agentes que controlan los horarios de los distintos quirófanos del hospital. Recibe las peticiones de CE. Entonces pide propuestas de posibles intervalos horarios que tengan libres los quirófanos, dentro del horario apto para la operación. Recibe las propuestas y se las pasa a CE para que éste elija la mejor opción, dependiendo de la compatibilidad con los horarios del personal.

En última instancia es Q quien decide cuándo hacer la operación, de entre los posibles horarios compatibles entre todos los que le ofrece CE, y en qué quirófano. Esta decisión la toma en función de los huecos que puedan quedar en los horarios de los quirófanos.

Q se comunica, pues, con CE y con todos los agentes que controlan el horario de cada uno de los quirófanos del hospital. En un caso excepcional también se comunica con CT, sólo cuando hay que hacer una cancelación de una operación ya

fijada. Como hemos comentado, recibe la hora de la operación a cancelar, y le pasa el resultado de nuevo a CT.

- **Coordinador de Personal Médico (CM)**: Este agente se encarga de buscar el personal necesario para una operación. Pasa la petición hacia abajo, para que M, A y E busquen horarios libres de sus agentes inferiores. Recibe todas estas propuestas e intenta coordinarlas, de tal manera que haya el número necesario de cada tipo de empleado (médicos, enfermeras y anestesistas). CM sabe qué personal es imprescindible para realizar cada tipo de operación, según de que órgano se trate. Cuando ya ha restringido los horarios posibles a los compatibles por todos, los pasa a CE. CE realiza el proceso de selección y le envía la hora definitiva de la operación, con lo que CM tiene que informar a los agentes inferiores de que hagan la actualización de horarios. Cuando recibe las confirmaciones de M, A y E, las pasa a CE y acaba su tarea.

- **Coordinador de Médicos (M)**: Este agente es muy similar a Q, y casi idéntico al Coordinador de Enfermeras y de Anestesistas. Recibe las peticiones de CM, y busca propuestas por parte de los agentes de horarios de médicos. Como los médicos tienen distintas especialidades, M necesita saber de qué órgano se trata, para comunicarse sólo con los médicos especialistas en ese órgano. Esto lo hace comunicándose con el DF, para que le informe de las direcciones de los médicos sólo de la especialidad requerida.

M pide propuestas a los agentes de horarios de médicos, las pasa a CM e informa a estos mismos agentes de la hora de la operación. Es aquí cuando decide qué médicos harán la operación, dependiendo, como en los quirófanos, de los huecos que haya en los horarios.

Al igual que Q también se comunica con CT, para cancelar operaciones ya confirmadas.

- **Coordinador de Anestesistas (A)**: Lo único que cambia en este agente respecto a M, es que no necesita saber de qué órgano se trata, ya que todos los anestesistas están capacitados para realizar cualquier tipo de operación. Debido a esto, cuando hace una búsqueda de anestesistas, también con el DF, pide propuestas a todos los agentes que estén registrados al sistema como anestesistas.

Por lo demás, es exactamente igual. Se comunica con CM, con los agentes inferiores de horarios y con CT para los casos de cancelación.

- **Coordinador de Enfermeras (E)**: El funcionamiento de este agente es idéntico al de A pero comunicándose con enfermeras.

Estos siete agentes principales son esenciales para el funcionamiento del sistema. Si falta cualquiera de ellos, el sistema no puede funcionar, son el motor del sistema. Pero evidentemente para que en un hospital se puedan hacer operaciones, se necesita personal médico y quirófanos. Aquí es donde entra en juego el último tipo de agente. No se consideran agentes principales porque su número es variable, incluso puede que

en algún hospital falte algún tipo de elemento (médicos, anestesistas, enfermeras o quirófanos),. Tenemos pues este último tipo de agentes:

- **Agente de Horarios (H):** Este es el único agente que tiene contacto directo con el usuario, que en nuestro caso serán los médicos, anestesistas, enfermeras y responsables de quirófanos. Se encarga de llevar los horarios de estos usuarios, tanto en lo que se refiere a consultas como a actualizaciones.

Este agente sólo se comunica con su coordinador y con los usuarios. H recibe peticiones donde se le indica la hora de llegada del órgano, la hora de caducidad y la duración de la operación. Siguiendo el protocolo Contract-Net, H le envía una propuesta a su coordinador conteniendo las horas que tiene libre el usuario, desde que llegará el órgano hasta que este será inutilizable, cumpliendo las condiciones, o sea la duración de la operación. Cuando ha comprobado el horario e informado de las horas libres, espera a que se realice todo el proceso por los agentes superiores. Entonces son informados de la hora elegida por el coordinador de quirófanos, que será la hora definitiva de la operación. H comprueba su horario y dependiendo de los huecos que pueden quedar si se ocupa esa hora, retorna una puntuación a su coordinador. Al final, los usuarios cuya propuesta ha sido aceptada reciben la hora a la que será la operación, y los rechazados reciben un mensaje de rechazo. Si se ha aceptado la propuesta, sólo queda actualizar el horario ocupando las horas de la operación.

La otra función de este agente es la de interacción exclusiva con el usuario, lo que le permite actualizar su horario y consultarlo, mediante un menú, que más adelante detallaremos.

Para ver más claro lo de las puntuaciones que se le otorgan a un intervalo horario, pondremos un pequeño ejemplo.

Supongamos que tenemos los siguientes horarios, donde los agentes M son médicos, A anestesistas, E enfermeras y Q quirófanos. Las horas ocupadas son las que están marcadas en gris, azul para los quirófanos.

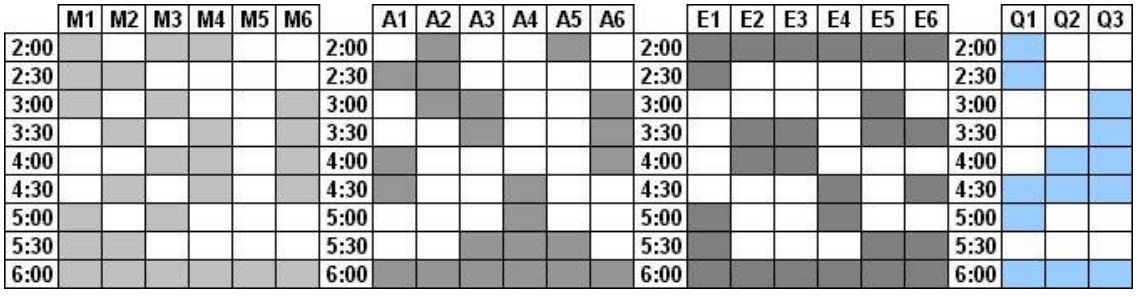


Figura 10 - Horarios

Imaginemos que se ha realizado casi todo el proceso, y se ha concluido que el único momento posible para una operación de una hora de duración, es de 3:00 a 4:00. En ese momento tenemos dos quirófanos disponibles, Q1 y Q2. Cuando Q1, Q2 y Q3 son informados de que la operación se puede realizar sólo de 3:00 a 4:00,

han de poner puntuación a este intervalo, para que Q elija el quirófano con mayor puntuación. Las puntuaciones se asignan de la siguiente manera. En el caso de Q3, la operación no se puede realizar a esa hora, por lo que retornaría como puntuación un 0. En caso de que la operación quepa en un hueco, el criterio que se sigue es el siguiente. Si la operación encaja perfectamente, es decir no quedan huecos ni antes ni después de la operación, se retorna un 3. Si quedan huecos por ambos lados, se retorna un 1, y finalmente, se retorna un 2 en el caso tanto de Q1 como de Q2, es decir, cuando queda un solo hueco, ya sea antes o después de la operación. Q elige Q1 o Q2 para realizar la operación, en este caso es indiferente porque los dos tienen la misma puntuación.

Supongamos ahora que sólo necesitamos un médico, la elección sería clara, el único médico disponible es M5, aunque tenga la peor puntuación posible 1. Necesitamos también 2 anestesistas y una enfermera. A escogería de entre las propuestas, la de A1, con puntuación 3 y cualquiera de las dos entre A4 y A5, que tienen la misma puntuación, 1. Para las enfermeras nos quedaríamos con E1, que tiene puntuación 2, frente al 1 de E4.

Con esta asignación de operaciones, lo que se pretende es que el tiempo se aproveche lo máximo posible, minimizando los huecos en los horarios, para que quede lo más compacto posible. Lo que no se tiene en cuenta es la repartición equitativa del trabajo entre todas las personas, podría ser una opción a añadir en el futuro.

4.3. Clases Java para los Agentes

Para implementar todos estos tipos de agentes, se usan una serie de clases en java, concretamente son siete.

4.3.1. Coord.java

Esta clase es la que maneja al agente Coordinador de Transplantes (CT). Hay cuatro comportamientos implementados, todos ellos se extienden de la clase CyclicBehaviour, por lo que se van ejecutando continua y concurrentemente mientras esté en funcionamiento el sistema.

- **RecibeDeC( )**: Lo único que hace este comportamiento es recibir mensajes del Coordinador General de Transplantes (C), ya sean de petición o cancelación de una operación, y los mete en una cola. La única comunicación que tiene C con este agente es a través de este comportamiento. Si se tuvieran que juntar los tres proyectos, también se le tendría que pasar la respuesta a C, pero como aquí C no es un agente concreto, tan sólo lo simulamos para enviar las peticiones.

- **RecibeDeCe( )**: Este comportamiento sirve para recibir la confirmación desde el Coordinador del Equipo (CE), de que todo está preparado para recibir una nueva petición. Cuando CT recibe un mensaje de este tipo, activa una variable para indicar que puede mandar una petición en cuanto tenga una en la cola. Además se muestra por pantalla el resultado de la búsqueda, especificando los detalles de la operación o el motivo del rechazo.

- **EnviaPeticion( )**: Si CE está preparado y hay peticiones en la cola, CT está dispuesto a enviar una. Primero se ha de mirar si lo que se va a enviar es una petición de transplante o una cancelación de uno ya programado. En el primer caso, se le envía la petición a CE. Si es una cancelación, CE no interviene, y CT la envía directamente a los Coordinadores de médicos, anestesistas, enfermeras y quirófanos, especificando que se trata de una cancelación. Una vez enviada la petición o cancelación, se elimina de la cola.

- **RecibeAceptación( )**: Si se ha enviado una cancelación, se espera a que todos los elementos (Q, M, A y E), envíen confirmación. Entonces se indica por pantalla que la cancelación se ha realizado correctamente y se activa la variable de preparado, para poder mandar otra petición.

4.3.2. Equipo.java

Esta es la clase perteneciente al agente Coordinador del Equipo (CE). Tan sólo tiene un comportamiento cíclico, y en él se realizan todas las tareas de este agente. Como toda la comunicación del sistema sigue un protocolo concreto, y este agente sólo interviene en las peticiones de transplante (no en las cancelaciones). En este comportamiento se envían y reciben mensajes ya establecidos. No hacen falta varios comportamientos concurrentes en el sistema, ya que la tarea de CE se realiza de una manera lineal.

- **RecibeDeCt( )**: Lo primero que se hace en este comportamiento es bloquear al agente hasta que se reciba un mensaje de CT, indicando una petición de transplante. La petición tiene el siguiente formato: "(hora_llegada_órgano hora_caducidad_órgano duración_operación tipo_ órgano)". Por ejemplo: "(02:00 06:00 01:00 corazon)". Se descompone la petición en tipo de órgano y horas. Se pasa la petición hacia CM, y se avisa a Q de que le va a llegar un órgano nuevo. Después de avisarle, se le envía la petición, pero sólo con las horas, porque a Q no le importa de qué órgano se trata. Después de esto, el proceso de búsqueda sigue, hasta que CE recibe dos mensajes, uno de CM y otro de Q, en los cuales se le comunican las horas libres que se han podido conseguir, de quirófano y de personal médico. Con estos intervalos horarios libres, se intenta encontrar alguno en que haya el personal médico necesario y quirófano. En caso de que haya horarios posibles, éstos se le envían a Q, para que sea él el que elija. Se espera la respuesta y se le pasa la hora elegida a CM. Sólo falta recibir la confirmación de CM. En caso de que haya algún tipo de error, se informa de qué se trata, y se para la búsqueda. Finalmente, se informa a CT del resultado de la búsqueda y de que ya vuelve a estar preparado para recibir otra petición.

4.3.3. Personal.java

A partir de esta clase surge el Coordinador de Personal Médico (CM). Al igual que en la clase anterior, aquí sólo hay un comportamiento, debido a que la comunicación es bastante lineal. Una particularidad de este agente es que posee una tabla en la cual está especificada la cantidad de médicos, anestesistas y enfermeras para cada tipo de órgano, pudiendo ser ampliada fácilmente para añadir nuevos órganos al sistema.

- **RecibeDeCe( )**: Este es el único comportamiento de esta clase, y que se encarga de la comunicación con CE, M, A y E. El comportamiento se activa al recibir un mensaje de CE, en el cual se informa de una petición de transplante. Al igual que en CE, aquí se descompone el mensaje en órgano y horas, porque el órgano únicamente le interesará al coordinador de médicos, para contactar sólo con los de la especialidad requerida. Entonces se avisa a M, A y E de que van a recibir una nueva petición, y posteriormente se le envía la propia petición. Como ya hemos dicho, a A y E sólo se le envían las horas, mientras que a M se le envían horas y órgano. El proceso continúa por abajo hasta que CM recibe respuesta de M, A y E, con los intervalos libres encontrados por los agentes de horarios. Para cada tipo de personal, rellena una tabla, que le permitirá luego saber si el número de personal de cada tipo que hay a una hora determinada es suficiente para realizar la operación. Este número lo sabe gracias a la tabla que hemos comentado. En caso de que haya personal suficiente de los tres tipos, mira a ver si puede hacer que concuerden las horas libres, y se le envían a CE estos intervalos. Si todo es correcto, o sea hay intervalos posibles, se espera el mensaje de CE en que se informará de la hora definitiva de la operación. Puede ser que en esta respuesta de CE, se indique que se ha de cancelar el proceso, porque no se hayan podido encontrar horas compatibles entre personal y quirófano. De no ser así, si todo es correcto, se informa a M, A y E de la hora definitiva, indicando también el número de personas necesario en cada caso, para que estos elijan al personal más adecuado de entre las posibilidades. Se esperan las confirmaciones y si todo es correcto, se informa a CE del personal que realizará la operación. Si durante la búsqueda faltaba algún elemento, o había otro tipo de error, se procede a cancelarla.

4.3.4. Mae.java

Esta clase sirve para los Coordinadores de Médicos, Anestesistas y Enfermeras, de ahí su nombre (MAE). Como ya hemos dicho, M, A y E son casi idénticos, por lo que no necesitan una clase cada uno, sino que con la misma y teniendo en cuenta el nombre del agente en la plataforma, se comportan de una u otra manera.

Esta clase es algo más compleja en lo que a comportamientos se refiere. El comportamiento principal se deriva del protocolo Contract-Net, para comunicarse con los agentes de Horarios. A su vez, este comportamiento está dentro de uno cíclico, que hace algunas cosas más. Hay también otro comportamiento cíclico para tratar las cancelaciones.

Antes que nada, al ejecutar el agente, se comprueba el nombre del mismo en la plataforma, puede ser M, A o E. Para cada caso, se actualizan unas variables que indican de qué tipo de agente se trata. Luego se añade el comportamiento RecibeDeCt(), que ahora comentaremos. Posteriormente se añade otro comportamiento cíclico, que es el siguiente:

- No tiene nombre, porque se define directamente en el setup( ) de la clase. Lo primero que hace es esperar un mensaje en el que se indique que hay una petición nueva. En este caso no bloqueamos el agente, porque al haber más de un comportamiento, hemos de permitir que se vayan ejecutando concurrentemente. Si se recibe un mensaje, procedente de CM, que indica que hay que tratar una nueva petición, se entra propiamente dentro del comportamiento. Primero se actualiza una variable que indica el día actual, que servirá para el tema de las cancelaciones. Se espera la petición en sí, desde CM. Entonces se buscan todas las personas que hay en el sistema de un determinado tipo, mediante el DF. A buscará enfermeras, E anestesistas y M buscará médicos de la especialidad indicada en la petición. En caso de encontrar personal, se les indica a estas personas que recibirán una nueva petición y se inicia el protocolo Contract-Net con los agentes de Horarios. Si no hay personal, se informa a CM, para que cancele la operación y se espera confirmación.

- **EnviaContractNet**: Este comportamiento, implementa la parte iniciadora del protocolo Contract-Net extendiendo el FipaContractNetInitiatorBehaviour. Hay dos métodos implementados, que son para manipular las propuestas que lleguen desde abajo, y para manipular los mensajes finales de los agentes de horarios. El protocolo se inicia enviando a los agentes de Horarios la petición de transplantes, para que estos realicen propuestas de horas libres. Una vez llegan las propuestas, entra en acción el primero de los dos métodos. Se juntan todos los intervalos propuestos, y se le envía a CM. Luego se espera a que CM informe de la hora definitiva de la operación. Si no se ha de cancelar el proceso, se envía a los agentes de Horarios la hora definitiva, para que éstos, según los posibles huecos que pudieran quedar si se aceptase la operación, retornen una puntuación. Una vez obtenidas estas puntuaciones, se busca el número de personas adecuado con la mayor puntuación posible, y a éstos se les envía un mensaje de aceptación de propuestas. El resto de propuestas se rechazan. En caso de que haya habido algún error, si falta algún tipo de empleado o algo así, se cancela el proceso y se para el protocolo. Cuando los empleados cuya propuesta ha sido aceptada reciben esta aceptación, indican que se ha efectuado la actualización del horario, y entonces se

ejecuta el segundo método. Cuando MAE recibe la confirmación de todos los elementos necesarios, se informa a CM de que todo es correcto. Cuando una operación es confirmada por todos los elementos, el agente MAE guarda en una tabla toda la información de la operación para poder cancelarla luego.

- **RecibeDeCt**: Este comportamiento es exclusivo para las cancelaciones, y se inicia cuando se recibe uno de estos mensajes desde CT. Cuando esto ocurre se actualiza la tabla donde se guarda el histórico de operaciones confirmadas, para comprobar si éstas pertenecen al día actual, y si no es así, eliminar las operaciones de días pasados. En el mensaje de CT, se pasa la hora de la operación que se quiere cancelar. Entonces se busca esta operación en el histórico, y si se encuentra, se envía un mensaje a todas las personas implicadas para que actualicen de nuevo su horario. Cuando se reciban todas las confirmaciones de los agentes de Horarios, se informa a CT. En caso de que no se haya encontrado la operación en el histórico, se informa a CT de que esa operación no ha sido concertada.

4.3.5. Quirof.java

Esta clase está implementada únicamente para el Coordinador de Quirófanos (Q). Es muy similar a la clase Mae.java, de hecho lo único que cambia es el método para evaluar las propuestas en el protocolo Contract-Net. Por tanto, esta es la única parte que trataremos aquí, el resto es igual, sólo que obviamente tratando con los agentes de Horarios de Quirófanos, y con CE en lugar de con CM.

- **EnviaContractNet( )**: Comentamos pues la forma de evaluar las propuestas de esta clase. Al recibir todas las propuestas de horas libres de quirófanos, directamente se evalúan para encontrar las horas en que se puede operar en cualquiera de los quirófanos. Estas horas libres son las que se envían a CE, para que las junte con las horas libres del personal. Entonces se reciben las posibles horas libres en las que se podrá operar, para que sea este agente el que elija. Si todo es correcto hasta el momento, es decir, hay algún quirófano libre, que a su vez concuerda con alguna hora libre del personal, no hay que cancelar. Es aquí cuando se tiene que escoger la hora. Para esto se hace una encuesta entre los quirófanos que han hecho propuestas, para saber qué hora es la mejor y en qué quirófano. Se escoge el quirófano y hora con mayor puntuación, se acepta esta propuesta y se rechazan las otras. Se espera confirmación de la propuesta aceptada y se envía a CE dicha confirmación, donde se indica la hora y el quirófano donde se hará la operación.

4.3.6. Horario.java

Esta es la clase que sirve para manipular los horarios de médicos, anestesistas y enfermeras. Para los tres tipos es exactamente igual, lo único que cambia es que habrán de comunicarse con lo que llamaremos padre, es decir M, A o E, dependiendo del tipo de usuario que sea, lo que se indicará al hacer la subscripción al DF.

Para esta clase se han implementado hasta cinco comportamientos. Como en los dos casos anteriores, uno de estos comportamientos pertenece al protocolo Contract-Net, pero esta vez extendiendo la clase FipaContractNetResponderBehaviour. Como antes también, este comportamiento está dentro de otro comportamiento cíclico, que se activa cuando se recibe un mensaje de nuevo órgano a tratar, pero que no bloquea el

agente, para que se puedan ejecutar concurrentemente el resto de comportamientos. El resto de comportamientos son:

- **Subscripcion( )**: Se encarga de suscribir al agente en cuestión al DF. Se pide por pantalla el tipo de agente del que se trata: médico, anestesista o enfermera. En el primer caso, se pregunta además la especialidad, de entre cuatro posibles por el momento: corazón, riñón, pulmón e hígado. Luego se pregunta el nombre del propietario. Con estos datos, se registra el agente al DF, con lo que será fácil hacer la búsqueda de médicos, enfermeras o anestesistas. Finalmente, informa por pantalla de la suscripción. Este comportamiento sólo se ejecuta una vez, al iniciar el agente.

- **RecibeMenu( )** Este comportamiento cíclico, tan sólo espera a que le lleguen mensajes pidiendo que ejecute el menú, pero sin bloquear el agente. Si llega uno de estos mensajes, ejecuta el menú. Las opciones del menú son: visualizar el horario, modificar un día, inicializar toda la semana, resetear toda la semana y modificar el día de hoy. Estas opciones las comentaremos más adelante. Cada vez que se ejecuta el menú, se actualiza una variable que indica el día actual, con lo que se borran las citas temporales y los transplantes del día anterior. Se supone que esto se realiza al menos una vez al día, por lo que sólo es necesario actualizar el día anterior.

- **Cancelar( )**: Aquí se espera, también sin bloquear, un mensaje del padre en que se pida una cancelación. Lo que se hace simplemente es actualizar el horario, para poner a libre las horas que antes contenían un transplante, e informar al padre cuando se haya hecho. Comportamiento cíclico.

- **RecibeContractNet( )**: Este es el comportamiento que se encarga del protocolo Contract-Net. Hay dos métodos implementados aquí, uno para elaborar las propuestas que pide el padre, y otro para manipular los mensajes de aceptación de propuestas. Vamos con el primero. Cuando se recibe un cfp desde el padre con la hora inicial, final y duración de la operación, el agente busca huecos libres, dentro de ese intervalo y con la duración mínima. También actualiza el día actual, para borrar las citas temporales de días anteriores. Los huecos encontrados los envía al padre siguiendo el protocolo y espera a recibir aceptación o rechazo. Cuando se recibe un rechazo no se hace nada, simplemente no se actualiza el horario y ya está. Cuando lo que se recibe es una aceptación, se cambia el horario, marcando las horas libres correspondientes como horas de transplante e informando de la actualización al padre.

- **RecibeDeArriba( )**: Hemos hablado en varias ocasiones ya de las puntuaciones de una posible hora dependiendo del horario. Este comportamiento cíclico es el que determina esa puntuación. Aquí como siempre, se espera, sin bloquear, un mensaje del padre pidiendo una puntuación para una hora determinada. Pues bien, hay cuatro puntuaciones: 0 => Indica que la operación no se puede realizar a la hora requerida porque no cabe. 3 => La operación encaja perfectamente en el hueco, es decir no queda hueco ni por arriba ni por abajo. Se aprovecha al máximo el tiempo. 2=> Si se hace la operación a esa hora, al usuario le queda un hueco, ya sea antes o después de la operación. 1 => Queda hueco antes y después de la operación. Es el peor de los tres casos, ya que el usuario no

aprovecha el tiempo. Una vez se ha decretado esta puntuación para la hora solicitada, se envía al padre.

4.3.7. HorarioQ.java

Esta clase es casi idéntica a la anterior, pero cambian algunas pequeñas cosas, por lo que se ha decidido hacerlo con otra clase distinta. Sirve para los agentes de los horarios de quirófanos. Por ejemplo, los comportamientos de Cancelar( ) y RecibeMenu( ) son idénticos. Cambia un poco el de Subscripcion( ), que no pregunta de qué tipo de agente se trata, porque ya se sabe que es un quirófano. La parte correspondiente al Contract-Net, también es igual, lo único que cambia es el RecibeDeArriba( ), que aquí se llama RecibeDeQ( ).

- **RecibeDeQ( )** El cambio más sustancial en este comportamiento, es que aquí el agente tiene que poner puntuación a una serie de horas, no es una sola como en el agente anterior. Recordamos que Q tiene que elegir hora y quirófano para la operación, lo que implica que este agente tenga que evaluar distintas horas para cada quirófano. Finalmente se queda con la mejor puntuación de todas y la envía a Q.

4.4. Características y Funcionalidad del Sistema

En todo momento se ha procurado que el sistema se asemeje lo máximo a la realidad. Sus características son:

- Permite programar operaciones de transplante, reclutando al personal médico necesario y reservando quirófano para la operación.

- Indica por pantalla el resultado de la búsqueda. En caso de poder realizarse la operación, se informa de detalles como son: la hora de operación, de qué órgano se trata, en qué quirófano se realiza y por quién, es decir: médicos, anestesistas y enfermeras. En caso de resultar infructuosa la búsqueda, se detallan los motivos del fracaso.

- Permite añadir y eliminar usuarios al sistema dinámicamente, gracias a la interficie gráfica de JADE. Al añadir un usuario, automáticamente se registra al DF. Se pide el tipo de agente por pantalla y se registran los servicios de este agente.

- Un usuario tiene plena interacción con su horario. Se pueden hacer consultas y modificaciones. Se muestra por pantalla el horario, en intervalos de media hora y para toda una semana. El usuario puede escoger entre modificar toda la semana, modificar un solo día, o simplemente modificar el día actual.

- Permite la cancelación de una operación después de ser confirmada por todas las partes implicadas, actualizando los horarios anteriormente modificados.

- Gracias al sistema utilizado, el proceso se realiza en pocos segundos, agilizando así el proceso de planificación de un transplante.

- Los horarios se actualizan diariamente, eliminando las citas no habituales o los transplantes que se hayan concertado en días anteriores. Para ver los posibles tipos de citas (temporales, fijas o transplantes), se explican en el Capítulo 5: Interficie con el Usuario.

- Se pueden tratar más órganos, tan sólo ampliando la pequeña base de datos que tiene el sistema, que es en realidad una simple tabla. Debido a que no sabemos demasiado sobre el tema de los transplantes, y este no es el objetivo del proyecto, aquí sólo se tratan cuatro posibles órganos: corazón, pulmón, hígado y riñón. En la base de datos tenemos la cantidad de personal necesaria para cada órgano.

- Se procura que la asignación de quirófanos y personal sea lo más inteligente posible, en el sentido de asignar la operación en los huecos más pequeños del horario de un usuario, para que éstos sean más compactos.

- Hay una cola de órganos pendientes, que permite al sistema recibir peticiones mientras se va procesando una. No se hace el proceso de varias peticiones concurrentemente, debido a que el tiempo de respuesta es muy pequeño y se complicaría el proceso sin necesidad.

- El sistema es capaz de parar la búsqueda e informar de ello cuando note que falta algún tipo de elemento para realizar la operación.

4.5. Seguimiento del Proceso Completo

Hasta ahora hemos visto más o menos por partes como se realiza el proceso, y ya nos hemos hecho una idea, pero veámoslo ahora de forma global, viendo todos los pasos y las etapas por las que pasa una petición, desde que es recibida por el Coordinador de Transplantes hasta que finaliza el proceso.

4.5.1. Pasos de una Petición de Transplante

Para seguir el tratamiento de una petición de transplante, pondremos un ejemplo concreto, para ver todos los pasos del proceso. Tenemos unos horarios de médicos, anestesistas, enfermeras y quirófanos, en la figura 11, donde las casillas coloreadas son intervalos ocupados.

	M1	M2	M3	M4	M5	M6		A1	A2	A3	A4	A5	A6		E1	E2	E3	E4	E5	E6		Q1	Q2	Q3
2:00							2:00							2:00							2:00			
2:30							2:30							2:30							2:30			
3:00							3:00							3:00							3:00			
3:30							3:30							3:30							3:30			
4:00							4:00							4:00							4:00			
4:30							4:30							4:30							4:30			
5:00							5:00							5:00							5:00			
5:30							5:30							5:30							5:30			
6:00							6:00							6:00							6:00			

Figura 11 - Horarios para simular una petición

En una petición intervienen todos los agentes del sistema, por tanto tenemos el esquema de la figura 12.

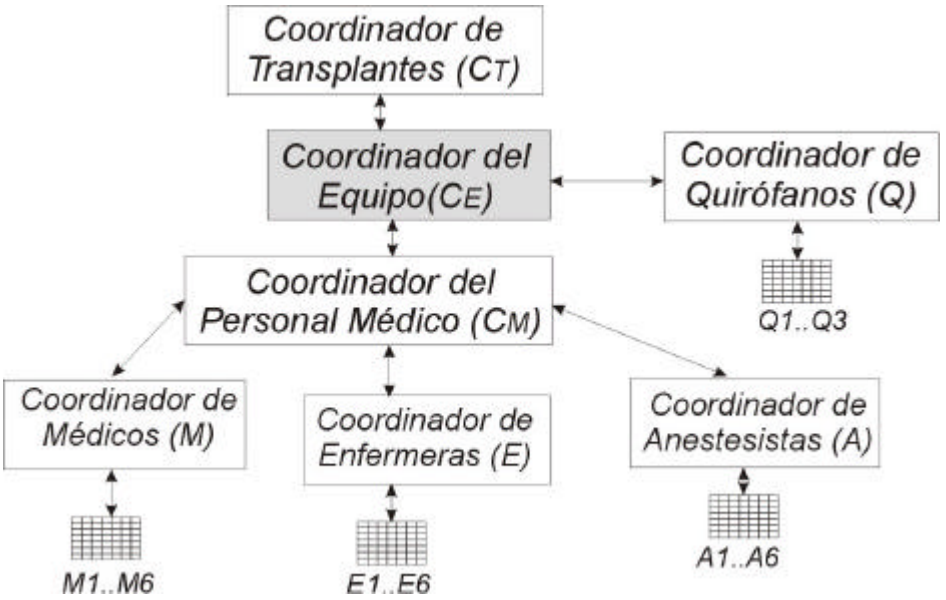


Figura 12 - Esquema de agentes en una petición

Para empezar el proceso, es necesario que CT reciba un mensaje del Coordinador General (C), con el formato "(hora_llegada hora_caducidad duración_operación órgano)", entonces lo meterá en la cola de órganos pendientes. En este caso, el mensaje que le enviamos tiene como contenido: "(02:00 06:00 01:00 corazón)". Debido a la longitud del mensaje, CT sabe que se trata de una petición y no de una cancelación. Cuando CE esté preparado para recibir una nueva petición, CT se la envía. Por su parte, CT sigue recibiendo peticiones desde C y metiéndolas en la cola.

CE recibe la petición y la descompone. Por un lado se guarda el tipo de órgano ("corazón") y por otro las horas de llegada, caducidad y duración ("(02:00 06:00 01:00)"). Envía las horas a Q, y a CM le envía todo, horas y órgano. Podemos ir siguiendo el proceso por los gráficos.

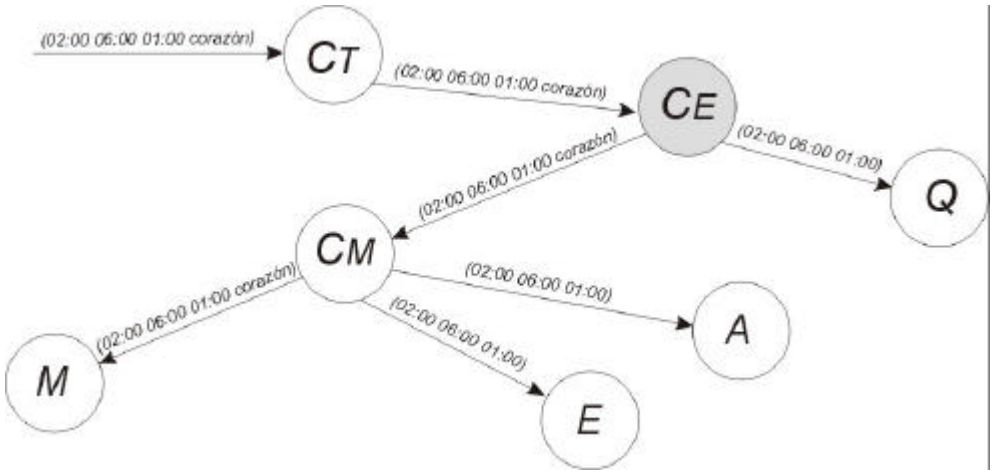


Figura 13 - Primeros pasos de una petición

CM hace lo mismo que CE con la petición, la descompone y envía sólo las horas a A y E, y todo el contenido a M. M es el único que necesita saber de qué órgano se trata, porque sólo tiene que contactar con los médicos especialistas (ver figura 13).

Ahora es cuando Q, M, A y E tienen que iniciar el protocolo Contract-Net con los agentes de Horarios. Recordemos que el Contract-Net consiste en que la parte iniciante, en este caso tomemos como ejemplo a Q, envía un cfp a sus hijos en busca de propuestas. Éstos le mandan sus propuestas, y después de evaluarlas, Q acepta algunas y rechaza el resto. Finalmente se realiza la acción expuesta en la propuesta. Veamos aquí el Contract-Net paso por paso. Q busca en DF todos los agentes que sean del tipo quirófano. Después envía un cfp con la petición como contenido a estos agentes. Los agentes Q1, Q2 y Q3 estudian su horario para hacer propuestas a Q, y le mandan las horas libres que tienen, cumpliendo las condiciones impuestas por Q.

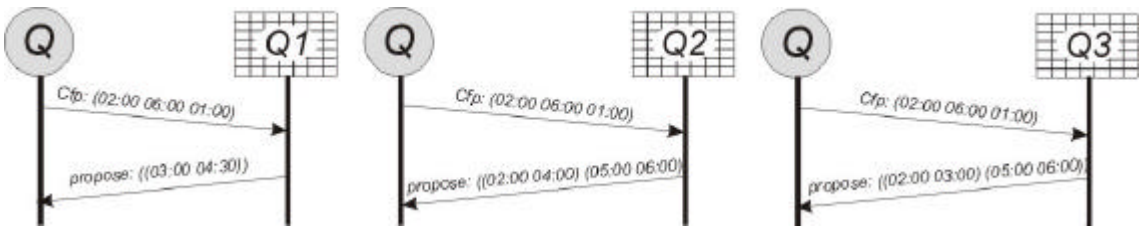


Figura 14 - Inicio del Contract-Net para quirófanos

Cuando Q ha obtenido todas las propuestas, las junta, para conseguir las horas posibles para operar, en que hay quirófano disponible. Q envía este conjunto de horas disponibles a CE. Lo mismo que ha hecho Q con el Contract-Net lo hacen M, A y E con sus respectivos hijos. M, A y E pasan las propuestas hacia arriba, a CM (figura 15.a). CM junta todos estos intervalos, teniendo en cuenta la cantidad necesaria de cada tipo de personal. Se obtienen así las horas en que hay personal disponible.

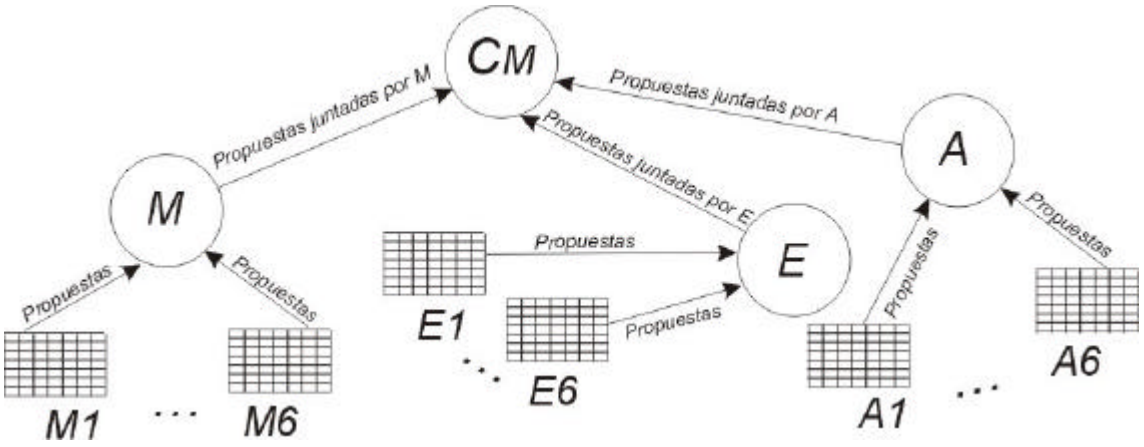


Figura 15.a

La situación ahora es la siguiente: CE tiene todas las horas en las que hay algún quirófano libre. CM por su parte tiene las horas donde hay personal suficiente disponible. CM envía estas horas a CE, que las junta con las obtenidas de Q y consigue las posibles horas en que se pueda operar, porque hay personal suficiente y quirófano disponible. Las horas con personal compatible suficiente son: *"((03:30 04:30)(04:00 05:00)(05:00 06:00))"*. Las de quirófano disponible: *"((02:00 03:00)(02:30 03:30)(03:00 04:00)(03:30 04:30)(05:00 06:00))"*. Por tanto las horas en que será posible operar son: *"((03:30 04:30)(05:00 06:00))"*.

Figura 15.b

Q se encuentra en este momento esperando respuesta de CE, lo que espera es que CE le envíe las horas posibles para operar, de las que Q podrá elegir la que más le convenga. Cuando Q obtiene estas horas posibles, realiza una encuesta entre sus hijos, para que den una puntuación a cada hora (figura 16.a). Esta comunicación entre Q y sus hijos es independiente del Contract-Net (figura 16.b). Recibe todas las puntuaciones y elige la hora y el quirófano con mayor puntuación. Informa de su elección a CE, e indica al quirófano aceptado que actualice su horario, al resto les envía el mensaje de rechazo del Contract-Net, con esto concluye este protocolo.

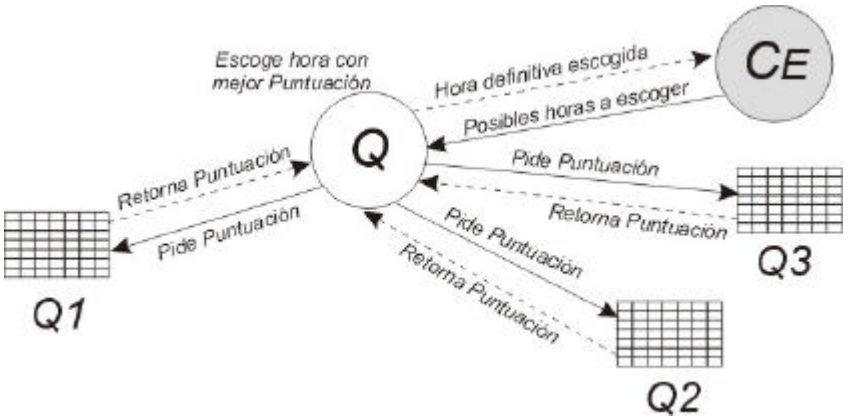


Figura 16.a

En nuestro caso, la mejor puntuación (3) se obtiene de 5:00 a 6:00 en los quirófanos Q2 y Q3. Por lo tanto Q escoge uno de los dos, supongamos, por ejemplo, Q2. Q envía un accept-proposal a Q2, indicando la hora escogida, y a los otros dos quirófanos les envía un reject-proposal. Para finalizar el protocolo Contract-Net, Q2 confirma que ha actualizado el horario, ocupando el intervalo de 5:00 a 6:00. Cuando Q recibe la confirmación, le indica a Q la hora escogida, además del quirófano donde se operará.

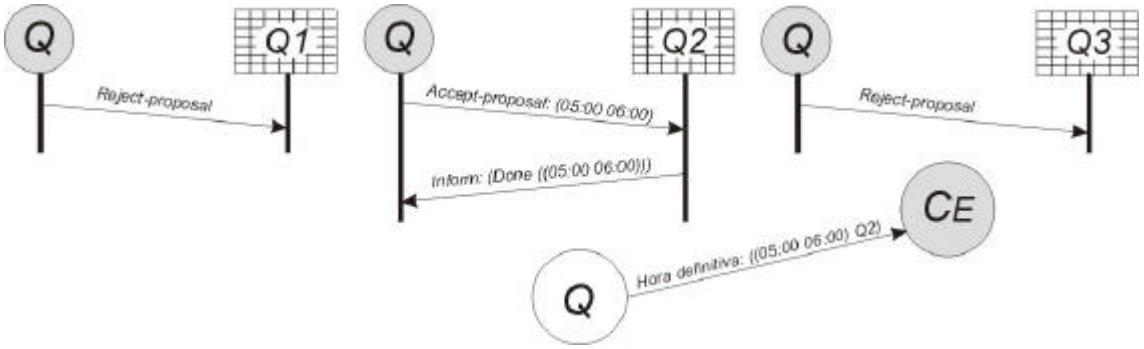


Figura 16.b - Final del Contract-Net

CE ya tiene la hora a la que será la operación, se la pasa a CM. CM se la envía a M, A y E, indicando también el número de personas de cada tipo necesarias. En este caso, para operar un corazón se necesitan 2 médicos, 3 anestesistas y 2 enfermeras. M, A y E realizan la encuesta como lo hizo Q, para ver a que personas les va mejor hacer la operación a la hora indicada. Se obtienen las mejores puntuaciones de M4 y M6 (3 puntos), A1 (3 puntos), A2 y A6 (2 puntos), E2 y E3 (2 puntos).

Una vez obtenidas las mejores puntuaciones, se envían accept-proposals a las propuestas aceptadas, y reject-proposals al resto. Esta parte es igual que el Contract-Net con Q. Los agentes cuya propuesta ha sido aceptada, envían confirmación de que han actualizado el horario a sus padres. M, A y E confirman a CM que todos los horarios han sido actualizados, e informan además del personal de la operación. CM sabe ya todo el personal que actuará en la operación, e informa de ello a CE.

Cuando CE ha obtenido las confirmaciones de CM y de Q, ya tiene toda la información acerca de la operación. La pasa a CT, indicando también que ya está

preparado para recibir otra petición. CT muestra la información por pantalla y manda otra petición. Así finaliza el proceso.

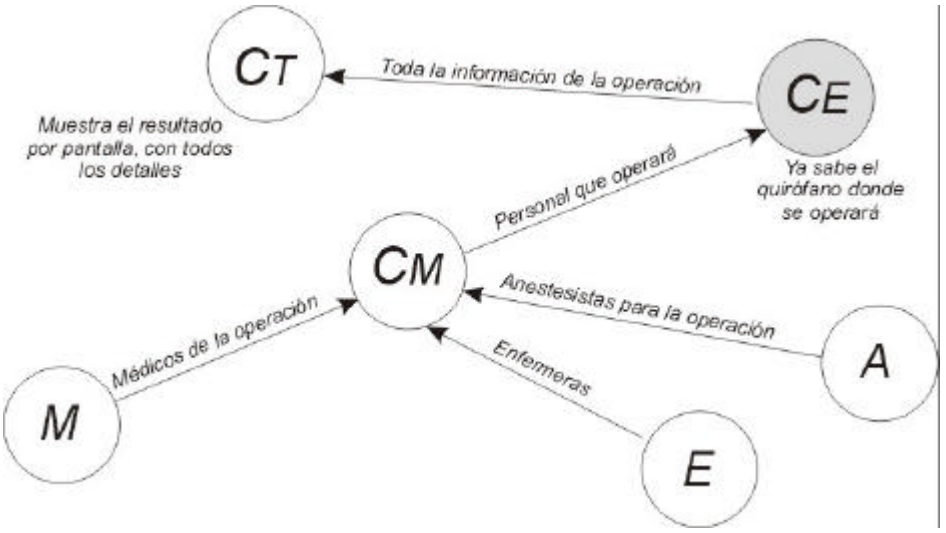


Figura 17 - Parte final del proceso de petición

En caso de que durante el proceso haya habido algún error, o sea, falte algún elemento o no se hayan podido coordinar todos, se cancela el proceso. En este caso, CE informa a CT de que la operación no se puede realizar, indicando los motivos.

También hay que ver los pasos que sigue la cancelación de una operación ya programada.

4.5.2. Pasos de una Cancelación

En el proceso de cancelación no intervienen CM ni CE, por lo tanto el esquema queda como en la figura 18.

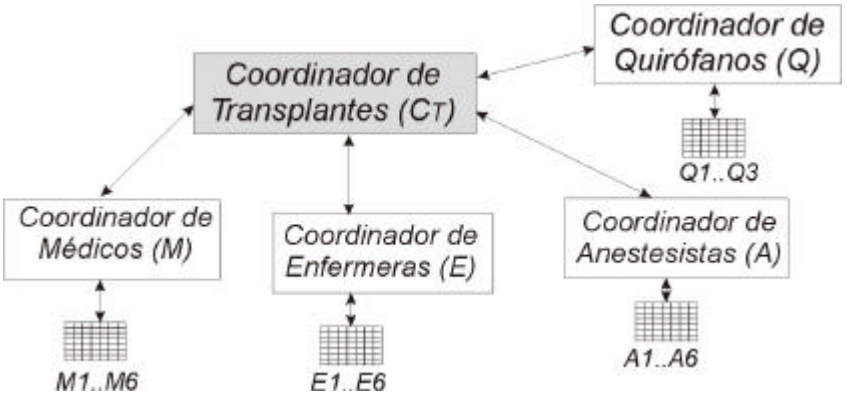


Figura 18 - Esquema del SMA para una cancelación

Al igual que con la petición, la cancelación al llegar a CT se encola, esperando a que el sistema esté preparado. Entonces CT envía la cancelación a M, A, E y Q.

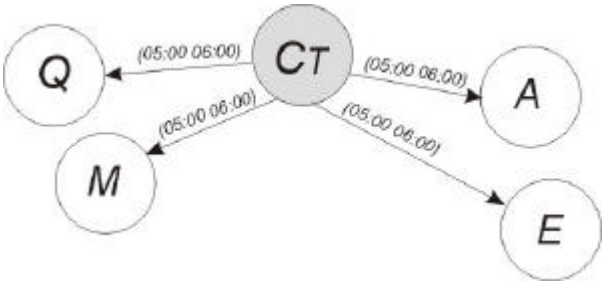


Figura 19 - Inicio del proceso de Cancelación

Los coordinadores reciben la cancelación, y buscan los datos de la operación correspondiente en una tabla donde se guardan las operaciones programadas. Esta información es la dirección de los agentes que intervienen en la operación. A estos agentes se les envía un mensaje de cancelación, indicando la hora de la operación que quieren cancelar.

Los agentes de Horarios actualizan los mismos, y lo confirman al padre correspondiente. Cuando M, A, E y Q tienen todas las confirmaciones, informan a CT, para que muestre el resultado por pantalla.

En caso de que haya algún error, que podría ser que no se encuentre la operación a cancelar en la tabla de operaciones programadas, se cancela el proceso, informando de que la operación no se puede cancelar porque no había sido confirmada previamente.

Hay una cosa a comentar en el caso de las cancelaciones. Se ha pensado en ello, pero no se ha podido corregir por problemas de tiempo. Cuando se pide una cancelación, se hace usando la hora programada para la operación. El problema es que puede darse el caso de que haya dos o más operaciones en distintos quirófanos exactamente a la misma hora. Es algo no muy probable pero posible y que se tendría que tener en cuenta. Se había pensado asignar algún tipo de identificador a cada operación programada, pero a última hora se ha visto que esto provocaría muchos cambios en el código y debido a la escasez de tiempo, se ha optado por dejarlo como un trabajo a realizar en el futuro.

5. INTERFICIE CON EL USUARIO

En este sistema, la única interficie que hay con el usuario, a parte de la propia del Jade, es la que nos ofrecen los agentes coordinadores de horarios.

La interficie de Jade es fundamental para la simulación del sistema. Con ella podemos añadir nuevos agentes al sistema, pararlos, eliminarlos, etc. Además permite enviar los mensajes para simular las peticiones y cancelaciones desde el Coordinador General.

Por otro lado, los únicos agentes que se comunican con el usuario son los de horarios. Se comunican en dos ocasiones. Primero en la suscripción del agente, se preguntan al usuario datos como el tipo de servicio que realiza o el propietario del agente. Segundo, tenemos el menú, con el que el usuario puede consultar y modificar su horario. Este menú se activa enviando un mensaje al agente deseado, cuyo contenido es "(Menu)". El mensaje se envía desde la interficie del Jade. Las opciones del menú, que podemos ver en la figura 20, son las siguientes:

1. **Visualizar Horario**: Se muestran seis horas del horario, que está dividido en intervalos de 30 minutos. Para ello se le pregunta al usuario a partir de qué hora quiere visualizarlo.

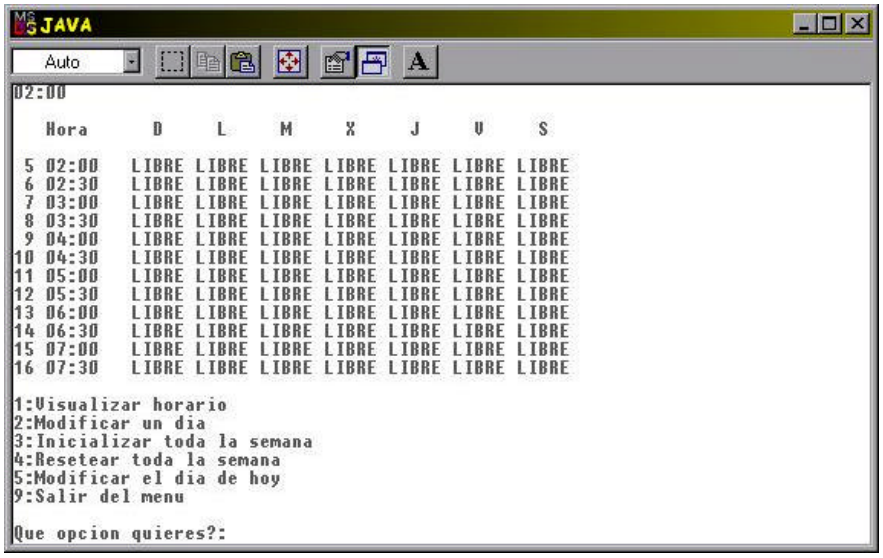


Figura 20 - Interficie con el usuario

2. **Modificar un día**: Se permite a un usuario añadir citas temporales a un día. Se le pregunta el día que quiere modificar, para que luego el usuario indique que intervalos de media hora quiere ocupar.

3. **Inicializar toda la semana**: Para cada día de la semana, se establece un horario, que será fijo hasta que no se vuelva a inicializar la semana. Es decir, cuando se actualiza el día actual, las horas ocupadas por este sistema no se borran. La diferencia entre este tipo de citas y las temporales, es que las primeras se

visualizan en el horario como "OCUP" y las segundas como "TEMP". Hay otro tipo de citas que también son temporales, son los transplantes, indicados como "TRANS".

4. **Resetear toda la semana**: Se pone en blanco todo el horario, dejándolo como estaba antes de introducir alguna cita.

5. **Modificar el día de hoy**: Igual que la opción 2, pero salta directamente al día actual, para que se modifique.

La interficie es muy sencilla, pero tampoco se necesita más para el manejo de los horarios. Quizás se podría haber hecho en modo gráfico, pero no es eso lo que se pretende en el proyecto, y con modo texto ya hay más que suficiente.

6. JUEGO DE PRUEBAS

Para comprobar el buen funcionamiento del sistema, se han hecho una serie de pruebas. Algunas en las que se puede encontrar hora para operar, y otras en cambio en las que es imposible encontrar algún hueco, por distintos motivos.

Hemos ejecutado la siguiente instrucción para ejecutar el sistema:

```
 java jade.Boot -gui -platform Ct:Coord Ce:Equipo Cm:Personal
M:Mae A:Mae E:Mae Q:Quirof M1:HoraM M2:HoraM M3:HoraM M4:HoraM
M5:HoraM M6:HoraM  A1:HoraA A2:HoraA A3:HoraA A4:HoraA A5:HoraA
A6:HoraA E1:HoraE E2:HoraE E3:HoraE E4:HoraE E5:HoraE E6:HoraE
Q1:HoraQ Q2:HoraQ Q3:HoraQ
```

Hay que comentar que HoraM, HoraE, HoraA y HoraQ, son unas clases expresamente modificadas para hacer las pruebas, que ya contienen un horario inicializado con lo que a nosotros nos interesa, figura 21.

Para ejecutar el sistema sin iniciar ningún horario sería con la instrucción:

```
java jade.Boot -gui -platform Ct:Coord Ce:Equipo Cm:Personal M:Mae
A:Mae E:Mae Q:Quirof
```

Entonces se iniciarían los agentes principales del sistema, sin ningún agente de horario. Los agentes de médicos iniciados, son todos especialistas de corazón en estas pruebas.

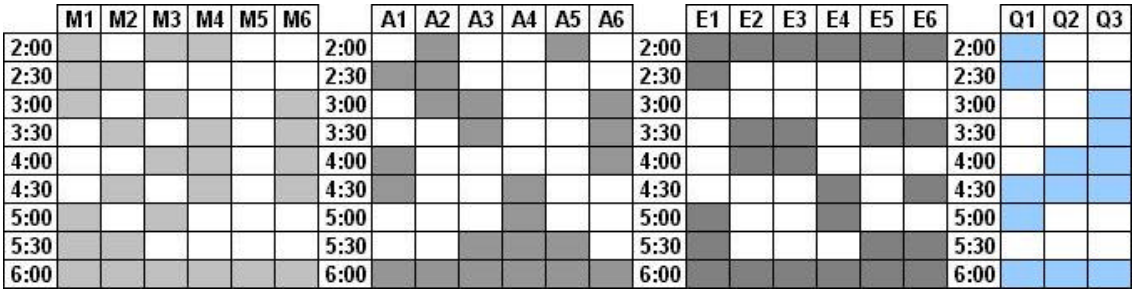


Figura 21 - Horarios para pruebas

A partir de este horario se ejecutan una serie de pruebas, veámoslas:

1.- Se envía un mensaje a "CT" desde "C" con *"(02:00 06:00 03:00 corazon)"* como contenido. El sistema tendría que concluir que no se puede realizar la operación, ya que no hay ningún hueco de 3 horas ni en los quirófanos ni en el personal necesario para esta operación de corazón (2 médicos, 3 anestesistas y 2 enfermeras). La respuesta obtenida por el sistema, que es correcta, se puede ver en la primera parte de la figura 22.

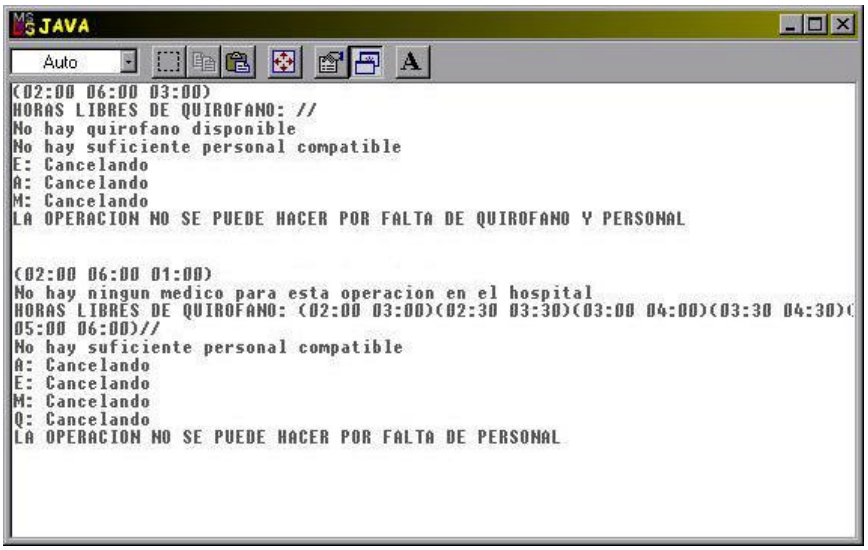


Figura 22 - Respuesta a las pruebas 1 y 2

2.- Ahora se busca hacer una operación de riñón. Se envía *"(02:00 06:00 01:00 rinyon)"*. Lo que tiene que ocurrir aquí es que el sistema diga que no se puede hacer. En este caso tenemos quirófano disponible (parte de abajo de la figura 22), tenemos anestesistas y enfermeras. Lo que nos faltan ahora son médicos especialistas en riñón, ya que como hemos dicho, todos los médicos que hay en estas pruebas son de corazón. El sistema nos indica que no hay médicos para este tipo de operación en el hospital y que no se puede realizar por falta de personal. Correcto.

3.- Ahora comienzan las pruebas donde se obtendrán resultados afirmativos. Enviamos *"(02:00 06:00 01:00 corazon)"*. Hay varias posibles horas para esta operación teniendo en cuenta los horarios. Se puede realizar en cualquiera de los tres quirófanos y a diversas horas. El sistema nos indica (parte de arriba de la figura 23), las horas libres que tiene el quirófano, y las horas en que se puede conseguir el personal compatible necesario. Esto lo coordina y nos da como resultado que la operación será de 5:00 a 6:00 en el quirófano Q3, con el personal que se nos muestra.

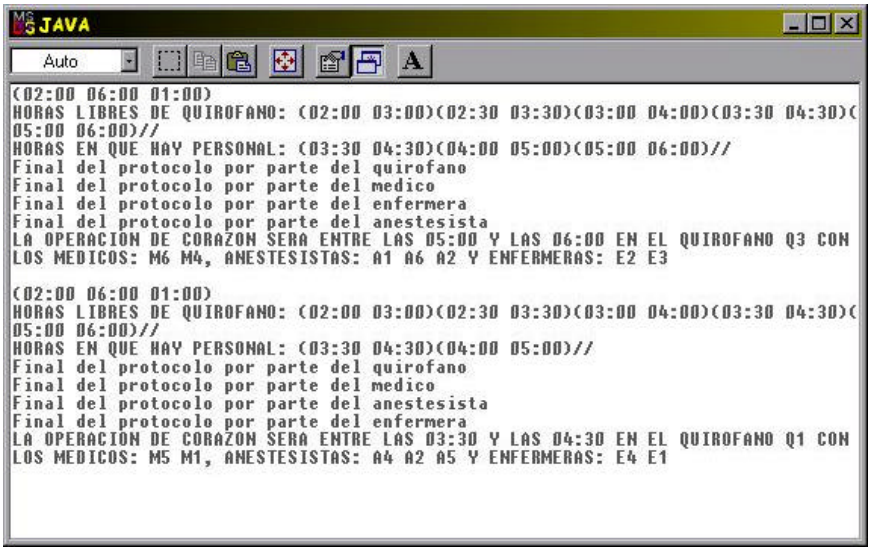


Figura 23 - Respuesta a las pruebas 3 y 4

4.- Ya tenemos algunos huecos ocupados. Vamos a enviar otra petición igual a la de antes *"(02:00 06:00 01:00 corazon)"*. De nuevo tenemos algunas posibilidades de elección, como se aprecia en la parte de debajo de la figura 23. Pero la única hora posible que coordina quirófano y personal es la elegida: de 3:30 a 4:30 en el quirófano Q1. Una vez programadas estas dos operaciones, los horarios nos quedan de la siguiente manera:

	M1	M2	M3	M4	M5	M6		A1	A2	A3	A4	A5	A6		E1	E2	E3	E4	E5	E6		Q1	Q2	Q3
2:00							2:00							2:00							2:00			
2:30							2:30							2:30							2:30			
3:00							3:00							3:00							3:00			
3:30	4				4		3:30		4		4	4		3:30	4			4			3:30	4		
4:00	4				4		4:00		4		4	4		4:00	4			4			4:00	4		
4:30							4:30							4:30							4:30			
5:00				3		3	5:00	3	3				3	5:00		3	3				5:00			3
5:30				3		3	5:30	3	3				3	5:30		3	3				5:30			3
6:00							6:00							6:00							6:00			

Figura 24 - Horarios con algunas operaciones

5.- Las operaciones están marcadas en los horarios con el número de la prueba correspondiente. Ahora ya no quedan demasiados huecos a priori, para otra operación de una hora. Comprobemos que pasa si pedimos otra, *"(02:00 06:00 01:00 corazon)"*. Aunque se podría obtener un quirófano libre, por ejemplo Q3 de 2:00 a 3:00, o Q2 de 2:00 a 4:00 y de 5:00 a 6:00, el problema aquí es que no podemos coordinar al personal médico suficiente. En la figura 25 arriba tenemos los motivos del rechazo de la petición:

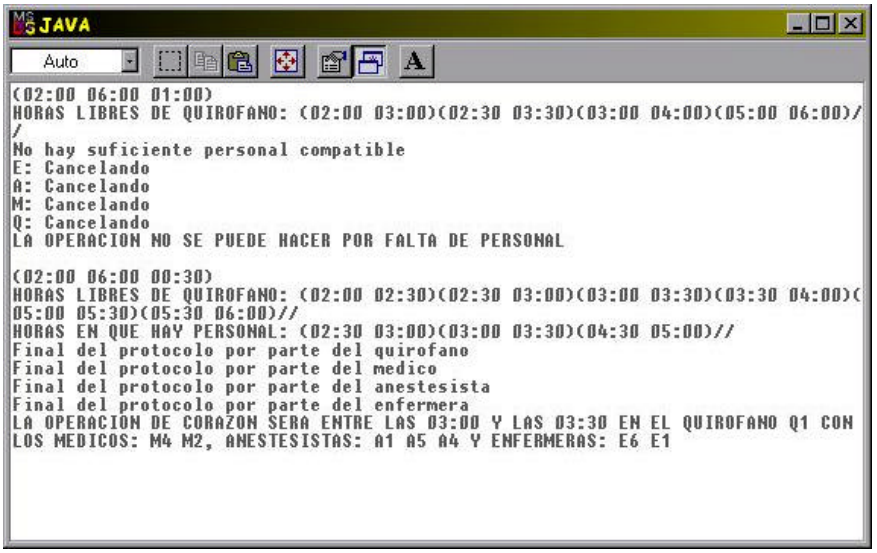


Figura 25 - Respuesta de las pruebas 5 y 6

6.- Ahora optamos por seguir haciendo peticiones que puedan ser aceptadas. Como tenemos que en los horarios ya no caben más operaciones de una hora o más, probamos de pedir una de media hora, siempre de corazón y en el intervalo que se ve en el horario, *"(02:00 06:00 00:30 corazon)"*. El resultado ahora vuelve a ser satisfactorio. Con media hora de duración, hay más posibilidades de elección, el sistema se decanta por programar la operación de 3:00 a 3:30 en el quirófano Q1, como se ve en la segunda parte de la figura 25.

7.- Seguimos con más operaciones de media hora, hasta que ya no quepan más, para ver que ocurre. Nos cabe otra más aún, *"(02:00 06:00 00:30 corazon)"*. Será de 2:30 a 3:00 en el quirófano Q3 (figura 26, arriba).

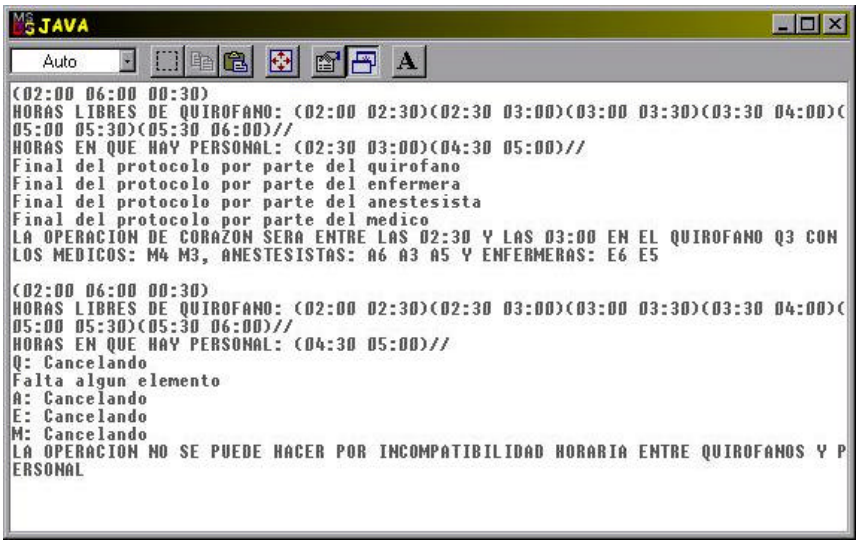


Figura 26 - Respuesta para las pruebas 7 y 8

8.- Se prueba una última operación de media hora, pero esta vez ya no cabe, ya que tenemos el horario muy completo (figura 27). Como se ve en la figura 28 abajo, tenemos quirófano disponible, personal compatible, pero no hay manera de coordinarlo todo, este es el motivo de que no se pueda realizar la operación.

	M1	M2	M3	M4	M5	M6		A1	A2	A3	A4	A5	A6		E1	E2	E3	E4	E5	E6		Q1	Q2	Q3
2:00							2:00							2:00							2:00			
2:30			7	7			2:30			7		7	7	2:30					7	7	2:30			7
3:00		6		6			3:00	6			6	6		3:00						6	3:00	6		
3:30	4				4		3:30		4		4	4		3:30	4			4			3:30	4		
4:00	4				4		4:00		4		4	4		4:00	4			4			4:00	4		
4:30							4:30							4:30							4:30			
5:00				3		3	5:00	3	3				3	5:00		3	3				5:00			3
5:30				3		3	5:30	3	3				3	5:30		3	3				5:30			3
6:00							6:00							6:00							6:00			

Figura 27 - Horarios con todas las operaciones posibles

9.- Ya hemos dicho que ya no caben más operaciones. Probemos ahora de cancelar alguna de las que ya están programadas. Para comprobar que realmente esto sucede, se puede comprobar el horario de M4, por ejemplo, antes y después de la cancelación (figura 28). Se ve que después de indicar que queremos cancelar la operación *"(03:00 03:30)"*, lo que era un "TRANS" en el horario, vuelve a ser un "LIBRE".

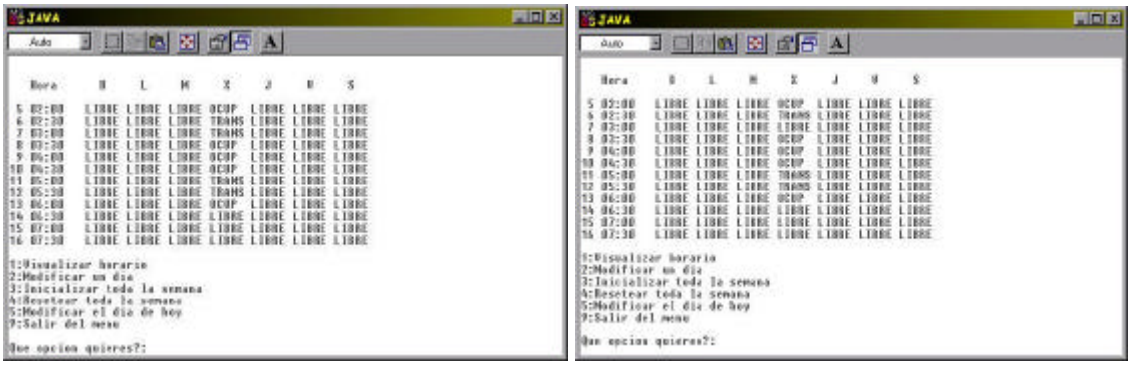


Figura 28 - Horario de M4 antes y después de una cancelación de 3:00 a 3:30

10.- Ya hemos comprobado que las cancelaciones se hacen correctamente. Probemos ahora que ocurre si queremos cancelar una operación no confirmada, por ejemplo de *"(02:00 02:30)"*. El resultado se ve en la parte de arriba de la figura 29. Se busca la operación entre las que se han programado, y como no la encuentra, se nos indica que la operación a cancelar no había sido confirmada antes.

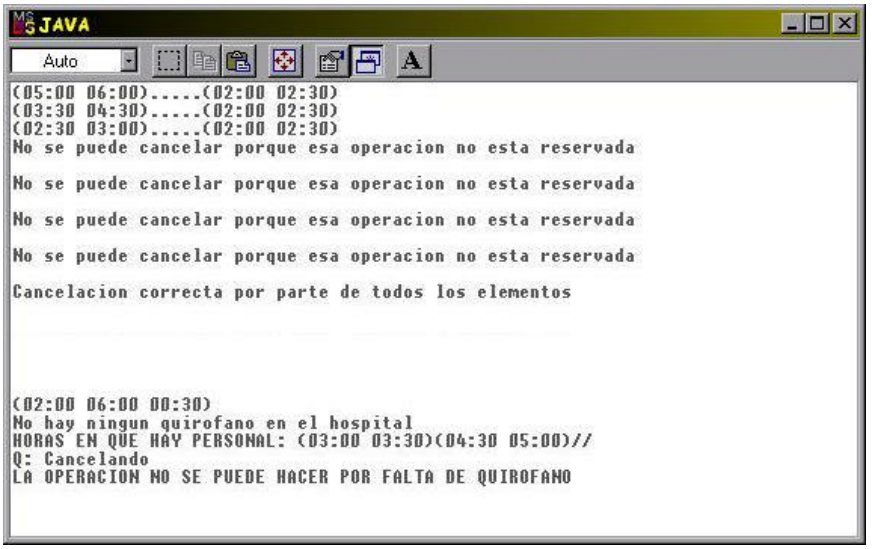


Figura 29 - Respuesta a las pruebas 10 y 11

11.- Finalmente, la última prueba que queda por hacer es comprobar que ocurre si no hay ningún quirófano en el sistema. Para eso eliminamos Q1, Q2 y Q3 del sistema, usando la interficie gráfica del Jade. Una vez eliminados, probamos de programar una operación de media hora *"(02:00 06:00 00:30 corazon)"*, que teóricamente cabría si hubiera quirófano, ya que acabamos de cancelar una igual. El resultado es el de la parte baja de la figura 29. Se aprecia que hay personal compatible, pero se nos informa de que no hay ningún quirófano en el hospital.

Pese a haber hecho todas estas pruebas con operaciones de corazón, se ha comprobado que funciona con todo tipo de órgano y duraciones. Además, el Jade aguanta bien el tener a tantos agentes (28) corriendo en el sistema, sin ningún tipo de problema.

7. CÓDIGO

Ahora veremos el código de todo el sistema, de las siete clases implementadas.

7.1. Coord..java

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

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.acl.ACLMessage;

public class Coord extends Agent {

   String peticiones[] = new String[10];
   int i=0;
   int j=0;
   int k;
   boolean preparado=true;
   boolean med=false;
   boolean enf=false;
   boolean an=false;
   boolean quir=false;

   public class RecibeDeC extends CyclicBehaviour {

      public RecibeDeC() {

      }

      public void action () {//Recibe los mensajes del coordinador externo ("C")y los
mete en una "cola"
        ACLMessage msg=receive(MessageTemplate.MatchSource("C"));
        if (msg!=null) {
          peticiones[i]=msg.getContent();
          i++;
          if (i>9) i=0;
        }
      }
   }

   public class RecibeDeCe extends CyclicBehaviour {//Recibe de Ce la señal de que éste
está preparado

      public RecibeDeCe() {

      }

      public void action () {
        ACLMessage msg=receive(MessageTemplate.MatchSource("Ce"));
        if (msg!=null) {
          if (msg.getOntology().equals("Preparado")) {
            preparado=true;
            System.out.println(msg.getContent());//Muestra por pantalla el resultado de
la petición de transplante
             System.out.println();

          }
        }
      }
   }

   public class RecibeAceptacion extends CyclicBehaviour {//Recibe la confirmación de
una cancelación

      public RecibeAceptacion() {
```

```
    }

    public void action () {

      MessageTemplate mt =
MessageTemplate.or(MessageTemplate.MatchSource("M"),MessageTemplate.MatchSource("E"));
      mt=mt.or(mt,MessageTemplate.MatchSource("A"));
      mt=mt.or(mt,MessageTemplate.MatchSource("Q"));
      mt=mt.and(mt,MessageTemplate.MatchType("agree"));
      int i;
      ACLMessage msg=null;

        //Espera mensajes de confirmación de M, A, E y Q, cuando ha recibido todos, lo
indica por pantalla

      msg=receive(mt);
      if (msg!=null) {
        if (msg.getSource().equals("M"))
          med=true;
        else if (msg.getSource().equals("E"))
          enf=true;
        else if (msg.getSource().equals("Q"))
          quir=true;
        else an=true;

        System.out.println(msg.getContent());
        System.out.println();

        if (med && an && enf && quir) {
          preparado=true; //Al recibir todas las confirmaciones, Ct ya puede
desencolar otra petición
          System.out.println("Cancelacion correcta por parte de todos los elementos");
          med=false;
          enf=false;
          an=false;
        }

      }
    }
  }

  public class EnviaPeticion extends CyclicBehaviour {//Envía un órgano a Ce cuando
está preparado

    public EnviaPeticion() {

    }

    public void action () {
      ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
      if (preparado && !peticiones[j].equals("")) { //Si Ce está preparado y hay
peticiones pendientes
        preparado=false;
        msg.setSource("Ct");
        msg.removeAllDests();
        msg.setContent(peticiones[j]);
        if (peticiones[j].length()<14) {//Se trata de una cancelación
          msg.addDest("M");                //por tanto se le envía a M, A, E y Q
          msg.addDest("E");
          msg.addDest("A");
          msg.addDest("Q");
          msg.setOntology("Cancelacion");
        } else { //Si es una petición de transplante se le envía a Ce
          msg.addDest("Ce");
          msg.setOntology("");
        }
        send(msg);
        peticiones[j]="";
        j++;
        if (j>9) j=0;
      }
    }
  }

  protected void setup() {
  for (k=0;k<10;k++)
    peticiones[k]="";
```

```
   addBehaviour(new RecibeDeCe());//Se añaden todos los comportamientos
   addBehaviour(new RecibeDeC());
   addBehaviour(new RecibeAceptacion());
   addBehaviour(new EnviaPeticion());
   }
}
```

7.2. Equipo.java

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

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.acl.ACLMessage;

public class Equipo extends Agent {

   String organo,str,hora,contenidoFinal;

   public class RecibeDeCt extends CyclicBehaviour {//Único comportamiento de esta clase

      public RecibeDeCt() {

      }

      public void action () {
         int i,j;
         String personal = new String();
         String quirofano = new String();
         String definitivo = new String();
         boolean faltaAlgo=false;
         boolean faltaPersonal=false;
         boolean faltaQuirof=false;
         boolean noHayQuir=false;

         ACLMessage msg = blockingReceive(MessageTemplate.MatchSource("Ct")); //Cuando
está preparado, recibe una petición
      str=msg.getContent();//con el formato (02:00 06:00 01:00 corazon)
      hora=str.substring(1,18);//lo descompone en horas y órgano
      organo=str.substring(19,str.length()-1);

      msg.setSource("Ce"); //Envía la petición hacia abajo, a Cm
      msg.setPerformative(ACLMessage.REQUEST);
      msg.removeAllDests();
      msg.addDest("Cm");
      send(msg);

      ACLMessage msgNuevo= new ACLMessage(ACLMessage.INFORM);
      msgNuevo.setContent("(Organo nuevo)");//Avisa a Q de que le llegará un órgano
nuevo
      msgNuevo.addDest("Q");
      send(msgNuevo);

      msg.setContent("("+hora+")");//Cuando Q ya está avisado, se le envía la petición,
tan sólo las horas,
      msg.removeAllDests();         //el órgano no le interesa
      msg.addDest("Q");
      send(msg);

      MessageTemplate mt =
MessageTemplate.or(MessageTemplate.MatchSource("Cm"),MessageTemplate.MatchSource("Q"));
            //Se espera respuesta de Q y de Cm, con los intervalos posibles de quirófano
y de personal
         for (i=1;i<3;i++) {
         msg = blockingReceive(mt);
         if (msg.getSource().equalsIgnoreCase("Cm"))
            personal=msg.getContent().substring(1,msg.getContent().length()-1); //Tenemos
las horas con personal disponible
         else   // Vendrá de "Q", pq lo comprobamos con el MessageTemplate
            quirofano=msg.getContent().substring(1,msg.getContent().length()-1); //y las
horas con quirófano disponible
         }

         if (quirofano.equals("No hay ningun quirofano en el hospital.")) { //Si no hay
quirófanos regsitrados en el hospital
            quirofano="";                                                    //se
activa un error, noHayQuir
            noHayQuir=true;
         }
```

```
       if (quirofano.equals("")) faltaQuirof=true;//Si no hay quirofano disponible, se
activa un error
       if (personal.equals("")) faltaPersonal=true;//lo mismo si no hay personal

       for (i=0;i<quirofano.length()-12;i+=13)//Se comprueba si concuerdan algún
intervalo de Q y Cm
         for (j=0;j<personal.length()-12;j+=13)
           if (quirofano.regionMatches(i,personal,j,13))
             definitivo+=(quirofano.substring(i,i+13));//Tenemos los posibles
intervalos con personal y quirófano

         //ya está fuera del "for"

       if (definitivo.equals("")) faltaAlgo=true;//Si no hay concordancia entre ningún
intervalo de Q y Cm => error

       if (! faltaAlgo) {//Si todo es correcto
         msg.reset();
         msg.setPerformative(ACLMessage.INFORM);
         msg.addDest("Q");
         msg.setSource("Ce");
         msg.setContent("("+definitivo+")");
         send(msg);  //Se le envía a Q los posibles horarios disponibles para que elija

         mt =
MessageTemplate.and(MessageTemplate.MatchSource("Q"),MessageTemplate.MatchType("inform")
);
             //Se espera respuesta de Q con el horario escogido
         msg=blockingReceive(mt);
         String horaDefinitiva=msg.getContent().substring(8,21); //Cogemos la hora
definitiva escogida por Q
         String
quirofanoDefinitivo=quitarDireccion(msg.getContent().substring(23,msg.getContent().lengt
h()-1));
             //y el quirófano donde se realizará la operación
         msg.setSource("Ce");
         msg.removeAllDests();
         msg.addDest("Cm");
         msg.setContent(horaDefinitiva);
         send(msg);
             //La hora definitiva se le pasa a Cm para que elija el personal
         msg=blockingReceive(MessageTemplate.MatchSource("Cm"));
         String
personalDefinitivo=msg.getContent().substring(1,msg.getContent().length()-1);
             //Se recibe la confirmación de Cm y ya está todo, la operación se
realizará
         contenidoFinal="LA OPERACION DE "+organo.toUpperCase()+ " SERA ENTRE LAS
"+horaDefinitiva.substring(1,6)+" Y LAS "+horaDefinitiva.substring(7,12)+" EN EL
QUIROFANO "+quirofanoDefinitivo+" CON LOS "+personalDefinitivo;
       } else { //Falta algún elemento y la operación no se puede realizar
         contenidoFinal="LA OPERACION NO SE PUEDE HACER POR INCOMPATIBILIDAD HORARIA
ENTRE QUIROFANOS Y PERSONAL";
         msg.reset();
         msg.setPerformative(ACLMessage.INFORM);
         if (!faltaQuirof)
           msg.addDest("Q");//En caso de que haya quirófano, se cancela el proceso de
búsqueda de quirófano
         else
           contenidoFinal="LA OPERACION NO SE PUEDE HACER POR FALTA DE QUIROFANO";

         if (!faltaPersonal && !noHayQuir)
           msg.addDest("Cm");//y en caso de que haya personal, también se cancela el
proceso
         else if (faltaPersonal)
           contenidoFinal="LA OPERACION NO SE PUEDE HACER POR FALTA DE PERSONAL";

         if ((faltaPersonal) && (faltaQuirof))
           contenidoFinal="LA OPERACION NO SE PUEDE HACER POR FALTA DE QUIROFANO Y
PERSONAL";
         else {
           msg.setSource("Ce");
           msg.setContent("(Cancelar)");
           send(msg);
         }
         if (noHayQuir) {
           msg.addDest("Q");
           msg.setSource("Ce");
```

```
            msg.setContent("(Cancelar)");
            send(msg);
          }
        }
        msg.removeAllDests();//Se le envía a Ct la señal de preparado, informando de
todos los detalles de la operación
        msg.addDest("Ct");   //si es que se realiza, o los motivos de porqué no se
puede llevar a cabo
        msg.setContent(contenidoFinal);
        msg.setOntology("Preparado");
        msg.setSource("Ce");
        send(msg);
      }
    }

    public String quitarDireccion(String s) { //Función que elimina de un string de tipo
dirección
      int i=0;                                        //los caracteres después de la @,
incluyendo ésta tb
      char c=s.charAt(i);
      String resp = new String();

      while ((c!='@') && (i<s.length())) {
        resp+=c;
        i++;
        c=s.charAt(i);
      }
      return(resp);
    }

    protected void setup() {
    addBehaviour(new RecibeDeCt());//Se añade el comportamiento cíclico a la cola de
comportamientos
    }

}
```

7.3. Personal.java

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

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.acl.ACLMessage;

public class Personal extends Agent {

   String organo,str,hora;
   String horas[] = new String[48];
   int medicos[] = new int[48]; //Tablas para comprobar las horas libres de médicos,
anestesistas y enfermeras
   int anestesistas[] = new int[48];
   int enfermeras[] = new int[48];
   int personal[][] = new int[4][3];//Tabla que contiene el nº necesario de persona
para operar cada tipo de órgano
   String organos[] = {"corazon","pulmon","rinyon","higado"};
   int indice=0;

   public class RecibeDeCe extends CyclicBehaviour { //Único comportamiento de esta clase

      public RecibeDeCe() {

      }

      public void action () {
         String interv,fuente;
         int i,numM,numE,numA;
         boolean noHayPers=false;
         String error = new String();

      for (i=0;i<48;i++) { //Inicialización de algunas estructuras
         medicos[i]=0;
         anestesistas[i]=0;
         enfermeras[i]=0;
      }

         ACLMessage msg = blockingReceive(MessageTemplate.MatchSource("Ce")); //Se espera
petición desde Ce
      str=msg.getContent();//con el formato(02:30 05:30 00:30 corazon)
      hora=str.substring(1,18);
      organo=str.substring(19,str.length()-1);

      for (i=0;i<4;i++)//Se comprueba de qué órgano se trata para acceder a la tabla de
personal necesario
         if(organo.equals(organos[i]))
            indice=i;

      int hora_ini=aEntero(hora.charAt(0),hora.charAt(1)); //Se convierten las horas de
String a entero para
         int min_ini=aEntero(hora.charAt(3),hora.charAt(4));  //usar las tablas en forma
de horario
         int hora_fin=aEntero(hora.charAt(6),hora.charAt(7));
         int min_fin=aEntero(hora.charAt(9),hora.charAt(10));
         int hora_dur=aEntero(hora.charAt(12),hora.charAt(13));
         int min_dur=aEntero(hora.charAt(15),hora.charAt(16));

         int dur=(hora_dur*2)+(min_dur%29); //duracion de la operacion en posiciones de la
tabla (cada posicion es media hora)
         int ini=(hora_ini*2)+(min_ini%29); //inicio de la operacion en la posicion ini de
la tabla
         int fin=(hora_fin*2)+(min_fin%29); //final de la operacion en la posicion fin

      ACLMessage msgNuevo= new ACLMessage(ACLMessage.INFORM);
      msgNuevo.setContent("(Organo nuevo)");//Se avisa a M,E y A de que llegará una
peticion nueva
      msgNuevo.addDest("M");
      msgNuevo.addDest("A");
      msgNuevo.addDest("E");
      send(msgNuevo);
```

```
    msg.setSource("Cm"); //Se envía la petición a M,A y E, tan sólo M necesita saber
el órgano, el resto tienen
    msg.setPerformative(ACLMessage.REQUEST);//suficiente con las horas
    msg.removeAllDests();
    msg.setOntology("INICIO");
    msg.addDest("M");
    send(msg);

    msg.setContent("("+hora+")");
    msg.removeAllDests();
    msg.addDest("A");
    send(msg);
    msg.removeAllDests();
    msg.addDest("E");
    send(msg);

      MessageTemplate mt =
MessageTemplate.or(MessageTemplate.MatchSource("M"),MessageTemplate.MatchSource("A"));
      mt = MessageTemplate.or(mt,MessageTemplate.MatchSource("E"));

      //Se esperan mensajes de M,A y E, con los intervalos posibles para realizar la
operación

      for (i=1;i<4;i++) {
      msg = blockingReceive(mt);
      interv=msg.getContent();//Se guardan los intervalos
      fuente=msg.getSource();  //y la fuente del mensaje

      if (fuente.equalsIgnoreCase("M")) {
        if (interv.charAt(1)!='N') {//Si el mensaje no es del tipo "No hay ninún
médico en el hospital"
          llenarTabla(interv,medicos,dur); //será que hay intervalos posibles. Se
llena la tabla de médicos con éstos
        } else {
          noHayPers=true; //Sino => Error, falta algún tipo de error en el hospital
          error+=interv;//Se guardan todos los errores
        }

      } else if (fuente.equalsIgnoreCase("A")) { //Lo mismo para anestesistas y
enfermeras
        if (interv.charAt(1)!='N') {
          llenarTabla(interv,anestesistas,dur);
        } else {
          noHayPers=true;
          error+=interv;
        }

      } else { //será de "E", gracias al MessageTemplate
        if (interv.charAt(1)!='N') {
          llenarTabla(interv,enfermeras,dur);
        } else {
          noHayPers=true;
          error+=interv;
        }
      }
      }

      String resp="";
      numM=personal[indice][0];//Número de médicos, anestesistas y enfermeras
necesarias,
      numA=personal[indice][1];//depende del órgano que le pasan, una simple tabla
      numE=personal[indice][2];

      msg.reset();
      msg.addDest("Ce");
      msg.setPerformative(ACLMessage.INFORM);
      msg.setSource("Cm");

      if (!noHayPers) //Si no hay errores de falta de personal
        resp=personalCompatible(dur,ini,fin,numM,numA,numE);//Personal compatible
entre si

      msg.setContent("("+resp+")");//Se le envían a Ce las horas con personal
disponible
      send(msg);
```

```
       if (resp.equals("")) noHayPers=true;//Si no hay personal compatible a ninguna
hora, activamos un error
       if (!noHayPers) {//Si todo es correcto, no falta personal

         System.out.println("HORAS EN QUE HAY PERSONAL: "+resp+"//");

         mt =
MessageTemplate.and(MessageTemplate.MatchSource("Ce"),MessageTemplate.MatchType("inform"
));

         //Se espera un mensaje de Ce donde se indicará la hora definitiva de la
operación, que habrá elegido Q

         msg=blockingReceive(mt);

         String horaDefinitiva=msg.getContent();

         if (!horaDefinitiva.equals("(Cancelar)")) {//Si no se ha de cancelar

           msg.setSource("Cm");

           msg.removeAllDests();
           msg.addDest("M");
           msg.setContent("("+horaDefinitiva+" "+numM+")");//Se envía la hora a M,A y
E, con el número de personas
           send(msg);                                                  //necesarias de
cada tipo para que elijan a qué
                                                                        //personas
les va mejor, por horario, hacer la operación
           msg.removeAllDests();
           msg.addDest("A");
           msg.setContent("("+horaDefinitiva+" "+numA+")");
           send(msg);

           msg.removeAllDests();
           msg.addDest("E");
           msg.setContent("("+horaDefinitiva+" "+numE+")");
           send(msg);

           mt =
MessageTemplate.or(MessageTemplate.MatchSource("M"),MessageTemplate.MatchSource("A"));
           mt = MessageTemplate.or(mt,MessageTemplate.MatchSource("E"));

           //Se espera la respuesta de los tres coordinadores

           boolean medicos,anestesistas,enfermeras;
           medicos=false;
           anestesistas=false;
           enfermeras=false;
           String med = new String();
           String an = new String();
           String enf = new String();

           for (i=1;i<4;i++) {
           msg = blockingReceive(mt);
           fuente=msg.getSource();

           if (fuente.equalsIgnoreCase("M")) {
             medicos=true; //Recibida confirmación de médicos
             med=msg.getContent(); //Medicos que harán la operación
           } else if (fuente.equalsIgnoreCase("A")) {
             anestesistas=true;//Confirmación de anestesistas
             an=msg.getContent(); //Anestesistas que trabajarán en la operación
             } else {  //será de "E" por el MessageTemplate
             enfermeras=true;   //De enfermeras
             enf=msg.getContent();//Enfermeras de la operación
             }
           }

           if (medicos && anestesistas && enfermeras) {//Si están todo los elementos
confirmados, se avisa a Ce
             msg.removeAllDests();
             msg.addDest("Ce");   //indicando el personal que intervendrá en la
operación
             msg.setSource("Cm");
             msg.setContent("(MEDICOS:"+med+", ANESTESISTAS:"+an+" Y
ENFERMERAS:"+enf+")");
```

```
            send(msg);
          }
        } else { //Si se ha de cancelar, por algún tipo de error
          System.out.println("Falta algun elemento");
          msg.removeAllDests();
          msg.addDest("M");//Se envía a M,A y E un mensaje de cancelación
          msg.addDest("A");
          msg.addDest("E");
          msg.setContent("(Cancelar)");
          msg.setSource("Cm");
          send(msg);
        }
      } else { //Si no hay personal compatible
        System.out.println("No hay suficiente personal compatible");
        msg.removeAllDests();
        msg.addDest("M");//lo mismo
        msg.addDest("A");
        msg.addDest("E");
        msg.setContent("(Cancelar)");
        msg.setSource("Cm");
        send(msg);
      }
    }
  }

  public String personalCompatible(int dur, int ini, int fin, int numM, int numA, int
numE) { //Determina las horas con
    int de=ini;
        //personal compatible
    int a=ini;
    int i;
    String resp=new String();

      //Usando las tablas de cada tipo, mira a ver si puede compatibilizar el personal
    for (i=ini;i<fin;i++) {
      if ((medicos[i]>=numM) && (anestesistas[i]>=numA) && (enfermeras[i]>=numE))
        resp=resp+"("+horas[i]+" "+horas[i+dur]+")";
    }
    return(resp);
  }

  public void llenarTabla(String str, int[] tabla, int dur) {
    int j,hora_ini,min_ini,hora_fin,min_fin,ini,fin,i,aux;

        //llena una tabla en forma de horario para comprobar si hay personal a una
determinada hora
    str=str.substring(1,str.length()-1);
    for (j=0;j<str.length()-12;j=j+13) {
        hora_ini=aEntero(str.charAt(j+1),str.charAt(j+2));
        min_ini=aEntero(str.charAt(j+4),str.charAt(j+5));
        hora_fin=aEntero(str.charAt(j+7),str.charAt(j+8));
        min_fin=aEntero(str.charAt(j+10),str.charAt(j+11));

        ini=(hora_ini*2)+(min_ini%29);
        fin=(hora_fin*2)+(min_fin%29);

        for (i=ini;i<fin;i++) {
          if (fin-i>=dur) tabla[i]++;
        }
    }
  }

  public int aEntero(char c1, char c2){//Convierte dos caracteres a un entero
    int n;
    n=java.lang.Character.getNumericValue(c1)*10;
    n+=java.lang.Character.getNumericValue(c2);
    return(n);
  }

  protected void setup() {
  int i,j,aux;
  String min,hora;
  hora="00";

  for (j=0;j<48;j++){
      if (j%2!=0) min="30";
      else min="00";
```

```
        aux=(j)/2;                          //Inicializaciones
        if (aux<10) hora=("0"+aux);
        else hora=hora.valueOf(aux);
        horas[j]=(hora+":"+min);
      }

      personal[0][0]=2;//Médicos necesarios para una operación de corazón
      personal[1][0]=3;//M pulmón
      personal[2][0]=1;//M riñón
      personal[3][0]=2;//M hígado
      personal[0][1]=3;//A corazón
      personal[1][1]=2;//A pulmón
      personal[2][1]=1;//A riñón
      personal[3][1]=3;//A hígado
      personal[0][2]=2;//E corazón
      personal[1][2]=3;//E pulmón
      personal[2][2]=2;//E riñón
      personal[3][2]=3;//E hígado

    addBehaviour(new RecibeDeCe()); //Se añade el comportamiento cíclico
    }
}
```

7.4. Mae.java

```
import java.io.*;
import java.util.Date;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Calendar;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.acl.ACLMessage;
import jade.proto.*;
import jade.core.AgentGroup;
import jade.domain.AgentManagementOntology.*;
import jade.domain.*;

public class Mae extends Agent {

   ACLMessage msg = new ACLMessage(ACLMessage.CFP);
   String organo = new String();
   String str,hora;
   int numPersonas;
   DFAgentDescriptor dfad=new DFAgentDescriptor();
   ServiceDescriptor sd=new ServiceDescriptor();
   DFSearchResult result=new DFSearchResult();
   String tipoAgente;
   boolean noHay;
   Vector operaciones = new Vector();
   ACLMessage msgCancel = new ACLMessage(ACLMessage.REQUEST);
   Calendar fecha=Calendar.getInstance();
   int hoy=fecha.get(Calendar.DAY_OF_WEEK);

   public class EnviaContractNet extends FipaContractNetInitiatorBehaviour {
        //Parte iniciante del prótocolo Contract-Net
    AgentGroup proposerAgents;
    boolean hasBeenReset=false;
    int state = 0;

     public EnviaContractNet(Agent a, ACLMessage msg, AgentGroup group) {
       super(a,msg,group);
       cfpMsg = (ACLMessage)msg.clone();
       proposerAgents = (AgentGroup)group.clone();
       state=0;
       hasBeenReset = false;
     }

    public EnviaContractNet(Agent a, ACLMessage msg) {
       this(a,msg,msg.getDests());
     }

     public Vector handleProposeMessages(Vector proposals) { //Método para tratar las
propuestas recibidas
        int i,j;
        ACLMessage msg,resp;
        String str = new String();
        Vector respuestas = new Vector();
        String str2;
        boolean error=false;

        ACLMessage msg3 = new ACLMessage(ACLMessage.INFORM);
        msg3.setSource(getLocalName());
        int num=0;
        int puntuacion=0;

        for (i=0;i<=proposals.size()-1;i++){//Se recorre el vector con todas las
propuestas recibidas
          msg=(ACLMessage)proposals.elementAt(i);
          str=str+msg.getContent().substring(1,msg.getContent().length()-1); //En un
string se juntan todas las horas propuestas
          msg3.addDest(msg.getSource()); //Se prepara un mensaje con las direcciones de
todos los agentes que han hecho propuestas
          num++; //Variable que indica el número de propuestas recibidas
        }
        ACLMessage msgCm = new ACLMessage(ACLMessage.INFORM);
```

```
        msgCm.setContent("("+str+")");
        msgCm.setSource(getLocalName());
        msgCm.removeAllDests();
        msgCm.addDest("Cm"); //Todas las propuestas se le envían a Cm
        send(msgCm);

        if (str==null) error=true; //En caso de que no haya propuestas recibidas => Error
        if (!error) { //Si hay propuestas

          msgCm=blockingReceive(MessageTemplate.MatchSource("Cm")); //Se espera
contestación de Cm
          if (!msgCm.getContent().equals("(Cancelar)")) { //Si no se ha de cancelar el
proceso
            String horaDefinitiva=msgCm.getContent().substring(1,14); //Se guarda la
hora definitiva

  numPersonas=java.lang.Character.getNumericValue(msgCm.getContent().charAt(15)); //Y el
número de personas necesarias
            msg3.setContent(horaDefinitiva);                                    //de
esta especialidad
            send(msg3); //Se envía a todos los agentes que han hecho propuestas, la hora
elgida, para que la puntúen

            String puntuacion3[] = new String[numPersonas];
            String puntuacion2[] = new String[numPersonas];
            String puntuacion1[] = new String[numPersonas];
            String puntuacion0[] = new String[numPersonas];

            for (i=0;i<num;i++) { //Se esperan todas las respuestas de los usuarios, que
devuelven la puntuación
              msg3=blockingReceive(MessageTemplate.MatchOntology("Puntuacion"));
              if (msg3!=null) {
                str2=msg3.getContent();
                puntuacion=java.lang.Character.getNumericValue(str2.charAt(1));
                switch (puntuacion) {
                  case 3: {
                    j=0;
                    while ((j<numPersonas) && (puntuacion3[j]!=null))
                      j++;
                    if (j<numPersonas) puntuacion3[j]=msg3.getSource();
                    break;
                  }
                  case 2: {
                    j=0;
                    while ((j<numPersonas) && (puntuacion2[j]!=null))
                      j++;
                    if (j<numPersonas) puntuacion2[j]=msg3.getSource();
                    break;
                  }
                  case 1: {
                    j=0;
                    while ((j<numPersonas) && (puntuacion1[j]!=null))
                      j++;
                    if (j<numPersonas) puntuacion1[j]=msg3.getSource();
                    break;
                  }
                  case 0: {
                    j=0;
                    while ((j<numPersonas) && (puntuacion0[j]!=null))
                      j++;
                    if (j<numPersonas) puntuacion0[j]=msg3.getSource();
                    break;
                  }
                }
              }
            }

            //Se llenan una serie de tablas con los agentes que tienen puntuación 3,2,1
y  0

            String mejorPuntuacion[] = new String[numPersonas];

            i=0;
            for (j=0;j<numPersonas;j++)
              if ((puntuacion3[j]!=null) && (i<numPersonas)){
                mejorPuntuacion[i]=puntuacion3[j];
                i++;
```

```
            }

          if (i<numPersonas)
            for (j=0;j<numPersonas;j++)
              if ((puntuacion2[j]!=null) && (i<numPersonas)){
                mejorPuntuacion[i]=puntuacion2[j];
                i++;
              }

          if (i<numPersonas)
            for (j=0;j<numPersonas;j++)
              if ((puntuacion1[j]!=null) && (i<numPersonas)){
                mejorPuntuacion[i]=puntuacion1[j];
                i++;
              }

          //Se buscan las mejores puntuaciones, de acorde con el número de personas
necesarias

          for (i=0;i<=proposals.size()-1;i++){
            msg=(ACLMessage)proposals.elementAt(i);
            if (estaEnTabla(msg.getSource(),mejorPuntuacion,numPersonas)) {
              ACLMessage aceptar = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
              aceptar.setSource(getName());
              aceptar.removeAllDests();//Se aceptan las propuesta que están entre
las mejores
              aceptar.addDest(msg.getSource());
              aceptar.setContent(horaDefinitiva);
              aceptar.setReplyTo(msg.getReplyWith());
              respuestas.addElement((ACLMessage)aceptar);
            } else {
              ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
              rechazar.setSource(getName());
              rechazar.removeAllDests();//Se rechaza el resto
              rechazar.addDest(msg.getSource());
              rechazar.setReplyTo(msg.getReplyWith());
              respuestas.addElement((ACLMessage)rechazar);
            }
          }
          msgCancel.setContent(horaDefinitiva); //Se empieza a rellenar un mensaje,
que servirá para mantener un histórico
          return(respuestas);                    //de las operaciones confirmadas
        } else {
          System.out.println(getLocalName()+": Cancelando");
          return(null); //Si hay que cancelar, se para el protocolo con los hijos
        }
      } else {
        System.out.println(getLocalName()+": No hay "+tipoAgente);
        return(null); //Lo mismo si falta algún tipo de agente
      }
    }

    public void handleOtherMessages(ACLMessage msg) { //Para tratar otro tipo de
mensajes, no hace nada

    }

    public Vector handleFinalMessages(Vector messages) { //Para tratar los mensajes
finales de los hijos
      int i;
      String str = new String();

      msgCancel.removeAllDests();

      for (i=0;i<=messages.size()-1;i++){
        msg=(ACLMessage)messages.elementAt(i);
        msgCancel.addDest(msg.getSource());//Se ponen en el mensaje que servirá para
el histórico, las direcciones de los
        str=str+" "+quitarDireccion(msg.getSource());//agentes que intervienen en la
operación
      }
        //Ha de haber tantos mensajes como personas necesarias
      if (messages.size()==numPersonas) {//Si se han recibido todas las
confirmaciones
        msg=(ACLMessage)messages.elementAt(0);
        ACLMessage msg2=new ACLMessage(ACLMessage.INFORM);
        msg2.setContent(str);
```

```
          msg2.removeAllDests();
          msg2.addDest("Cm");//Se envía la confirmación a Cm
          msg2.setSource(getLocalName());
          send(msg2);
          msgCancel.setLanguage(numPersonas+"");
          System.out.println("Final del protocolo por parte del "+tipoAgente);
          operaciones.add((ACLMessage)msgCancel.clone()); //Se guarda la operación
actual en el histórico
        }
        return(null); //Finaliza el protocolo

      }

      public String createCfpContent(String cfpContent, String receiver) {

        return(cfpContent);

      }

    }

    public boolean estaEnTabla(String str, String[] tabla, int num) { //Busca un String en
una tabla
      boolean esta=false;
      int i;

      for (i=0;i<num;i++)
        if (tabla[i].equals(str)) esta=true;

      return(esta);
    }

    public String quitarDireccion(String s) {//Quita la direccion, a partir de la @, de
un String
      int i=0;
      char c=s.charAt(i);
      String resp = new String();

      while ((c!='@') && (i<s.length())) {
        resp+=c;
        i++;
        c=s.charAt(i);
      }
      return(resp);

    }

    public class RecibeDeCt extends CyclicBehaviour { //Sirve para las cancelaciones

      public RecibeDeCt() {

      }

      public void action () {
        int i,numMens,j;
        String str;
        MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource("Ct"),MessageTemplate.MatchOntology("Can
celacion"));
        boolean encontrado=false;

        ACLMessage msgDeCt = receive(mt);
        if(msgDeCt != null) { //Se espera un mensaje de Ct pidiendo una cancelación, si
no llega no se hace nada

          fecha=Calendar.getInstance();
          if (hoy!=fecha.get(Calendar.DAY_OF_WEEK)) {
            hoy=fecha.get(Calendar.DAY_OF_WEEK);
            for (j=1;j<operaciones.size();j++)
              operaciones.remove(operaciones.elementAt(j));
          } //Se actualiza el histórico, eliminando las operaciones ya realizadas el día
anterior

          str=msgDeCt.getContent();
          for (i=0;i<operaciones.size();i++) { //Se comprueba si la operación a cancelar
está confirmada
            ACLMessage msgACancelar=(ACLMessage)operaciones.elementAt(i);
```

```
              if (msgACancelar.getContent().equals(str)) {
                encontrado=true;

  numMens=java.lang.Character.getNumericValue(msgACancelar.getLanguage().charAt(0));
                send(msgACancelar); //Se les envía a los agentes implicados un mensaje
para que cancelen
                operaciones.remove(msgACancelar); //Se elimina la operación del
histórico
                for (j=0;j<numMens;j++)
                  blockingReceive(MessageTemplate.MatchOntology("Cancelada"));
                    //Se esperan todas las confirmaciones de los agentes inferiores
                ACLMessage msgAceptacion = new ACLMessage(ACLMessage.AGREE);
                msgAceptacion.addDest("Ct"); //Se informa de la cancelación correcta a
Ct
                msgAceptacion.setSource(getLocalName());
                msgAceptacion.setContent("Cancelacion correcta por parte del
coordinador de "+tipoAgente+"s");
                send(msgAceptacion);
              }
          }
          if (!encontrado) { //Si no se ha encontrado la operación en el histórico
            ACLMessage msgAceptacion = new ACLMessage(ACLMessage.AGREE);
            msgAceptacion.addDest("Ct"); //Se informa a Ct del hecho
            msgAceptacion.setSource(getLocalName());
            msgAceptacion.setContent("No se puede cancelar porque esa operacion no esta
reservada");
            send(msgAceptacion);
          }
        }
      }

  }

  protected void setup() {
  if (getLocalName().equals("A"))//Se mira el tipo de agente según el nombre
     tipoAgente="anestesista";
  else if (getLocalName().equals("E"))
     tipoAgente="enfermera";
  else if (getLocalName().equals("M"))
     tipoAgente="medico";

  msgCancel.setOntology("Cancelacion");
  msgCancel.setSource(getLocalName());

  addBehaviour(new RecibeDeCt());

  addBehaviour(new CyclicBehaviour(this) { //Se añade un comportamiento cíclico, que se
actica cuando Cm
       public void action() {                 //avisa a este agente de que llegará un
órgano nuevo
        noHay=false;
        ACLMessage msgNuevo=receive(MessageTemplate.MatchContent("(Organo nuevo)"));
        if (msgNuevo!=null) {
          msgNuevo.removeAllDests();
          fecha=Calendar.getInstance();
            hoy=fecha.get(Calendar.DAY_OF_WEEK);

            ACLMessage msgDeCm =
blockingReceive(MessageTemplate.and(MessageTemplate.MatchSource("Cm"),MessageTemplate.Ma
tchOntology("INICIO")));
          str=msgDeCm.getContent(); //Se espera el mensaje con la petición desde Cm
          hora=str.substring(1,18);//Se guarda la hora
          if (tipoAgente.equals("medico")) {//En caso de que este agente sea M, se
guarda también el órgano
            organo=str.substring(19,str.length()-1);
              sd.setType(tipoAgente+"-"+organo);
          } else
            sd.setType(tipoAgente);  //Buscará agentes con este servicio

           dfad.addAgentService(sd);     //añadimos el servicio al descriptor de agente
DF, para la busqueda

           msg.removeAllDests();

            try {
              result=searchDF("df",dfad,null);//procedemos a buscar en "df" lo que
queremos (dfad) sin constraints (null)
```

```
               Enumeration e = result.elements();//miramos los resultados uno a uno
               if (!e.hasMoreElements()) //En caso de que no haya agentes disponibles en
el sistema
                  noHay=true;

               while(e.hasMoreElements()) {
                    //convertimos el resultado a DFAgentDescriptor para poder extraer el
nombre del agente encontrado
               AgentManagementOntology.DFAgentDescriptor actual =
(AgentManagementOntology.DFAgentDescriptor)e.nextElement();
               msg.addDest(actual.getName());//Con esto obtenemos el nombre del agente
               msgNuevo.addDest(actual.getName());
               }

            } catch (FIPAException exc) {
               System.out.println( exc.toString() );
            }
            if (!noHay) { //Si hay agentes disponibles
               msg.setContent("("+hora+")");

               send(msgNuevo); //Comienza el Contract-Net con ellos

              addBehaviour(new EnviaContractNet(Mae.this,msg));
            } else { //Si no hay agentes disponibles, se informa de ello, tanto a Cm
como por pantalla
               System.out.println("No hay ningun "+tipoAgente+" para esta operacion en
el hospital");
               msg.reset();
               msg.setSource(getLocalName());
               msg.setContent("(No hay ningun "+tipoAgente+" para esta operacion en el
hospital. )");
               msg.addDest("Cm");
               send(msg);
               msg=blockingReceive(MessageTemplate.MatchSource("Cm")); //Se recibe un
mensaje de Cm donde indicará que se ha de
               System.out.println(getLocalName()+": Cancelando");    //cancelar el
proceso
            }
       }
       }
    });
  }


}
```

7.5. Quirof.java

```
import java.io.*;
import java.util.Date;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Calendar;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.lang.acl.ACLMessage;
import jade.proto.*;
import jade.core.AgentGroup;
import jade.domain.AgentManagementOntology.*;
import jade.domain.*;

public class Quirof extends Agent {

   ACLMessage msg = new ACLMessage(ACLMessage.CFP);
   String str,hora;
   String horas[] = new String[48];
   int quirofanos[] = new int[48];
   int hora_ini,min_ini,hora_fin,min_fin,hora_dur,min_dur,ini,fin,dur;
   DFAgentDescriptor dfad=new DFAgentDescriptor();
  ServiceDescriptor sd=new ServiceDescriptor();
  DFSearchResult result=new DFSearchResult();
   int j,aux;
   boolean noHay;
   Vector operaciones = new Vector();
   ACLMessage msgCancel = new ACLMessage(ACLMessage.REQUEST);
   Calendar fecha=Calendar.getInstance();
   int hoy=fecha.get(Calendar.DAY_OF_WEEK);

   public class EnviaContractNet extends FipaContractNetInitiatorBehaviour {
        //Parte iniciante del prótocolo Contract-Net
    AgentGroup proposerAgents;
    boolean hasBeenReset=false;
    int state = 0;

     public EnviaContractNet(Agent a, ACLMessage msg, AgentGroup group) {
       super(a,msg,group);
       cfpMsg = (ACLMessage)msg.clone();
       proposerAgents = (AgentGroup)group.clone();
       state=0;
       hasBeenReset = false;
    }

    public EnviaContractNet(Agent a, ACLMessage msg) {
       this(a,msg,msg.getDests());
    }

     public Vector handleProposeMessages(Vector proposals) { //Método para tratar las
propuestas recibidas
       int i,num;
       ACLMessage msg;
       String str = new String();
       String hora=new String();
       String quirofano=new String();
       int puntuacion=0;
       String str2;
       Vector respuestas = new Vector();
       boolean noHayQuir=false;

       ACLMessage msgQ = new ACLMessage(ACLMessage.INFORM);
       msgQ.setSource("Q");

       num=0;

       for (i=0;i<=proposals.size()-1;i++){//Se recorre el vector con todas las
propuestas recibidas
         msg=(ACLMessage)proposals.elementAt(i);
         str=str+msg.getContent().substring(1,msg.getContent().length()-1); //En un
string se juntan todas las horas propuestas
```

```
          msgQ.addDest(msg.getSource()); //Se prepara un mensaje con las direcciones de
todos los agentes que han hecho propuestas
          num++; //Variable que indica el número de propuestas recibidas
        }

        llenarTabla("("+str+")",quirofanos,dur); //Se buscan las horas donde hay
quirófano disponible
        String resp=quirofanoLibre(dur,ini,fin);

        System.out.println("HORAS LIBRES DE QUIROFANO: "+resp+"//");

        ACLMessage msgCe = new ACLMessage(ACLMessage.INFORM);
        msgCe.setContent("("+resp+")");
        msgCe.setSource("Q"); //Estas horas se envían a Ce
        msgCe.removeAllDests();
        msgCe.addDest("Ce");
        send(msgCe);

        if (resp=="") noHayQuir=true; //Se actica un error si no había ningún quirófano
        if (!noHayQuir) { //Si no hay error
          msgCe=blockingReceive(MessageTemplate.MatchSource("Ce"));
          String paraEscoger=msgCe.getContent(); //Se espera respuesta de Ce
          if (!paraEscoger.equals("(Cancelar)")) {
            msgQ.setContent(paraEscoger);
            send(msgQ);
              //Se guardan las horas de entre que Q tiene que escoger una y se piden
puntuaciones a los quirófanos
            for (i=0;i<num;i++) {
              msgQ=blockingReceive(MessageTemplate.MatchOntology("Puntuacion"));
              if (msgQ!=null) {
                 str2=msgQ.getContent();
                 if (java.lang.Character.getNumericValue(str2.charAt(15))>puntuacion) {
                   puntuacion=java.lang.Character.getNumericValue(str2.charAt(15));
                   hora=str2.substring(1,14);
                   quirofano=msgQ.getSource();
                 }
              }
            } //Se escoge el quirófano y hora con mejor puntuación, de entre las
propuestas recibidas

            for (i=0;i<=proposals.size()-1;i++){
              msg=(ACLMessage)proposals.elementAt(i);
              if (msg.getSource().equals(quirofano)) {
                 ACLMessage aceptar = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                 aceptar.setSource(getName());
                 aceptar.removeAllDests();
                 aceptar.addDest(msg.getSource());
                 aceptar.setContent(hora); //Se acepta la propuesta con mejor puntuación
                 aceptar.setReplyTo(msg.getReplyWith());
                 respuestas.addElement((ACLMessage)aceptar);
              } else {
                 ACLMessage rechazar = new ACLMessage(ACLMessage.REJECT_PROPOSAL);
                 rechazar.setSource(getName());
                 rechazar.setReplyTo(msg.getReplyWith());
                 rechazar.removeAllDests();//y se rechaza el resto
                 rechazar.addDest(msg.getSource());
                 respuestas.addElement((ACLMessage)rechazar);
              }
            }
            msgCancel.setContent(hora);//Se pone la hora de la operación, en el mensaje
que servirá para el histórico
            return(respuestas);
          } else { //Si hay error
              System.out.println("Q: Cancelando");
              return(null);
          }
        } else { //Si no hay ningún quirófano disponible
            System.out.println("No hay quirofano disponible");
            return(null);
        }
    }

    public void handleOtherMessages(ACLMessage msg) {

    }

    public Vector handleFinalMessages(Vector messages) {
```

```
        //Aquí sólo ha de haber un mensaje, sólo se operará en un quirófano

      msgCancel.removeAllDests();

      if (messages.size()==1) {
        msg=(ACLMessage)messages.elementAt(0);
        msgCancel.addDest(msg.getSource());
        ACLMessage msg2=new ACLMessage(ACLMessage.INFORM);
        msg2.setContent("("+msg.getContent()+msg.getSource()+")");
        msg2.removeAllDests();
        msg2.addDest("Ce");//Se informa a Q de que la operación se realizará en
cierto quirófano a cierta hora
        msg2.setSource("Q");
        send(msg2);
        operaciones.add((ACLMessage)msgCancel.clone()); //Se guarda la operación en el
histórico
        System.out.println("Final del protocolo por parte del quirofano");
      }
      return(null); //Finaliza el protocolo

    }

    public String createCfpContent(String cfpContent, String receiver) {

      return(cfpContent);

    }

  }

  public String quirofanoLibre(int dur, int ini, int fin) {
    int de=ini;
    int a=ini;
    int i;
    String resp="";

    for (i=ini;i<fin;i++) {
      if (quirofanos[i]>=1) resp=resp+"("+horas[i]+" "+horas[i+dur]+")";
    }

    return(resp);
  }

  public void llenarTabla(String str, int[] tabla, int dur) {
    int j,hora_ini,min_ini,hora_fin,min_fin,ini,fin,i,aux;

    str=str.substring(1,str.length()-1);
    for (j=0;j<str.length()-12;j=j+13) {
        hora_ini=aEntero(str.charAt(j+1),str.charAt(j+2));
        min_ini=aEntero(str.charAt(j+4),str.charAt(j+5));
        hora_fin=aEntero(str.charAt(j+7),str.charAt(j+8));
        min_fin=aEntero(str.charAt(j+10),str.charAt(j+11));

        ini=(hora_ini*2)+(min_ini%29);
        fin=(hora_fin*2)+(min_fin%29);

        for (i=ini;i<fin;i++) {
          if (fin-i>=dur) tabla[i]++;
        }
    }
  }


  public int aEntero(char c1, char c2){
    int n;
    n=java.lang.Character.getNumericValue(c1)*10;
    n+=java.lang.Character.getNumericValue(c2);
    return(n);
  }

  public class RecibeDeCt extends CyclicBehaviour { //Sirve para las cancelaciones

    public RecibeDeCt() {

    }

    public void action () {
```

```
       int i,j;
       String str;
       MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource("Ct"),MessageTemplate.MatchOntology("Can
celacion"));
       boolean encontrado=false;

       ACLMessage msgDeCt = receive(mt);
       if(msgDeCt != null) { //Se espera un mensaje de Ct pidiendo una cancelación, si
no llega no se hace nada

         fecha=Calendar.getInstance();
         if (hoy!=fecha.get(Calendar.DAY_OF_WEEK)) {
           hoy=fecha.get(Calendar.DAY_OF_WEEK);
           for (j=1;j<operaciones.size();j++)
             operaciones.remove(operaciones.elementAt(j));
         } //Se actualiza el histórico, eliminando las operaciones ya realizadas el día
anterior

         str=msgDeCt.getContent();
         for (i=0;i<operaciones.size();i++) { //Se comprueba si la operación a cancelar
está confirmada
           ACLMessage msgACancelar=(ACLMessage)operaciones.elementAt(i);
             System.out.println(msgACancelar.getContent()+"....."+str);
             if (msgACancelar.getContent().equals(str)) {
               encontrado=true;
               send(msgACancelar); //Se les envía a los agentes implicados un mensaje
para que cancelen
               operaciones.remove(msgACancelar); //Se elimina la operación del
histórico

               blockingReceive(MessageTemplate.MatchOntology("Cancelada"));
                 //Se espera la confirmación del quirófano y se informa a Ct de que
la cancelación es correcta
               ACLMessage msgAceptacion = new ACLMessage(ACLMessage.AGREE);
               msgAceptacion.addDest("Ct");
               msgAceptacion.setSource(getLocalName());
               msgAceptacion.setContent("Cancelacion correcta por parte del
coordinador de quirofanos");
               send(msgAceptacion);
             }
         }
         if (!encontrado) { //Si no se ha encontrado la operación en el histórico
           ACLMessage msgAceptacion = new ACLMessage(ACLMessage.AGREE);
           msgAceptacion.addDest("Ct"); //Se informa a Ct
           msgAceptacion.setSource(getLocalName());
           msgAceptacion.setContent("No se puede cancelar porque esa operacion no esta
reservada");
           send(msgAceptacion);
         }
       }
     }

   }

   protected void setup() {

   msgCancel.setOntology("Cancelacion");
   msgCancel.setSource("Q");

   addBehaviour(new RecibeDeCt());

   addBehaviour(new CyclicBehaviour(this) { //Se añade un comportamiento cíclico, que se
actica cuando Ce
       public void action() {                 //avisa a este agente de que llegará un
órgano nuevo
       noHay=false;
       ACLMessage msgNuevo=receive(MessageTemplate.MatchContent("(Organo nuevo)"));
       if (msgNuevo!=null) {
         msgNuevo.removeAllDests();
         fecha=Calendar.getInstance();
           hoy=fecha.get(Calendar.DAY_OF_WEEK);

           String min,hora;
           hora="00";

         for (j=0;j<48;j++){
```

```
              if (j%2!=0) min="30";
              else min="00";//Algunas inicializaciones
              aux=(j)/2;
              if (aux<10) hora=("0"+aux);
              else hora=hora.valueOf(aux);
              horas[j]=(hora+":"+min);
            }
          for (j=0;j<48;j++) {
            quirofanos[j]=0;
          }

          ACLMessage msgDeCm = blockingReceive(MessageTemplate.MatchSource("Ce")); //Se
espera la petición desde Ce
          str=msgDeCm.getContent();
          System.out.println(str);
          hora=str.substring(1,18);

            hora_ini=aEntero(hora.charAt(0),hora.charAt(1));
            min_ini=aEntero(hora.charAt(3),hora.charAt(4));//Conversiones de String a
entero
            hora_fin=aEntero(hora.charAt(6),hora.charAt(7));
            min_fin=aEntero(hora.charAt(9),hora.charAt(10));
            hora_dur=aEntero(hora.charAt(12),hora.charAt(13));
            min_dur=aEntero(hora.charAt(15),hora.charAt(16));

            dur=(hora_dur*2)+(min_dur%29); //duracion de la operacion en posiciones de
la tabla (cada posicion es media hora)
            ini=(hora_ini*2)+(min_ini%29); //inicio de la operacion en la posicion ini
de la tabla
            fin=(hora_fin*2)+(min_fin%29); //final de la operacion en la posicion fin

          sd.setType("quirofano");

            dfad.addAgentService(sd);     //añadimos el servicio al descriptor de agente
DF, para la busqueda de quirófanos

            msg.removeAllDests();

            try {
              result=searchDF("df",dfad,null);//procedemos a buscar en "df" lo que
queremos (dfad) sin constraints (null)

              Enumeration e = result.elements();//miramos los resultados uno a uno
              if (!e.hasMoreElements()) //Si no hay quirófanos => Error
                noHay=true;

              while(e.hasMoreElements()) {
                   //convertimos el resultado a DFAgentDescriptor para poder extraer el
nombre del agente encontrado
              AgentManagementOntology.DFAgentDescriptor actual =
(AgentManagementOntology.DFAgentDescriptor)e.nextElement();
              msg.addDest(actual.getName());//Con esto obtenemos el nombre del agente
              msgNuevo.addDest(actual.getName());
              }

            } catch (FIPAException exc) {
                System.out.println( exc.toString() );
            }
            if (!noHay) { //Si hay quirófanos disponibles
              msg.setContent("("+hora+")");

              send(msgNuevo); //Comienza el Contract-Net con ellos

              addBehaviour(new EnviaContractNet(Quirof.this,msg));
            } else { //Si no hay quirófanos disponibles, se informa de ello, tanto a Ce
como por pantalla
              System.out.println("No hay ningun quirofano en el hospital");
              msg.reset();
              msg.setSource(getLocalName());
              msg.setContent("(No hay ningun quirofano en el hospital.)");
              msg.addDest("Ce");
              send(msg);
              msg=blockingReceive(MessageTemplate.MatchSource("Ce")); //Se recibe un
mensaje de Ce donde indicará que se ha de
              System.out.println(getLocalName()+": Cancelando");      //cancelar el
proceso
            }
```

```
        }
      }
    });
  }

}
```

7.6. Horario.java

```
import java.io.*;
import java.util.Calendar;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.proto.*;
import jade.domain.AgentManagementOntology.*;
import jade.domain.*;


public class Horario extends Agent {

  String semana[][] = new String[49][8]; //Tabla que representa el horario del quirófano
  int hoy;
  ACLMessage mensaje;
  String padre = new String();
  Calendar fecha=Calendar.getInstance();

     public void inicializarSemana() { //Inicializa el horario, dejando todas las horas
libres

      String min,hora;
     int i,j,aux;

     hora="00";

      for (i=1;i<49;i++)
        for (j=1;j<8;j++)
          semana[i][j]="LIBRE";

      semana[0][0]="SEMANA";

    semana[0][1]="D";
    semana[0][2]="L";
    semana[0][3]="M";
    semana[0][4]="X";
    semana[0][5]="J";
    semana[0][6]="V";
    semana[0][7]="S";

      for (i=1;i<49;i++){
        if (i%2==0) min="30";
        else min="00";
        aux=(i-1)/2;
        if (aux<10) hora=("0"+aux);
        else hora=hora.valueOf(aux);
        semana[i][0]=(hora+":"+min);
      }

      fecha=Calendar.getInstance();
      hoy=fecha.get(Calendar.DAY_OF_WEEK);
   }

  class Subscripcion extends OneShotBehaviour { //Comportamiento de suscripición, sólo
se ejecuta una vez,
                                                 //al iniciar el agente
      public Subscripcion () {

      }

      public void action () {
           //accion del comportamiento, subscribimos un agente al DF con el servicio
que se pide por pantalla
          int len = 0;     //en caso de ser médico, se pide además la especialidad
          int opcion=4;
          byte[] buffer = new byte[1024];

         DFAgentDescriptor dfad =new DFAgentDescriptor();
         ServiceDescriptor sd =new ServiceDescriptor();
         boolean correcto=false;

         try {
```

```
            while (!correcto) {
               System.out.println("Que servicio tiene el agente? 1:Medico,
2:Enfermera, 3:Anestesista");
               len = System.in.read(buffer);
               if (len>2) {
                  String entrada = new String(buffer,0,len-2);
                  opcion = aEntero('0',entrada.charAt(0));
                  correcto=true;
               }
            }
            switch (opcion) {
            case 1: {
               sd.setName("medico");
               padre="M";
               correcto=false;
               while (!correcto) {
                    System.out.println("De que especialidad? 1:Corazon, 2:Pulmon,
3:Rinyon, 4:Higado");
                    len = System.in.read(buffer);
                    if (len>2) {
                       String entrada = new String(buffer,0,len-2);
                       opcion = aEntero('0',entrada.charAt(0));
                       correcto=true;
                    }
                 }
                 switch (opcion) {
                 case 1: {
                      sd.setType("medico-corazon");
                   break;
                 }
                 case 2: {
                   sd.setType("medico-pulmon");
                   break;
                 }
                 case 3: {
                   sd.setType("medico-rinyon");
                   break;
                 }
                 case 4: {
                   sd.setType("medico-higado");
                   break;
                 }
                 }
               break;
            }
            case 2: {
               sd.setName("enfermera");
               sd.setType("enfermera");
               padre="E";
               break;
            }
            case 3: {
               sd.setName("anestesista");
               sd.setType("anestesista");
               padre="A";
               break;
            }
            }

          System.out.print("Nombre del propietario? ");
           len = System.in.read(buffer);
          if(len > 2) dfad.setOwnership(new String(buffer,0,len-2));
          else dfad.setOwnership("nadie");

        } catch(IOException e) {
           System.out.println(e);
        }
        dfad.addAgentService(sd);
        dfad.removeAddresses();
        dfad.addAddress(getAddress());
        dfad.setName(getName());
        dfad.setType ("fipa-agent");
        dfad.setDFState("active");

        try { //Se informa del registro por pantalla
            registerWithDF("df", dfad);
```

```
            System.out.println("Agente "+getLocalName()+" de tipo "+sd.getType()+"
registrado al DF por "+dfad.getOwnership());
          } catch (FIPAException exc) {
               System.out.println( exc.toString() );
          }
          menu();
      }
  }

  public class RecibeContractNet extends FipaContractNetResponderBehaviour {
        //Parte "respondiente" del Contract-Net con M, A o E

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

    public ACLMessage handleCfpMessage(ACLMessage cfp) {

      String str,respuesta;
      int j,i,n;
      char c1,c2;

      diaDeHoy(); //Se actualiza el horario

      str=cfp.getContent(); //Se guarda la petición llegada desde el padre

      ACLMessage msg = new ACLMessage(ACLMessage.PROPOSE);
      msg.removeAllDests();
    msg.addDest(cfp.getSource());

    respuesta=elaborarPropuesta(str); //Se envían las propuestas
    msg.setContent(respuesta);
      return(msg);
    }

    public String elaborarPropuesta(String str) { //A partir de una petición, busca
horas libres cumpliendo las condiciones

      int i,dur,ini,fin,de,a;
      String resp;

      int hora_ini=aEntero(str.charAt(1),str.charAt(2));//Convertimos el String a
enteros,para trabajar con la tabla
      int min_ini=aEntero(str.charAt(4),str.charAt(5));
      int hora_fin=aEntero(str.charAt(7),str.charAt(8));
      int min_fin=aEntero(str.charAt(10),str.charAt(11));
      int hora_dur=aEntero(str.charAt(13),str.charAt(14));
      int min_dur=aEntero(str.charAt(16),str.charAt(17));

      dur=(hora_dur*2)+(min_dur%29);//Duración en cuadros de la tabla de horario. 1
cuadro => 30'
      ini=(hora_ini*2)+(min_ini%29)+1; //Hora de llegada del órgano en el cuadro ini
      fin=(hora_fin*2)+(min_fin%29)+1; //Hora de caducidad en el cuadro fin

      de=ini;
      a=ini;
      resp="";

      for (i=ini;i<fin;i++) {//Recorremos el horario buscando huecos libres donde
quepa la operación
        if (semana[i][hoy]=="LIBRE") a++;
        else {
          if ((a-de)>=dur) resp=resp+"("+semana[de][0]+" "+semana[a][0]+")";
          de=i+1; //Si encontramos un hueco, lo añadimos a la lista de horas
disponibles
          a=i+1;
        }
      }
      if ((a-de)>=dur) resp=resp+"("+semana[de][0]+" "+semana[a][0]+")";

      return("("+resp+")");
    }

    public ACLMessage handleAcceptProposalMessage(ACLMessage msg) { //Método para
manipular los mensajes de aceptación
      String str=msg.getContent();                                    //de
propuesta
```

```
       int i; //Nos pasan la hora que hemos de ocupar en el horario, donde se hará la
operación

       int hora_ini=aEntero(str.charAt(1),str.charAt(2));
       int min_ini=aEntero(str.charAt(4),str.charAt(5));
       int hora_fin=aEntero(str.charAt(7),str.charAt(8));
       int min_fin=aEntero(str.charAt(10),str.charAt(11));

       int ini=(hora_ini*2)+(min_ini%29)+1;
       int fin=(hora_fin*2)+(min_fin%29)+1;//Conversiones igual que antes

       for (i=ini;i<fin;i++)
         semana[i][hoy]="TRANS"; //Ocupamos los cuadros desde la hora de inicio hasta
el final de la operación

       ACLMessage msgDone = new ACLMessage(ACLMessage.INFORM);
       msgDone.addDest(msg.getSource());
       msgDone.setContent("(Done ("+str+"))"); //Informamos de que la actualización ha
sido un éxito

       return(msgDone);
     }

     public void handleRejectProposalMessage(ACLMessage msg) { //No hace nada, aunque
sirve para manejar los mensajes
                                                                    //de
rechazo desde el padre
     }

     public void handleOtherMessages(ACLMessage msg) {//Igual que antes, es para
manipular otro tipo de mensajes

     }

   }

   public int aEntero(char c1, char c2) { //Pasa a entero dos caracteres
     int n;
     n=java.lang.Character.getNumericValue(c1)*10;
     n+=java.lang.Character.getNumericValue(c2);
     return(n);
   }

   public void visualizarSemana(int hora) { //Se visualiza por pantalla el horario, seis
horas a partir de la hora indicada

     int i,j;

     if (hora>37) hora=37;
     if (hora<1) hora=1;
     System.out.println();
     System.out.print("   Hora      ");
     for (j=1;j<8;j++)
       System.out.print(semana[0][j]+"     ");
     System.out.println();
     System.out.println();
     for (i=hora;i<hora+12;i++) {
       for (j=0;j<8;j++){
         if (j==0) {
           if (i<10) System.out.print(" ");
           System.out.print(i+" ");
         }
         System.out.print(semana[i][j]+" ");
         if (j==0) System.out.print("  ");
       }
       System.out.println();
     }
     System.out.println();

   }

   public void actualizarDia(int dia, String str) { //Actualiza el día indicado, con un
determinado tipo de cita
                                                          //"OCUP" => fija, "TEMP"
=> temporal, "TRANS" => transplante
     String entrada = null;
     int len = 0;
```

```
    int hora=0;
    byte[] buffer = new byte[1024];

    try{
      System.out.print("Que intervalo horario quieres que este ocupado? (ENTER para
finalizar): ");
      len = System.in.read(buffer);
      while (len > 2) {
        entrada = new String(buffer,0,len-2);
        if (len==3) hora = aEntero('0',entrada.charAt(0));
        else hora = aEntero(entrada.charAt(0),entrada.charAt(1));
        if ((hora>0) && (dia>0) && (hora<49) && (dia<8))
          if (semana[hora][dia]!="TRANS") semana[hora][dia]=str;
          else System.out.println("No se puede ocupar una hora de transplante");
        else System.out.println("Datos incorrectos");
        visualizarSemana(hora-8);
        hora = 0;
        System.out.print("Que intervalo horario quieres que este ocupado? (ENTER para
finalizar): ");
        len = System.in.read(buffer);
      }
    } catch(IOException e) {
      System.out.println(e);
    }
  }

  public void diaDeHoy() {//Actualiza el día actual, eliminando citas temporales y
transplantes anteriores

                                //Supondremos que se ejecuta al menos una vez al día,
ya sea por consulta/modificación del
    int i,j;                  //horario, o petición de transplante, etc.. o sea que
sólo hará falta borrar las citas
                                //temporales del día anterior
    fecha=Calendar.getInstance();
    if (hoy!=fecha.get(Calendar.DAY_OF_WEEK)) {
      hoy=fecha.get(Calendar.DAY_OF_WEEK);
      for (j=1;j<49;j++)
        if ((semana[j][hoy-1]=="TEMP ") || (semana[j][hoy-1]=="TRANS"))
          semana[j][hoy-1]="LIBRE";
    }

  }

  public void menu() { //Menú, las opciones están explicadas en el capítulo 5 de la
documentación

    String entrada = null;
    int len = 0;
    int hora,dia,opcion;
    byte[] buffer = new byte[1024];
    boolean fin=false;

    diaDeHoy();

    opcion=8;
    try{
      while (! fin) {
        System.out.println("1:Visualizar horario");
        System.out.println("2:Modificar un dia");
        System.out.println("3:Inicializar toda la semana");
        System.out.println("4:Resetear toda la semana");
        System.out.println("5:Modificar el dia de hoy");
        System.out.println("9:Salir del menu");
        System.out.println();

       System.out.print("Que opcion quieres?:  ");
        len = System.in.read(buffer);
        if (len>2) {
          entrada = new String(buffer,0,len-2);
          opcion = aEntero('0',entrada.charAt(0));
        } else opcion=8;

        switch (opcion) {  //1:Visualizar horario
        case 1: {
          System.out.println("A partir de que hora? (intervalos de 30', se veran 6
horas, ENTER=>00:00): ");
```

```
            len = System.in.read(buffer);
            if (len==7) {
                 entrada = new String(buffer,0,len-2);

                 int hora_ini=aEntero(entrada.charAt(0),entrada.charAt(1));
                 int min_ini=aEntero(entrada.charAt(3),entrada.charAt(4));

                 hora=(hora_ini*2)+(min_ini%29)+1;

            } else hora=1;
            visualizarSemana(hora);
            break;
          }
          case 2: {   //2:Modificar un dia
            visualizarSemana(1);
            System.out.println("Que dia quieres modificar? (1:Domingo, 2:Lunes,
3:Martes, ...): ");
            len = System.in.read(buffer);
              if (len>2) {
                 entrada = new String(buffer,0,len-2);
                 dia = aEntero('0',entrada.charAt(0));
              } else dia=hoy;
            actualizarDia(dia,"TEMP ");
            break;
          }
          case 3: {   //3:Inicializar toda la semana
            visualizarSemana(1);
            for (dia=1;dia<8;dia++) {
              System.out.println("Modificando dia "+ semana[0][dia]+": ");
              actualizarDia(dia,"OCUP ");
            }
            break;
          }
          case 4: {   //4:Resetear toda la semana
            inicializarSemana();
            break;
          }
          case 5: {   //5:Modificar el dia de hoy
            visualizarSemana(1);
            actualizarDia(hoy,"TEMP ");
            break;
          }
          case 8: {   //8:Nada
               break;
          }
          case 9: {   //9:Salir del menu
               fin=true;
               break;
          }
          }
        }
      } catch(IOException e) {
          System.out.println(e);
      }
    }

public class RecibeMenu extends CyclicBehaviour { //Comportamiento que recibe un mensaje
cada vez que se ha
                                                          //de activar el menú de
este agente
    public RecibeMenu() {

    }

    public void action () {
      ACLMessage msgMenu = receive(MessageTemplate.MatchContent("(Menu)"));
      if(msgMenu != null) {
        System.out.println();
        menu();
      }
    }

}

public class RecibeDeArriba extends CyclicBehaviour { //Se recibe desde el padre la hora
de la operación
```

```
                                                                    //para que este
agente la puntúe en función del horario
  public RecibeDeArriba() {

  }

  public void action () {
    String str;
    MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource(padre),MessageTemplate.MatchType("inform
"));
    int hora_ini,min_ini,hora_fin,min_fin,puntuacion,ini,fin,i;

    ACLMessage msgDeArriba = receive(mt); //Si el mensaje no es el que queremos
recibir, simplemente no se hace nada
    if(msgDeArriba != null) {
      str=msgDeArriba.getContent(); //Nos quedamos con el intervalo

      hora_ini=aEntero(str.charAt(1),str.charAt(2));
      min_ini=aEntero(str.charAt(4),str.charAt(5));
      hora_fin=aEntero(str.charAt(7),str.charAt(8));
      min_fin=aEntero(str.charAt(10),str.charAt(11));
      puntuacion=3; //Encaja perfectamente, si no se cambia la puntuación, es que no
hay huecos ni antes ni después
                        //de la hora de la operación
      ini=(hora_ini*2)+(min_ini%29)+1;
      fin=(hora_fin*2)+(min_fin%29)+1;

      for (i=ini;i<fin;i++)
        if (semana[i][hoy]!="LIBRE") puntuacion=0;//Si no están libres todos los
intervalos de media hora
                                                                    //en que se hará la
operación => No se puede hacer a esa hora
      if (puntuacion!=0)
        if ((semana[ini-1][hoy]=="LIBRE") || (semana[fin][hoy]=="LIBRE")) {
          puntuacion=2; //Hay hueco por un lado
          if ((semana[ini-1][hoy]=="LIBRE") && (semana[fin][hoy]=="LIBRE"))
puntuacion=1; //Hay hueco por los dos lados
        }

      ACLMessage msgArriba = new ACLMessage(ACLMessage.INFORM);
      msgArriba.setContent("("+puntuacion+")");
      msgArriba.addDest(padre);//Se envía la puntuación al padre
      msgArriba.setOntology("Puntuacion");
      msgArriba.setSource(getName());
      send(msgArriba);
    }
  }

}

public class Cancelar extends CyclicBehaviour { //Para cancelar operaciones ya
confirmadas

  public Cancelar() {

  }

  public void action () {
    String str;
    MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource(padre),MessageTemplate.MatchType("reques
t"));
    mt = mt.and(mt,MessageTemplate.MatchOntology("Cancelacion"));
    int hora_ini,min_ini,hora_fin,min_fin,ini,fin,i;

    ACLMessage msgCancelar = receive(mt); //Se espera un mensaje desde el padre en el
que se indique una cancelación
    if(msgCancelar != null) {
      str=msgCancelar.getContent();

      hora_ini=aEntero(str.charAt(1),str.charAt(2)); //Conversiones
      min_ini=aEntero(str.charAt(4),str.charAt(5));
      hora_fin=aEntero(str.charAt(7),str.charAt(8));
      min_fin=aEntero(str.charAt(10),str.charAt(11));

      ini=(hora_ini*2)+(min_ini%29)+1;
```

```
       fin=(hora_fin*2)+(min_fin%29)+1;

       for (i=ini;i<fin;i++)  //Si en el horario aparecían las horas marcadas como
transplante, las dejamos de nuevo libres
         if (semana[i][hoy].equals("TRANS")) semana[i][hoy]="LIBRE";

       msgCancelar.reset();
       msgCancelar.setSource(getLocalName());
       msgCancelar.addDest(padre);
       msgCancelar.setOntology("Cancelada");
       send(msgCancelar);//Enviamos confirmación al padre de la cancelación

    }
  }

}


protected void setup() {
  addBehaviour(new Subscripcion()); //Se añade el comportamiento de suscripción
  inicializarSemana();

    addBehaviour(new RecibeMenu()); //Se añaden tres comportamientos más
    addBehaviour(new RecibeDeArriba());
    addBehaviour(new Cancelar());

    addBehaviour(new CyclicBehaviour(this) { //Se añade este comportamiento cíclico que
añadirá el de RecibeContractNet
      public void action() {                //cada vez que se vaya a recibir un
órgano nuevo
       ACLMessage msgNuevo=receive(MessageTemplate.MatchContent("(Organo nuevo)"));
       if (msgNuevo!=null) {
            addBehaviour(new RecibeContractNet(Horario.this));
       }
      }
    });
  }
}
```

7.7. HorarioQ.java

```
import java.io.*;
import java.util.Calendar;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.proto.*;
import jade.domain.AgentManagementOntology.*;
import jade.domain.*;


public class HorarioQ extends Agent {

   String semana[][] = new String[49][8]; //Tabla que representa el horario del quirófano
   int hoy;
   ACLMessage mensaje;
   Calendar fecha=Calendar.getInstance();

    public void inicializarSemana() { //Inicializa el horario, dejando todas las horas
libres

     String min,hora;
     int i,j,aux;
     hora="00";

     for (i=1;i<49;i++)
       for (j=1;j<8;j++)
          semana[i][j]="LIBRE";

     semana[0][0]="SEMANA";

   semana[0][1]="D";
   semana[0][2]="L";
   semana[0][3]="M";
   semana[0][4]="X";
   semana[0][5]="J";
   semana[0][6]="V";
   semana[0][7]="S";

     for (i=1;i<49;i++){
       if (i%2==0) min="30";
       else min="00";
       aux=(i-1)/2;
       if (aux<10) hora=("0"+aux);
       else hora=hora.valueOf(aux);
       semana[i][0]=(hora+":"+min);
     }

     fecha=Calendar.getInstance();
     hoy=fecha.get(Calendar.DAY_OF_WEEK);

  }

  class Subscripcion extends OneShotBehaviour { //Comportamiento de suscripición, sólo
se ejecuta una vez,
                                                          //al iniciar el agente
      public Subscripcion () {

      }

      public void action () {
            //accion del comportamiento, subscribimos un agente al DF con el servicio de
quirófano
           int len = 0;
           byte[] buffer = new byte[1024];

          DFAgentDescriptor dfad =new DFAgentDescriptor();
          ServiceDescriptor sd =new ServiceDescriptor();

          sd.setName("quirofano");
           sd.setType("quirofano");

          try {
```

```
            System.out.print("Nombre del propietario? ");
            len = System.in.read(buffer);
            if(len > 2) dfad.setOwnership(new String(buffer,0,len-2));
            else dfad.setOwnership("nadie");
          } catch(IOException e) {
             System.out.println(e);
          }

          dfad.addAgentService(sd);
          dfad.removeAddresses();
          dfad.addAddress(getAddress());
          dfad.setName(getName());
          dfad.setType ("fipa-agent");
          dfad.setDFState("active");

          try { //Se informa del registro por pantalla
              registerWithDF("df", dfad);
              System.out.println("Agente "+getLocalName()+" de tipo "+sd.getType()+"
registrado al DF por "+dfad.getOwnership());
          } catch (FIPAException exc) {
                System.out.println( exc.toString() );
          }
          menu();
      }
  }

  public class RecibeContractNet extends FipaContractNetResponderBehaviour {
         //Parte "respondiente" del Contract-Net con Q

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

    public ACLMessage handleCfpMessage(ACLMessage cfp) { //Para manipular los mensajes
cfp enviados desde Q

       String str,respuesta;
       int j,i,n;
       char c1,c2;

       diaDeHoy(); //Se actualiza el horario

       str=cfp.getContent();//Se guarda la petición de Q

       ACLMessage msg = new ACLMessage(ACLMessage.PROPOSE);
       msg.removeAllDests();
    msg.addDest(cfp.getSource());

    respuesta=elaborarPropuesta(str); //Se envían las propuestas
    msg.setContent(respuesta);
       return(msg);
    }

    public String elaborarPropuesta(String str) { //A partir de una petición, busca
horas libres cumpliendo las condiciones

       int i,dur,ini,fin,de,a;
       String resp;

       int hora_ini=aEntero(str.charAt(1),str.charAt(2));//Convertimos el String a
enteros,para trabajar con la tabla
       int min_ini=aEntero(str.charAt(4),str.charAt(5));
       int hora_fin=aEntero(str.charAt(7),str.charAt(8));
       int min_fin=aEntero(str.charAt(10),str.charAt(11));
       int hora_dur=aEntero(str.charAt(13),str.charAt(14));
       int min_dur=aEntero(str.charAt(16),str.charAt(17));

       dur=(hora_dur*2)+(min_dur%29);//Duración en cuadros de la tabla de horario. 1
cuadro => 30'
       ini=(hora_ini*2)+(min_ini%29)+1; //Hora de llegada del órgano en el cuadro ini
       fin=(hora_fin*2)+(min_fin%29)+1; //Hora de caducidad en el cuadro fin

       de=ini;
       a=ini;
       resp="";
```

```
       for (i=ini;i<fin;i++) {//Recorremos el horario buscando huecos libres donde
quepa la operación
          if (semana[i][hoy]=="LIBRE") a++;
          else {
            if ((a-de)>=dur) resp=resp+"("+semana[de][0]+" "+semana[a][0]+")";
            de=i+1; //Si encontramos un hueco, lo añadimos a la lista de horas
disponibles
            a=i+1;
          }
       }
       if ((a-de)>=dur) resp=resp+"("+semana[de][0]+" "+semana[a][0]+")";

       return("("+resp+")");
     }

     public ACLMessage handleAcceptProposalMessage(ACLMessage msg) { //Método para
manipular los mensajes de aceptación
       String str=msg.getContent();                                          //de
propuesta
       int i; //Nos pasan la hora que hemos de ocupar en el horario, donde se hará la
operación

       int hora_ini=aEntero(str.charAt(1),str.charAt(2));
       int min_ini=aEntero(str.charAt(4),str.charAt(5));
       int hora_fin=aEntero(str.charAt(7),str.charAt(8));
       int min_fin=aEntero(str.charAt(10),str.charAt(11));

       int ini=(hora_ini*2)+(min_ini%29)+1;
       int fin=(hora_fin*2)+(min_fin%29)+1;//Conversiones igual que antes

       for (i=ini;i<fin;i++)
         semana[i][hoy]="TRANS"; //Ocupamos los cuadros desde la hora de inicio hasta
el final de la operación

       ACLMessage msgDone = new ACLMessage(ACLMessage.INFORM);
       msgDone.addDest(msg.getSource());
       msgDone.setContent("(Done ("+str+"))"); //Informamos de que la actualización ha
sido un éxito

       return(msgDone);
     }

     public void handleRejectProposalMessage(ACLMessage msg) { //No hace nada, aunque
sirve para manejar los mensajes
                                                                               //de
rechazo desde Q
     }

     public void handleOtherMessages(ACLMessage msg) {//Igual que antes, es para
manipular otro tipo de mensajes

     }

   }

   public int aEntero(char c1, char c2) { //Pasa a entero dos caracteres
     int n;
     n=java.lang.Character.getNumericValue(c1)*10;
     n+=java.lang.Character.getNumericValue(c2);
     return(n);
   }

   public void visualizarSemana(int hora) { //Se visualiza por pantalla el horario, seis
horas a partir de la hora indicada

     int i,j;

     if (hora>37) hora=37;
     if (hora<1) hora=1;
     System.out.println();
     System.out.print("   Hora      ");
     for (j=1;j<8;j++)
       System.out.print(semana[0][j]+"     ");
     System.out.println();
     System.out.println();
     for (i=hora;i<hora+12;i++) {
       for (j=0;j<8;j++){
```

```
          if (j==0) {
            if (i<10) System.out.print(" ");
            System.out.print(i+" ");
          }
          System.out.print(semana[i][j]+" ");
          if (j==0) System.out.print("  ");
        }
        System.out.println();
      }
      System.out.println();

    }

    public void actualizarDia(int dia, String str) { //Actualiza el día indicado, con un
determinado tipo de cita
                                                          //"OCUP" => fija, "TEMP"
=> temporal, "TRANS" => transplante
      String entrada = null;
      int len = 0;
      int hora=0;
      byte[] buffer = new byte[1024];

      try{
        System.out.print("Que intervalo horario quieres que este ocupado? (ENTER para
finalizar): ");
        len = System.in.read(buffer);
        while (len > 2) {
          entrada = new String(buffer,0,len-2);
          if (len==3) hora = aEntero('0',entrada.charAt(0));
          else hora = aEntero(entrada.charAt(0),entrada.charAt(1));
          if ((hora>0) && (dia>0) && (hora<49) && (dia<8))
            if (semana[hora][dia]!="TRANS") semana[hora][dia]=str;
            else System.out.println("No se puede ocupar una hora de transplante");
          else System.out.println("Datos incorrectos");
          visualizarSemana(hora-8);
          hora = 0;
          System.out.print("Que intervalo horario quieres que este ocupado? (ENTER para
finalizar): ");
          len = System.in.read(buffer);
        }
      } catch(IOException e) {
        System.out.println(e);
      }
    }

    public void diaDeHoy() {//Actualiza el día actual, eliminando citas temporales y
transplantes anteriores

                                  //Supondremos que se ejecuta al menos una vez al día,
ya sea por consulta/modificación del
      int i,j;                  //horario, o petición de transplante, etc.. o sea que
sólo hará falta borrar las citas
                                  //temporales del día anterior
      fecha=Calendar.getInstance();
      if (hoy!=fecha.get(Calendar.DAY_OF_WEEK)) {
        hoy=fecha.get(Calendar.DAY_OF_WEEK);
        for (j=1;j<49;j++)
          if ((semana[j][hoy-1]=="TEMP ") || (semana[j][hoy-1]=="TRANS"))
            semana[j][hoy-1]="LIBRE";
      }

    }

    public void menu() { //Menú, las opciones están explicadas en el capítulo 5 de la
documentación

      String entrada = null;
      int len = 0;
      int hora,dia,opcion;
      byte[] buffer = new byte[1024];
      boolean fin=false;

      diaDeHoy();

      opcion=8;
      try{
        while (! fin) {
```

```
          System.out.println("1:Visualizar horario");
          System.out.println("2:Modificar un dia");
          System.out.println("3:Inicializar toda la semana");
          System.out.println("4:Resetear toda la semana");
          System.out.println("5:Modificar el dia de hoy");
          System.out.println("9:Salir del menu");
          System.out.println();

         System.out.print("Que opcion quieres?:  ");
         len = System.in.read(buffer);
         if (len>2) {
           entrada = new String(buffer,0,len-2);
           opcion = aEntero('0',entrada.charAt(0));
         } else opcion=8;

         switch (opcion) {  //1:Visualizar horario
         case 1: {
           System.out.println("A partir de que hora? (intervalos de 30', se veran 6
horas, ENTER=>00:00): ");
           len = System.in.read(buffer);
           if (len==7) {
                entrada = new String(buffer,0,len-2);

                int hora_ini=aEntero(entrada.charAt(0),entrada.charAt(1));
                int min_ini=aEntero(entrada.charAt(3),entrada.charAt(4));

                hora=(hora_ini*2)+(min_ini%29)+1;

           } else hora=1;
           visualizarSemana(hora);
           break;
         }
         case 2: {  //2:Modificar un dia
           visualizarSemana(1);
           System.out.println("Que dia quieres modificar? (1:Domingo, 2:Lunes,
3:Martes, ...): ");
           len = System.in.read(buffer);
             if (len>2) {
               entrada = new String(buffer,0,len-2);
               dia = aEntero('0',entrada.charAt(0));
             } else dia=hoy;
           actualizarDia(dia,"TEMP ");
           break;
         }
         case 3: {  //3:Inicializar toda la semana
           visualizarSemana(1);
           for (dia=1;dia<8;dia++) {
             System.out.println("Modificando dia "+ semana[0][dia]+": ");
             actualizarDia(dia,"OCUP ");
           }
           break;
         }
         case 4: {  //4:Resetear toda la semana
           inicializarSemana();
           break;
         }
         case 5: {  //5:Modificar el dia de hoy
           visualizarSemana(1);
           actualizarDia(hoy,"TEMP ");
           break;
         }
         case 8: {  //8:Nada
              break;
         }
         case 9: {  //9:Salir del menu
              fin=true;
              break;
         }
         }
       }
     } catch(IOException e) {
        System.out.println(e);
     }
   }

public class RecibeMenu extends CyclicBehaviour { //Comportamiento que recibe un mensaje
cada vez que se ha
```

```
                                                                    //de activar el menú de
este agente
  public RecibeMenu() {

  }

  public void action () {
    ACLMessage msgMenu = receive(MessageTemplate.MatchContent("(Menu)"));
    if(msgMenu != null) {
      System.out.println();
      menu();
    }
  }

}

public class RecibeDeQ extends CyclicBehaviour { //Se reciben desde Q las posibles horas
para realizar la operación
                                                                  //para que este agente las
puntúe en función del horario
  public RecibeDeQ() {

  }

  public void action () {
    String str;
    String mejorHora=new String();
    MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource("Q"),MessageTemplate.MatchType("inform")
);
    int j,hora_ini,min_ini,hora_fin,min_fin,puntuacion,ini,fin,i;
    int mejorPuntuacion=0;

    ACLMessage msgDeQ = receive(mt); //Si el mensaje no es el que queremos recibir,
simplemente no se hace nada
    if(msgDeQ != null) {
      str=msgDeQ.getContent().substring(1,msgDeQ.getContent().length()-1); //Nos
quedamos con los intervalos
      for (j=0;j<str.length()-12;j=j+13) {
        hora_ini=aEntero(str.charAt(j+1),str.charAt(j+2)); //Convertimos a enteros...
        min_ini=aEntero(str.charAt(j+4),str.charAt(j+5));
        hora_fin=aEntero(str.charAt(j+7),str.charAt(j+8));
        min_fin=aEntero(str.charAt(j+10),str.charAt(j+11));
        puntuacion=3; //Encaja perfectamente, si no se cambia la puntuación, es que
no hay huecos ni antes ni después
                        //de la hora de la operación
        ini=(hora_ini*2)+(min_ini%29)+1;
        fin=(hora_fin*2)+(min_fin%29)+1;

        for (i=ini;i<fin;i++)
          if (semana[i][hoy]!="LIBRE") puntuacion=0;//Si no están libres todos los
intervalos de media hora
                                                                  //en que se hará la
operación => No se puede hacer a esa hora
        if (puntuacion!=0)
          if ((semana[ini-1][hoy]=="LIBRE") || (semana[fin][hoy]=="LIBRE")) {
            puntuacion=2; //Hay hueco por un lado
            if ((semana[ini-1][hoy]=="LIBRE") && (semana[fin][hoy]=="LIBRE"))
puntuacion=1; //Hay hueco por los dos lados
          }

        if (puntuacion>mejorPuntuacion) { //Se busca la mejor puntuación de las horas
posibles
          mejorHora="("+str.substring(j,j+13)+" "+puntuacion+")";
          mejorPuntuacion=puntuacion;
        }
      }

      if (mejorPuntuacion==0) mejorHora="((XX:XX XX:XX) 0)"; //Si no se puede hacer a
ninguna hora, se envía esto

      ACLMessage msgAQ = new ACLMessage(ACLMessage.INFORM);
      msgAQ.setContent(mejorHora);//Finalmente se envía la mejor hora obtenida
      msgAQ.addDest("Q");
      msgAQ.setOntology("Puntuacion");
      msgAQ.setSource(getName());
      send(msgAQ);
```

```
    }
  }

}

public class Cancelar extends CyclicBehaviour { //Para cancelar operaciones ya
confirmadas

  public Cancelar() {

  }

  public void action () {
    String str;
    MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchSource("Q"),MessageTemplate.MatchType("request"
));
    mt = mt.and(mt,MessageTemplate.MatchOntology("Cancelacion"));
    int hora_ini,min_ini,hora_fin,min_fin,ini,fin,i;

    ACLMessage msgCancelar = receive(mt); //Se espera un mensaje desde Q en el que se
indique una cancelación
    if(msgCancelar != null) {
      str=msgCancelar.getContent();

      hora_ini=aEntero(str.charAt(1),str.charAt(2)); //Conversiones
      min_ini=aEntero(str.charAt(4),str.charAt(5));
      hora_fin=aEntero(str.charAt(7),str.charAt(8));
      min_fin=aEntero(str.charAt(10),str.charAt(11));

      ini=(hora_ini*2)+(min_ini%29)+1;
      fin=(hora_fin*2)+(min_fin%29)+1;

      for (i=ini;i<fin;i++)  //Si en el horario aparecían las horas marcadas como
transplante, las dejamos de nuevo libres
        if (semana[i][hoy].equals("TRANS")) semana[i][hoy]="LIBRE";

      msgCancelar.reset();
      msgCancelar.setSource(getLocalName());
      msgCancelar.addDest("Q");
      msgCancelar.setOntology("Cancelada");
      send(msgCancelar);//Enviamos confirmación a Q de la cancelación

    }
  }

}

protected void setup() {
  addBehaviour(new Subscripcion()); //Se añade el comportamiento de suscripción
  inicializarSemana();

    addBehaviour(new RecibeMenu()); //Se añaden tres comportamientos más
    addBehaviour(new RecibeDeQ());
    addBehaviour(new Cancelar());

    addBehaviour(new CyclicBehaviour(this) { //Se añade este comportamiento cíclico que
añadirá el de RecibeContractNet
      public void action() {                 //cada vez que se vaya a recibir un
órgano nuevo
       ACLMessage msgNuevo=receive(MessageTemplate.MatchContent("(Organo nuevo)"));
       if (msgNuevo!=null) {
            addBehaviour(new RecibeContractNet(HorarioQ.this));
       }
      }
    });
  }
}
```

8. VALORACIÓN Y CONCLUSIONES

Los objetivos pretendidos en la realización de este proyecto se han cumplido. Se ha conseguido un sistema realista, con variedad de opciones y capaz de ser implantado en un hospital, aunque obviamente, no tal cual está ahora.

Gracias al proyecto, he podido conocer un mundo totalmente nuevo, como es el de los agentes, y en concreto los SMA. Un mundo inexplorado hasta el momento por mí, y que ofrece muchas posibilidades. Hace apenas unos meses no sabía ni lo que era un agente, pero en este tiempo he ido descubriendo como avanza la inteligencia artificial, de la que los SMA son una parte importante. Pese a esto, los SMA no tienen demasiado que ver con la clase de IA que vimos en asignaturas como Introducción a la Inteligencia Artificial (IIA), o Métodos y Técnicas de la IA (MTIA), que son las que personalmente he cursado relacionadas con el tema. Y es que de hecho hay muchas ramas de la IA, y cada una es bastante diferente de las otras.

El proyecto, también ayuda en la programación, en este caso en el manejo de Java, que es el lenguaje utilizado. Lógicamente, cuanto más se programa, más soltura se coge.

Es positiva la creación de un grupo como el GruSMA, que intenta acercarnos más a este mundo de los SMA, proponiendo proyectos, en los cuales los miembros del grupo puedan intercambiar sus dudas y descubrimientos.

Muchas de las cosas que no se han hecho, no han sido posibles por la limitación del tiempo, pero se podrían realizar en el futuro.

En definitiva, este proyecto, ha servido básicamente, para tomar un primer contacto con el mundo de los Sistemas Multi-agente y profundizar en la Inteligencia Artificial.

9. TRABAJOS FUTUROS

Ya hemos comentado varias veces las limitaciones de este proyecto en algunas cuestiones. Estas limitaciones se podrían subsanar en el futuro, siguiendo las líneas ya marcadas por este proyecto.

Algunas de las cosas que se podrían hacer son:

- Generalizar el sistema para otros temas. Aprovechando la arquitectura del sistema, se podría hacer algo similar, pero cambiando el contexto, como ya se comenta al principio del documento.

- Mejorar aspectos de este proyecto como la interficie con el usuario. Se podría hacer en modo gráfico y permitir más opciones.

- Distribuir el sistema en diferentes máquinas. A priori no habría que modificar mucho el sistema, en cuanto a código, y permitiría hacer posible la implantación en un hospital.

- Ampliar el sistema para tratar más tipos de órgano o diferentes características de los quirófanos. Más especialidades, también en enfermeras y anestesistas.

- Mejorar la propia IA de sistema, sobretodo a la hora de asignar hora, quirófano y personal a una operación. También se tendrían que prever cosas como que haya operaciones a media noche, cosa que no trata este sistema.

- Repartir las operaciones de forma equitativa entre el personal y los quirófanos, para que no queden algunos saturados de operaciones y otros sin apenas trabajo.

- Tener en cuenta a la hora de cancelar una operación ya confirmada, que puede haber varias a la misma hora, por lo que se tendría que hacer algún tipo de modificación.

- Conseguir coordinar las tres partes del proyecto global. Como ya hemos comentado, no debería entrañar demasiada dificultad.

Resumiendo, el camino que queda por recorrer es aún muy largo, y de hecho lo será por muchas cosas que se hagan en el futuro. Porque como todo actualmente, el mundo de la IA no para de avanzar y surgen cosas nuevas a cada momento.

10. BIBLIOGRAFÍA

[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*