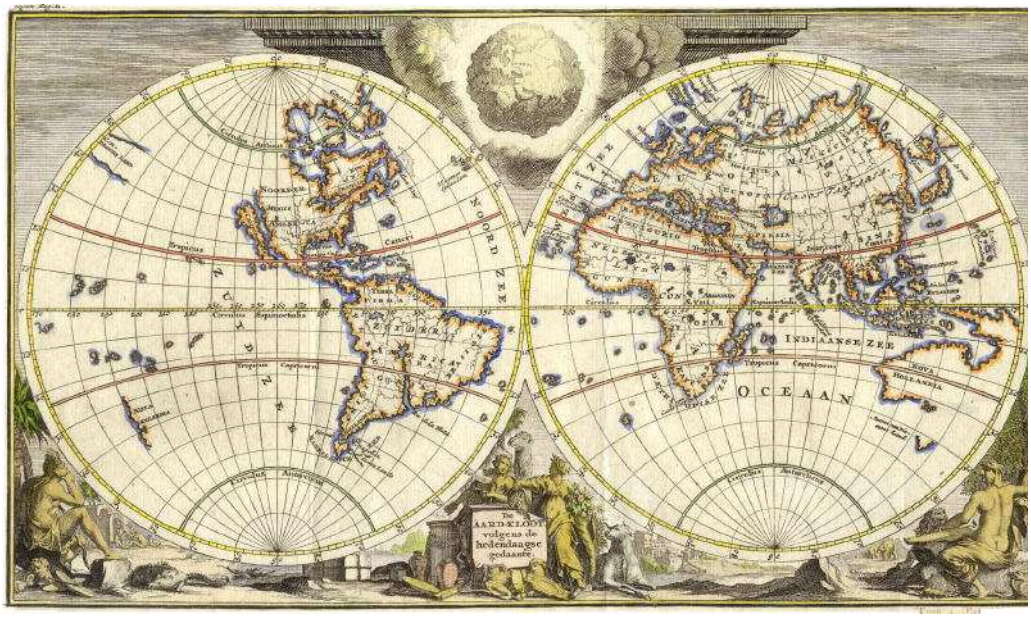
ESCOLA TÈCNICA SUPERIOR ENGINYERIA

UNIVERSITAT ROVIRA I VIRGILI

SMA PFC Travel



"Cualquier destino del mundo, a un solo clic"

Título: SMA PFC Travel
Autor: Fernando M. Sacasa López
Director de Proyecto: Dr. Antonio Moreno Ribas
Estudios: Ingeniería en Informática
Centro Universitario: Universitat Rovira i Virgili (URV)
Departamento: Departament d'Enginyeria Informàtica i Matemàtiques
Fecha Conclusión: Septiembre de 2005

[DΣIM]

1. Introducción

La explosión de Internet como medio de lucro e investigación ha repercutido directamente en la filosofía viajera de nuestros tiempos. La creciente expansión de este medio, y la relativa facilidad y bajo coste del mismo, ha permitido que entre en nuestros hogares de manera progresiva, casi sin darnos cuenta; abriéndonos un abanico infinito de posibilidades, donde saber lo qué buscamos y cómo lo buscamos adquiere una importancia trascendental, ya que la información no se encuentra siempre fácilmente accesible ni ordenada. Más bien, podríamos considerar la red de redes como un camino laberíntico donde muchos no consiguen encontrar el modo de solucionar sus necesidades. Muchas veces, se podría considerar como una jungla sombría en la cual es fácil perderse si no se dispone de los medios adecuados.

Antaño, la filosofía viajera se centraba en el descubrimiento de lo desconocido, en explorar nuevos parajes y realizar guías y relatos para que los futuros viajeros supieran a qué atenerse.

Eran viajes largos, pesados, lentos y llenos de peligros. Se sabía de donde se partía, pero no hasta donde se llegaría.

Con el avance de los tiempos, la proliferación de negocios como las posadas o pensiones, que permitían un fácil modo de alojamiento, lejos de la peligrosidad de las acampadas en los flancos de los caminos, así como también la evolución en cuanto a seguridad en los estados más avanzados, generaron una nueva visión para el viajero.

Desde entonces, se convirtió en algo mucho más ocioso. Ya se sabía donde se iría, y cómo hospedarse. El descubrimiento y la aventura, se sustituyó por la simple necesidad de colmar la curiosidad respecto otros modos de vivir, u otros parajes distintos al nuestro.

La evolución llegó a tal punto que, desde hace relativamente poco, el tipo de viajeros pasaron a llamarse turistas, a los cuales se les incluía en paquetes de viaje prediseñados, donde estaba todo resuelto y planeado.

Juntamente a la evolución de Internet, hubo otro factor determinante. La aparición de compañías de vuelo de bajo coste abría a los usuarios nuevos horizontes donde dirigir sus intereses.

Desde entonces, existe un nuevo fenómeno en cuanto a la filosofía de los viajeros. De alguna manera se ha vuelto a la búsqueda de la aventura, a recorrer parajes desconocidos, a huir de las aglomeraciones turísticas y buscar los destinos más tranquilos y variopintos, que ninguna empresa o agencia de viaje ofrece como producto, debido a que los gustos e intereses son tan dispares, que no pueden asumir riesgos de crear productos con demandas tan difusas.

Muchos viajeros prefieren planear el viaje por sí mismos, para ello tienen una herramienta muy potente. Como hemos comentado, en Internet podemos encontrar todas las alternativas a cualquier problema. Y este, no podía ser una excepción. Pero, como también hemos comentado, es realmente complicado evaluar todas las alternativas existentes, sin perderse o desesperarse, debido a lo complicado que resulta combinar diferentes tipos de transportes, horarios y todo al mejor precio.

Es por ello que resulta altamente interesante el desarrollo de una aplicación que realice todas estas búsquedas de manera autónoma y eficiente. Haciendo que el proceso de planificación pase de varios días, o incluso semanas, a apenas unos minutos en los casos más complicados.

“Divide et vinces". Esta frase célebre de Julio César nos inspira el uso de un entorno distribuido, donde se pueda descomponer el problema para que sea tratado independientemente cada uno de los componente del mismo, lo cual trae implícitas un gran número de ventajas que discutimos en siguientes apartados de este mismo documento.

Si a esto le añadimos las necesidades de exploración, autonomía y comunicación en el caso que nos atañe. Podemos pensar que un eficiente método para resolverlo sea el uso de un Sistema Multi Agente que cooperen y se comuniquen entre ellos.

1.1 Grupo de investigación Banzai - GruSMA

El grupo de investigación en el campo de la Inteligencia Artificial llamado BANZAI forma parte de la Universidad Rovira i Virgili, situada en Tarragona (España).

Es uno de los equipos de investigación enmarcados dentro del departamento de Ingeniería Informática y Matemáticas (DEIM) y está formado por diversos profesores y estudiantes de Doctorado.

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

GruSMA proporciona un entorno de trabajo en el cual los alumnos de Informática de la ETSE, interesados en las nuevas tecnologías de la Inteligencia Artificial, pueden compartir conocimientos y experiencias en el campo de los Sistemas Multi Agente. Dicho grupo trabaja en el desarrollo de sistemas utilizando una herramienta para el desarrollo de SMAs llamada JADE, la cual cumple con las especificaciones de la FIPA.

1.2 FIPA

Para definir estándares (de arquitectura, lenguajes, protocolos…) se creó la FIPA, una fundación sin animo de lucro nacida con el objetivo de establecer las reglas para el diseño e implementación de un SMA y garantizar la interoperabilidad que comentábamos en el apartado anterior. Esta organización inició su andadura en 1996, haciendo pública su primera especificación, y desde entonces se ha ido imponiendo hasta conseguir ser el estándar seguido de facto por un gran número de grupos de investigación académicos e industriales.

Una vez conseguida su meta principal de obtener un grueso de especificaciones útiles a la hora de crear SMAs ínter operables. El siguiente paso era conseguir que los Agentes vean garantizada la comunicación con tecnologías que no se basan en ellos. Dentro de esta perspectiva, en marzo del 2005, la FIPA se unió a la IEEE Computer Society, pasándose a llamar FIPA Standards Committe.

El IEEE FIPA Standards Committe actúa como órgano independiente, con sus propias políticas, estructuras, procedimientos…pero beneficiándose de las grandes ventajas que obtiene al encontrarse bajo la organización de la IEEE Computer Society.

En nuestro grupo de trabajo, hemos partido de las especificaciones aportadas por la FIPA para desarrollar todo un conjunto de proyectos que desean explotar las ventajas de los Agentes, siguiendo el modelo impuesto por la FIPA. En el caso de nuestro estudio, nos hemos valido de una herramienta de desarrollo de SMAs que cumple las especificaciones FIPA; por tanto, nos extenderemos más adelante en los comentarios sobre estas reglas tanto de diseño como de implementación.

1.3 Objetivos del proyecto

1.3.1 Objetivos técnicos

Nuestro camino se marca como meta investigar cómo aglomerar, en un mismo aplicativo, tecnologías de lo más dispares, pero que cuentan como punto de convergencia, la naturaleza distribuida de las mismas.

Es por ello que hemos decidido crear una aplicación con un alto grado de matices en cuanto al uso de técnicas nunca usadas hasta ahora en nuestro departamento, relacionándolas todas ellas con los Agentes.

Así pues, nos adentramos en Agentes capaces de interactuar con Internet, realizar búsquedas, consultas, interactuar con formularios, realizar el papel tanto de cliente como de servidor, de usar invocaciones remotas a métodos, creación dinámica de Agentes por medio de JSP's, y un largo etcétera que iremos detallando paso a paso a lo largo del documento. No obstante los nombraremos en la siguiente enumeración:

- Creación e inclusión en plataforma SMA de un Agente de manera dinámica a través de formularios HTML y de páginas JSP.
- Uso de métodos de invocación remota dentro de un Agente a través de CORBA por medio de un servidor especializado.
- Acceso a bases de datos.

- Capacidad de exploración en Internet. Realizando funciones de cliente, de servidor y de usuario. Creamos peticiones nuevas y atendemos peticiones tipo GET o POST.
- Total manejo de estado de una conexión por medio de sesiones y cookies (tratamiento del contexto de la conexión).

1.3.2 Objetivos funcionales

Nuestra aplicación permitirá al usuario realizar planificaciones de rutas entre dos puntos. Es decir, introduciendo un punto de partida y un punto de destino, el sistema proporcionará una secuencia de subtrayectos encadenados que se deberían seguir para conseguir llegar de un punto a otro. Cada subtrayecto tiene asociadas varias características, como puede ser el precio del mismo, el medio de transporte usado, las horas de partida y de llegada, el día que se efectúa e información adicional dependiendo del medio usado; por ejemplo, nos puede dar el número de vuelo, aeropuertos y compañía aérea si se trata de un trayecto realizado en avión; o bien, nos puede decir la distancia y los pueblos que deberemos ir atravesando si lo realizamos por carretera; si es por tren, nos indicaría las estaciones, el tipo de tren, los días que ese mismo tren efectúa el servicio, etc.

Comentar también que el usuario, a partir de un formulario, podrá personalizar la consulta para obtener los resultados deseados. Los parámetros disponibles se explicarán en apartados sucesivos.

Gracias a esta aplicación, que engloba y accede a toda la información necesaria en cuanto a medios de transporte, un usuario no tendrá que estar navegando por un innumerable número de páginas realizando malabares para encontrar combinaciones adecuadas, cuadrando horarios, escalas, etc. El sistema lo realizará de manera automática y presentará los resultados en una secuencia de pasos ordenados que lo llevarán, prácticamente, hasta donde sus deseos de viajar y moverse le permitan.

1.4 Estructura del documento

Se inicia el documento con un capítulo introductorio, donde se explican algunos de los conceptos genéricos principales necesarios para comprender el contexto de nuestro trabajo; el cual nos ayudará a ubicarnos en la parcela donde nos centramos, dentro del gran universo de la inteligencia artificial.

Dedicaremos íntegramente el siguiente capítulo para profundizar en el campo de los Agentes y de los Sistemas Multi Agente (SMA).

Muy relacionado con el anterior, se incluye, en el tercer punto, un amplio estudio, sin afán de ser exhaustivo, sobre JADE, la tecnología con la cual implementaremos nuestros Agentes.

Seguiremos el estudio de las tecnologías empleadas en el cuarto apartado. Donde se comentarán de manera introductoria, todos aquellos elementos que entran en juego en el desarrollo de la aplicación.

Posteriormente se hará un análisis completo del Diseño de nuestra plataforma SMA y de los diferentes módulos que la complementan, como pueden ser la aplicación Web, los servicios implementados en CORBA, etc.

El capítulo seis está dedicado a orientar tanto al usuario final como al administrador del sistema, en el uso de la misma, para el primero y de la configuración, arranque y mantenimiento de la aplicación para el segundo.

El séptimo apartado se encarga de comentar algunos ejemplos de funcionamiento, para comprobar que sean correctos. Por tanto, se efectúan varios juegos de pruebas para entender los procedimientos y los casos de excepción, así como también aquellos casos de flujo normal.

Aprovecharemos el octavo capítulo para explicar en qué punto se encuentra nuestro proyecto, tanto a nivel de desarrollo interno, como en el contexto tecnológico donde se emplaza; también tendremos la oportunidad de compararlo con otros sistemas similares.

Los dos últimos capítulos están destinados a recopilar las referencias bibliográficas útiles para el desarrollo de las diferentes partes de la aplicación, y para añadir información extra que pueda ser de interés para el lector en forma de anexos.

1.5 Recursos utilizados

Para finalizar el primer apartado, haremos un breve repaso a los recursos que han sido de utilidad en la creación de nuestra aplicación.

Hardware:

- Ordenador portátil Acer TM529ATX PIII 900Mhz, 384 Mb RAM (S.O. Windows XP)
- Ordenador sobremesa AMD Duron 750Mhz, 384 Mb RAM. (S.O. Windows XP)

Software:

- J2SE v .4.2 de Sun Microsystems
- JADE v3.1 de Telecom Italia Lab
- MySQL Server v3.23.47-nt
- Eclipse
- CORBA
- Macromedia DreamWeaver
- Protege 2.1 beta
- Apache Tomcat 4.1

2. Agentes y Sistemas Multi Agente

Los Sistemas Multi Agente están compuestos por unidades más pequeñas que cooperan entre sí, denominadas Agentes.

Podemos encontrar diversas definiciones de Agente, todas ellas recogidas en el Desenvolupament de Sistemes Multi Agent en JADE, Report de Recerca [Isern et al., 2001]. De todos modos, nos quedaremos con la explicación ofrecida por el doctor Wooldridge en [Wooldridge02]:

"Un Agente inteligente es un proceso computacional capaz de realizar tareas de forma autónoma y que se comunica con otros Agentes para resolver problemas mediante cooperación, coordinación y negociación. Habitan en un entorno complejo y dinámico con el que interaccionan en tiempo real para conseguir un conjunto de objetivos."

2.1 Propiedades de los Agentes

Las propiedades indispensables para un Agente son:

- **Autonomía:** es la capacidad de operar sin la intervención directa de los humanos, y de tener algún tipo de control sobre las propias acciones y el estado interno.
- **Sociabilidad / Cooperación:** los Agentes han de ser capaces de interactuar con otros Agentes a través de algún tipo de lenguaje de comunicación.
- **Reactividad:** los Agentes perciben su entorno y responden en un tiempo razonable a los cambios detectados.
- **Pro-actividad o iniciativa:** deben ser capaces de mostrar que pueden tomar la iniciativa en ciertos momentos.

Otras propiedades destacables serían:

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

- **Continuidad temporal:** los Agentes están continuamente ejecutando procesos.
- **Veracidad:** un Agente no comunicará información falsa premeditadamente.
- **Benevolencia:** es la propiedad que indica que un Agente no tendrá objetivos conflictivos, y que cada Agente intentará hacer lo que se le pide.
- **Racionalidad:** el Agente ha de actuar para conseguir su objetivo.
- **Aprendizaje:** mejoran su comportamiento con el tiempo.
- **Inteligencia:** usan técnicas de IA para resolver los problemas y conseguir sus objetivos.

2.2 Sistemas Multi Agente

Llamaríamos Sistema Multi Agente, SMA o MAS a aquel en el que un conjunto de Agentes cooperan, coordinan y se comunican para conseguir un objetivo común.

2.2.1 Ventajas de un Sistema Multi Agente

Las principales ventajas de la utilización de un Sistema Multi Agente son:

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

2.2.2 Gestión de un Sistema Multi Agente

La administración de Agentes establece el modelo lógico para la creación, registro, comunicación, movilidad y destrucción de Agentes. En este proyecto vamos a seguir los estándares establecidos por la FIPA.

En la ilustración 1 se puede ver el modelo de administración de Agentes para este entorno.

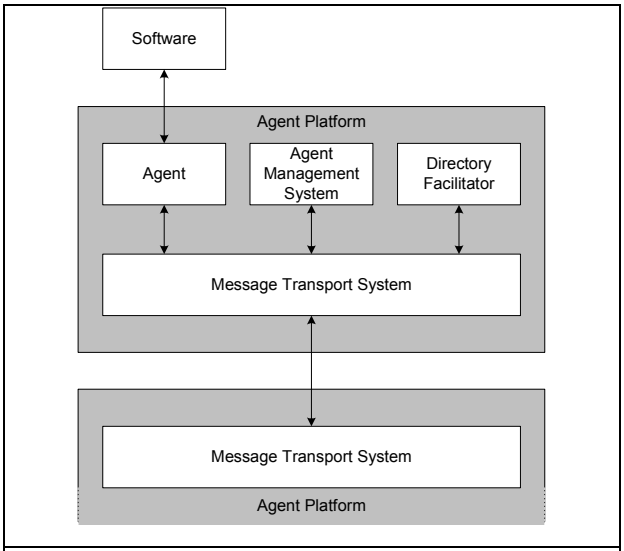


Ilustración 1: Arquitectura FIPA de un SMA

Los componentes básicos son:

- **Agente:** unidad básica. Se podría definir como un programa que contiene y ofrece una serie de servicios.
- **Directory Facilitator (DF):** Agente que proporciona un servicio de *páginas amarillas* dentro del sistema, es decir, conoce los diferentes servicios que ofrecen el resto de Agentes de la plataforma. Los Agentes se han de registrar en el DF para ofrecer sus servicios. Será de especial utilidad a la hora de requerir algún servicio, puesto que nos informará de quien puede proporcionárnoslo.
- **Agent Management System (AMS):** es el Agente que controla el acceso y uso de la plataforma. Almacena las direcciones de los Agentes que se han dado de

alta, ofreciendo un servicio de *páginas blancas*. Dado que JADE se fundamenta en entornos distribuidos, será básico para conocer la localización de los Agentes.

- **Message Transport System (MTS):** facilita la comunicación entre los Agentes de diferentes plataformas.
- **Agent Platform (AP):** proporciona la infraestructura básica para crear y ejecutar Agentes.
- **Software:** cualquier programa accesible desde un Agente.

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

Tal como se define en la FIPA, en cada plataforma se inician dos Agentes que tienen unas funciones vitales para el funcionamiento del sistema y que acabamos de comentar, se trata del *DF* y del *AMS*.

Gracias a estos Agentes la comunicación en la plataforma es muy sencilla de realizar y permite una gran flexibilidad y transparencia. Por ejemplo, si un Agente A quiere enviar un mensaje a un Agente B que desarrolla la tarea T, el Agente A preguntará al DF qué Agentes me pueden proporcionar dicho servicio T. Cuando se quiera conocer la dirección física de un Agente determinado, preguntará al AMS.

Para que este sistema funcione correctamente será necesario que, durante el proceso de inicialización del Agente, éste informe al DF de qué servicios dispone y el tipo de Agente qué es (para poder identificarlo); al hacerlo, AMS le dará una dirección física (dependiendo de dónde se ejecute), para que otros Agentes se puedan poner en contacto con él.

2.2.3 Lenguaje de comunicación entre Agentes (ACL)

Los Agentes individualmente proporcionan una funcionalidad interesante, pero lo que les hace tan adecuados para ciertos sistemas es su capacidad de cooperar para resolver problemas. Para poder hacerlo, los Agentes se han de comunicar entre sí, utilizando un lenguaje común: ACL *(Agent Communication Language)*. Para garantizar la homogeneidad y compatibilidad entre los diversos Agentes, la FIPA determina que

forma ha de tener un mensaje y su utilización; para ello, esta organización elabora las *FIPA Specifications*.

2.2.3.1 Componentes de un mensaje

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

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

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

    - **Accept-proposal:** aceptamos una propuesta hecha anteriormente para realizar una acción.
    - **Agree:** aceptación de una acción.
    - **Cancel:** cancelación de una acción.
    - **Cfp:** para proponer una acción (Call For Proposals).
    - **Confirm:** el emisor informa al receptor que una proposición es cierta, cuando el receptor lo dudaba.
    - **Disconfirm:** el emisor informa al receptor que una proposición es falsa, cuando el receptor pensaba que era cierta.

- **Failure:** indica que la acción pedida anteriormente ha fallado por algún motivo.
- **Inform:** el emisor informa al receptor que una acción se ha realizado correctamente.
- **Not-understood:** el emisor informa al receptor que no ha entendido una petición realizada anteriormente.
- **Propose:** acción de proponer la realización de una acción, dadas ciertas precondiciones.
- **Query-if:** acción de preguntar a otro Agente si un hecho es cierto o no.
- **Refuse:** negación a la realización de una acción dada.
- **Request:** el emisor pide al receptor la realización de una acción.
- **Subscribe:** el emisor pide ser avisado en el momento que se cumpla una condición.

- **Participante de la comunicación:** son los identificadores de los Agentes.

  - **sender** (quien envía el mensaje)
  - **receiver** (quien lo recibe)
  - **reply-to** (a quien tiene que ir destinado el siguiente mensaje de la conversación).

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

- **Descripción del contenido:** permite que el Agente que reciba el mensaje pueda identificar qué se le está enviando y en qué formato.

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

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

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

2.2.3.2 Protocolos de comunicación

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

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

3. JADE

JADE es una herramienta de programación que contiene un conjunto de librerías escritas en JAVA para el desarrollo de Sistemas Multi Agente. Además de proporcionar las funciones de manejo de Agentes a bajo nivel y las interfaces que facilitan la programación, también nos presenta un entorno de ejecución donde los Agentes podrán ejecutarse e interactuar.

Para la realización del proyecto hemos trabajado con la versión 3.1.
Comentaremos brevemente las especificaciones de JADE dejando para el lector la posibilidad de profundizar más en su funcionamiento consultando el *JADE Programmer's guide*, que encontraremos junto a mucha más información en la página oficial de JADE, o leyendo anteriores proyectos presentados por el GRUSMA.

3.1 Paquetes de JADE

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

- ***JADE.core*** es el núcleo del sistema. Proporciona la clase JADE.core.Agent que es imprescindible para la creación de SMA. También contiene el paquete JADE.core.behaviour que incluye las clases de los comportamientos que ha de tener todo Agente.
- ***JADE.lang*** contiene las clases específicas para soportar un lenguaje de comunicación entre Agentes. Esta formado por el paquete java.lang.acl.
- ***JADE.content*** contiene las clases específicas para soportar un lenguaje de comunicación entre Agentes (ACL). Está formado por los paquetes JADE.content.lang.acl y JADE.content.lang.sl. En éste mismo encontramos todas las clases necesarias para la realización e implementación de la ontología (JADE.content.onto).
- ***JADE.domain*** contiene todas las clases para representar la plataforma de Agentes y los modelos del dominio, tales como entidades para la administración de Agentes, lenguajes y ontologías (AMS, DF, ACC…).

- ***JADE.proto*** paquete que contiene los protocolos de interacción entre Agentes FIPA.
- ***JADE.tools*** contiene algunas herramientas que facilitan el desarrollo de aplicaciones y la administración de la plataforma:
    - *Remote Management Agent* (RMA): es el Agente que se encarga de la interfaz gráfica de JADE para la administración y control del sistema.
    - *Dummy Agent*: es una herramienta de monitorización y depuración, compuesta por una interfaz gráfica y un Agente JADE. Permite la creación de mensajes ACL y enviarlos a otros Agentes pudiendo visualizarlos.
    - *Sniffer*: Agente que intercepta mensajes ACL y los visualiza, resulta muy útil para el seguimiento de una conversación.
    - *Introspector Agent*: Agente que permite la monitorización del ciclo de vida de un Agente.
    - *SocketProxyAgent* Agente que actúa como gateway bidireccional entre la plataforma JADE y una conexión TCP/IP.
- ***JADE.gui*** contiene herramientas para construir interfaces gráficas

3.2 Agentes y comportamientos (Behaviours)

Para poder crear Agentes en JADE deberemos extender la clase **JADE.core.Agent** que ya está definida y deberemos rellenar los campos y las funciones básicas que nos proporciona la interfaz. Dos métodos muy importantes que se han de implementar son **setup()** y **takedown()** que inicializan y finalizan el Agente respectivamente. Como inciso de interés comentaremos la existencia de la clase **JADE.gui.GuiAgent**, una extensión de la clase Agente estándar, creada para facilitar la implementación de los Agentes con GUI, que no ha sido utilizada en este proyecto dado que se fundamenta en la librería grafica *swing* y nosotros, como presentación usamos una aplicación para Internet, basada en HTML.

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

3.2.1 Comportamientos SIMPLES

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

• **SimpleBehaviour:** representa un comportamiento atómico que se ejecuta sin interrupciones. Podemos diferenciar entre:

- OneShotBehaviour*:* sólo se ejecuta una vez.
- CyclicBehaviour: el comportamiento se ejecuta cíclicamente.
- TickerBehaviour: periódicamente se ejecuta un trozo de código indicado por el usuario.
- WakerBehaviour: Oneshot task que se ejecuta tras producirse un time-out.

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

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

3.2.2 Comportamientos COMPUESTOS

Son los comportamientos que pueden tener hijos, es decir, podemos asociar uno o más *subbehaviours* a un comportamiento complejo, que serán gestionados por un ***scheduler*** independiente del Agente. Estos comportamientos derivan de la clase *JADE.core.behaviours.ComplexBehaviour*. Los métodos que se pueden sobrecargar son **onStart**() y **onEnd**(). El primero en ejecutarse será el **onStart**(), que será dónde añadiremos los subbehaviours. Cuando finaliza la ejecución de este método se lanzarán todos los subbehaviours, y cuando todos hayan acabado se ejecutará el método **onEnd**(). Cabe destacar que en la antigua versión 2.61 estos métodos eran llamados *preAction()* y *postAction()* respectivamente, apunte que será de interés a aquellos más curtidos en la versión antigua. Hay varios tipos de comportamientos complejos:

- **SequentialBehaviour:** ejecuta los hijos de manera secuencial.

- **ParallelBehaviour:** ejecuta los hijos según una política de Round Robin.
- **FSMBehaviour:** ejecuta los hijos según una máquina de estados finita.

3.3 Lenguaje de Comunicación

El lenguaje FIPA-SL (Semantic Language) es un lenguaje formal con el cual se pretende poder resolver una gran variedad de problemas. En FIPA-SL se pueden formar expresiones lógicas, intercambiar información entre Agentes y expresar acciones a realizar. La notación utilizada es similar al Lisp: se definen un conjunto de predicados genéricos y cada uno de ellos tiene una serie de atributos y parámetros.

3.4 Ontología

JADE también nos permite crear ontologías. Éstas son muy importantes, ya que definen el vocabulario utilizado en nuestro SMA. Es por eso que antes de definir los Agentes, hay que diseñar la ontología que se adapte a nuestras necesidades.

El elemento básico de una ontología es el **frame**. Éste a su vez se divide en **slots**, que pueden ser de diferentes tipos: simples (integer, string, boolean...) u otros frames.

Para la creación de una ontología completa se han de tener 3 elementos en cuenta:

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

3.5 Protocolos de comunicación

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

1. El que inicia la conversación: INITIATOR
2. El que la sigue: RESPONDER

JADE en su condición de plataforma que cumple las especificaciones de la FIPA, nos proporciona los elementos necesarios (*behaviours* y *performatives*) para definir los protocolos descritos por esta organización. Los protocolos entre los que podemos elegir son: FIPA-Request, FIPA-Propose, FIPAQuery, FIPA-Request-When, FIPA-Recruiting, FIPA-Brokering y FIPA-Contract-Net.

3.5.1 FIPA Request

Se utiliza cuando un Agente quiere pedir la realización de una acción a otro Agente. En la ilustración 2 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder*.

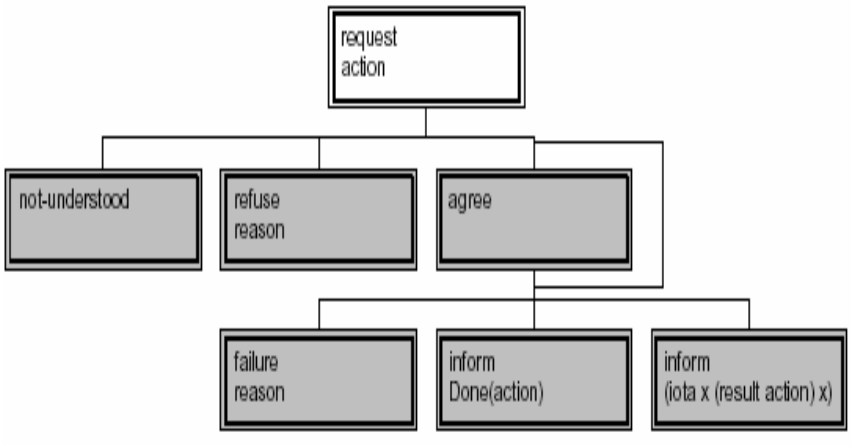


Ilustración 2: Protocolo FIPA Request

3.5.2 FIPA Query

Se utiliza cuando un Agente quiere pedir cualquier tipo de información a otro Agente. En la ilustración 3 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder*.

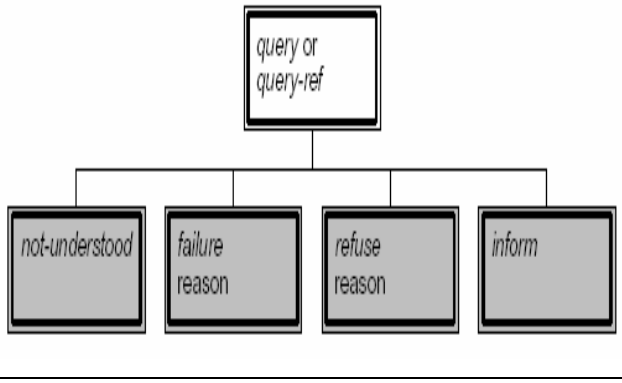


Ilustración 3: Protocolo FIPA Query

3.5.3 FIPA Contract Net

Se utiliza cuando se quiere realizar una negociación entre varios Agentes. En la ilustración 4 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder.*

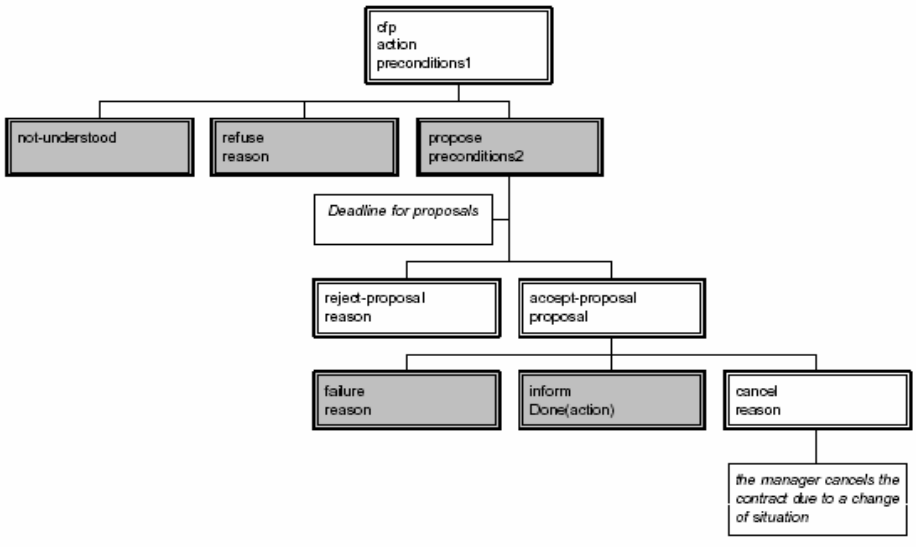


Ilustración 4: Protocolo FIPA Contract Net

3.5.4 Clases proporcionadas por JADE

JADE proporciona diferentes clases para poder manipular los protocolos de comunicación establecidos según la FIPA:

- AchieveREInitiator / AchieveREResponder
- ContractNetInitiator / ContractNetResponder
- SubscriptionInitiator / SubscriptionResponder

3.6 Herramientas de JADE

JADE ofrece una interfaz gráfica para la administración de la plataforma, así como herramientas para facilitar la depuración y fase de test de las aplicaciones.
Para iniciar una sesión y crear los Agentes, deberemos escribir la siguiente línea de comandos:

- *java JADE.Boot [options] –platform [lista de Agentes]*

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

Para la lista de Agentes debemos colocar una cadena de texto donde cada uno de ellos estará separado con espacios en blanco y tendrá el siguiente formato:

- *<NombreAgente>:ClaseAgenteJava*

En el caso que las clases las tengamos dentro de un *package* deberemos colocar la siguiente línea de texto:

- *<NombreAgente>:PackageClase.ClaseAgenteJava*.

En caso que las clases requieran parámetros para su ejecución, éstos serán introducidos usando paréntesis al final de cada clase.

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

De todas formas, existe la posibilidad de iniciar el entorno gráfico de JADE sin ningún Agente y posteriormente con las herramientas que te ofrece ir cargando los Agentes que se deseen y pudiendo visualizar su comportamiento, nosotros no obstante,

por comodidad, los cargaremos todos de inicio, utilizando además un fichero *.bat*, que agilizará el procedimiento.

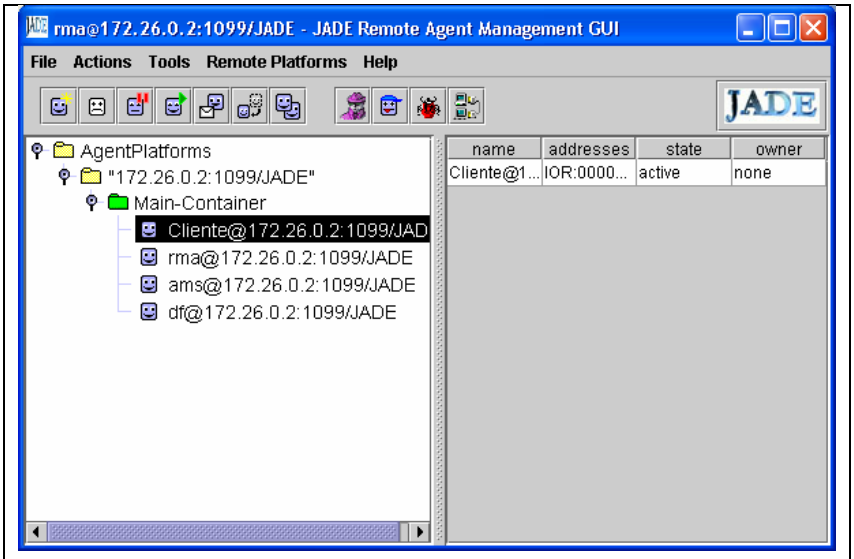


Ilustración 5: Interfaz gráfica de JADE

Cuando iniciamos el entorno gráfico, tal como se muestra en la ilustración 5, aunque no inicialicemos ningún Agente, se crean algunos automáticamente, que son los Agentes básicos de la plataforma, encargados de que todo funcione correctamente. Aunque ya han sido comentados en apartados anteriores, damos un breve repaso.

- *Agente RMA*: será el encargado de controlar la interfaz gráfica de JADE.
- *Agente DF*: donde se subscribirán los servicios de los Agentes. Es muy importante, ya que se utiliza para buscar Agentes con unas características indicadas (páginas amarillas).
- *Agente AMS*: donde se guardan las direcciones de los Agentes de la plataforma.
- *Agente Sniffer:* es una aplicación JAVA creada para rastrear el intercambio de mensajes entre los Agentes JADE. Cuando lo activas se abre una ventana y sólo podrás observar que existe una caja llamada **others**. Para que el Agente funcione correctamente y te muestre el intercambio de mensajes que se puede mostrar en la ilustración 6 debes indicarle qué Agentes quieres visualizar. Esta herramienta es muy útil para la fase de test del correcto funcionamiento del paso de mensajes. Si se hace doble click en las flechas se abre una ventana, donde tendrás toda la información necesaria sobre ese mensaje en concreto. Remarcar

que otra posibilidad que nos ofrece es guardar los mensajes y posteriormente poderlos abrir.

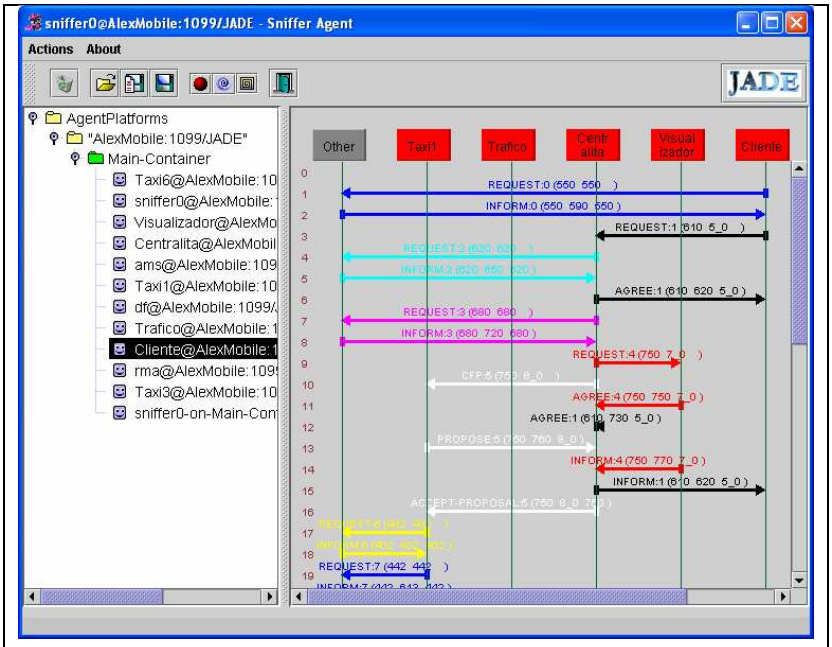


Ilustración 6: Agente Sniffer

- *Agente Dummy (Dummy-Agent):* permite enviar mensajes ACL entre los Agentes, recibirlos, inspeccionarlos, leer y salvarlos en un fichero (ilustración 7).

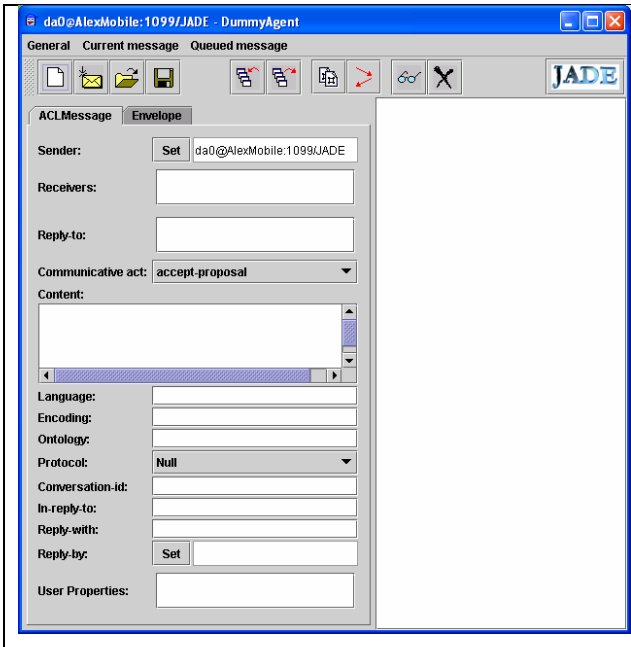


Ilustración 7: Dummy Agent

- *Agente DF:* Otra herramienta muy útil para la realización de este tipo de sistemas, es la activación de la interfaz del Agente DF (menú *tools*), donde podemos observar los Agentes que se han registrado y qué servicio ofrecen, tal como podemos observar en la ilustración 8.

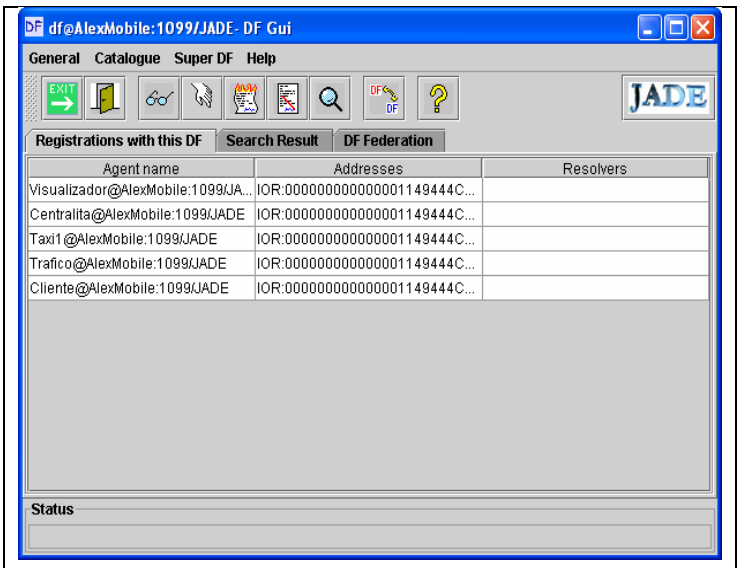


Ilustración 8: Agente DF

- *Agente Introspector (Introspector Agent):* nos permite monitorizar el ciclo de vida del Agente

4. Otras arquitecturas y tecnologías

En este apartado se comentan las distintas arquitecturas y tecnologías que se han usado para dar a luz la aplicación sobre la que estamos tratando, a parte de la ya comentada JADE.

El conjunto y la unión natural de conceptos y filosofías de cada una de ellas, es la garantía de que puedan funcionar en equipo para satisfacer los distintos objetivos planeados para este proyecto.

Nota: Parte de esta información ha sido obtenida de http://es.wikipedia.org

4.1 Servidor Web Apache Tomcat

Tomcat (también llamado **Jakarta Tomcat** o **Apache Tomcat**) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los Servlets y de JavaServer Pages (JSP) de Sun Microsystems. Se le considera un servidor de aplicaciones. Para nosotros era importante el aspecto de trato de JSP por parte del servidor, pues hemos decidido usar esta alternativa respecto los Servlets, debido al gran contenido de código HTML contenido en los módulos de presentación e interacción con el usuario final (Aplicación Web).

4.1.1 Historia

Tomcat empezó siendo una implementación de la especificación de los servlets comenzada por James Duncan Davidson, que trabajaba como arquitecto de software en Sun y que posteriormente ayudó a hacer el proyecto *open source* y en su donación a la Apache Software Foundation.

Duncan Davidson inicialmente esperaba que el proyecto se convirtiese en *open source* y dado que la mayoría de los proyectos open source tienen libros de O'Reilly asociados con un animal en la portada, quiso ponerle al proyecto nombre de animal. Eligió *Tomcat*

(gato), pretendiendo representar la capacidad de cuidarse por sí mismo, de ser independiente.

4.1.2 Entorno

Tomcat funciona con cualquier servidor Web con soporte para Servlets y JSPs. Tomcat incluye el compilador Jasper, que compila JSPs convirtiéndolas en Servlets. El motor de Servlets del Tomcat a menudo se presenta en combinación con el servidor Web Apache.

También puede funcionar como servidor Web por sí mismo. Opera de tal manera en entornos de desarrollo poco exigentes en términos de velocidad y de manejo de transacciones. Esta es la opción que nosotros hemos escogido. Pudiendo migrar a un servidor Apache, de reconocido prestigio y rendimiento, de manera natural, incorporando Tomcat como módulo del mismo.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual.

4.1.3 Estado de su desarrollo

Desarrollado y lo mantenido por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software Licence*. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 5.x, que implementan las especificaciones de Servlet 2.4 y de JSP 2.0. Nosotros usamos la versión 4.1, pues al inicio del desarrollo del proyecto, las 5.x no estaban lo suficientemente maduras y eran objetos de muchas críticas debido a bugs, fallas y vulnerabilidades que, con el paso del tiempo, parece que definitivamente se han corregido.

4.1.4 Características del producto

Tomcat 3.x (distribución inicial)

- Implementado a partir de las especificaciones Servlet 2.2 y JSP 1.1
- recarga de Servlets

Tomcat 4.x

- Implementado a partir de las especificaciones Servlet 2.3 y JSP 1.2
- contenedor de servlets rediseñado como Catalina
- motor JSP rediseñado con Jasper
- conector Coyote
- Java Management Extensions (JMX), JSP Y administración basada en Struts

Nota: Nosotros usamos la versión 4.1 concretamente.

Tomcat 5.x

- implementado a partir de las especificaciones Servlet 2.4 y JSP 2.0
- recolección de basura reducida
- capa envolvente nativa para Windows y Unix para la integración de las plataformas

4.1.5 Estructura de directorios

La jerarquía de directorios de instalación de Tomcat incluye:

- bin - arranque, cierre, y otros scripts y ejecutables
- common - clases comunes que pueden utilizar Catalina y las aplicaciones Web
  Dentro de este directorio se encuentra un directorio 'lib' donde incluiremos nuestros modulos .JAR necesarios para la aplicación. Lo veremos más adelante en la guía del Administrador.
- conf - ficheros XML y los correspondientes DTDs para la configuración de Tomcat.

Tendremos que realizar pequeñas modificaciones en archivos de este directorio para habilitar nuestra aplicación. También lo comentaremos posteriormente.

- logs - logs de Catalina y de las aplicaciones
- server - clases utilizadas solamente por Catalina
- shared - clases compartidas por todas las aplicaciones Web
- webapps - directorio que contiene las aplicaciones Web.
  Dentro de esta carpeta incluiremos nuestro directorio de la aplicación, donde estará contenida parte del módulo de presentación e interacción con el usuario.
- work - almacenamiento temporal de ficheros y directorios

4.2 JSP y Servlets

Como hemos comentado, Tomcat convierte automáticamente JSP en Servlets. Por eso explicamos ambos para entender qué tipo de tecnología tenemos entre manos.

Java Server Pages (**JSP**) es la tecnología para generar páginas Web de forma dinámica en el servidor, desarrollado por Sun Microsystems, basado en scripts que utilizan una variante del lenguaje java.

La tecnología JSP, o de JavaServer Pages, es una tecnología Java que permite a los programadores generar dinámicamente HTML, XML o algún otro tipo de página Web. Esta tecnología permite al código Java y a algunas acciones predefinidas ser empotradas en el contenido estático. En las jsp, se escribe el texto que va a ser devuelto en la salida (normalmente código HTML como en nuestro caso) empotrando código java dentro de él para poder modificar o generar contenido dinámicamente. El código java se incluye dentro de las marcas de etiqueta <% y %>.

Los **Servlets** son componentes del servidor. Estos componentes pueden ser ejecutados en cualquier plataforma o en cualquier servidor debido a la tecnología Java que se usa para implementarlos. Los Servlets incrementan la funcionalidad de una aplicación Web. Se cargan de forma dinámica por el entorno de ejecución Java del servidor cuando se necesitan. Cuando se recibe una petición del cliente, el contenedor / servidor Web inicia el Servlet requerido. El Servlet procesa la petición del cliente y envía la respuesta de vuelta al contenedor / servidor, que es enrutada al cliente.

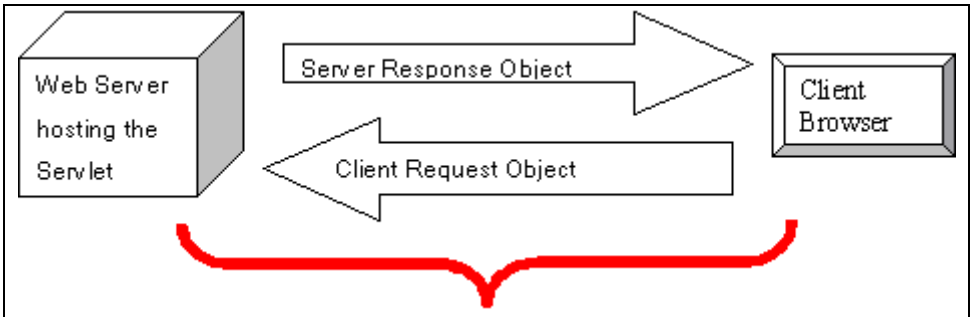


Ilustración 9: Modelo de respuesta a peticiones http

La interacción cliente / servidor basada en Web usa el protocolo HTTP. EL protocolo HTTP es un protocolo sin estados (por lo que se hace necesario el uso de variables de sesión, explicadas más adelante) basado en un modelo de petición y respuesta con un número pequeño y finito de métodos de petición como GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT, etc. La respuesta contiene el estado de la respuesta y meta-información describiendo dicha respuesta. La mayor parte de las aplicaciones Web basadas en Servlets se construyen en el marco de trabajo del modelo petición / respuesta HTTP (ilustración 9).

4.3 Variables de sesión de Internet

Como hemos comentado anteriormente, el protocolo principal sobre el que corre Internet (http) es un protocolo sin estado; es decir, el servidor no tiene información alguna sobre el cliente.

Una manera de solucionar este pequeño 'handicap' es mediante las llamadas sesiones, o cookies, que es información que se guarda en el lado del cliente y es consultada por el servidor para comprobar el estado del primero.

En toda aplicación son necesarias ciertas variables accesibles desde lugares muy distantes y que mantengan su valor durante todo el momento de la sesión.

El problema de las cookies, es que no son aceptadas por todos los navegadores; o pueden desactivar la opción de aceptarlas.

Los objetos Session se guardan en memoria (no en ficheros de texto como las cookies) y pueden ser usados en cualquier momento.

La duración de una sesión viene definida en unos cuantos minutos; tras el paso de los cuales, si no se hace alguna acción sobre ella, ésta caduca. La sesión también se puede destruir de manera explícita (como sucede en nuestro caso cuando el usuario cierra sesión) o bien cerrando el navegador directamente.

En la aplicación que aquí estamos comentando, el uso de sesiones se hace patente para guardar información sobre la validación del actor 'Usuario', sin la cual, el sistema no efectuará consultas ni actualizaciones alguna.

4.4 Puente ODBC-JDBC: Conexión a base de datos

JDBC es la API estándar de acceso a Bases de Datos con Java, y se incluye con el Kit de Desarrollo de Java (JDK) a partir de la versión 1.1. Sun optó por crear una nueva API, en lugar de utilizar APIs ya existentes, como ODBC, con la intención de obviar los problemas que presenta el uso desde Java de estas APIs, que suelen ser de muy bajo nivel y utilizar características no soportadas directamente por Java, como punteros, etc. Aunque el nivel de abstracción al que trabaja JDBC es alto en comparación, por ejemplo, con ODBC, la intención de Sun es que sea la base de partida para crear librerías de más alto nivel, en las que incluso el hecho de que se use SQL para acceder a la información sea invisible.

Para trabajar con JDBC es necesario tener controladores (drivers) que permitan acceder a las distintas Bases de Datos: cada vez hay más controladores nativos JDBC. Sin embargo, ODBC es hoy en día la API más popular para acceso a Bases de Datos: Sun admite este hecho, por lo que, en colaboración con Intersolv (uno de principales proveedores de drivers ODBC) ha diseñado un puente que permite utilizar la API de JDBC en combinación con controladores ODBC. Nosotros utilizaremos este puente JDBC/ODBC para acceder a nuestras Bases de Datos creadas con MySQL (SGBD para el que también existe un driver JDBC nativo, InterClient).

Para todo ello es necesario habilitar un DataSource en Windows XP y dar de alta nuestros servicios MySql en nuestro caso.

4.5 MySql

MySQL es uno de los Sistemas Gestores de Bases de Datos más populares desarrolladas bajo la filosofía de código abierto.

La desarrolla y mantiene la empresa MySql AB pero puede utilizarse gratuitamente y su código fuente está disponible.

Inicialmente, MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas Web con contenido dinámico, justamente por su simplicidad; aquellos elementos faltantes fueron llenados por la vía de las aplicaciones que la utilizan.

Poco a poco los elementos faltantes en MySQL están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

Según las cifras del fabricante, existirían más de seis millones de copias de MySQL funcionando en la actualidad, lo que supera la base instalada de cualquier otra herramienta de bases de datos.

4.6 CORBA (e invocación remota a métodos)

Siguiendo nuestro objetivo de usar los recursos necesarios con tal de dar a luz una aplicación totalmente distribuida, se han implementado llamadas remotas a métodos. En este apartado hacemos una breve introducción a CORBA, del cual nos hemos servido.

4.6.1 Definición

Common Object Request Broker Architecture (**CORBA**) es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

CORBA fue definido y está controlado por el *Object Management Group* (OMG) que define la arquitectura OMA (Object Management Arquitecture), las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida para la invocación de métodos sobre objetos en entornos heterogéneos.

Los entornos heterogéneos son aquellos en los que las arquitecturas que constituyen el entorno pueden ser sistemas Microsoft Windows, máquinas Unix de diferentes fabricantes (Linux entre otros) e inclusos sistemas como MacOS o OS/2. Y es más, dentro de la heterogeneidad también se incluyen los sistemas de comunicaciones (protocolos de comunicación como TCP/IP, IPX ...) o los lenguajes de programación utilizados en las diferentes arquitecturas. Es el mundo informático en su complejidad más alta y por tanto sólo una arquitectura muy flexible y potente puede cubrir todos estos aspectos.

En un sentido general CORBA "envuelve" el código escrito en otro lenguaje en un paquete que contiene información adicional sobre las capacidades del código que contiene, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) desde la red. En este sentido

CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras pero con más información.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar los interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, **Java** y Python. Hay también implementaciones para Perl y TCL. Posteriormente veremos nuestro modelo IDL creado para esta aplicación.

Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el implementador del objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como *stub*, el cual incluye un *proxy* (representante) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos *skeletons* (esqueletos) que el desarrollador tiene que rellenar para implementar los métodos del objeto.

De esta forma se logra una abstracción a la heterogeneidad que permite que el uso de CORBA no sea nada complejo. Como veremos cuando expliquemos su uso en nuestro proyecto, la forma de desarrollar con CORBA sigue una metodología concreta y fácil de seguir.

4.6.2 ¿Por qué CORBA?

Con el triunfo de las redes de ordenadores y su implantación, las aplicaciones distribuidas son las únicas que pueden explotar todo el potencial de este nuevo modelo informático.

Por otro lado, CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad, transacciones, gestión de las comunicaciones, informando a clientes y servidores de caídas de los canales de comunicación.

Las aplicaciones distribuidas se caracterizan por su ejecución coordinada en diferentes máquinas comunicadas, así como también por una alta complejidad en todas las etapas de su desarrollo, debido a la gran cantidad de factores que influyen en su ejecución, siendo uno de los principales la gestión de las comunicaciones.

Por ejemplo, hay que tener en cuenta que los enlaces entre las máquinas se pueden caer, y prever que hacer en esos casos, y como se detectan dichas caídas. Otro ejemplo sería que contra una máquina *A* se pueden conectar varias más, y debe ser capaz esta máquina *A* de manejar todas las conexiones. Las máquinas se tienen que poder encontrar y tienen que poder localizar servicios remotos. Podemos citar también que los entornos en red suelen ser heterogéneos, hay que comunicar diferentes máquinas con distintas configuraciones software y hardware.

CORBA abstrae de muchos de estos detalles y hace la distribución de la aplicación un proceso mucho menos complejo y costoso.

Se encarga de organizar los servicios que se pueden encontrar en la red a través de los interfaces IDL. Como hemos comentado, es independiente de la plataforma y del lenguaje de desarrollo lo que facilita el desarrollo en entornos heterogéneos. Gestiona las comunicaciones, informando a clientes y servidores de caídas de los canales de comunicación.

5.6.3 Arquitectura CORBA

Comentemos por encima la arquitectura CORBA, por lo menos para entender los elementos que usamos nosotros. Empecemos echando un vistazo a la arquitectura, representada en niveles, que podemos ver en la ilustración 10.

En este caso, no pretendemos hacer un estudio exhaustivo de esta arquitectura, de hecho simplemente nos interesa saber para qué sirve su corazón, es decir, el servicio ORB, pues también lo usamos.

ORB es posiblemente la parte más importante de CORBA. Es lo que se conoce como el bus de los objetos. El ORB se encarga de poner en contacto a los clientes y a los objetos de forma transparente con respecto a la distribución.

Sus responsabilidades principales son:

- Suministrar mecanismos para que el cliente encuentre la Implementación de la Interfaz
- Proporcionar la preparación de la Implementación de la Interfaz para que pueda recibir invocaciones remotas

- Gestionar la comunicación de los datos que hacen posible la petición (argumentos de la función y parámetros de retorno)

Cuando el cliente quiere hacer una petición a la Implementación de la Interfaz puede utilizar dos caminos diferentes: utilizar un proxy OMG IDL, o bien utilizar la interfaz de invocación dinámica (DII). Pero esto ya se escapa de la finalidad de esta documentación.

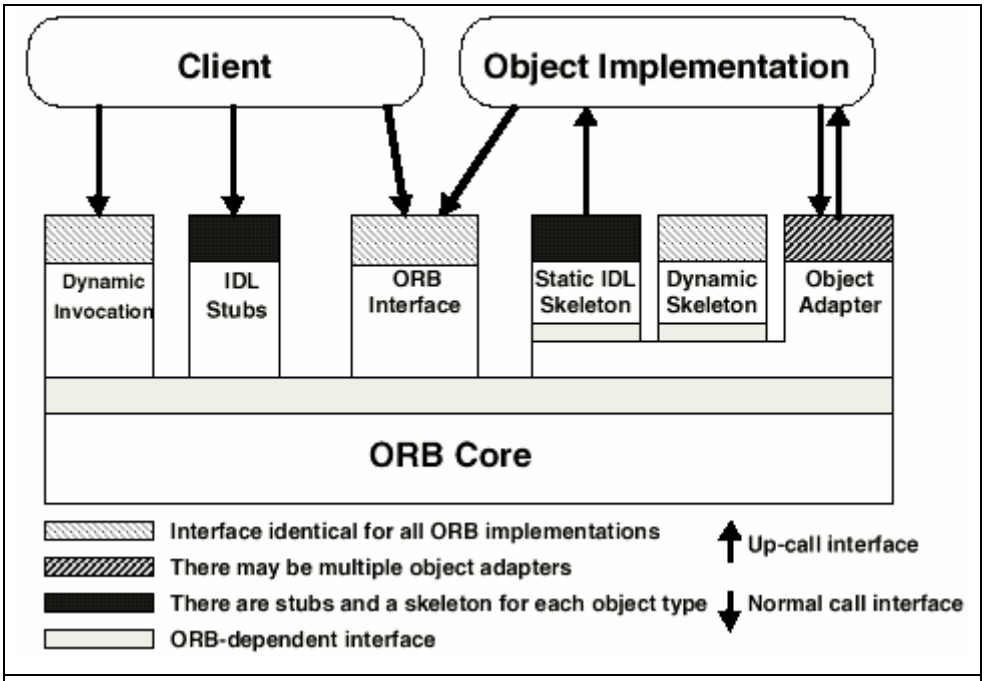


Ilustración 10: Arquitectura CORBA

4.7 Microsoft Internet Explorer (IE)

Tras abatir a lo que fue anteriormente, su principal competidor (Netscape), IE se ha convertido en el navegador más extendido y que cumple con los estándares actuales en cada momento. Netscape reaccionó liberando su código fuente y dando pié a un navegador libre, convirtiéndose en Mozilla y Firefox, con los que está recuperando una parte del terreno perdido.

Un navegador es una aplicación cuya finalidad es la de explorar a través de Internet, visitando todo tipo de páginas Web; cuyo contenido se ha enriquecido notablemente con elementos multimedia, applets Java, formularios... Todo ello soportado por IE (versión 5 o superior).

Es una herramienta vital para nuestra aplicación pues es la única manera que tiene el usuario de interactuar con el aplicativo.

4.8 JavaScript

Las páginas Web en general hacen uso de javascript para controles en el lado del cliente, que aceleran la gestión de errores provocados por el mal uso por parte del usuario, además de para incluir nuevas funcionalidades en forma de controles que van más allá de los habituales en formularios HTML.

En nuestro caso, encontramos JavaScript en los controles de calendario y en los distintos formularios de inserción de datos donde se comprueba que se hayan rellenado todos los campos necesarios, y que estos sean correctos. Por ejemplo, se introduce dos veces el password y controlamos que sea el mismo, para descartar equivocaciones del usuario, o se comprueban las fechas cuando hacemos viaje de Ida y Vuelta, para que la de Vuelta sea posterior a la de Ida, etc. En caso de no cumplirse alguna de estas condiciones, el sistema informa mediante una ventana emergente de alerta, del error, e invita al usuario a resolverlo antes de volver a intentar enviar de nuevo el formulario.

4.9 Mecanismos de Seguridad del sistema

El primer mecanismo de seguridad es la validación del actor Usuario, mediante login y password; los cuales están administrados en el servidor y situados en una base de datos interna.

El siguiente mecanismo es la existencia de una sesión en Internet, que asegura que el actor que está interactuando con el sistema sea realmente el que se validó en su momento.

El punto fundamental es la fragilidad, en cuanto seguridad, ofrece el protocolo http. El servidor Web usado, Tomcat, nos ofrece la posibilidad de configurarlo para trabajar con el protocolo shttp (secure http), el cual nos garantiza una conexión segura y sin riesgos.

De esta manera guardamos la confidencialidad de las consultas de los usuarios, así como también sus preferencias y datos personales, a la vez que dejamos preparado el

sistema para futuras ampliaciones en cuanto a posibilidades de transacciones económicas, compras vía Internet a través de los Agentes, y un largo etcétera.

5. Diseño SMA

En este apartado realizamos un estudio en profundidad sobre los diferentes aspectos que se han tenido en cuenta en el momento de realizar el diseño de nuestra aplicación.

5.1 Justificaciones

Una vez evaluados los requisitos funcionales de la aplicación, hemos tomado las decisiones oportunas para llevar a cabo un diseño coherente con la filosofía que impulsa la aplicación, dando relevante importancia a factores como:

- **Facilidad de uso:** Se ha dotado a la aplicación de una interfaz Web, con la que la mayoría de usuarios están habituados a interactuar. Esto, junto a la ordenación lógica de las distintas opciones y los enlaces entre ellas que guían al usuario, conlleva a un uso intuitivo y rápido de las funcionalidades que ofrece la aplicación.
- **Robustez / Distribución:** Bajo este concepto se han elegido las distintas tecnologías que entran en juego en el desarrollo de la aplicación. Integradas de manera coherente y natural, proporcionando un diseño distribuido, donde el aplicativo se puede dividir en varias máquinas, lo que proporciona que, en caso de caída de alguna de ellas, la funcionalidad no se vea resentida siempre y cuando se usen técnicas de replicación.
- **Eficiencia:** Ligado estrechamente con la distribución, surge la importancia de la eficiencia, que queda totalmente garantizada gracias a la posibilidad de usar paralelismo en consultas, lo que disminuye el tiempo global de operación.

5.2 *Descripción de los casos de uso*

Pasamos a detallar las principales funcionalidades que presenta la aplicación. Primero mediante diagramas de casos de uso y después mediante la descripción textual de los mismos.

Estructura de la descripción textual:

Para cada funcionalidad, presentada como un caso de uso, se especifican varias partes: Resumen de la funcionalidad, Papel dentro del trabajo del usuario, Actores, Casos de uso relacionados (donde aparecen sólo los más importantes, y que merezcan ser estudiados a parte), Precondición, Poscondición y Proceso normal principal.

Convenciones:

- Nombre de actores en **negrita**
- Terminología del usuario en *cursiva*
- Referencias a otros casos de uso subrayadas

5.2.1 Diagramas de casos de uso

En la ilustración 11, vemos las diferentes opciones que tiene el usuario para interactuar con el sistema.

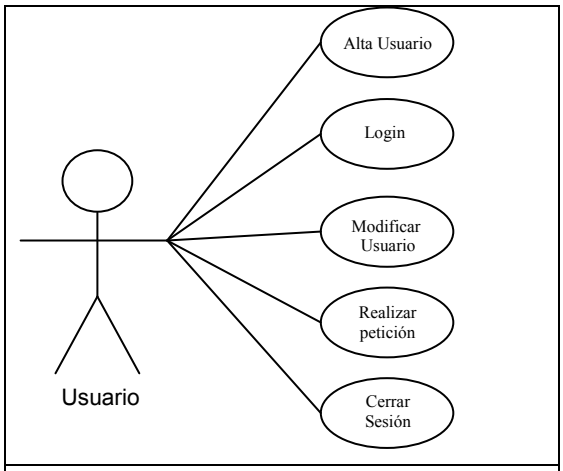


Ilustración 11: Casos de uso del usuario

5.2.2 Descripción textual de casos de uso

5.2.2.1 Alta Usuario

Resumen:

Permite introducir en la base de datos información sobre un nuevo *usuario*.

Papel:

Eventual

Actores:

Usuario

Casos de uso relacionados:

- Login
- modif. / baja usuario
- Realizar petición

Precondición:

El *usuario* no ha de estar previamente introducido en la base de datos.

Poscondición:

Información sobre nuevo usuario queda almacenada en la base de datos.

Proceso normal principal:

Se introducen los datos correspondientes a los siguientes campos:

- Usuario
- Password
- Confirmar Password
- Nombre
- Apellido
- E-mail
- Sexo
- Fecha naciemiento

- Lista transportes preferidos
- Lista transportes prohibidos
- Número escalas
- Días de margen
- Cuestionario 'Deseas mantenerte informado de las últimas ofertas de viajes a través de tu correo electrónico?'
- Cuestionario 'Permites usar tus datos para propósitos de mejora de los productos ofrecidos?'

Gracias a este caso de uso, el usuario quedará registrado en la base de datos, y tendrá privilegios para usar la aplicación siempre que lo desee. También permite al sistema conocer algunos aspectos del perfil del usuario para adecuar las búsquedas a las necesidades del mismo.

5.2.2.2 Login

Resumen:

Permite autenticar al *usuario* registrado para poder interactuar con la aplicación.

Papel:

Habitual

Actores:

Usuario

Casos de uso relacionados:

- Alta Usuario
- modif. / baja usuario
- Realizar petición

Precondición:

El usuario ha de estar previamente introducido en la base de datos (mediante caso de uso Alta Usuario)

Poscondición:

Queda registrado el login del usuario y permite entrar en las zonas de acceso restringido.

Proceso normal principal:

Se introducen los datos correspondientes a los siguientes campos:

- Usuario
- Password

5.2.2.3 Modificar Usuario

Resumen:

Permite modificar en la base de datos información sobre un *usuario*.

Papel:

Eventual

Actores:

Usuario

Casos de uso relacionados:

- Alta usuario
- Login

Precondición:

El usuario ha de estar previamente introducido en la base de datos y correctamente autenticado.

Poscondición:

Información sobre *usuario* queda modificada en la base de datos.

Proceso normal principal:

Se pueden modificar los datos correspondientes a los siguientes campos:

- Nombre de Usuario
- Antiguo Password
- Password
- Confirmar Password
- Nombre

- Apellido
- E-mail
- Sexo
- Fecha nacimiento
- Lista transportes preferidos
- Lista transportes prohibidos
- Número escalas
- Días de margen
- Cuestionario 'Deseas mantenerte informado de las últimas ofertas de viajes a través de tu correo electrónico?'
- Cuestionario 'Permites usar tus datos para propósitos de mejora de los productos ofrecidos?'

5.2.2.4 Realizar petición

Resumen:

Permite efectuar una búsqueda de planificación entre dos localidades. Creando una ruta que nos indica cómo llegar desde un punto a otro.

Papel:

Habitual

Actores:

Usuario

Casos de uso relacionados:

- Alta usuario
- Login

Precondición:

El usuario ha de estar previamente introducido en la base de datos y correctamente autenticado.

Poscondición:

El sistema presenta los resultados de la búsqueda.

Proceso normal principal:

Se introducen los datos correspondientes a los siguientes campos:

- Ciudad Origen
- País Origen
- Ciudad Destino
- País Destino
- Día ida
- Margen días ida
- Día vuelta
- Margen días vuelta
- Lista transportes preferidos
- Lista transportes prohibidos
- Número escalas
- Importe máximo

Algunos de estos datos, al acceder al formulario saldrán con valores por defecto tomados de las preferencias generales del usuario, pero pueden ser libremente modificados.

5.2.2.5 Cerrar Sesión

Resumen:

Permite desconectar al *usuario* para no poder interactuar con la aplicación.

Papel:

Habitual

Actores:

Usuario

Casos de uso relacionados:

- Alta Usuario
- Login

Precondición:

El usuario ha de estar previamente introducido en la base de datos (mediante caso de uso Alta Usuario) y autenticado.

Poscondición:

Queda inutilizada la sesión; por tanto, se cierra el acceso a las funcionalidades de la aplicación. Para volver a usarlas, deberá volver a realizar Login.

Proceso normal principal:

Se presiona sobre la opción del menú simplemente.

5.3 Diseño del proceso de planificación

En este punto se detalla todo el proceso de planificación, comentando ampliamente los aspectos más representativos en cada una de las fases por las que pasa.

5.3.1 Visión global del proceso

Como vemos en la ilustración 12, el primer paso para realizar la planificación consta en el acceso al sistema a través de nuestra aplicación Web. Como se ve en el diagrama, este acceso se podría exportar al protocolo Wap, para poder tener acceso mediante dispositivos móviles. Para ello sería necesario un servidor Wap, el cual no hemos incluido en nuestra versión. Quedando, de esta manera, como una línea futura de investigación.

Una vez autenticados podremos acceder al formulario destinado a recoger los datos necesarios para emprender el camino hacia la resolución del problema planteado por el usuario. En el diagrama se muestran otros formularios de interacción con el mismo y con una Base de Datos, los cuales explicaremos detalladamente más adelante, junto al diseño completo que también veremos en profundidad en capítulos posteriores. En este diagrama vemos qué operación realizará cada formulario sobre la Base de Datos, algunos insertando o modificando información y otros simplemente consultándola.

Es en este punto cuando la aplicación Web (a través de un JSP) crea dinámicamente un nuevo Agente y lo registra en la plataforma SMA para empezar a interactuar con el resto de Agentes que puedan proporcionar la ayuda necesaria para conseguir satisfacer las necesidades del usuario. Este es el denominado *ClienteAgente*.

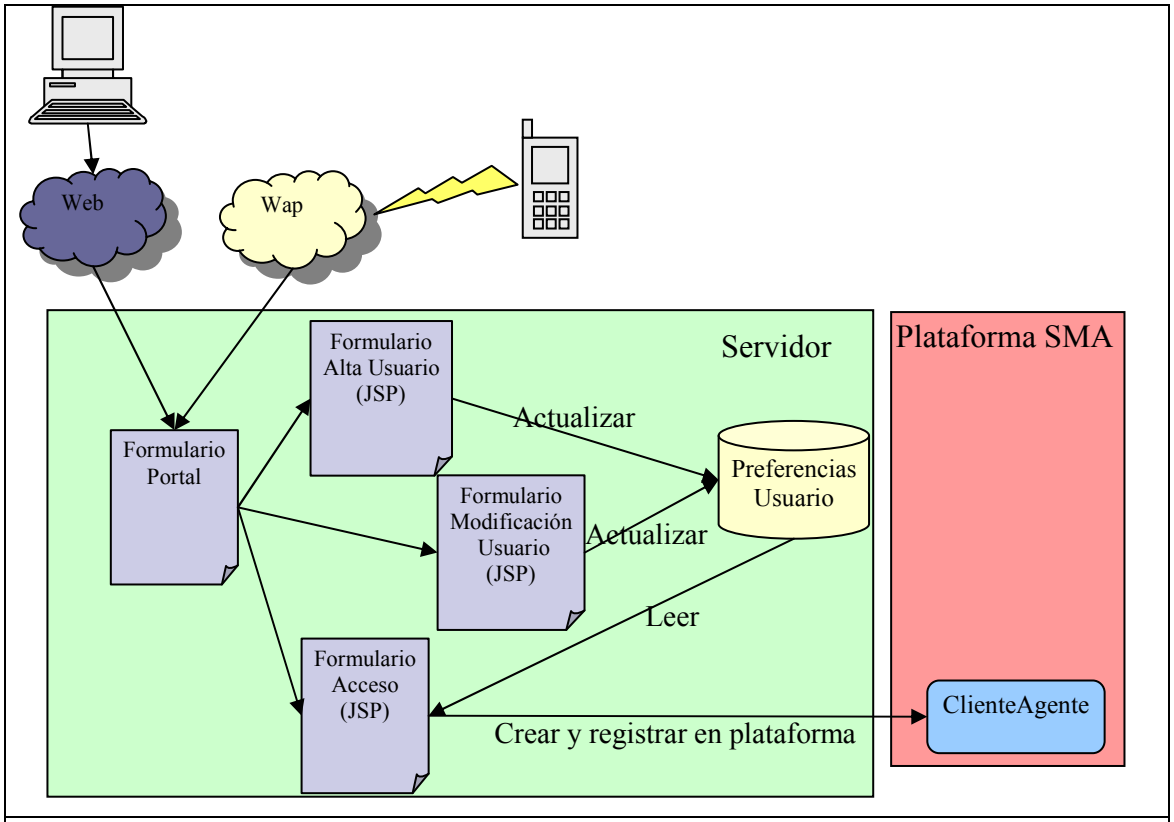


Ilustración 12: Acceso al sistema

Una vez nuestro *ClienteAgente* logra introducirse en la plataforma virtual (SMA) empieza una serie de tareas para, de algún modo, delegar sus obligaciones en otro Agente especializado. Veamos previamente la ilustración 13 para hacernos una idea del proceso, antes de detallarlo textualmente.

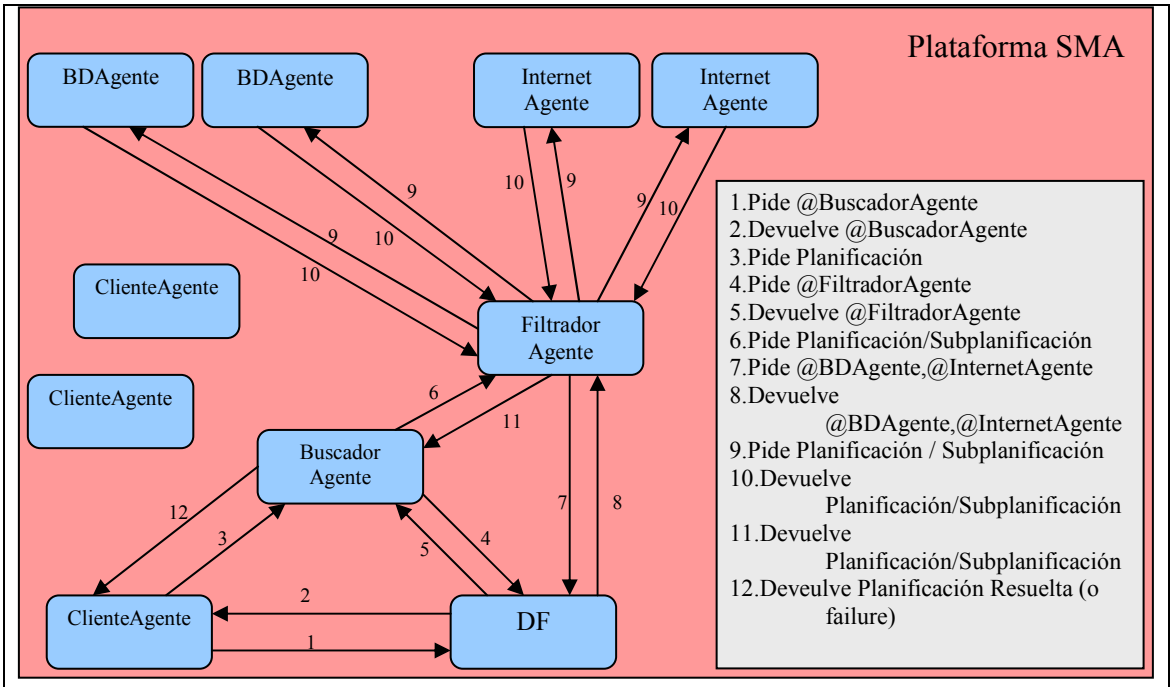


Ilustración 13: Esquema de interacción entre Agentes

Comentaremos la interacción, obviando las fases de registro y de baja de los Agentes en la plataforma SMA.

El primer paso para el *ClienteAgente* es localizar (gracias al DF) Agentes de tipo *BuscadorAgente*. De entre todos los encontrados, elegirá arbitrariamente uno de ellos con el que interactuará. *BuscadorAgente* aceptará la petición siempre y cuando esté ocioso, es decir, no ocupado con otra petición en curso. Esta misma filosofía se aplicará en todas y cada una de las peticiones en el resto de Agentes.

Una vez se haya encontrado un *BuscadorAgente* dispuesto a ayudarnos, se delegará la búsqueda en él. Pasándole los parámetros de la búsqueda a través de un mensaje, usando la Ontología adecuada, creada especialmente para nuestra aplicación y que veremos comentada en apartados sucesivos.

Cuando *BuscadorAgente* obtenga la petición, se dedicará a buscar otro tipo de Agente que le ayudará. Se trata de *FiltradorAgente*.

La delegación en este punto no es algo trivial ni inmediata, pues cada uno de los Agentes hasta ahora comentados tiene un rol específico que interviene de manera complementaria en la resolución del problema. A saber:

- **ClienteAgente:**

    - Creado dinámicamente a través de JSP
    - Recibe parámetros de búsqueda especificados por cliente
    - Se registra en la plataforma SMA
    - Busca algún *BuscadorAgente* ocioso
    - Delega la búsqueda en el mismo
    - Espera respuesta
    - Devuelve respuesta a JSP
    - Se desregistra de la plataforma SMA y muere

- **BuscadorAgente:**

    - Creado y registrado en plataforma SMA desde consola en el arranque de la aplicación, aunque pueden crearse en cualquier momento posterior.
    - Atiende peticiones de *ClienteAgente*
    - Busca en el DF algún *FiltradorAgente*
    - Lanza petición de subplanificación a *FiltradorAgente*
    - Espera propuestas de planificación
    - Mientras propuestas recibidas sean insuficientes, lanza petición a *FiltradorAgente* de subplanificación, atendiendo a criterios de pertenencia (Algoritmo de Generalización – Especificación) de los pueblos, ciudades…
    - En el supuesto que el viaje sea de ida y vuelta, repetirá la búsqueda para el trayecto de vuelta, siempre y cuando el de ida haya sido satisfactorio
    - Si se consigue componer una propuesta que satisfaga la petición del cliente, devuelve a *ClienteAgente* propuesta.
    - Si en las fechas seleccionadas no consigue resultado válido, jugará con los días de margen.
    - También tendrá en cuenta el importe máximo introducido por el usuario.
    - En caso contrario, devuelve ‘failure’
    - Queda de nuevo libre para atender nuevas peticiones

- **FiltradorAgente:**

    - Creado y registrado en plataforma SMA desde consola en el arranque de la aplicación, aunque pueden crearse en cualquier momento posterior.
    - Atiende peticiones de *BuscadorAgente*
    - Busca en el DF todos los:
        - *InternetAgente*
        - *BDAgente*
    - Selecciona *BD* e *InternetAgente's* según los transportes prohibidos del usuario
    - Pide subplanificaciones a *InternetAgente's BDAgente's*
    - Recibe propuesta de ellos
    - Filtra respuestas eliminando las que considere inadecuadas.
    - Ordena los resultados obtenidos según los transportes preferidos del usuario y otros parámetros de eficiencia y devuelve sólo el mejor de todos ellos.
    - Comunica respuesta o failure a *BuscadorAgente*
    - Cuando acaba, queda listo para nuevas peticiones

Se han comentado dos nuevos tipos de Agentes con los que se interactúa al final de la cadena. Se trata nada más y nada menos de los encargados de realizar las búsquedas concretas de itinerarios. Estos dos Agentes pueden interactuar, según su naturaleza, directamente con Bases de Datos estáticas, o bien con la mayor Base de Datos existente, nos referimos a Internet.

Nuestro proyecto da mayor hincapié a este segundo tipo de interacciones, siendo unos de los objetivos del mismo el implementar Agentes capaces de efectuar búsquedas inteligentes a través de la Red de Redes. Posteriormente se dedica un apartado exclusivamente a tratar este tema.

Cuando uno de estos dos tipos de Agentes recibe una petición de algún *FiltradorAgente*, aprovecha sus capacidades para realizar la búsqueda y devolver el resultado que se espera.

Veamos las funciones principales de los mismos:

- **BDAgente:**

    - Creado y registrado en plataforma SMA desde consola desde el arranque de la aplicación, aunque pueden crearse en cualquier momento posterior.
    - Atiende peticiones de *FiltradorAgente*
    - Accede a una BD para solucionar planificaciones
    - Devuelve planificación resuelta o failure
    - Queda listo para responder nuevas peticiones

- **InternetAgente:**

    - Creado y registrado en plataforma SMA desde consola desde el arranque de la aplicación, aunque pueden crearse en cualquier momento posterior.
    - Atiende peticiones de *FiltradorAgente*
    - Accede a formularios (jsp, servlets, cgi…) para solucionar planificaciones
    - Devuelve planificación resuelta o failure
    - Queda listo para atender nuevas propuestas

Las respuestas de cada Agente se suceden en cadena en sentido contrario a las peticiones, llegando hasta *ClienteAgente*, que será el encargado de, al finalizar todo el proceso, mostrar al usuario los resultados obtenidos, ya sean satisfactorios o no.

5.3.2 Aspectos más profundos del proceso de planificación

Llegados a este punto, al lector se le pueden plantear una serie de dudas de índole más técnica y de más bajo nivel. Por tanto explicaremos los detalles del funcionamiento de todo el proceso de planificación que no hayan quedado claros en la visión general del mismo.

5.3.2.1 Chunk Search

Cuando *BuscadorAgente* recibe una petición con un origen y un destino (además del resto de parámetros, claro está) intenta realizar un trayecto directo entre esos puntos

a través de la cooperación con el resto de Agentes de la plataforma. Pero esto no es siempre posible, pues no existen conexiones directas entre todas las poblaciones.

Es por ello que muchas veces debe dividir esta búsqueda en trozos, lo cual hemos denominado Chunk Search, para completar cada uno de estos trozos como una planificación autónoma (siempre teniendo en cuenta los parámetros generales, y otros que aparecerán en este proceso). Lo explicaremos mediante un ejemplo, para que quede lo más claro posible.

Ejemplo: *Deseamos partir el 19 de septiembre de 2005 desde Els Pallaresos (Tarragona, España) y llegar hasta Londres (Reino Unido), no tenemos predilección en cuanto a los medios de transporte usados ni a las escalas permitidas, tampoco queremos días de margen y nuestro presupuesto es ilimitado. El día de vuelta será el 26 de ese mismo mes.*

El sistema, al recibir esta petición intentará buscar un camino directo entre estas dos poblaciones usando avión, barco, tren o, en algunos casos, carretera. Debemos comentar que, cuando las localidades pertenecen a dos países diferentes, en los primeros intentos por resolver la planificación, se excluye (de manera puntual) el transporte por vehículo, pues es un transporte muy versátil y que prácticamente siempre proporciona resultados, que no siempre son los óptimos. Por tanto, estas primeras veces priorizamos otros medios de transporte más eficientes, como el avión. Aunque así renunciemos a obtener rutas directas en algunos casos, en general es mucho mejor hacer alguna escala y viajar por aire, pues reduciremos el coste en cuanto a tiempo y, en algunos casos, incluso económico.

Como es de esperar, no hay ningún barco, tren o vuelo directo entre estos dos puntos. Aquí entra en juego la descomposición en trozos y la búsqueda de alternativas de las poblaciones mediante el algoritmo de Generalización – Especificación, el cual comentaremos en breve.

Nota: El algoritmo se aplicaría tanto para el origen como para el destino. Pongamos un ejemplo imaginario. No existe conexión entre Tarragona y Londres, pero en cambio, si la existe entre Tarragona y el cercano pueblo a Londres de Strattford. Entonces deberíamos encontrar la ruta que existiría entre:

Els Pallaresos ➔ Tarragona ➔ Strattford ➔ Londres.

En nuestro punto de desarrollo de la aplicación, comentar que el algoritmo Generalización – Especificación sólo es aplicable a poblaciones de nuestro país, debido a que no están insertadas en la base de datos las de otros países. Aunque, el sistema está preparado para ello, e, inmediatamente después de introducirlas en la Base de Datos de CORBA, sería perfectamente funcional, sin realizar absolutamente ninguna modificación adicional. Aunque para Londres no obtengamos resultados, podría darse el caso de querer realizar el trayecto al contrario, Londres ➔ Els Pallaresos, donde origen y destino intercambian los papeles.

Volviendo al hilo del problema y al caso real que hemos planteado, se realizará la búsqueda de alternativas en origen.

Generalización – Especificación proporciona solo poblaciones colindantes a la nuestra y, según el grado de profundidad, serán de la misma provincia que la nuestra, de la misma comunidad, del mismo país o del mismo continente. Siempre empezamos por el grado mínimo, y los resultados suelen aparecer, como máximo, en un segundo grado como regla general. Solo abstraemos a poblaciones con posibilidades dotadas de posibilidades en cuanto a comunicación, entendiéndose aquellas con aeropuerto o estación de tren por ejemplo. Devuelve los resultados ordenados por aquellas localidades con aeropuerto, lo cual representa un aumento de más del 37 % en el rendimiento de la aplicación respecto a los mismos resultados con otra ordenación diferente, según pruebas efectuadas empíricamente.

En una primera aproximación nos devolvería dos resultados. Serían Tarragona (con estación de tren) y Reus (con estación de tren y aeropuerto). El sistema intentará la búsqueda de una trayectoria Reus ➔ Londres. Supongamos que no la encuentra, en este caso, intentará realizar Tarragona ➔ Londres. Al no existir ninguna conexión, ampliaremos el nivel de profundidad de Generalización – Especificación, situándonos a nivel de comunidad autónoma. De nuevo, se aplicaría tanto en origen como en destino. Obteniendo una serie de resultados como Girona, Lleida, Sabadell, Barcelona, etc, con el mismo criterio de ordenación.

Siguiendo análogo procedimiento, y tras fallar en algunos casos, encontrará una conexión entre Barcelona y Londres.

En este punto tenemos la planificación de la siguiente manera:

Origen: Els Pallaresos	Destino: Londres
(¿? ➔ ¿?) + (Barcelona ➔ *Londres*)	

Los trozos (chunks) se ordenan de tal manera que sabemos cual es el orden que nos queda por calcular. En este caso, no sabemos como llegar hasta Barcelona, por tanto podemos probar calcular Els Pallaresos ➔ Barcelona. El sistema reinicia sus parámetro (profundidad de Generalización – Especificación, lista de ciudades ya examinadas, etc) y busca esta ruta teniendo en cuenta los horarios del avión, para no perderlo (en caso de que fuera un trozo al otro lado, tendría en cuenta la hora de llegada del avión, para no coger el siguiente transporte antes de llegar).

El sistema halla ruta directa por carretera para este trozo que falta y así lo incluye. Ya tenemos el viaje de ida completo.

Origen: Els Pallaresos	Destino: Londres
(*Els Pallaresos* ➔ Barcelona) + (Barcelona ➔ *Londres*)	

Ahora realizaríamos la planificación inversa para el viaje de vuelta Londres ➔ Els Pallaresos, con los parámetros del mismo.

Intentaría directo, al no encontrarlo buscaríamos alternativas en destino (pues en Londres, recordamos que no generamos en esta versión). Entonces el primer candidato sería Reus. Encontramos vuelo Londres ➔ Reus, por tanto, sólo nos quedaría realizar por carretera Reus ➔ Els Pallaresos.

Como curiosidad, comentar que no hemos encontrado vuelo Reus ➔ Londres, pero sí Londres ➔ Reus. Esto es debido a la fecha de ida que ha escogido el usuario, en la cual no existe compañía que ofrezca este servicio.

La planificación está ahora completa y devolvemos los resultados al usuario final.

Comentar que para saber cuáles son los trozos que faltan. El sistema, al recibir cada nuevo trozo, lo inserta de manera ordenada en la cadena de trozos ya planificados. (Lista de Subplanifiaciones). Tras esto, realiza una búsqueda encadenada desde el punto de partida definido por el usuario al origen del primer trozo, desde el destino de este

trozo al origen del siguiente, y así sucesivamente. De esta manera localiza los trozos que faltan por completar.

BuscadorAgente comprueba que el sistema cumple las restricciones marcadas por el usuario en cuanto a importe del viaje, o número máximo de escalas.

Por otro lado, hay que tener en cuenta que si no encuentra rutas para un determinado día, podrá resetear la búsqueda, descartar los trozos ya planificados (si los hubiera) e intentar realizar la planificación para días sucesivos o anteriores, según lo indicado por el usuario en cada caso.

5.3.2.2 Selección mejor propuesta

FiltradorAgente recibe propuestas de otros varios Agentes, pero ¿cómo elige cuál es la mejor entre todas las recibidas?

Realmente se basa en una pequeña función heurística, que puntúa los resultados obtenidos ateniéndose a una serie de criterios.

Se pretenderá priorizar aquellas propuestas que:

- Tengan un menor tiempo de viaje.
- La diferencia entre la hora de partida y la hora mínima sea lo más baja posible, lo que implica menor tiempo de espera (e.g. Llega un vuelo a las 12:15h (hora mínima) del trozo anterior. Encontramos dos propuestas, una que sale a las 13:40h (hora de partida) y otra a las 15:03h (hora de partida), priorizaríamos la primera).
- La diferencia entre la hora de llegada y la hora máxima sea lo más baja posible, lo que implica menor tiempo de espera (e.g. Debemos coger un vuelo a las 12:15h (hora máxima) del trozo posterior. Encontramos dos propuestas, una que llega a las 10:40h (hora de partida) y otra a las 9:03h (hora de partida), priorizaríamos la primera).
- El medio de transporte para esta propuesta esté incluido en la lista de transportes preferidos del cliente.
- El precio del trayecto sea más bajo.

Notar que *FiltradorAgente* sólo recibe propuestas válidas, pues *BuscadorAgente* se encarga de eliminar las peticiones que no se ajustan a los transportes permitidos por el usuario y *BDAgentes* e *InternetAgentes*, se encargan de descartar aquellas propuestas que no respeten las fechas u horarios impuestos en la petición.

Los controles respecto importe, número de escalas, o margen de días, corren a cargo de *BuscadorAgente* en última instancia, que decidirá si probar otras alternativas o indicar al usuario que cambie algunos de sus parámetros de búsqueda pues no ha habido resultados satisfactorios.

5.3.3 Creación dinámica de Agentes mediante JSP

En este apartado damos referencias básicas de cómo es el mecanismo usado por parte de nuestra página JSP en la creación de un Agente nuevo e inserción del mismo en la plataforma SMA.

5.3.3.1 Proceso general (Pseudo – código) de la página JSP

1. Validación de sesión para comprobar que sea un usuario registrado
2. Obtención de parámetros especificados por el usuario para realizar la búsqueda
3. Acceder al RunTime de JADE
4. Crear un perfil por defecto
5. Crear otro contenedor conectando al contenedor principal del SMA
6. Crear un Agente nuevo de la clase *ClienteAgente* y pasarle los parámetros pertinentes, como objetos
7. Dotar a nuestro Agente con un identificador único para que no hayan colisiones
8. Crear un Controlador de Agentes
9. Lanzar el Agente
10. Redirigir la salida a otro JSP que esperará que *ClienteAgente* devuelva los resultados y muera, antes de mostrarlos por pantalla al usuario final

5.3.4 Algoritmo Generalización – Especificación

Uno de los puntos fundamentales en nuestra estrategia de planificación usada, estriba en el uso de un algoritmo creado especialmente para proporcionar una mayor efectividad en la búsqueda de rutas. Se puede dividir en dos partes:

1. **Fase de preparación e invocación:** En primer lugar se debe preparar el sistema para poder operar con métodos remotos. Para ello, es necesario contactar con nuestro servidor CORBA implementado para atender nuestras peticiones particulares.

2. **Fase de ejecución y presentación de resultados:** Una vez haya sido invocado el método de interfaz, este se ejecutará en la máquina donde esté corriendo el servidor CORBA. En el momento de obtener los resultados necesarios, éstos serán enviados de nuevo hacia el Agente que lo haya invocado.

Veamos con mayor detalle en qué consiste exactamente nuestro algoritmo:

5.3.4.1 Pseudo - código Fase de preparación e invocación (lado cliente)

- Obtenemos los parámetros de inicialización desde un fichero de configuración. Necesitamos saber la dirección de la máquina (host) donde está corriendo el servidor CORBA, así como también el puerto por donde acepta las peticiones.
- Creamos un objeto CORBA con el que conectaremos al servidor remoto. Nos proporciona un objeto remoto (de la clase *CorbaBDImpl*) con el cual podemos operar transparentemente como un objeto local. A partir de este momento se pueden invocar todos sus métodos.
- Realizamos la invocación del método ***public boolean*** *realizarGeneralizacion(String lugar,* ***int*** *nivelProfundidad)*, que explicaremos en el siguiente apartado, pasando como parámetros el lugar que queremos tratar (pueblo, ciudad…) y el nivel hasta donde queremos realizar la Generalización (localidad, provincia, comunidad, país o

continente) según nos interese obtener más o menos resultados y más o menos distantes los unos de los otros.

- Si se han producido resultados, realizamos nuevamente invocaciones remotas para recuperarlos desde el servidor, esta vez sobre los métodos ***public*** *String getNombre()* y ***public boolean*** *avanzaFila()* principalmente.
- En el caso que solo haya un resultado obtenido, debemos incrementar el nivel de profundidad y volver a realizar la operación, debido a que este tipo de casos no nos ayudaría en nada, pues se produce por ejemplo cuando el nombre de la localidad y de la provincia, se llaman igual (Ej. Badajoz – Badajoz).

5.3.4.2 Fase de ejecución y presentación de resultados (lado servidor)

Explicamos el métod ***public boolean*** *realizarGeneralizacion(String lugar,* ***int*** *nivelProfundidad),* ya mencionado en el anterior apartado, debido a que es el más importante de esta fase, por ser encargado de realizar la Generalización y Especificación en última instancia. El resto de métodos se pueden consultar directamente en el código fuente y en los documentos creados con JavaDoc (incluidos en el CD de distribución).

- Establecemos conexión con la base de datos que contiene la información a cerca de cada localidad y su localización jerárquica dentro de la ordenación territorial mundial. A partir de aquí se harán una serie de consultas SQL que tendrán por finalidad completar los pasos siguientes. Para mayor detalle, deberíamos recurrir al anexo, al apartado de código fuente.
- Se comprueba a que **provincia** pertenece la *supuesta localidad*.
- Si localizamos la *supuesta localidad*, empezamos la generalización. Buscamos todas las localidades pertenecientes a la misma provincia que tengan buenas comunicaciones, como aeropuerto o estación de tren.
    - En caso de no haber encontrado ninguna localidad o de querer más profundidad, buscamos a nivel de **comunidad**. Vemos a que comunidad pertenece la provincia.
    - Buscamos todas las localidades de esa comunidad.

- En caso de no haber encontrado ninguna localidad o de querer más profundidad, buscamos a nivel de **país**. Vemos a que país pertenece la comunidad.
- Buscamos todas las localidades de ese país.
- En caso de no haber encontrado ninguna localidad o de querer más profundidad, buscamos a nivel de **continente**. Vemos a que continente pertenece el país.
- Buscamos todas las localidades de ese continente.

- En el caso que la *supuesta localidad* no sea localizada, comprobaremos si es una *supuesta provincia*, una *supuesta comunidad*, un *supuesto país* o un *supuesto continente*. Iniciando la Generalización y Especificación desde ese nivel.

Se debe tener en cuenta que no devolverá todas las localidades de un determinado territorio, sino que lo sólo lo hará con aquellas con buenas comunicaciones, es decir, que dispongan de aeropuerto o estación de tren por ejemplo. Además se dan ordenadas de manera que los medios potencialmente mejores tengan preferencia, en este sentido las localidades provistas de aeropuerto se llevan la prioridad más alta. Con esta ordenación, se mejora el rendimiento en torno al 40 %. Con el descarte de localidades mal comunicadas se realiza una poda de caminos que seguramente no aportarían ninguna solución al problema. La mejora de rendimiento con esta medida es inmensa.

5.3.4.3 Conclusiones

Resumiendo, este algoritmo toma una determinada localidad y realiza una Generalización en el sentido que abstraemos a un nivel superior, para mirar con mayor perspectiva. Es como si hiciéramos un *zoom* de alejamiento sobre un mapa imaginario. Entonces vemos las zonas delimitadoras de esta localidad, en el nivel deseado, y realizamos la fase de Especificación, haciendo una selección de las localidades más interesantes. Es decir, subimos **n** niveles, para volver a bajarlos inmediatamente después.

Una vez más queda patente la naturaleza distribuida de nuestro proyecto. El algoritmo se divide en tres espacios físicos:

1. Lugar donde se ejecuta el Agente, que es quien realiza las invocaciones a los métodos.
2. Lugar donde corre el Servidor CORBA y donde se realizan las operaciones implementadas en los métodos.
3. Lugar donde se hospeda la Base de Datos que consultamos para obtener la información territorial.

Esta división implica una serie de ventajas, entre las que podemos comentar:

- El no tener que disponer cada Agente del código del algoritmo, pudiendo ser accedido directamente con una simple llamada como si fuera un método ordinario.
- Al tener la codificación centralizada, se pueden efectuar cambios internos de funcionamiento de manera transparente a los usuarios, dado que existe un contrato en forma de interfaz que garantiza la compatibilidad en todo momento.
- Podemos tener máquinas dedicadas especialmente a cada servicio, lo que aumenta el rendimiento notablemente.

5.3.5 CORBA en nuestro proyecto

Aprovechando que acabamos de comentar el algoritmo donde entra en juego la parte de CORBA en nuestra aplicación, inauguramos este apartado con el objetivo de hacer entender al lector las nociones esenciales de esta tecnología, el papel de la misma en nuestro proyecto y la manera en que la usamos.

5.3.5.3 Nuestro desarrollo bajo CORBA

Enumeraremos los pasos que se deben seguir para desarrollar un servicio CORBA y, paralelamente, explicaremos como lo hemos hecho nosotros en cada uno de ellos.

1. Se diseña el servicio y se crea la interfaz IDL. En la ilustración 14 observamos la nuestra. Estos serán lo métodos accesibles remotamente a través del objeto

CORBA.

```
module CorbaBD

{

 interface CorbaBD

 {

  boolean realizarQuery(in string consulta);

  boolean realizarGeneralizacion(in string lugar, in long nivelProfundidad);

  string imprimirResultadoConsulta();

  string getPertenece();

  string getNombre();

  boolean avanzaFila();

  oneway void shutdown();

 };

};
```

Ilustración 14: Interfaz IDL de nuestro servicio

2. Se elige la plataforma y el lenguaje de implementación y se busca la herramienta CORBA para esa plataforma y ese lenguaje. En nuestro caso hemos elegido JAVA corriendo sobre Windows.
3. La herramienta CORBA debe tener un compilador IDL que, a partir de la interfaz IDL, generará los proxys que permiten enganchar la implementación de la interfaz con la arquitectura CORBA. Nosotros hemos usado el compilador IDL proporcionado por JAVA en su distribución habitual de desarrollo (nosotros usamos *idlj.exe* de J2SDK v1.4.2.02).
4. Implementamos la interfaz IDL en el lenguaje elegido. En nuestro caso se ha resuelto en la clase *CorbaBDImpl.java*. La cual encapsula los métodos necesarios para interactuar con la BD usada por el algoritmo Generalizar – Especificar.
5. Creamos un servidor CORBA que se encargue de registrar la nueva funcionalidad en CORBA, utilizando la implementación realizada por nosotros y

los proxys generados a partir de la interfaz IDL. Para eso hemos creado la clase *CorbaBDServer.java*.

6. Los nuevos servicios ya están disponibles en CORBA a la espera de la llegada de clientes que los quieran utilizar. Para que pueda prestar servicio, debemos lanzar el servicio ORB y nuestro servidor, que ya sabemos cual es. Pero… ¿qué es el ORB?

También comentar que en nuestro proyecto usamos el servicio ORB proporcionado, al igual que el compilador de IDL, por J2SDK v1.4.2.02, con la aplicación *orb.exe*. Y que deberemos lanzar antes que nuestro servidor, pues será el que éste usará para registrar sus servicios.

En la guía del administrador podemos seguir paso a paso todos los servicios y el orden de activación de los mismos, que necesitamos para que nuestra aplicación funcione.

5.4 Agentes interactuando con Internet (InternetAgente)

En estas líneas comentaremos la manera en la que hemos dotado a nuestros Agentes con la posibilidad de interactuar directamente con Internet. Se detallarán algunos aspectos a nivel de implementación debido a que si no lo hiciéramos, sería una explicación muy genérica y poco didáctica. Además, es uno de los puntos clave, desde el principio fue uno de los principales objetivos, por tanto merece un tratamiento más minucioso.

5.4.1 Ventajas del uso de Agentes para búsquedas en Internet

Idealmente, un Agente es un ente autónomo con iniciativa y capacidad de realizar tareas complementarias para resolver un problema complejo. Si realizamos un pequeño paralelismo con el perfil de un usuario corriente realizando búsquedas en Internet, vemos varios potenciales puntos en común. Entonces… ¿por qué no crear un Agente que realice esta pesada tarea? Pues, gracias a las ventajas y características de los

mismos en general, se presentan como unos extraordinarios candidatos para llevarlas acabo de la manera más eficiente posible.

5.4.2 Problemas relacionados con la interacción

Internet es un universo dinámico, en continua evolución y cambio. Por ello se debía encontrar una solución fácilmente adaptable y modificable, que fuera lo más flexible posible. A parte de esto, debía ser eficiente, rápida en el manejo de conexiones, y lo suficientemente hábil para poder relacionarse con cualquier recurso interesante de los disponibles en la red.

La información que nos interesa la podemos encontrar en varias URL de compañías dedicadas al transporte de pasajeros. Cada una de ellas, con unas peculiaridades específicas, pero todas comparten muchos aspectos en común. Por tanto se ha tratado de diseñar los Agentes de tal manera que tuvieran una parte de la funcionalidad común, que es la que servirá como base a cualquier consulta, ampliado con otras funcionalidades adaptadas a cada caso. Es algo que podemos realizar de manera natural con el uso de lenguajes orientados a objetos con herencia y especialización de clases.

A parte, debido a la evolución de tecnología en Internet, los paquetes básicos de comunicación de Java no soportan los métodos adecuados para poder hacer frente a cualquier situación. Es por ello que hemos recurrido a paquetes extra proporcionados por terceros. En nuestro caso hemos recurrido a Jakarta y su módulo *commons httpclient* en su versión 2.0.2 como piedra angular de las conexiones.

Todas estas decisiones, las veremos detalladas a continuación.

5.4.3 Diseño y detalles de implementación

Se ha diseñado una arquitectura que requerirá del empleo de dos clases extra para cada nueva interacción con un recurso diferente de Internet.

Es posible que una misma clase maneje varios recursos. Sin embargo, no se aconseja, facilitando de esta manera la modificación de las clases en caso de ser necesario.

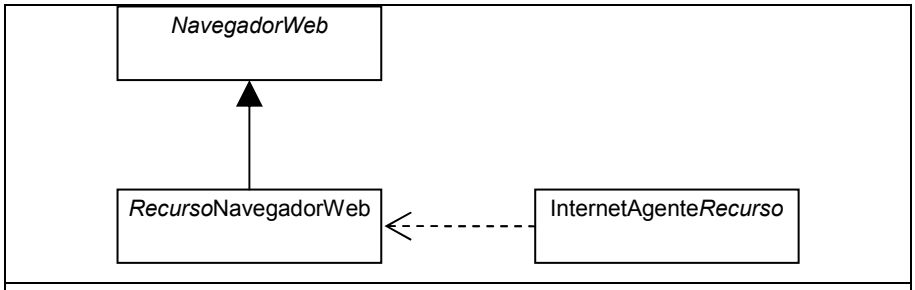


Ilustración 15: Diagrama de clases relacionadas con *InternetAgente*

En la ilustración anterior se muestra la disposición de las clases involucradas en la implementación de *InternetAgente*. Veamos la función de cada una.

- *NavegadorWeb*: Es una clase abstracta en forma de interfaz Java. En ella se denota la manera con la que se debería interactuar con Internet. El uso de los métodos necesarios que se debería implementar en sus subclases a fin de mantener un estándar. Los métodos son:
    - *public void connect(String horaSalida,String ciudadOrigen,String ciudadDestino,String diaPartida,String mesPartida).* Encargado de conectar a la página y preparar el flujo de datos para realizar las consultas y obtener las respuestas.
    - *public void readContents(Vector listaAlternativas).* Encargado de obtener e interpretar los resultados que nos llegan.
    - *public void disconnect();* Encargado de desconectar la conexión y liberar recursos.
- Recurso*NavegadorWeb*: Es una clase que codifica los métodos descritos en la interfaz que implementa (clase *NavegadorWeb*). Es la clase encargada de encapsular toda la lógica de conexión e interacción a Internet para el recurso que nos interese en cada caso.
  Notar que el prefijo 'Recurso' se debería sustituir en cada caso por el nombre del recurso al cual se desea conectar (e.g. ViaMichelin, Renfe, Lastminute).
- *InternetAgente*Recurso: Es la clase del Agente. Define su comportamiento en cada situación. Atenderá peticiones e intentará resolverlas haciendo alarde de su capacidad de interacción con Internet que le proporciona el uso de la clase Recurso*NavegadorWeb*.
  Notar en este caso que el sufijo 'Recurso' se debería sustituir en cada caso por el

nombre del recurso al cual se desea conectar (e.g. ViaMichelin, Renfe, Lastminute) y que éste debería coincidir por convención interna con el prefijo de Recurso*NavegadorWeb*.

5.4.4 Proceso de ingeniería inversa sobre sitios

En este apartado trataremos de explicar la metodología general que se sigue para conseguir interactuar con las diferentes páginas a las que hemos accedido, y que es válido para conectar con la mayoría de *sitios* que ofrecen productos similares.

Seguiremos el proceso de ingeniería inversa conjuntamente con un ejemplo basado en la Web de Renfe a la cual hemos accedido, detallando en cada paso los factores a tener en cuenta.

Para una correcta interpretación de los datos nos harán falta conocimientos sobre:

- Aplicaciones sobre Internet (JSP, CGI, etc)
- Lenguajes de script usados en páginas HTML (JavaScritp, jScritp, etc)
- HTML

5.4.4.1 Localizar formulario de la página que lanza la petición

El primer paso a seguir es encontrar el formulario que genera la consulta y que tiene todos los parámetros necesarios. En la siguiente ilustración mostramos la página inicial de Renfe, en la que nos hemos basado.

Ilustración 16: Pagina principal de Renfe.es

Localizamos rápidamente cuál es el formulario principal y ya empezamos a ver cuáles son los parámetros principales que necesitaremos rellenar para personalizar nuestra búsqueda, aunque deberemos saber cuáles son sus nombres y qué valores toman, pues pueden sufrir transformaciones de manera interna, ya sea mediante lenguajes de script, o mediante simples combos con valores diferentes a la etiqueta que lee el usuario (como en este caso).

Debemos editar el código fuente del *sitio* en cuestión y localizar en él, los diferentes campos de entrada que tenemos. En este caso aparece la estación de origen y de destino, en forma de combo, de la misma manera que podemos introducir el día, mes y año en que queremos realizar el trayecto con este mismo tipo de controles. Se puede hacer con la ayuda de una herramienta visual como DreamWeaver para localizarlos más rápidamente. En el ejemplo de la ilustración tenemos seleccionado el combo que aparece al lado de 'Mes' con valor 'Sin Fijar', y se nos resalta el código HTML correspondiente de manera directa.

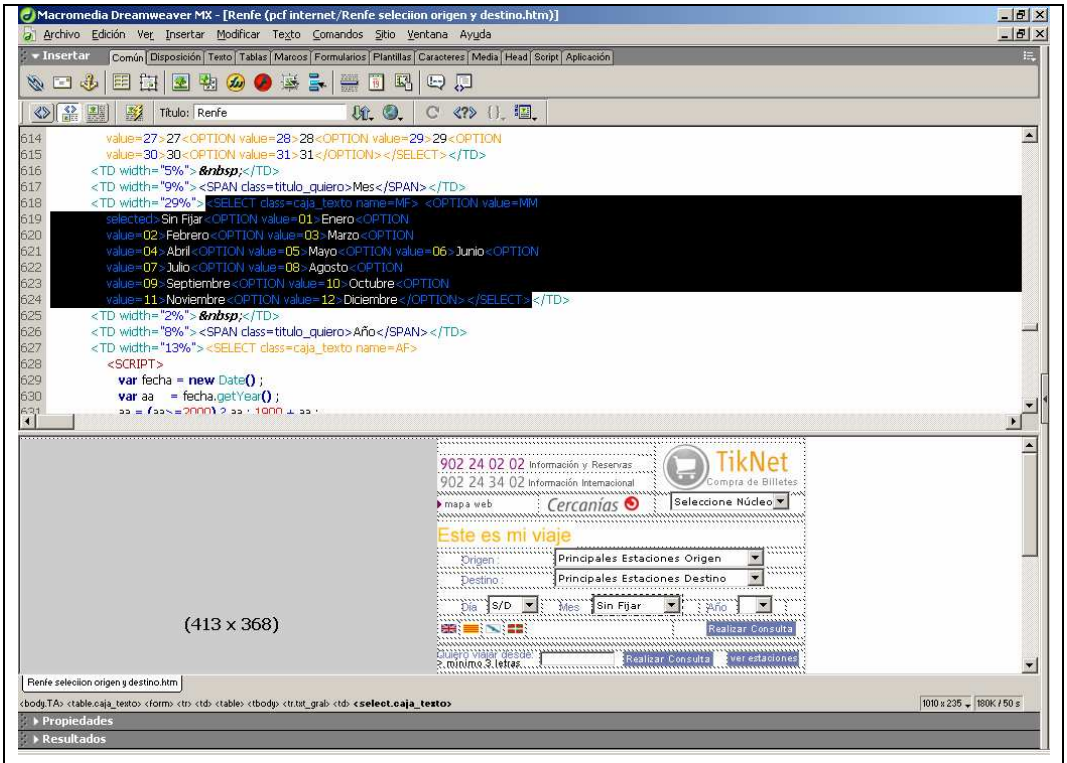


Ilustración 17: Localización de datos con DreamWeaver

Una vez localizado todos los campos 'rellenables' por el usuario, debemos anotar su nombre, pues lo deberemos pasar nosotros también.

En general, los campos combo, nos servirán para comprobar si se hace alguna transformación sobre los datos. Por ejemplo, seleccionamos estación Barcelona y el sistema envía un valor diferente codificado. Esto nos servirá para realizar una serie de diccionarios para realizar las correspondencias.

En el recuadro de la ilustración 18, observamos lo que comentábamos. Los campos más importantes de cada parámetro son el nombre y el valor que toma, que puede ser variable (seleccionable por un control combo), modificable (como un campo de texto), o predeterminado (generalmente en campos de tipo *hidden*).

En nuestro ejemplo vemos como las ciudades se corresponden con códigos numéricos, que es lo que en última instancia reconocerá el servidor final. También localizamos que el nombre del parámetro es '*o*', lógico, si sabemos que estamos eligiendo la estación de origen. Por tanto vemos que tendremos que transformar la ciudad que nos diga el usuario en nuestra aplicación a un código que la represente y ponerlo como valor en el atributo '*o*'.

```
<SELECT class=caja_texto name=o>
<OPTION value=? selected>Principales Estaciones Origen 
<OPTION value=31412>A Coruña/La Coruña
<OPTION value=60911>Alacant-Terminal/Alicante Término
<OPTION value=60600>Albacete
<OPTION value=60400>Alcázar de San Juan
<OPTION value=55020>Algeciras
<OPTION value=56312>Almería
<OPTION value=10400>Avila
…
```

Ilustración 18: Parámetro de tipo combo con valores distintos a la elección de usuario

Otro de los parámetros importantes son los denominados *hidden*, es decir que están ocultos al usuario pues no son modificables por el mismo. Suelen tener un valor por defecto predefinido que nunca se modifica. Deberemos anotar el nombre de los mismos y el valor correspondiente en cada caso, pues también deberemos enviarlos.

En el código, también deberemos observar la cabecera del formulario de envío, de dónde extraeremos la dirección del servidor que procesa las peticiones y el método sobre el que se hace.

```
<FORM action=http://horarios.renfe.es/hir/index.jsp? method=get>
<INPUT type=hidden value=s name=ID>
<INPUT type=hidden value=hjhir110.jsp name=page>
…
```

Ilustración 19: Ejemplo de campos hidden y de cabecera de formulario

En este ejemplo vemos un par de atributos tipo *hidden*, que deberemos pasar igualmente, como puede ser el caso de '*ID*' que siempre llevará el valor '*s*'. En cuanto a los formularios, los datos importantes son el que pone '*method'* y el que pone '*action*' que corresponden con el método de acceso y con la URL del servidor al que accederemos.

Para asegurarnos de no dejarnos ningún campo sin parametrizar, es muy útil, en muchas ocasiones, analizar la barra de direcciones de las páginas de resultados, donde se muestra la URL seguida de los parámetros recibidos y sus valores asociados.

Por ejemplo, en Renfe.es se produce la siguiente URL: '*http://horarios.renfe.es/hir/index.jsp?page=hjhir130.jsp&O=71500&D=BARCE&AF=2005&MF=09&DF=15&SF=NaN&ID=s&FHORA=23.36.01*'

En ella vemos las diferentes partes, en primer lugar el servidor que nos proporciona el servicio con un primer parámetro adjunto (http://horarios.renfe.es/hir/index.jsp?page=hjhir130.jsp), a partir de aquí se suceden el resto de parámetros, como origen parte des Tarragona, pero está codificado según el combo, como hemos visto anteriormente (O=71500), el destino es Barcelona (D=BARCE) y también se ve la fecha para la cual hemos realizado la consulta dividida en año, mes y día (AF=2005&MF=09&DF=15), luego existen un par de parámetros de tipo *hidden* que anteriormente habíamos localizado en el código y que siempre tienen el mismo valor (SF=NaN&ID=s) y, finalmente, la hora a la que hemos hecho la consulta (FHORA=23.36.01).

5.4.4.2 Localizar formulario modo de acceso a la página de resultados

Debemos descubrir como se accede a la página de resultados. Hay sitios que los muestran de manera directa tras realizar la petición (como es el caso de Renfe.es). Otros, en cambio, nos dejan a la espera con una pantalla que simula una barra de progreso (léase Lastminute.es). En apartados sucesivos veremos como actuar en cada caso.

5.4.4.3 Localizar información en la página de resultados

Una vez hayamos localizado la página de resultados, deberemos nuevamente acceder a su código fuente, de la misma manera que lo hicimos en el formulario. Buscaremos los campos que nos aportan información, así como la manera de localizarlos de manera genérica.

Por suerte, el motor de generación de respuestas de cada *sitio* actúa siempre del mismo modo, por tanto, una vez hayamos conseguido discretizar los valores interesantes, nos valdrá para todas las veces.

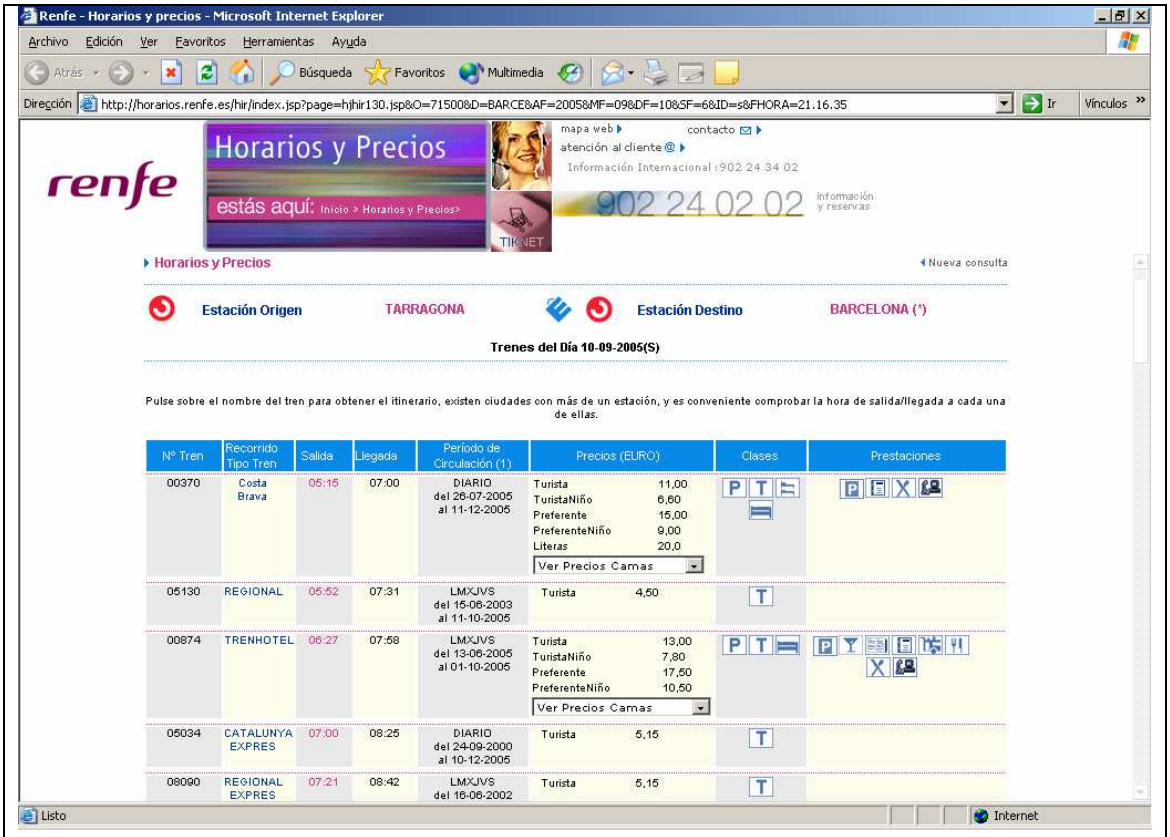


Ilustración 20: Resultados de la búsqueda

En este caso, vemos la información repartida en varias filas, a lo largo de las cuales se muestran los datos de cada trayecto.

Debemos encontrar la manera de localizar estos campos a través del código fuente. En este caso podemos obtener el Nº Tren, Recorrido Tipo Tren, Salida, etc.

```
<td class="txt_gral" align="center" valign="top" width="52" bgcolor="FCFCEB">
<a href=/hir/hjhir140.jsp?&T=00370&G=1&ID=s&F=2005-07-25 class="link_azul">Costa
Brava</a>
</td><td class="txt_tiknet01" align="center" valign="top" width="65"
bgcolor="EAE9E9">01:50</td>
```

Ilustración 21: Código HTML de la página de resultados

Debemos encontrar Strings que se repitan justo antes de cada campo en todos los casos. En el siguiente cuadro vemos como lo hemos solucionado nosotros.

```
/* Buscamos TipoTren */
indiceIni = lineaLeida.indexOf("\"link_azul\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 12;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	tipoTren = lineaLeida.substring(indiceIni,indiceFi);
}
indiceIni = indiceFi;
/* Buscamos horaSalida */
indiceIni = lineaLeida.indexOf("\"EAE9E9\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 9;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	horaSalida = lineaLeida.substring(indiceIni,indiceFi);
}
```

Ilustración 22: Interpretación del tipo de tren y la hora de salida

Obviamente, antes nos tenemos que posicionar en la región donde empiezan los datos del trayecto. Pero se haría de manera análoga.

5.4.4.4 Conclusiones

Hemos presentado una metodología genérica que nos orienta a encontrar la manera de interpretar tanto las páginas de elaboración de consultas, como las de presentación de resultados. Sin embargo, cada sitio tiene sus peculiaridades y tecnologías dispares, por lo que habría que adaptarlo a cada caso en concreto.

Por otro lado, también deberemos tener un poco de '*ideas felices*', sobretodo en las primeras páginas que adaptemos, cuando actuemos con tecnologías desconocidas.

Para conseguir buenos resultados, será imprescindible una fase de '*Trial and Error*', hasta encontrar la manera en que se hace exactamente.

5.4.5 Páginas con las que realizamos las consultas

En este capítulo comentamos las tres páginas a las cuales accedemos y cómo lo hacemos en cada caso.

Se ha escogido una página que nos lleva a través de carretera, otra que nos lleva por tren, y otra por aire.

5.4.5.1 Página de tren Renfe.es

Mediante Renfe.es podemos acceder a los horarios e itinerarios entre las principales estaciones españolas, y alguna fuera de nuestras fronteras.

Conexión

Para conectarnos a través de nuestro *InterenetAgenteRenfe*, se ha implementado la clase *RenfeNavegadorWeb.java* que encapsula los métodos necesarios para la conexión e interpretación de la información.

Lo primero que hacemos es acceder al servicio: "horarios.renfe.es/hir/hjhir130.jsp", al cual tenemos que pasar mediante el método *post* una serie de parámetros para realizar la búsqueda, estos son los que podemos ver en la siguiente tabla:

```
/* Introducimos los parametros comunes a todas las llamadas */
post.addParameter("ID","s");
post.addParameter("SF","2");
/* Introducimos los parametros particulares de cada llamada */
post.addParameter("O",this.ciudadOrigen);
post.addParameter("D",this.ciudadDestino);
post.addParameter("FHORA",horaSalida);
post.addParameter("AF",anyFecha);
post.addParameter("MF",mesFecha);
post.addParameter("DF",diaFecha);
```

Ilustración 23: Parámetros necesarios para realizar la consulta

Debemos tener en cuenta que Renfe.es realiza una sustitución de localidades por códigos de las mismas, para ello, hemos implementado un diccionario que realiza esta traducción, veámoslo en esta tabla:

```
/* Diccionario de paises y codigos usados por Renfe.es */
ciudades = new Hashtable(76);
ciudades.put("a coruña","31412");
ciudades.put("alicante","60911");
ciudades.put("albacete","60600");
ciudades.put("alcazar de san juan","60400");
ciudades.put("algeciras","55020");
ciudades.put("almeria","56312");
ciudades.put("avila","10400");
ciudades.put("badajoz","37606");
ciudades.put("barcelona","BARCE");
ciudades.put("bilbao","13200");
ciudades.put("bobadilla","54400");
ciudades.put("burgos","11100");
```

```
ciudades.put("caceres","35400");
ciudades.put("cadiz","51405");
ciudades.put("calatayud","70600");
ciudades.put("cartagena","61307");
ciudades.put("castellon","65300");
ciudades.put("ciudad real","37200");
ciudades.put("cordoba","50500");
ciudades.put("cuenca","66100");
ciudades.put("san sebastian","11511");
ciudades.put("ferrol","21010");
ciudades.put("figueres","79309");
ciudades.put("gandia","69110");
ciudades.put("gijon","GIJON");
ciudades.put("girona","79300");
ciudades.put("granada","05000");
ciudades.put("guadalajara","GUADA");
ciudades.put("huelva","43019");
ciudades.put("huesca","74200");
ciudades.put("irun","11600");
ciudades.put("jaen","03100");
ciudades.put("jerez de la frontera","51300");
ciudades.put("leon","15100");
ciudades.put("linares","50300");
ciudades.put("lleida","78400");
ciudades.put("logroño","81100");
ciudades.put("lorca","06006");
ciudades.put("lugo","20309");
ciudades.put("madrid","MADRI");
ciudades.put("malaga","54413");
ciudades.put("medina del campo","10500");
ciudades.put("merida","37500");
ciudades.put("miranda de ebro","11200");
ciudades.put("monforte de lemos","20300");
ciudades.put("monzon","78301");
ciudades.put("navalmoral de la mata","35206");
ciudades.put("ourense","22100");
ciudades.put("oviedo","15211");
ciudades.put("palencia","14100");
ciudades.put("pamplona","80100");
ciudades.put("ponferrada","20200");
ciudades.put("pontevedra","23004");
ciudades.put("portbou","79315");
ciudades.put("puerto de santa maria","51400");
ciudades.put("puertollano","37300");
ciudades.put("reus","71400");
ciudades.put("sahagun","15009");
ciudades.put("salamanca","30100");
ciudades.put("santander","14223");
ciudades.put("santiago de compostela","31400");
ciudades.put("segovia","12100");
ciudades.put("sevilla","51003");
ciudades.put("soria","82100");
ciudades.put("tarragona","71500");
ciudades.put("teruel","67200");
ciudades.put("toledo","92102");
ciudades.put("tudela","81202");
ciudades.put("valdepeñas","50102");
ciudades.put("valencia","VALEN");
ciudades.put("valladolid","10600");
ciudades.put("vigo","22303");
ciudades.put("villena","60902");
ciudades.put("vitoria","11208");
ciudades.put("zamora","30200");
ciudades.put("zaragoza","ZARAG");
```

Ilustración 24: Diccionario de ciudades y códigos de Renfe.es

En la siguiente ilustración se muestra el formulario de búsqueda de la página que estamos consultando.

Ilustración 25: Pagina principal de Renfe.es

Obtención pagina resultados

Con el proceso anterior, emulamos el comportamiento de un navegador Web en el envío de los datos de un formulario. La página de resultados se obtiene directamente en la inmediata respuesta del servidor. Simplemente hay que abrir un flujo de entrada por donde leerla.

Interpretación y elección de resultados

Podemos obtener los campos que podemos observar en la tabla siguiente:

```
/* Buscamos el numero de tren elegido */
indiceIni = lineaLeida.indexOf("\"EAE9E9\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 9;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	numeroTren = lineaLeida.substring(indiceIni,indiceFi);
}
indiceIni = indiceFi;
/* Buscamos TipoTren */
indiceIni = lineaLeida.indexOf("\"link_azul\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 12;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	tipoTren = lineaLeida.substring(indiceIni,indiceFi);
}
indiceIni = indiceFi;
/* Buscamos horaSalida */
indiceIni = lineaLeida.indexOf("\"EAE9E9\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 9;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	horaSalida = lineaLeida.substring(indiceIni,indiceFi);
}
/* Buscamos horaLlegada */
indiceIni = lineaLeida.indexOf("\"FCFCEB\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 9;
	indiceFi = lineaLeida.indexOf("<",indiceIni);
	horaLlegada = lineaLeida.substring(indiceIni,indiceFi);
}
/* Buscamos periodo de circulacion */
indiceIni = lineaLeida.indexOf("\"EAE9E9\">",indiceIni);
if(indiceIni!=-1)
{
	indiceIni += 9;
	indiceFi = lineaLeida.indexOf("</td>",indiceIni);
	periodoCirculacion = lineaLeida.substring(indiceIni,indiceFi);
}
/* Buscamos precio mas economico */
indiceIni = lineaLeida.indexOf("<td class=\"txt_gral\">",indiceIni);
indiceIni = lineaLeida.indexOf("<td class=\"txt_gral\">",indiceIni+21);
if(indiceIni!=-1)
{
	indiceIni += 21;
	indiceFi = lineaLeida.indexOf("</td>",indiceIni);
	precioTrayecto = lineaLeida.substring(indiceIni,indiceFi);
	precioTrayecto = precioTrayecto.replace(',','.');
}
```

Ilustración 26: Obtención de los campos necesarios

Como vemos, vamos analizando el código mediante la búsqueda de cadenas de texto que preceden, suceden e identifican unívocamente el dato al cual queremos tener acceso. Nos quedamos con el valor que más nos convenga según horario, pues los resultados se muestran ordenados por hora de partida y de llegada, por tanto podemos localizar la mejor opción teniendo en cuenta nuestra hora mínima de salida (hora antes

de la cual no podemos partir) y hora máxima de llegada (hora máxima a la que podemos llegar a un destino para no perder el siguiente transporte).

En la siguiente ilustración vemos la página de resultados que vemos en un navegador y de la cual hemos extraído estos datos.

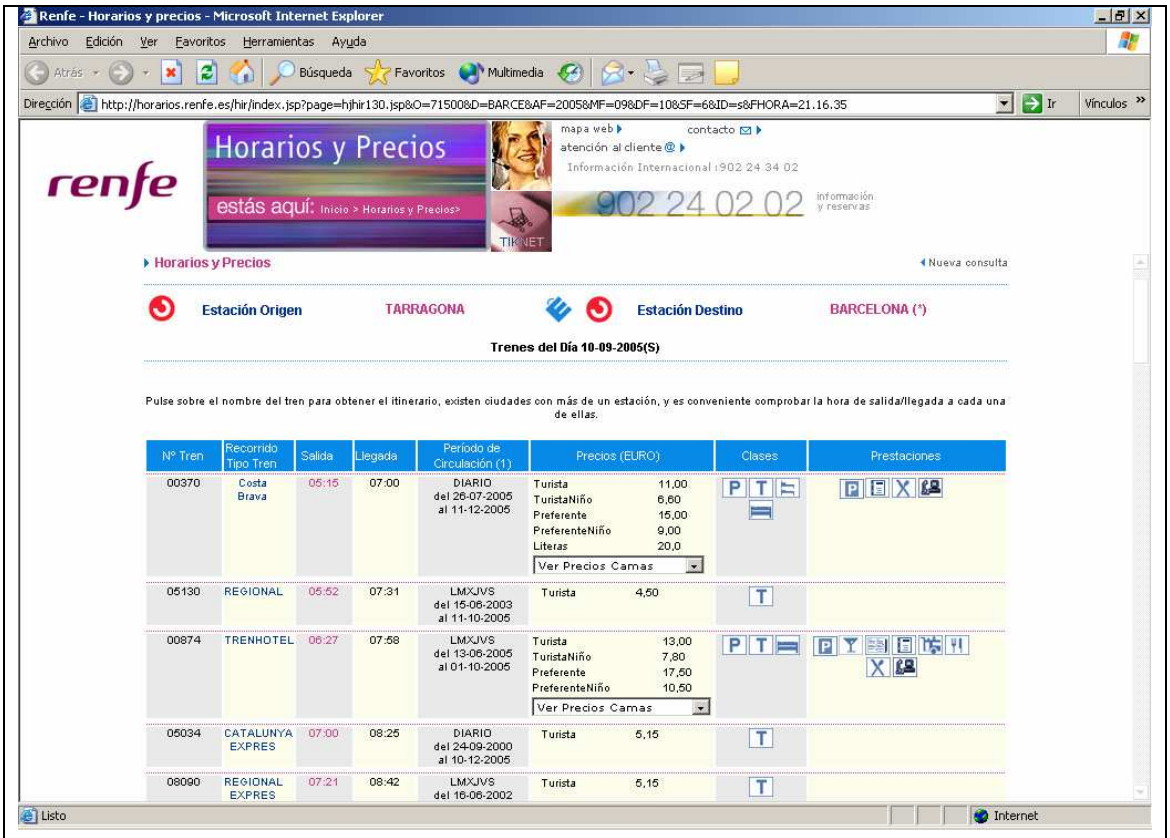


Ilustración 27: Resultados de la búsqueda

5.4.5.2 Página de carretera Viamichelin.es

Mediante Viamichelin.es podemos acceder a las rutas por carretera entre prácticamente la totalidad de poblaciones europeas. Calcula el trayecto óptimo, en cuanto a tiempo, distancia, calidad de carreteras, incidencias en las mismas como obras, etc.

Conexión

Para conectarnos a través de nuestro *InterenetAgenteViamichelin*, se ha implementado la clase *ViaMichelinNavegadorWeb.java* que encapsula los métodos necesarios para la conexión e interpretación de la información.

Lo primero que hacemos es acceder al servicio: "www.viamichelin.es/viamichelin/esp/dyn/controller/ItiWGPerformPage", al cual tenemos que pasar mediante el método *post* una serie de parámetros para realizar la búsqueda, estos son los que podemos ver en la siguiente tabla:

```
/* Introducimos los parametros comunes a todas las llamadas */
post.addParameter("E_wg","210505208jS1405207172827765965ITIWG2f10133esp0023100h100
4101004101001001072005207039.004-1.001100010010010010010010003esp011");
post.addParameter("pim","true");
post.addParameter("strStartAddress","");
post.addParameter("strStartCP","");
post.addParameter("strDestAddress","");
post.addParameter("strDestCP","");
post.addParameter("strStep1City","");
post.addParameter("strStep2City","");
post.addParameter("strStep3City","");
post.addParameter("dtmDeparture","26%2F07%2F2005");
post.addParameter("intItineraryType","1");
post.addParameter("intOneCountryCheck","true");
post.addParameter("unit","km");
post.addParameter("vh","CAR");
post.addParameter("conso","6");
post.addParameter("carbCost","1.00");
post.addParameter("devise","1.0%7CEUR");
post.addParameter("devise2","Otro");
post.addParameter("image.x","12");
post.addParameter("image.y","12");
/* Introducimos los parametros particulares de cada llamada */
post.addParameter("strStartCity",this.ciudadOrigen);
post.addParameter("strStartCityCountry",this.paisOrigen);
post.addParameter("strDestCity",this.ciudadDestino);
post.addParameter("strDestCityCountry",this.paisDestino);
post.addParameter("strStep3CityCountry",this.paisDestino);
post.addParameter("strStep2CityCountry",this.paisDestino);
post.addParameter("strStep1CityCountry",this.paisDestino);
```

Ilustración 28: Parámetros necesarios para realizar la consulta

En la siguiente ilustración se muestra el formulario de búsqueda de la página que estamos consultando. Exactamente en el apartado *ITINERARIOS*.

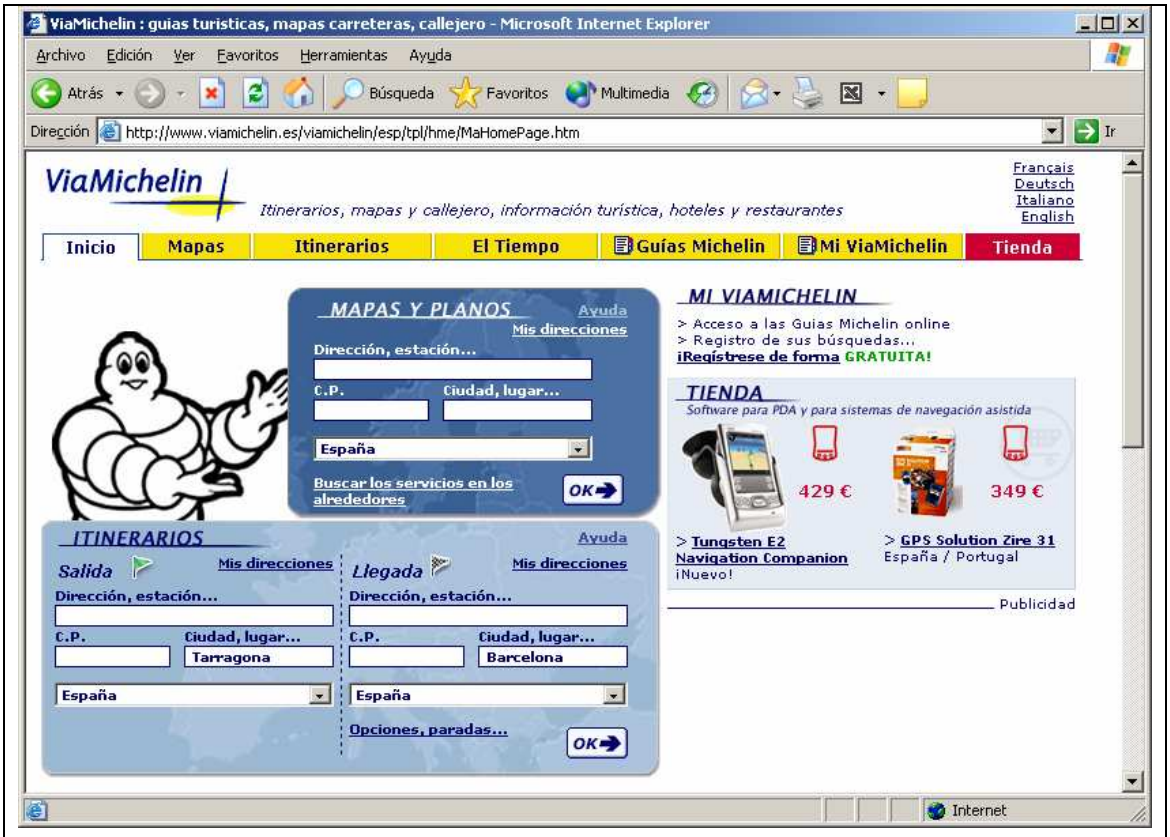


Ilustración 29: Pagina principal de ViaMichelin.es

Esta vez, sabemos que ViaMichelin.es realiza una sustitución de países por códigos de los mismos, para ello, hemos implementado un diccionario que realiza esta traducción, veámoslo en esta tabla:

```
/* Diccionario de paises y codigos usados por viaMichelin.es */
paises = new Hashtable(43);
paises.put("Alemania","240");
paises.put("Andorra","856");
paises.put("Austria","106");
paises.put("Belarus","1794");
paises.put("Belgica","311");
paises.put("Bosnia y Herzegovina","1025343");
paises.put("Bulgaria","1025340");
paises.put("Canada","1145795");
paises.put("Checa, Republica","1694");
paises.put("Croacia","1752");
paises.put("Dinamarca","1473");
paises.put("Eslovaquia","1697");
paises.put("Eslovenia","1746");
paises.put("España","844");
paises.put("Estados Unidos","1145799");
paises.put("Estonia","1860861");
paises.put("Finlandia","1792");
paises.put("Francia","1424");
paises.put("Grecia","1945835");
paises.put("Hungria","1741");
paises.put("Irlanda","919");
paises.put("Italia","612");
paises.put("Letonia","1851089");
```

```
paises.put("Liechtenstein","108");
paises.put("Lituania","1851066");
paises.put("Luxemburgo","247");
paises.put("Macedonia","1025334");
paises.put("Moldova","1025352");
paises.put("Monaco","852");
paises.put("Noruega","1574");
paises.put("Paises Bajos","285");
paises.put("Polonia","1743");
paises.put("Portugal","669");
paises.put("Reino Unido","1138");
paises.put("Rumania","1025349");
paises.put("Rusia","1851058");
paises.put("San Marino","318");
paises.put("Serbia y Montenegro","1025346");
paises.put("Suecia","1507");
paises.put("Suiza","185");
paises.put("Turquia","2059154");
paises.put("Ucrania","1749");
paises.put("Vaticano","2066810");
```

Ilustración 30: Diccionario de países y códigos de ViaMichelin.es

Obtención pagina resultados

Con el proceso anterior, emulamos el comportamiento de un navegador Web en el envío de los datos de un formulario. La página de resultados se obtiene directamente en la inmediata respuesta del servidor como en el caso de Renfe.es. Simplemente hay que abrir un flujo de entrada por donde leerla.

Interpretación y elección de resultados

Podemos obtener los campos que podemos observar en la tabla siguiente:

```
/* Buscamos el tiempo que tarda en realizar el trayecto */
indiceIni = lineaLeida.indexOf("itiTotalTime = ",indiceIni);
if(indiceIni!=-1)
{
	indiceFi = lineaLeida.indexOf(";",indiceIni+15);
	tiempo = lineaLeida.substring(indiceIni+15,indiceFi);
	tiempoInt = ((new Integer(tiempo)).intValue()) / 60; /* Lo pasamos a minutos */
}
indiceIni = lineaLeida.indexOf("document.getElementById(\"timeH\").innerHTML =
mkHour(",indiceIni);
if(indiceIni!=-1)
{
	indiceFi = lineaLeida.indexOf(")",indiceIni+52);
	distancia = lineaLeida.substring(indiceIni+52,indiceFi);
	distanciaInt = ((new Integer(distancia)).intValue()) / 35;
}
/* Buscamos las principales paradas */
indiceIni = 0;
String ciudades = "";
boolean quedanCiudades = true;
while(quedanCiudades==true)
{
	indiceIni = lineaLeida.indexOf("FDRCity",indiceIni);
	if(indiceIni!=-1)
```

```
        {
                indiceIni = lineaLeida.indexOf("\"",indiceIni)+1;
                indiceFi = lineaLeida.indexOf("\"",indiceIni);
                ciudades = ciudades + lineaLeida.substring(indiceIni,indiceFi) + "<br>";
                indiceIni = indiceFi;
        }
        else quedanCiudades = false;
}
```

Ilustración 31: Obtención de los campos necesarios

Como vemos, una vez más vamos analizando el código mediante la búsqueda de cadenas de texto que preceden, suceden e identifican unívocamente el dato al cual queremos tener acceso. En este caso solo tenemos una opción para escoger, al estar libres de horarios.

Veamos la pantalla de resultados que vemos en un navegador y de la cual hemos extraído estos datos.

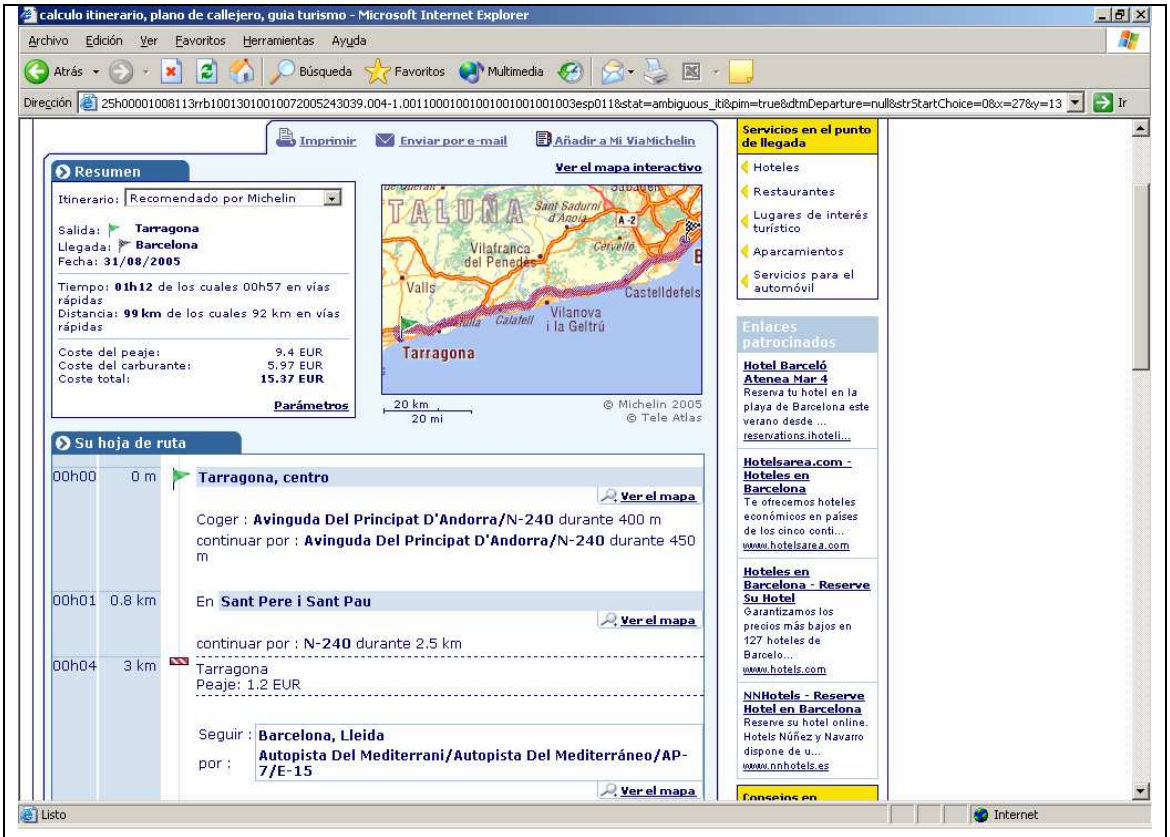


Ilustración 32: Resultados de la búsqueda

5.4.5.3 Página de vuelos LastMinute.es

Mediante LastMinute.es podemos acceder a los vuelos que se realizan a nivel mundial con las compañías aéreas más importantes de cada país. Nos devuelve una lista de alternativas, ordenadas por precio y horario.

Conexión

Para conectarnos a través de nuestro *InterenetAgenteLastMinute*, se ha implementado la clase *LastMinuteNavegadorWeb.java* que encapsula los métodos necesarios para la conexión e interpretación de la información.

Lo primero que hacemos es acceder al servicio: "www.es.lastminute.com/lmn/lfe/flights/search/search_helper.jhtml", al cual tenemos que pasar mediante el método *post* una serie de parámetros para realizar la búsqueda, estos son los que podemos ver en la siguiente tabla:

```
/* Introducimos los parametros comunes a todas las llamadas */
post.addParameter("_DARGS","%2Flmn%2Fhomepage%2Fes_ES%2Fhomepage_es
_flights_newui.jhtml");
post.addParameter("lfe_num_adult", "1");
post.addParameter("lfe_ret_time", "0001");
post.addParameter("lfe_ticket_type", "false");
post.addParameter("lfe_ret_day", "%2");
post.addParameter("lfe_ret_month", "%2");
post.addParameter("lfe_num_child", "0");
post.addParameter("lfe_num_infant", "0");
post.addParameter("lfe_best_price", "true");
post.addParameter("lfe_sub_agent", "LM2");
post.addParameter("lfe_failure_url","/lmn/lfe/flights/homepage/
homepage.jhtml?CATID=94829");
post.addParameter("lfe_success_url","/lmn/lfe/flights/results/
results.jhtml?CATID=94829");
post.addParameter("lfe_delegate_to_ts", "false");
post.addParameter("lfe_currency", "EUR");
post.addParameter("lfe_language", "ES");
post.addParameter("lfe_number_of_results", "100");
post.addParameter("new_lfe_locale", "es_ES");
post.addParameter("new_lfe_inc_pub", "true");
post.addParameter("new_lfe_inc_nett", "true");
post.addParameter("new_lfe_inc_lcc", "true");
post.addParameter("new_lfe_inc_charter", "true");
post.addParameter("new_lfe_inc_it", "true");
post.addParameter("new_lfe_inc_trains", "true");
post.addParameter("lfe_ext_search", "true");
post.addParameter("/com/lastminute/lfe/dynamo/handler/
SearchFormHandler.multiSearch","false");
post.addParameter("_D:/com/lastminute/lfe/dynamo/handler/
SearchFormHandler.multiSearch","%2");
post.addParameter("lfe_cabin_class", "X");
post.addParameter("lfe_num_senior", "0");
```

```
post.addParameter("lfe_direct_only", "false");
post.addParameter("lfe_success_tq_url",
"/lmn/lfe/flights/results/tq_results.jhtml?CATID=85");
/* Introducimos los parametros particulares de cada llamada */
post.addParameter("lfe_origin", ciudadOrigen);
post.addParameter("lfe_dep_day", diaPartida);
post.addParameter("lfe_dep_month", mesPartida);
post.addParameter("lfe_destination", ciudadDestino);
post.addParameter("lfe_dep_time", horaSalida);
```

Ilustración 33: Parámetros necesarios para realizar la consulta

En la siguiente ilustración se muestra el formulario de búsqueda de la página que estamos consultando. Exactamente en el apartado *Vuelos*.

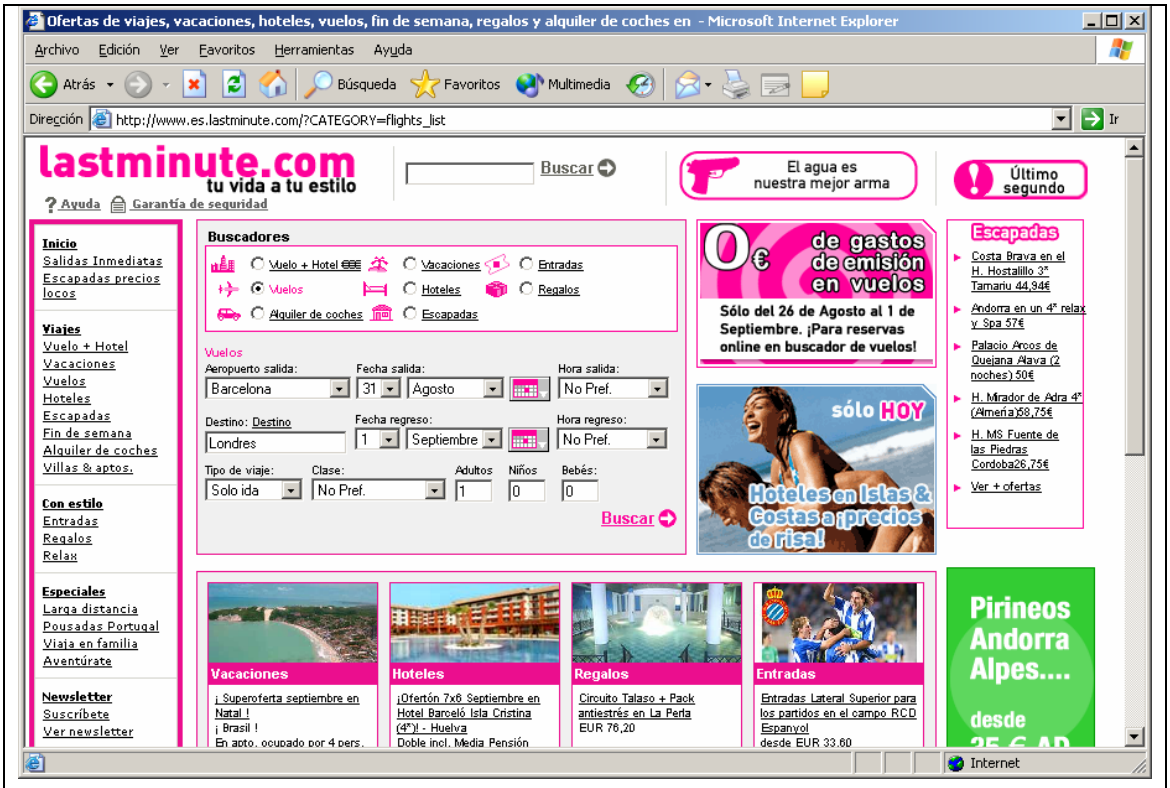


Ilustración 34: Pagina principal de LastMinute.es

Esta vez, no necesitamos diccionario alguno, las ciudades se pasan tal cual.

Obtención pagina resultados

Como en los proceso anteriores, emulamos el comportamiento de un navegador Web en el envío de los datos de un formulario. El servidor nos devuelve una página de espera y tras ella, nos redirecciona a la página de resultados final.

Para solventar este problema tenemos que leer la página intermedia, guardar el contexto de la aplicación (cookies, sesión, etc), conectar a la nueva página con el contexto proporcionado anteriormente mediante un método *get* y abrir flujo de datos de nuevo para leer lo que nos llega.

Ilustración 35: Pagina de espera de LastMinute.es

Interpretación y elección de resultados

Podemos obtener los campos que podemos observar en la tabla siguiente:

```
if(lineaLeida.indexOf("<!-- SHOW FARE START-->")!=-1)
{
        flagRecopilarInfo = true;
}
if(flagRecopilarInfo)
{
        ofertaAtratar += lineaLeida;
}
if(lineaLeida.indexOf("<!-- SHOW FARE END -->")!=-1)
{
        indiceIni = 0;
        /* Analizamos la información recopilada */
        boolean quedanVuelos = true;
        indiceIni = ofertaAtratar.indexOf("EUR ",indiceIni)+9;
        indiceFi = ofertaAtratar.indexOf("\t",indiceIni);
        String precio = ofertaAtratar.substring(indiceIni,indiceFi);
        indiceIni = indiceFi;
        while(quedanVuelos==true)
        {
```

```
                /* Buscamos la compañía de vuelo */
                indiceIni = ofertaAtratar.indexOf("alt=\"",indiceIni)+5;
                indiceFi = ofertaAtratar.indexOf("\"",indiceIni);
                String compañia = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
                /* Buscamos ciudad de origen */
                indiceIni = ofertaAtratar.indexOf("Desde: ",indiceIni)+7;
                indiceFi = ofertaAtratar.indexOf(" ",indiceIni);
                String origen = ofertaAtratar.substring(indiceIni,indiceFi);
                /* Comprobamos que nos haya devuelto correctamente el resultado, pues a
                veces nos devuelve aeropuertos equivocados */
                String origenTemp = origen.toLowerCase();
                if(origenTemp.indexOf(ciudadOrigen.toLowerCase())==-1) throw new
Exception("Aeropuerto de origen invalido: "+origen+"!="+ciudadOrigen);
                origen += " "+ofertaAtratar.substring(indiceFi+6,indiceFi+11);
                indiceIni = indiceFi;
                /* Buscamos hora salida */
                indiceIni = ofertaAtratar.indexOf("Salida ",indiceIni)+7;
                indiceFi = ofertaAtratar.indexOf(" ",indiceIni);
                String horaSalida = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
                /* Buscamos fecha salida */
                indiceIni = ofertaAtratar.indexOf(" ",indiceIni)+1;
                indiceFi = ofertaAtratar.indexOf("\t",indiceIni);
                String fechaSalida = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
                /* Buscamos ciudad destino */
                indiceIni = ofertaAtratar.indexOf("a: ",indiceIni)+3;
                indiceFi = ofertaAtratar.indexOf(" ",indiceIni);
                String destino = ofertaAtratar.substring(indiceIni,indiceFi);
                /* Comprobamos que nos haya devuelto correctamente el resultado, pues a
                veces nos devuelve aeropuertos equivocados */
                String destinoTemp = destino.toLowerCase();
                if(destinoTemp.indexOf(ciudadDestino.toLowerCase())==-1)  throw new
Exception("Aeropuerto de destino invalido: "+destino+"!="+ciudadDestino);
                destino += " "+ofertaAtratar.substring(indiceFi+6,indiceFi+11);
                indiceIni = indiceFi;
                /* Buscamos hora llegada */
                indiceIni = ofertaAtratar.indexOf("Llegada: ",indiceIni)+9;
                indiceFi = ofertaAtratar.indexOf(" ",indiceIni);
                String horaLlegada = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
                /* Buscamos fecha llegada */
                indiceIni = ofertaAtratar.indexOf(" ",indiceIni)+1;
                indiceFi = ofertaAtratar.indexOf("\t",indiceIni);
                String fechaLlegada = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
                /* Buscamos numero de vuelo */
                indiceIni = ofertaAtratar.indexOf("N° de vuelo ",indiceIni)+12;
                indiceFi = ofertaAtratar.indexOf("\t",indiceIni);
                String numVuelo = ofertaAtratar.substring(indiceIni,indiceFi);
                indiceIni = indiceFi;
```

Ilustración 36: Obtención de los campos necesarios

Como vemos, una vez más vamos analizando el código mediante la búsqueda de cadenas de texto que preceden, suceden e identifican unívocamente el dato al cual queremos tener acceso. Obtenemos información de todos los vuelos resultantes, siendo esta vez el *InternetAgente* el encargado de seleccionar la mejor opción (que será la primera pues es la más económica).

A continuación mostramos la pantalla de resultados que vemos en un navegador y de la cual hemos extraído estos datos.

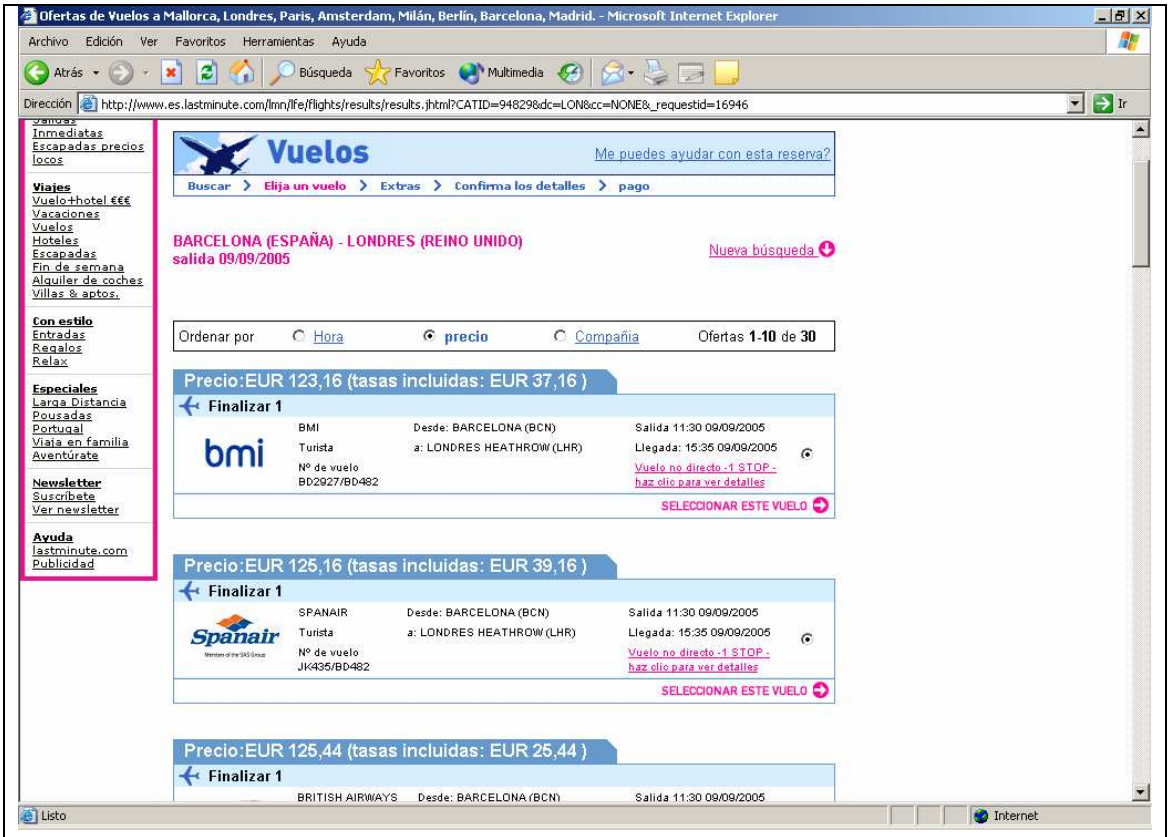


Ilustración 37: Resultados de la búsqueda

5.4.6 Páginas inaccesibles

5.4.6.1 GuiaCampsa.com

Esta metodología, a pesar de tener un amplio margen de éxito, no es infalible y, en algunos casos, deberemos ir más allá pues no obtendremos los resultados esperados tan fácilmente.

Por ejemplo, como alternativa a ViaMichelin.es (http://www.viamichelin.es) existe la conocida GuiaCampsa.com (*http://www.guiacampsa.com*). Paralelamente a la primera, intentamos acceder a esta con la misma metodología. Analizamos el código fuente de la página donde está contenido el formulario de envío de petición, de donde extraímos los campos que en él aparecían, así como también la URL de conexión, formando una cadena con el siguiente formato:

http://www.guiacampsa.com/bienvenidoalinfinito/gcampsa/ruta/calcular/itinerario.aspx?frmorigenpais=ESP&frmdestinopais=ESP&frmorigenlocalidad=tarragona&frmdestinol

ocalidad=barcelona&frmorigendireccion=&frmorigennum=&frmdestinodireccion=&frmdestinonum=&frmpaisp1=&frmlocalidadp1=&frmdireccionp1=&frmnump1=&frmpaisp2=&frmlocalidadp2=&frmdireccionp2=&frmpaisp3=&frmlocalidadp3=&frmdireccionp3=&frmnump3=&itinerario=1&evitarpeajes=0&evitarincidencias=0&estaciones=1&dia=1&vehiculo=1&consumo=&precio=

Pero, al introducir esta dirección en la barra de direcciones del navegador, recibimos otra página distinta a la que esperábamos, con otro formulario como vemos en la ilustración adjunta, en vez de los resultados que buscábamos.

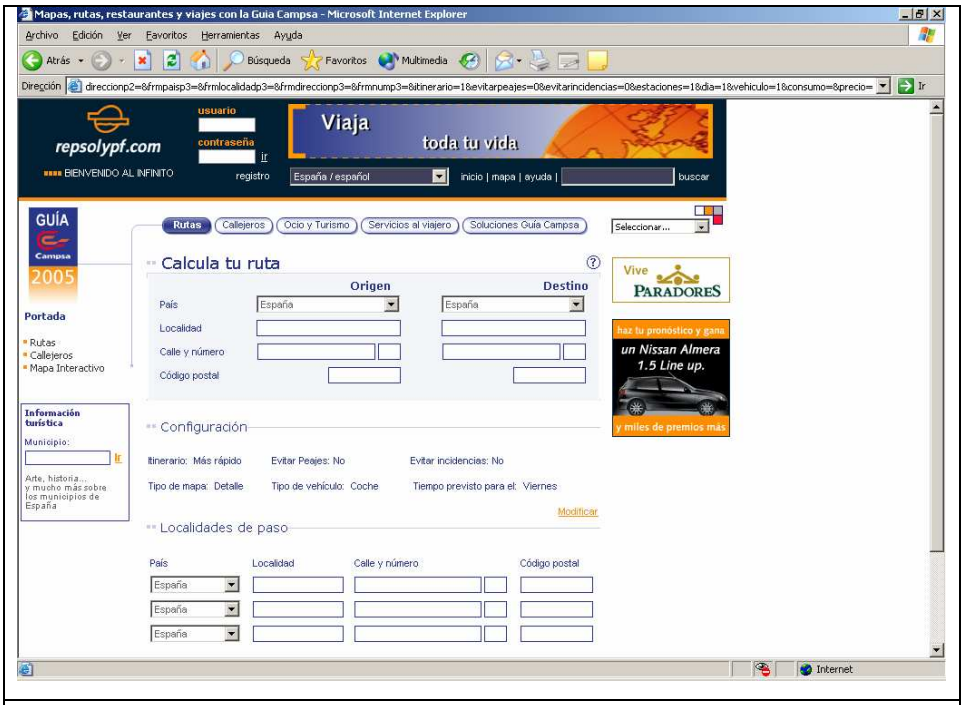


Ilustración 38: Nuevo formulario

En este caso, podríamos volver a iniciar nuestro método sobre el código de esta página, pues quizás nos diera más pistas. Aunque nosotros, teniendo la alternativa de ViaMichelin.es, decidimos descartar esta otra opción.

Algo similar nos pasó con el recurso Edreams.es (http://www.edreams.es). Aunque examinando bien el código, descubrimos que el problema residida en que había algún campo que no habíamos tenido en cuenta, por lo que los resultados no eran satisfactorios. Finalmente pudimos lograr la conexión y la consulta con el servicio,

obteniendo los resultados deseados. Esto nos hace pensar que en el caso de GuiaCampsa.com suceda algo similar.

Comentar que decidimos descartar usar el recurso de Edreams.es pues proporciona resultados muy parecidos a LastMinute.com, el cual conseguimos acceder en primer lugar. Aunque cabe destacar que Edreams.es tiene una ventaja respecto LastMinute.com; y es que proporciona vuelos de compañías de bajo coste, razón por la cual es un candidato muy serio a ser tratado por un nuevo Agente.

5.4.7 Resultados y conclusiones

Este diseño simplifica de manera notable la creación de nuevos recursos, pues *InternetAgente*Recurso permanece prácticamente igual en todos los casos, cambiando únicamente la clase del objeto de conexión, seleccionando la clase Recurso*NavegadorWeb* apropiada en cada caso.

Por otro lado, muchas de las conexiones con recursos siguen un esquema similar. Algunos son más completos, otros requieren del manejo de contextos, algunos otros necesitan varios pasos para obtener soluciones o bien realizar pequeñas traducciones, por ejemplo de nombres de ciudades o países por códigos.

Gracias a esta similitud, el código de unas y otras clases se puede aprovechar para la nueva, pues ya hemos recogido los modos de acceso más comunes en cuanto a acceso. Simplemente habría que analizar el nuevo recurso que se desea añadir para localizar el comportamiento que se debería usar en cada caso.

Todo esto conlleva un desarrollo de nuevos recursos y una modificación de los existentes ágil, rápida, eficiente y coherente.

5.5 Agentes interactuando con Bases de Datos

5.5.1 Interacción semi - local (*BDAgente*)

El comportamiento y estructura son análogos a los *InternetAgente*. De la misma manera que hemos abstraído y encapsulado los métodos necesarios en varias clases en el apartado anterior. En este podemos hacer exactamente igual, cambiando en cada caso,

la fuente de origen de datos y usando las consultas apropiadas en cada caso sobre la Base de datos.

Esta vez, es aun más sencillo debido a que ya existe un estándar sobre el que podemos trabajar. Se trata de SQL. Además, Java tiene métodos muy maduros para la interacción con Bases de Datos, por lo que podemos usar los módulos que trae ya consigo el lenguaje y que colman nuestras necesidades.

Para finalizar, comentar que la hemos denominado 'semi – local' puesto que la fuente de origen de datos puede estar en la misma computadora donde corre el Agente, o bien en otra diferente, pero, sea de una manera o de otra, usaremos siempre métodos y objetos locales. Sin uso de la tecnología CORBA como sí haremos en el siguiente apartado.

5.5.2 Interacción remota (*BuscadorAgente*)

En apartados anteriores se ha hablado sobre el acceso remoto a objetos y del uso de consultas sobre bases de datos en estas circunstancias como parte del algoritmo de Generalización – Especificación.

Sin darnos a penas cuenta, se ha introducido otra manera de acceder a fuentes de datos por parte de los Agentes. Ya hemos hablado de las ventajas que conllevaba y como se adaptaba a la filosofía de nuestro proyecto.

Gracias a CORBA y, siempre y cuando tengamos una potente interfaz que nos permita realizar cualquier tipo de consulta necesaria, estaremos ante un sistema limpio, potente, transparente y eficiente (aunque sea algo más lento que accesos locales, debido a la latencia y el retraso de los bits circulando desde el origen al destino a través de las distintas redes).

5.6 Diseño de las distintas bases de datos

Ligado estrechamente al apartado anterior, vamos a comentar las Bases de Datos a las cuales accedemos desde nuestra aplicación.

Inicialmente existen dos BD's importantes. Por un lado contamos con la encargada de gestionar el acceso al sistema, la información de los usuarios y las

preferencias de los mismos (que detallaremos en un apartado posterior). Por otro lado, encontramos la base de datos que nos permite desarrollar el algoritmo Generalización – Especificación.

A parte de estas dos, cada *BDAgente* podrá interactuar hasta con ***n*** BD's más, dependiendo de su configuración. Podrían ser totalmente diferentes y ajenas a nuestra aplicación, conociendo sólo métodos o interfaces de acceso.

5.6.1 Base de Datos de Usuarios

Es la primera BD con la que interactúa la aplicación en el flujo normal de ejecución.

Sus funciones consisten en:

- **Control de Acceso:** Tener almacenados los usuarios registrados en nuestra aplicación, lo cual es sinónimo de los usuarios que tienen acceso al sistema y pueden efectuar consultas.
- **Creación Perfiles:** Almacenar información personal de estos usuarios, a fin de obtener perfiles de los mismos para futuros estudios con *Data Mining*.
- **Adaptación a usuarios:** Almacenar las preferencias generales de cada uno de los usuarios para que no tengan que introducir cada vez todos los criterios de búsqueda y que ésta se adecue lo máximo posible a las expectativas de los mismos.

En la siguiente ilustración vemos el diagrama de Entidad Relación de Chen asociado a esta Base de datos, como se observa es realmente sencilla.

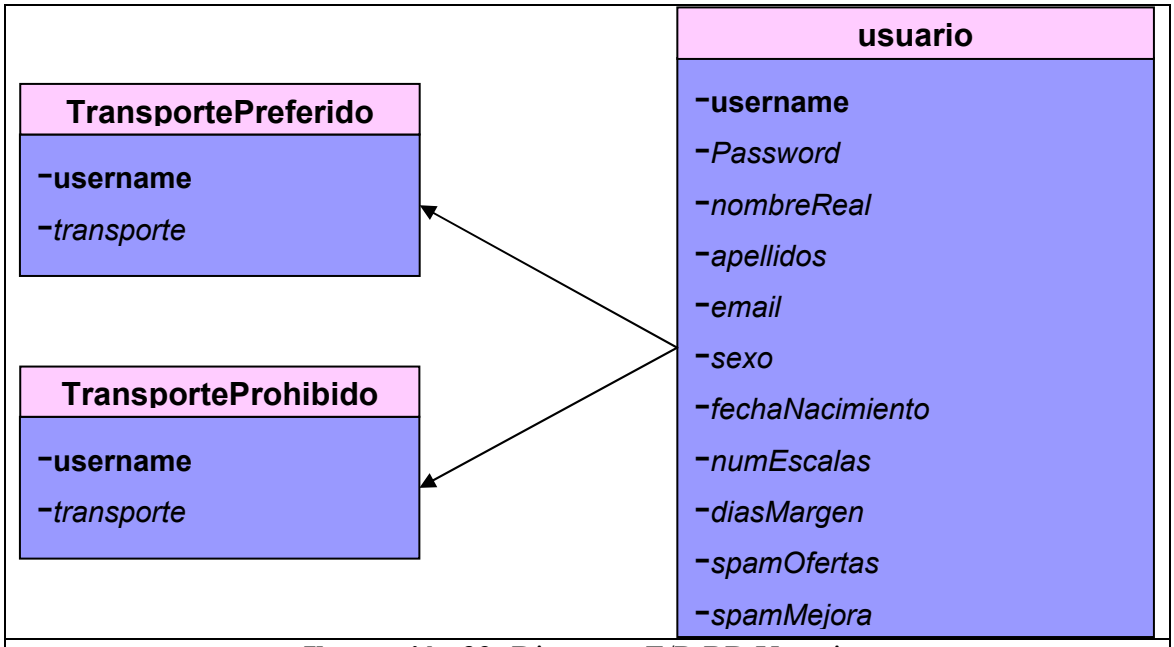


Ilustración 39: Diagrama E/R BD Usuarios

5.6.2 Base de datos de Generalización – Especificación

Esta BD es usada por parte del algoritmo que da título al apartado. El acceso a la misma se realiza a través de métodos remotos, que proporcionan una interfaz para obtener los resultados deseados.

La funcionalidad de esta Base de Datos es permitir englobar jerárquicamente todas las localidades existentes. Nuestra aplicación trabaja a nivel nacional, pues hemos introducidos la totalidad de localidades censadas en España, aunque los datos pueden ampliarse para trabajar a nivel mundial. El diseño está preparado para ello, simplemente hay que introducir la información del resto de países, de manera jerarquizada, como lo hemos hecho con las poblaciones de España mediante un Script, que pueden encontrarse en los anexos de esta documentación.

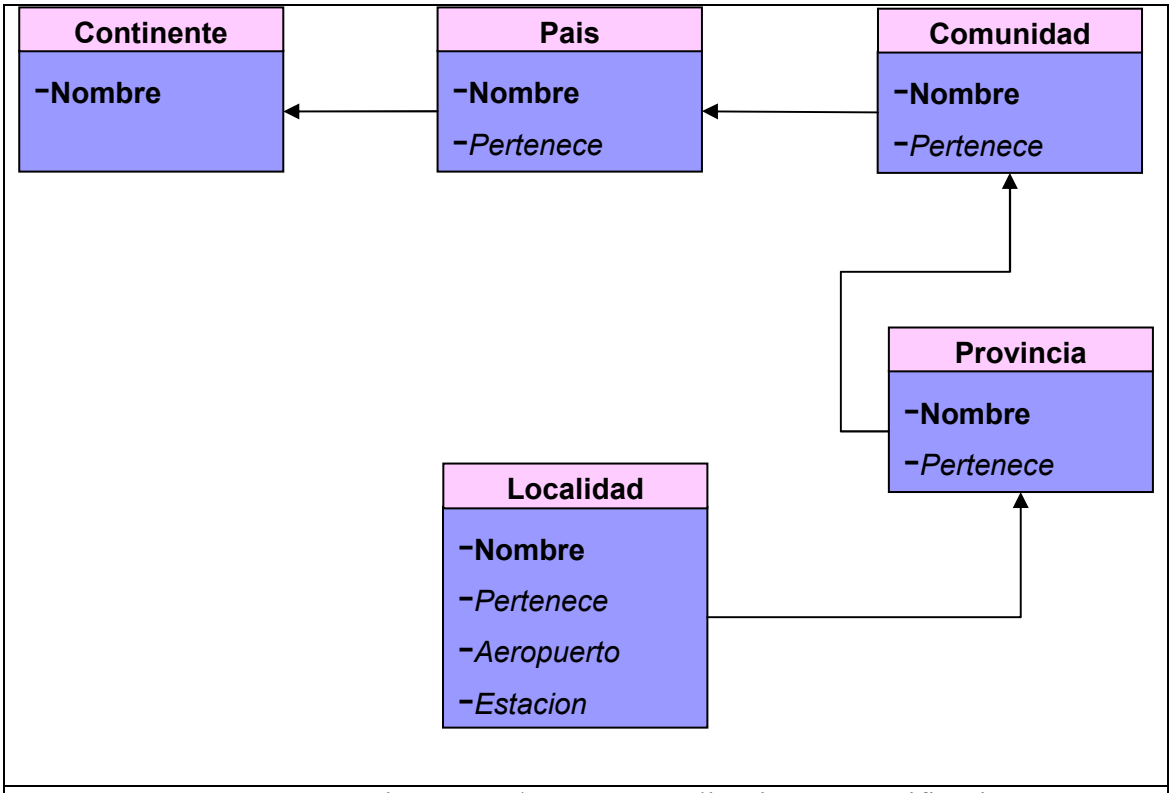


Ilustración 40: Diagrama E/R BD Generalización – Especificación

5.7 Diseño de la ontología

Esta sección está dedicada a la explicación de la ontología usada para comunicar los distintos Agentes que entran en juego en el transcurso de la ejecución de la aplicación.

En la siguiente ilustración vemos los distintos conceptos (o frames) que componen nuestra ontología, y los tipos simples que quedan incluidos dentro de cada uno de ellos, denominados slots.

Las flechas que salen del recuadro '*Travel (Dominio)*' indican relación jerárquica, mientras que el resto indican que un determinado slot tiene como tipo un valor determinado dentro de un conjunto finito (como es el caso de los Medios de transporte) o bien como tipo tiene un elemento de otro frame.

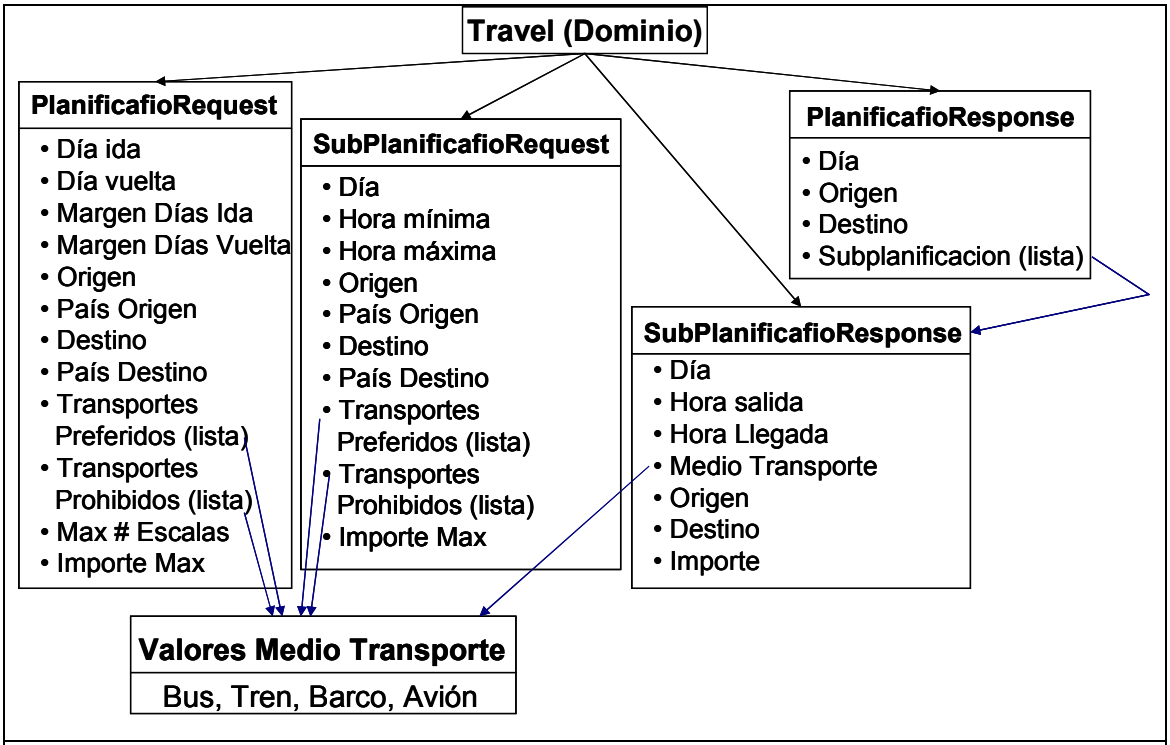


Ilustración 41: Conceptos de la Ontología

5.7.1 Uso de los frames

Cada uno de estos conceptos tiene un uso predefinido y que se usa en una fase u otra de la planificación. En este diagrama vemos cuándo se usa cada uno de ellos.

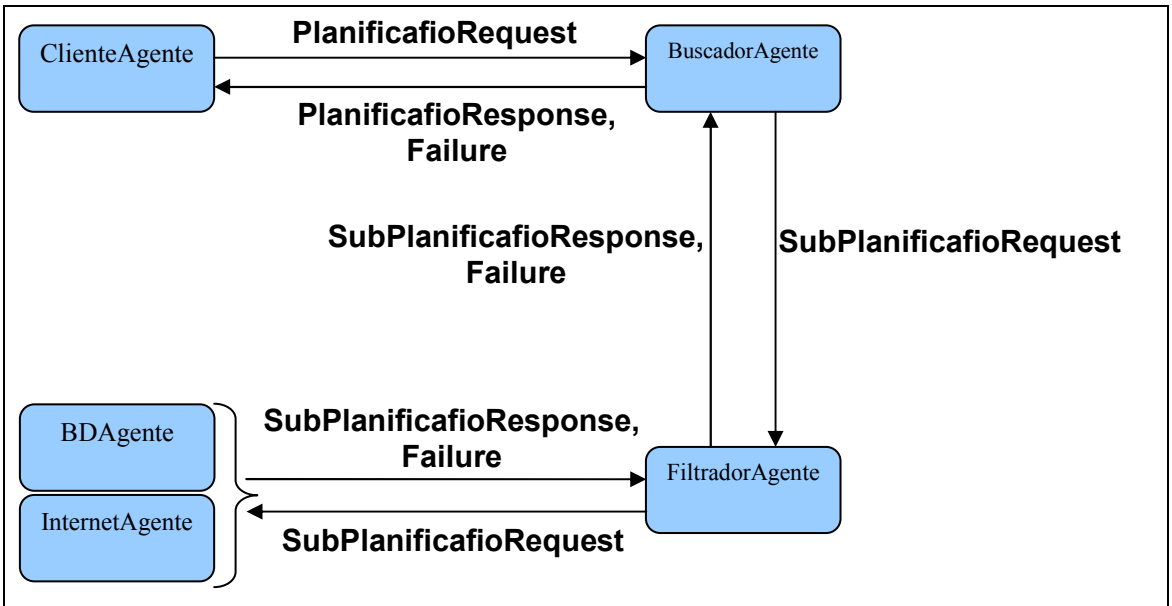


Ilustración 42: Intercambio de mensajes entre Agentes

ClienteAgente es el encargado de empezar la cadena de mensajes. Para ello se pone en contacto con *BuscadorAgente*, enviándole un mensaje de tipo *PlanificacioRequest*. *BuscadorAgente*, descompone este mensaje según sea necesario, si sólo necesita viaje de ida, o de ida y vuelta, o bien, según el desarrollo de la respuesta, ve que necesita realizar subtrayectos auxiliares no contemplados en la petición inicial y empieza a realizar peticiones de tipo *SubPlanificacioRequest* a *FiltradorAgente* (e.g. Se desea ir de Tarragona a Berlín, al no encontrar camino directo, tendrá que hacer varias descomposiciones. El resultado será Tarragona → Barcelona y Barcelona → Berlín).

FiltradorAgente no modifica la petición recibida, pero prioriza unas a otras y determina cuál será la respuesta óptima, después de realizar las investigaciones oportunas.

Finalmente, *BDAgente* e *InternetAgente*, los que finalizan el camino de los mensajes, descomponen los mensajes recibidos, los interpretan y buscan soluciones para satisfacer la petición. Entonces se inicia el viaje de vuelta. Donde las peticiones *(Requests*) pasan a ser respuestas (*Responses*).

FiltradorAgente recibe las respuestas emitidas por los distintos BD e *InternetAgente* en forma de *SubPlanificacioResponse*, las filtras, selecciona la mejor y la devuelve hacia *BuscadorAgente*, quien la agrega a la *PlanificacioResponse* que está preparando y que no devolverá a *ClienteAgente* hasta que esté totalmente completa, o haya agotado los caminos para solucionar el problema. Para ello, quizás tuviera que reiniciar la transmisión de nuevo, repitiendo el proceso desde este punto, para completar los caminos que hayan podido quedar cortados.

5.7.2 Acciones

Todo concepto tiene asociada una determinada acción, que puede ser realizada por los Agentes. En el siguiente listado, vemos como se corresponden unas a otras:

- **RealizarPlanificacioRequest**
  - *PlanificafioRequest*
- **RealizarSubplanificacioRequest**
  - *SubPlanificafioRequest*
- **TornarSubPlanificacioResponse**

- *SubPlanificafioResponse*

- **TornarPlanificacioResponse**
  - *PlanificafioResponse*

5.8 Preferencias de Usuarios

Cada vez se hace más importante la diferenciación. En un mundo tan bombardeado de información y posibilidades, el conocer al cliente a quien deseamos ofrecer un servicio resulta determinante en el éxito de los nuevos productos.

Es por ello, que para ceñirnos a las expectativas de los mismos, debemos conocerlos, debemos saber cuáles son sus predilecciones, gustos, afinidades y también que es aquello que no le gusta o detesta. Así podremos seguir conociéndoles, ofrecerles productos adaptados a ellos y fidelizar su compromiso con nosotros.

Nuestra aplicación permite la definición de una serie de parámetros que creemos interesantes para poder adaptar las búsquedas al perfil del usuario.

Estas preferencias se recogen en el momento del registro del usuario, pero pueden modificarse en cualquier momento, tanto temporal, como 'permanentemente' con el caso de uso apropiado.

Captamos tanto información personal, como información puramente funcional destinada a parametrizar la búsqueda. En el apartado destinado a la Guía de Usuario, se enumeran estos datos que recopilamos.

6. Guías

En este apartado se muestra, por una parte, como el usuario debe usar la aplicación y por otra como el administrador del sistema debe inicializarlo y mantenerlo.

6.1 Guía de Usuario

6.1.1 Pantalla principal

Con lo primero que se encuentra el usuario una vez abre un Navegador y accede a la aplicación, es con una pantalla de bienvenida, dividida en dos partes. Por un lado encontramos un frame en la parte superior, donde se encuentra un menú con todos los accesos a las diferentes funcionalidades. Debajo de este menú se encuentra el frame principal. Encargado de recopilar los datos necesarios en cada caso y de informar al usuario sobre el resultado de las operaciones realizadas.

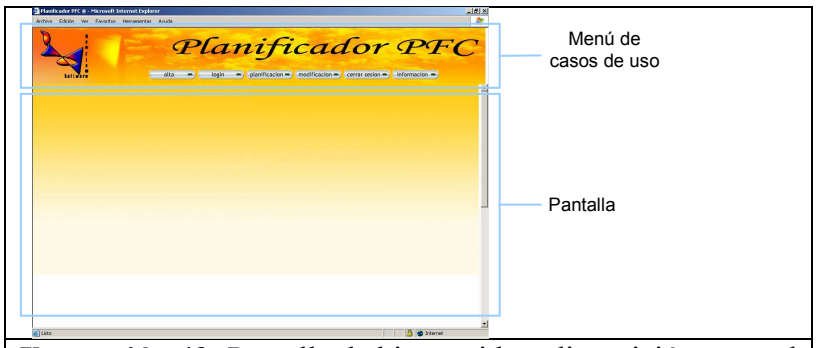


Ilustración 43: Pantalla de bienvenida y disposición general

6.1.2 Alta de usuario

La primera cosa que todo usuario ha de hacer la primera vez que accede a la aplicación es registrarse. Es decir darse de alta como usuario, adquiriendo de esta manera los privilegios necesarios para poder usar el resto de funciones. Para ello ha de

pulsar en el menú sobre el botón ‘Alta’ y rellenar y enviar el formulario que aparece en el frame principal. A partir de entonces, podrá pasar al siguiente apartado.

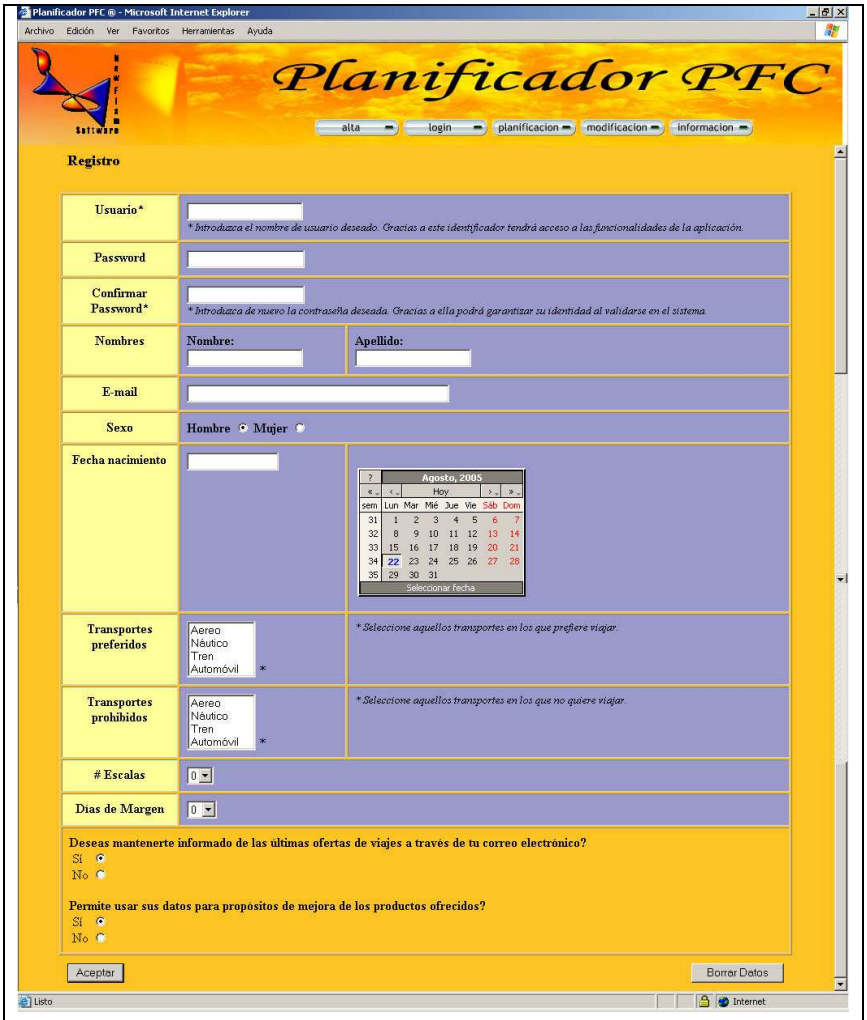


Ilustración 45: Pantalla Alta Usuario

Los campos a rellenar serían los siguientes:

- **Usuario:** Se especificará un nombre de usuario, que será exclusivo.
- **Password:** Se introduce una palabra de paso para poder acceder al sistema de manera segura.
- **Confirmar Password:** Habrá que repetir el valor del campo anterior para que no hayan errores en el password.
- **Nombre:** Nombre real del usuario.
- **Apellido:** Apellidos del usuario.
- **E-mail:** Correo electrónico del usuario.
- **Sexo:** Se podrá escoger entre *hombre* y *mujer*.

- **Fecha Nacimiento:** Se especificará la fecha de nacimiento del usuario. Se ha de usar el calendario situado a la derecha para escogerla.
- **Transportes Preferidos:** Se escogerán los transportes que el usuario considere mejores para él. De esta manera se harán las búsquedas priorizando estos medios. Manteniendo presionada la tecla control (Ctrl), se podrá realizar una selección múltiple.
- **Transportes Prohibidos:** Se escogerán los transportes que el usuario considere inapropiados para él. De esta manera se harán las búsquedas descartando estos medios. Manteniendo presionada la tecla control (Ctrl), se podrá realizar una selección múltiple.
- **# Escalas:** Se indica el número de veces que se puede cambiar de transporte en una planificación. Es decir, si escogemos '2', no podrá hacer un viaje que primero realice trayecto en autobús, luego en tren y finalmente en avión, pues nos pasamos de uno. Tampoco se podrían hacer dos viajes en autobús y luego otro en tren, pues nos seguiríamos pasando de uno. En cambio si se podrían hacer dos en avión, dos en tren, uno en barco y otro por carretera, etc
- **Días de Margen:** Se escoge el margen de días que podemos demorar o anticipar el viaje. Esto es útil cuando para la fecha exacta en que realizamos la consulta, no hay servicio disponible.
- **Permisos en cuanto a manejo de información:** Se pregunta al usuario si permite que se le envíen correos electrónicos de información sobre mejoras, nuevas propuestas, productos con destinos interesantes. También se pregunta si permite que sus datos sean base de estudio para realizar, por ejemplo, operaciones de Data Mining, ajustando los productos ofrecidos a sus gustos personales.

El sistema comprobará, en el lado del cliente mediante JavaScript, que hayamos rellenado todos los campos de texto, pues es información imprescindible para completar el proceso de alta. También comprueba que el valor de *Password* y de *Confirmar Password* sea el mismo.

6.1.3 Login

Una vez hayamos hecho el registro, y cada vez que accedamos de nuevo a la aplicación, deberemos pasar por esta pantalla de autenticación para poder usar el resto de funcionalidades.

Simplemente deberemos introducir correctamente nuestro nombre de usuario (*User Name*) y nuestra palabra de paso (*Password*).

Ilustración 46: Pantalla de autenticación

El sistema comprobará, en el lado del cliente mediante JavaScript, antes de enviar el formulario que los dos campos estén introducidos. De lo contrario nos avisará del error y nos invitará a solucionarlo antes de enviarlo.

6.1.4 Planificación

Es la piedra angular de nuestra aplicación. La función para la cual está concebida y, por tanto, la pantalla más importante.

Tras especificar una serie de parámetros que se han de rellenar en un formulario que se nos es presentado, se lanzará una petición al sistema para que empiece a trabajar en la resolución de la misma. Tras unos instantes, se presentarán los resultados en el frame principal.

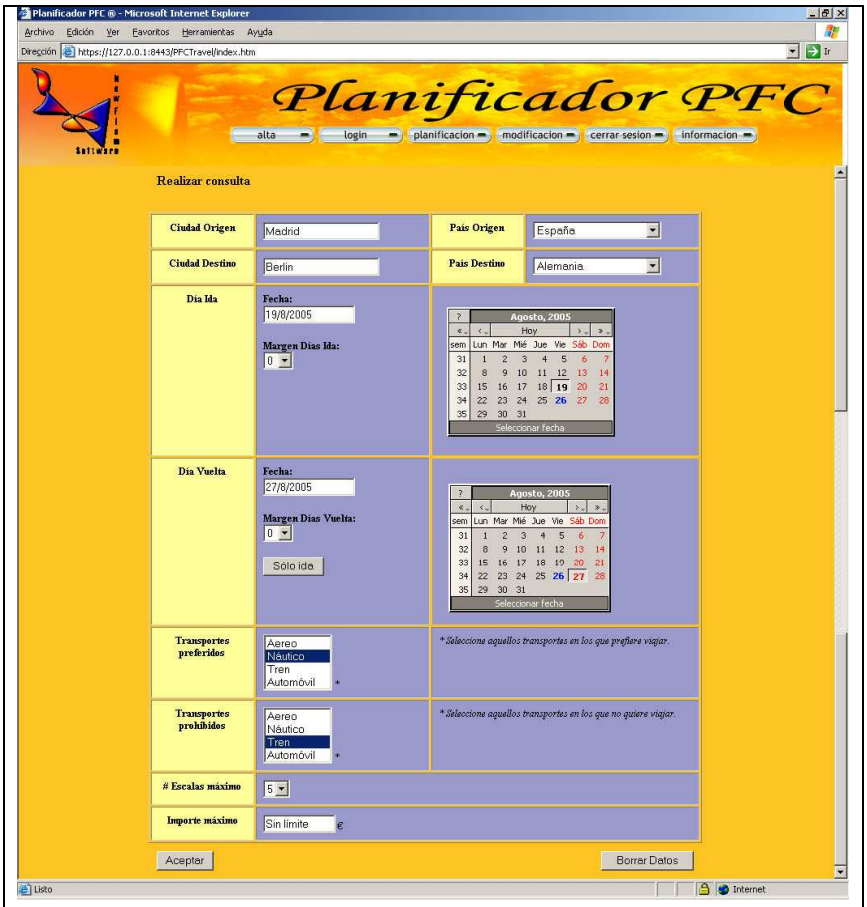


Ilustración 47: Pantalla Planificación

Los campos a rellenar en el formulario son los siguientes:

- **Ciudad Origen:** Nombre de localidad (ciudad, pueblo, urbanización…) desde donde queremos iniciar el viaje.
- **País Origen:** País al que pertenece la ciudad origen.
- **Ciudad Destino:** Nombre de localidad (ciudad, pueblo, urbanización…) donde queremos finalizar el viaje.
- **País Destino:** País al que pertenece la ciudad destino.
- **Día Ida:** Información relativa al viaje de ida.
    - **Fecha:** Fecha de partida.
    - ***Margen Días Ida:*** Margen de días que podemos demorar o anticipar la partida.
- **Día Vuelta:** Información relativa al viaje de ida.

    - **Fecha:** Fecha de regreso. Sí sólo estamos interesados en viaje de ida, dejaremos este campo en blanco.
    - ***Margen Días Vuelta:*** Margen de días que podemos demorar o anticipar la vuelta.
    - ***Sólo Ida:*** Este botón sirve para indicar que sólo queremos viaje de ida, en caso que hayamos introducido previamente algún valor y queramos volver atrás en nuestra decisión.
- ***Transportes preferidos:*** Transportes que el usuario considere mejores para él. De esta manera se harán las búsquedas priorizando estos medios. Manteniendo presionada la tecla control (Ctrl), se podrá realizar una selección múltiple.
- ***Transportes prohibidos:*** Transportes que el usuario considere inapropiados para él. De esta manera se harán las búsquedas descartando estos medios. Manteniendo presionada la tecla control (Ctrl), se podrá realizar una selección múltiple.
- **# *Escalas máximo:*** Se indica el número de veces que se puede cambiar de transporte en un trayecto. Es decir, si escogemos '2', no podrá hacer un viaje que primero realice trayecto en autobús, luego en tren y finalmente en avión, pues nos pasamos de uno.
- **Importe Máximo:** Presupuesto que estamos dispuestos a gastar como máximo. Cualquier propuesta que supere esta cantidad será descartada. Se puede dejar en blanco o en valor por defecto ('Sin límite') para indicar que nuestro presupuesto es ilimitado.

El sistema comprobará, en el lado del cliente mediante JavaScript, que, por lo menos, hayamos introducido el valor de Ciudad Origen, Ciudad Destino y Fecha de viaje de Ida, como mínimo. Pues son los parámetros imprescindibles para realizar la consulta.

En caso de viaje de ida y vuelta, el sistema comprobará que la fecha de vuelta sea posterior a la de ida.

Nota: *Los campos señalados en cursiva, tendrán por valor por defecto aquellos que tenemos almacenados en preferencias de usuario, pudiéndose cambiar por otros de manera puntual, si así lo desea el usuario.*

6.1.5 Modificación Usuario

Esta opción permite modificar todos los datos relativos al usuario, tanto información personal como sus preferencias. Al acceder al formulario, aparecerán por defecto los valores actuales de cada campo; pudiéndose modificar al antojo del usuario.

Destacar que es la misma información que se proporciona en el proceso de Alta de Usuario. El formulario, tiene el mismo comportamiento que en aquel caso. Para mayor información, referirse al apartado mencionado.

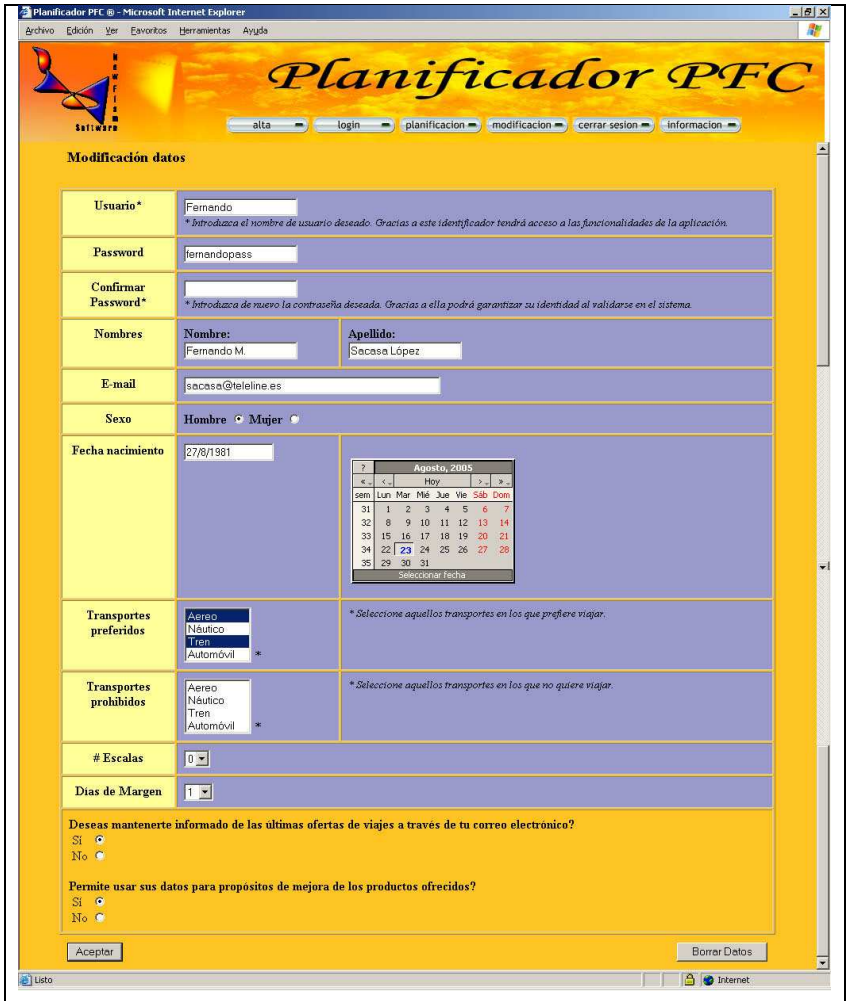


Ilustración 48: Pantalla Modificación Usuario

El sistema comprobará, en el lado del cliente mediante JavaScript, que hayamos rellenado todos los campos de texto, pues es información imprescindible para completar el proceso de alta. También comprueba que el valor de *Password* y de *Confirmar Password* sea el mismo.

6.1.6 Cerrar sesión

Como medida de seguridad, es conveniente antes de abandonar la aplicación, invalidar la sesión activa. De esta manera impedimos que algún actor malintencionado use nuestra cuenta detrás de nosotros.

Simplemente se presiona sobre la opción en el menú principal y se cerrará de manera inmediata. Una vez se muestra el resultado de la operación, nos indica que para acceder de nuevo a la aplicación debemos realizar Login de nuevo, y nos ofrece un acceso directo para ello.

Ilustración 49: Cerrar Sesión

6.1.7 Información

Es un apartado de información general sobre la aplicación, como autor de la misma, su propósito, etc. También quedan reflejados todos los links de la bibliografía, para que puedan ser consultados de manera rápida y sencilla.

Ilustración 50: Pantalla de información

6.1.8 Pantallas de respuesta

Aquí se engloban todas las pantallas que genera la aplicación tras completar, o intentar completar cualquier tipo de operación, desde el estado de registro o autenticación hasta los resultados producidos tras completar una planificación satisfactoriamente. También quedan incluidas las pantallas de error que pueden producirse en algún momento.

Todas estas pantallas son meramente informativas, la única interacción con ellas se limita, y sólo en algunos casos determinados, a links que enlazan con posibles soluciones.

Ilustración 51: Pantalla de respuesta con link incluido

6.2 Guía de Administrador

6.2.1 Guía de Arranque

Detallamos los pasos a seguir para lanzar de manera correcta la aplicación:

1. **Lanzar servicio MySQL WinMySQLadmin 1.3:** Con esto conseguimos tener acceso a las bases de datos del sistema.
   Anteriormente se habrán tenido que incluir las mismas en '*Panel de Control→Herramientas Administrativas→Orígenes de datos (ODBC)*', de Windows.
   Otro requisito previo, consiste en haber creado las bases de datos con los scripts adecuados y haber rellenado la información de la base de datos de CORBA a través de la aplicación Java encargada de esto.

2. **Lanzar servicio ORB daemon:** Con lo que conseguimos habilitar las funcionalidades CORBA genéricas.
   Se puede hacer desde Eclipse, en el menú run, o bien ejecutando desde consola el comando orb.exe incluido en la distribución de desarrollo J2SDK (*e.g. D:\Programacion\j2sdk1.4.2_02\bin\orbd.exe).*

3. **Lanzar servidor CORBA CorbaBDServer.java:** Con lo que habilitamos nuestras funcionalidades CORBA usadas en la aplicación.

4. **Lanzar servidor de aplicaciones Apache Tomcat:** Así posibilitamos el acceso a la aplicación Web.
   Podemos acudir al archivo de configuración de Tomcat incluido en la distribución de la aplicación para saber qué modificaciones se deberá hacer sobre el fichero de configuración del servidor para que funcione correctamente. Sencillamente consiste en la creación de un contexto de trabajo y acceso.
   Notar que la primera vez deberemos instalar la aplicación Web para habilitar la interfaz de usuario, para ello copiaremos la carpeta TravelPFC (incluida en el CD de la distribución) dentro del directorio '..\Apache Tomcat 4.0\webapps'.

Por otro lado, deberemos instalar los .JAR necesarios para la ejecución, que encontramos en el CD, dentro de '..\Apache Tomcat 4.0\common\lib'.

5. **Lanzar la plataforma SMA con los Agentes básicos:** Es la plataforma virtual donde nuestros Agentes creados dinámicamente entrarán para interactuar con el resto. Inicialmente se pueden crear un número determinado de Agentes de cada tipo (*BuscadorAgente*, *FiltradorAgente*, *InternetAgente* y *BDAgente*) dependiendo con la carga que queramos que soporte el sistema. A más Agentes, más peticiones simultáneas podremos realizar.

6.2.2 Guía de mantenimiento

Para el mantenimiento y mejora de la aplicación se pueden usar las mismas herramientas usadas para el desarrollo de la misma; a saber:

1. **Protege 2.1 beta:** Para la ontología.
2. **Eclipse:** Existen 2 proyectos creados, uno que recoge los Agentes y sus ontologías y otro que recoge la parte de CORBA.
3. **MySQL:** Se pueden mantener las Bases de Datos con las utilidades que nos proporciona.
4. **Apache Tomcat:** Cualquier modificación en algún JSP, será compilado automáticamente por Tomcat cuando lo coloquemos en la carpeta de la aplicación.
5. **Macromedia DreamWeaver:** Pudiéndose sustituir por cualquier otro editor de páginas Web que nos facilite el trabajo de programar directamente en HTML.

7. Juego de Pruebas

En este capítulo realizaremos algunos juegos de pruebas con lo que comprobaremos el buen funcionamiento del sistema y veremos un ejemplo de cómo operar con la aplicación.

7.1 Login y Alta

Si intentamos hacer login sin saber nuestro nombre de usuario y password. El sistema nos avisa que es un acceso inválido.

Ilustración 52: Proceso fallido de Login

Podemos seguir el link que nos sugiere el mensaje, para registrarnos, o bien pulsar directamente sobre la opción *Alta Usuario* del menú. Como vemos el proceso es totalmente guiado.

En este punto, si intentamos acceder a cualquier otra funcionalidad que no sea Información o Alta Usuario, el sistema nos lo impedirá. Probemos entrar en Modificación.

Ilustración 53: Acceso sin permiso a secciones protegidas

Nos registramos, para obtener cuenta de usuario.

Ilustración 54: Proceso Alta de Usuario

Realizamos de nuevo la autenticación y vemos como ahora sí podemos acceder.

Ilustración 55: Proceso de Login correcto

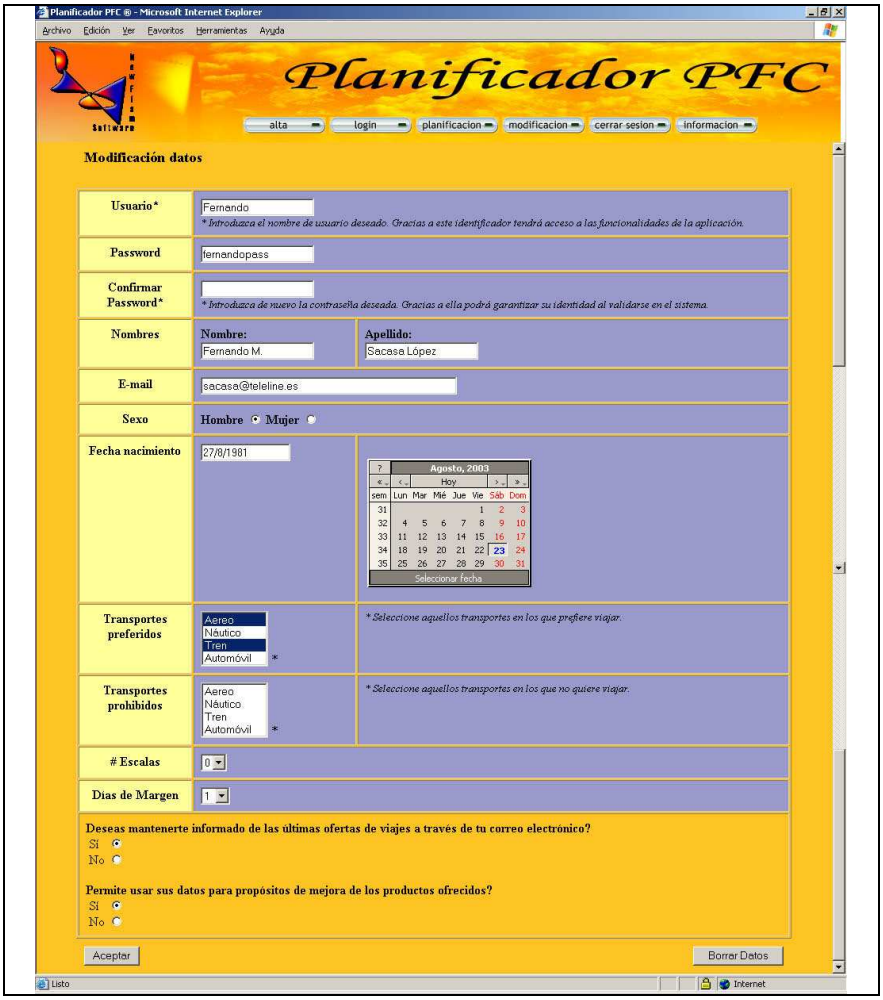
Ahora tenemos acceso a todas las funcionalidades restantes.

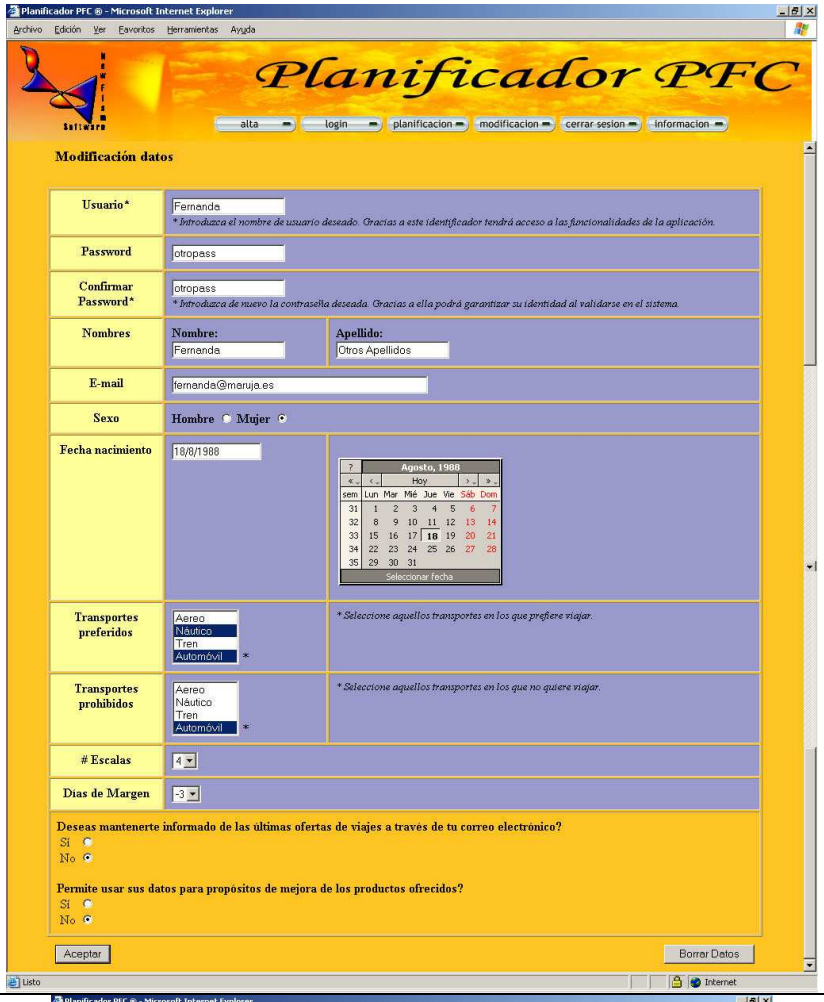
7.2 Modificación Usuario

En la siguiente ilustración vemos los distintos pasos que son necesarios para completar la modificación de los datos del usuario.

Lo primero que haremos es acceder a la opción *modificación* del menú principal. Se nos abre un formulario con los datos de nuestro usuario ya cargados. El siguiente paso, sería modificar los campos deseados. Nosotros, cambiaremos, a modo de ejemplo, todos los datos.

Tras ello, el sistema nos indica si se ha realizado correctamente la modificación. Vemos en la ilustración, que ha sido un éxito. Por último, el sistema nos cambia de identidad la sesión si es necesario. En este caso, ya no somos Fernando, ahora ha cambiado a Fernanda como usuario, por tanto se adecua a la modificación de manera automática. Así lo comprobamos al volver al menú de modificación, como esta vez retoma los nuevos datos. Observamos que no es necesario cerrar y abrir la sesión, lo cual es muy cómodo para el usuario.




Planificador PFC ® - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Planificador PFC
alta
login
planificacion
modificacion
cerrar sesion
informacion
Modificación datos
Usuario*
Fernanda
* Introduzca el nombre de usuario deseado. Gracias a este identificador tendrá acceso a las funcionalidades de la aplicación.
Password
otropass
Confirmar Password*
otropass
* Introduzca de nuevo la contraseña deseada. Gracias a ella podrá garantizar su identidad al validarse en el sistema.
Nombres
Nombre:
Fernanda
Apellido:
Otros Apellidos
E-mail
fernanda@maruja.es
Sexo
Hombre
Mujer
Fecha nacimiento
18/8/1988
Agosto, 1988
Hoy
Seleccionar fecha
Transportes preferidos
Aereo
Nautico
Tren
Automóvil
* Seleccione aquellos transportes en los que prefiere viajar.
Transportes prohibidos
Aereo
Nautico
Tren
Automóvil
* Seleccione aquellos transportes en los que no quiere viajar.
# Escalas
Dias de Margen
Deseas mantenerte informado de las ultimas ofertas de viajes a través de tu correo electrónico?
Si
No
Permite usar sus datos para propósitos de mejora de los productos ofrecidos?
Si
No
Aceptar
Borrar Datos
Listo
Internet


Planificador PFC ® - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Planificador PFC
alta
login
planificacion
modificacion
cerrar sesion
informacion
Modificacion realizada.
Usuario modificado correctamente
Transporte Nautico añadido como preferido
Transporte Automovil añadido como preferido
Transporte Automovil añadido como prohibido
Recuaerde que pueden haber cambiado los parametros de login
Listo
Internet

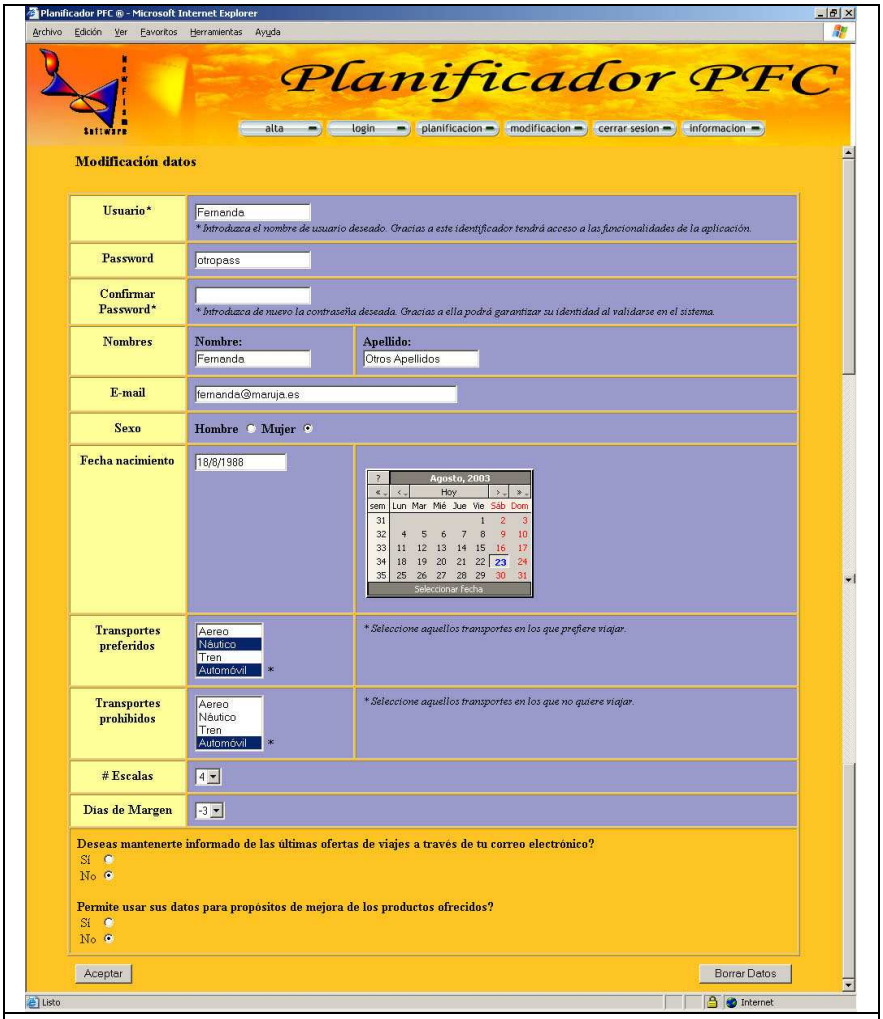


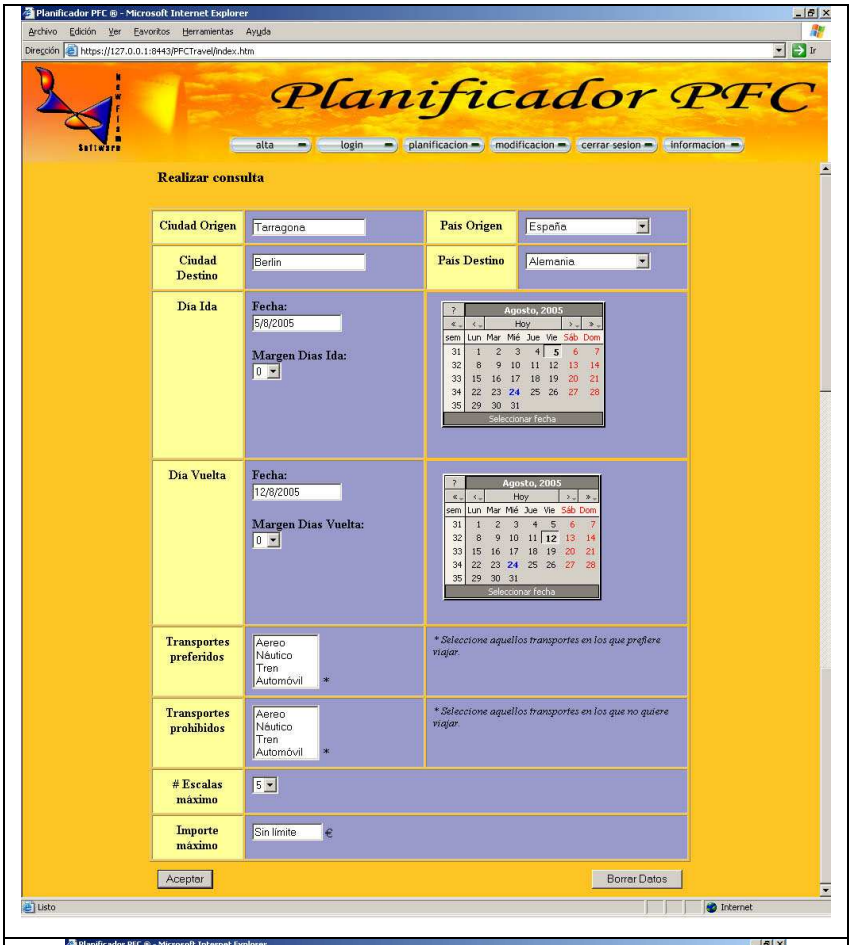
Ilustración 56: Proceso de modificación de datos del usuario

7.3 Petición de planificación

Veamos como reacciona el sistema ante peticiones del usuario. Probaremos varias combinaciones de parámetros para comprobar como éstos influyen de manera sensible en los resultados obtenidos.

7.3.1 Combinaciones no encontradas

Buscaremos la ruta Tarragona ←→ Berlin ida y vuelta para unas determinadas fechas, dejando el resto de parámetros por defecto.


Planificador PFC ® - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección https://127.0.0.1:8443/PFCTravel/index.htm
Planificador PFC
alta
login
planificacion
modificacion
cerrar sesion
informacion
Realizar consulta
Ciudad Origen
Tarragona
Pais Origen
España
Ciudad Destino
Berlin
Pais Destino
Alemania
Dia Ida
Fecha:
5/8/2005
Margen Dias Ida:
0
Agosto, 2005
Seleccionar fecha
Dia Vuelta
Fecha:
12/8/2005
Margen Dias Vuelta:
0
Agosto, 2005
Seleccionar fecha
Transportes preferidos
Aereo
Náutico
Tren
Automóvil
*Seleccione aquellos transportes en los que prefiere viajar.
Transportes prohibidos
Aereo
Náutico
Tren
Automóvil
*Seleccione aquellos transportes en los que no quiere viajar.
Escalas máximo
5
Importe máximo
Sin limite
€
Aceptar
Borrar Datos
Listo
Internet


Planificador PFC ® - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Planificador PFC
alta
login
planificacion
modificacion
cerrar sesion
informacion
Petición realizada.
Aguarde mientras los Agentes de nuestro SMA determinan cuál es la mejor opción a su consulta...
Listo
Internet
Inicio
imagenes
Planificador ...
Tomcat
Acceso direc...
Java - Corba...
Planificado...
Adobe Photo...
16:29

Ilustración 57: Proceso de Planificación inválido

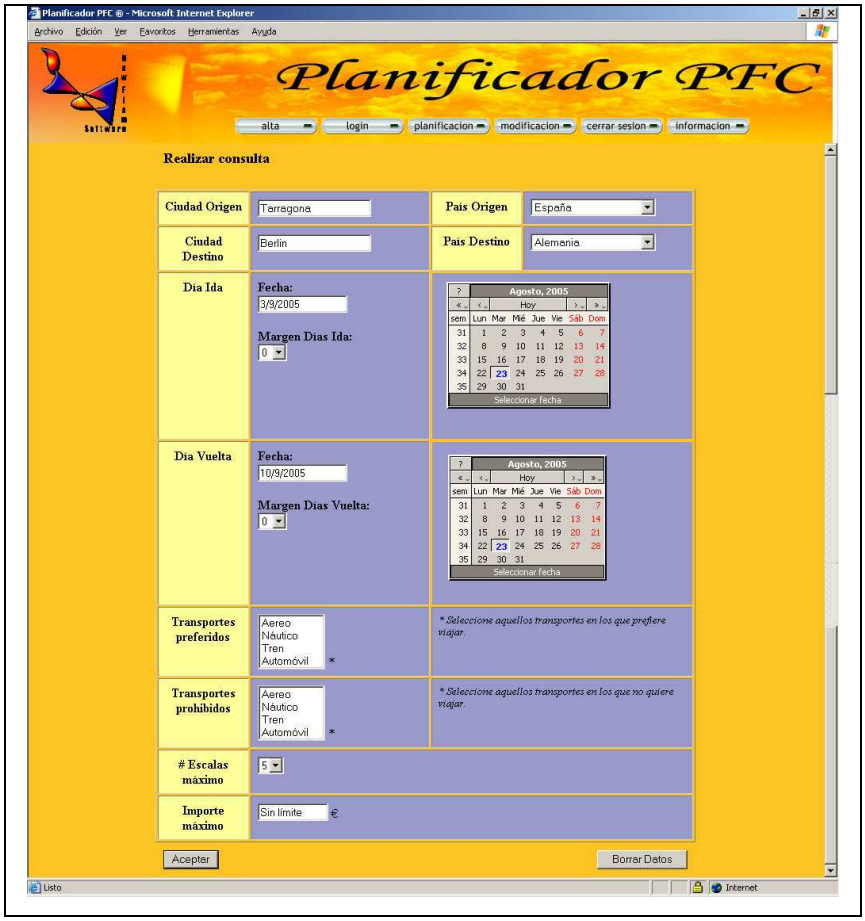
En este primer ejemplo, observamos como, tras unos instantes de espera, mientras los Agentes buscan soluciones compatibles entre sí, se muestra un mensaje de error indicando que no se han encontrado alternativas para las fechas escogidas.

7.3.2 Viaje Ida y Vuelta, nivel 2

Probemos modificando los días de partida y de regreso, el resto de campos permanecen como en el ejemplo anterior.

Esta vez sí se presentan los resultados. Como hemos dicho, en este caso se ha escogido tanto viaje de ida como de vuelta. Se detallan cada uno por separado, englobándolos después en uno solo, aunando ambos.

Como se observa en los resultados, el algoritmo de Generalización – Especificación ha llegado a nivel 2 (de comunidad autónoma), pues pasamos por Barcelona, perteneciente a una provincia distinta de Tarragona, pero de la misma comunidad autónoma, Cataluña.


Planificador PFC ® - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Planificador PFC
alta
login
planificacion
modificacion
cerrar sesion
informacion
Realizar consulta
Ciudad Origen
Terragona
Pais Origen
España
Ciudad Destino
Berlin
Pais Destino
Alemania
Dia Ida
Fecha:
3/9/2005
Margen Dias Ida:
0
Agosto, 2005
Hoy
Seleccionar fecha
Dia Vuelta
Fecha:
10/9/2005
Margen Dias Vuelta:
0
Transportes preferidos
Aereo
Náutico
Tren
Automóvil
* Seleccione aquellos transportes en los que prefiere viajar.
Transportes prohibidos
* Seleccione aquellos transportes en los que no quiere viajar.
# Escalas máximo
5
Importe máximo
Sin límite
€
Aceptar
Borrar Datos
Listo
Internet

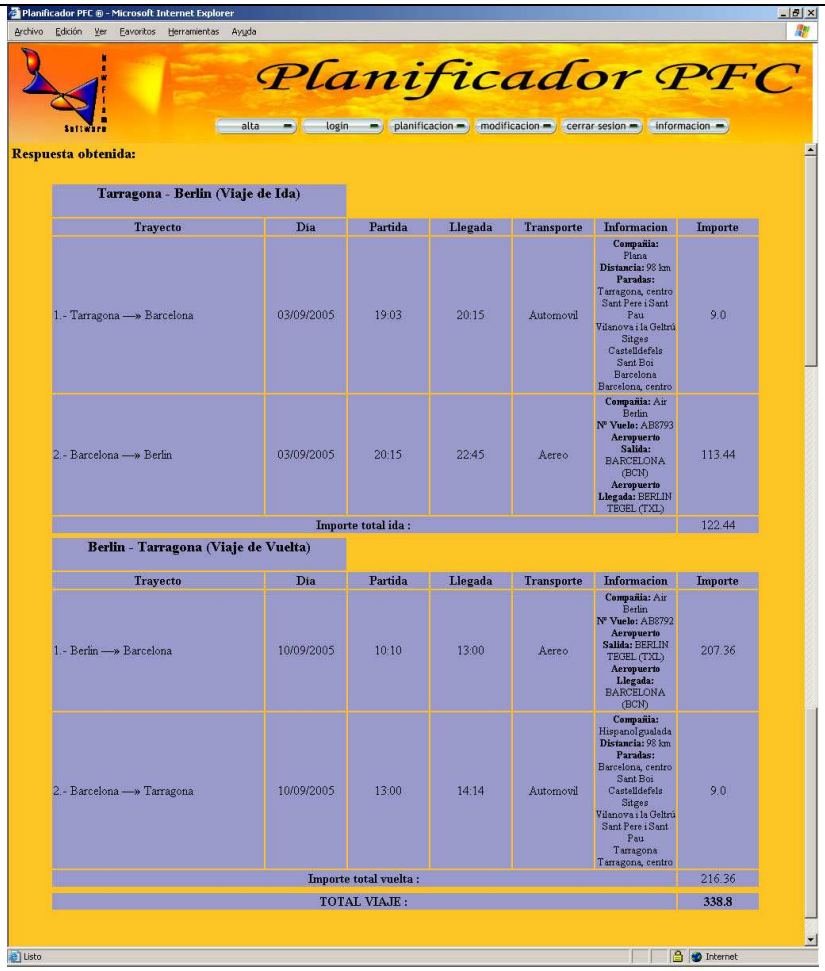


Ilustración 58: Proceso de Planificación correcto

7.3.3 Viaje Ida y Vuelta, nivel 2, transporte preferido

Todos los parámetros que modifiquemos, influirán de manera determinante en los resultados obtenidos. A modo de prueba, comprobaremos como repercute en los mismos la elección de un transporte preferido determinado. En nuestro caso, el Tren.

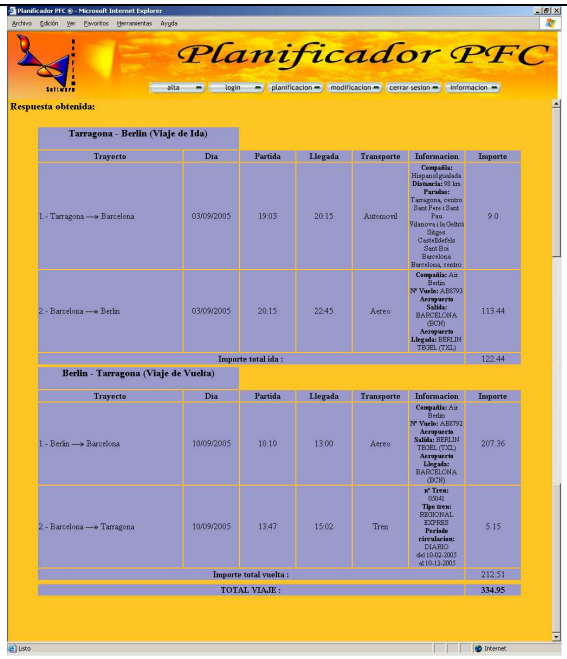


Ilustración 59: Proceso de Planificación correcto con diferentes parámetros

Se puede observar como el sistema ha sustituido el transporte por carretera por el de sobre raíles en el viaje de vuelta. Priorizando de esta manera las peticiones del usuario y adaptando las respuestas a sus intereses.

En el de ida, a pesar de ser un transporte preferido, el sistema ha determinado que era mejor ir por carretera, seguramente por tema horario.

7.3.4 Viaje Ida y Vuelta, nivel 2, transporte prohibido

En esta prueba comprobamos como el sistema cuando prohibimos un transporte, busca alternativas con los restantes. En condiciones normales, suele realizar el trayecto Tarragona ➔ Barcelona por carretera. En cambio esta vez, al haberlo prohibido, lo realiza en Tren.

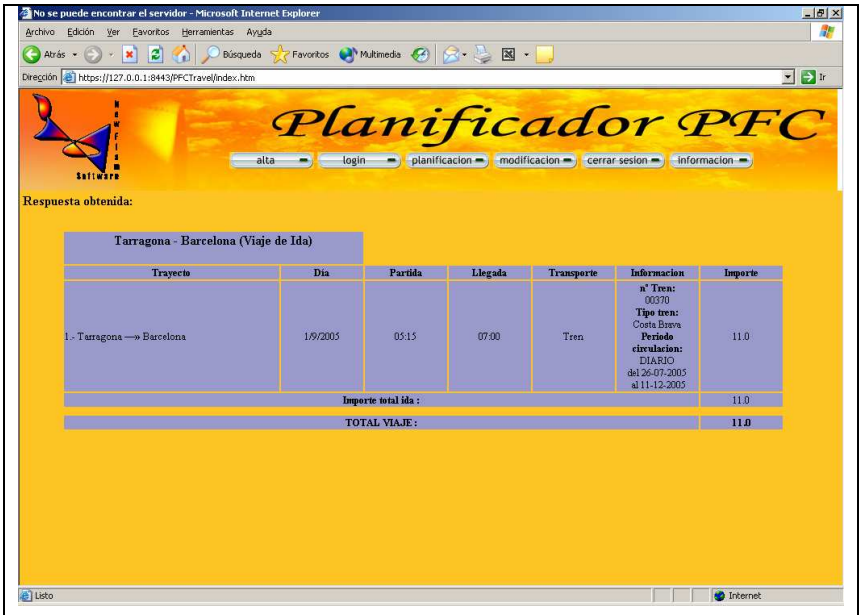


Ilustración 60: Transporte Automóvil prohibido

7.3.5 Trayecto directo

Las dos siguientes pruebas consistirán en comprobar como el sistema es capaz de realizar tanto planificaciones simples como compuestas. A partir de unos resultados directos, vemos como reacciona perfectamente a otra petición más complicada.

Introduciremos unos valores para buscar un trayecto directo. Por ejemplo Barcelona ←→ Londres (existe conexión directa a través de avión, como veremos en los resultados). Haremos tanto viaje de ida como de vuelta.

La aplicación realiza la planificación a una velocidad muy alta, demostrando su gran eficiencia en este tipo de casos. Pero, ¿qué sucede si complicamos el problema realizando búsquedas que no sean directas? Lo vamos a comprobar en el siguiente ejemplo.

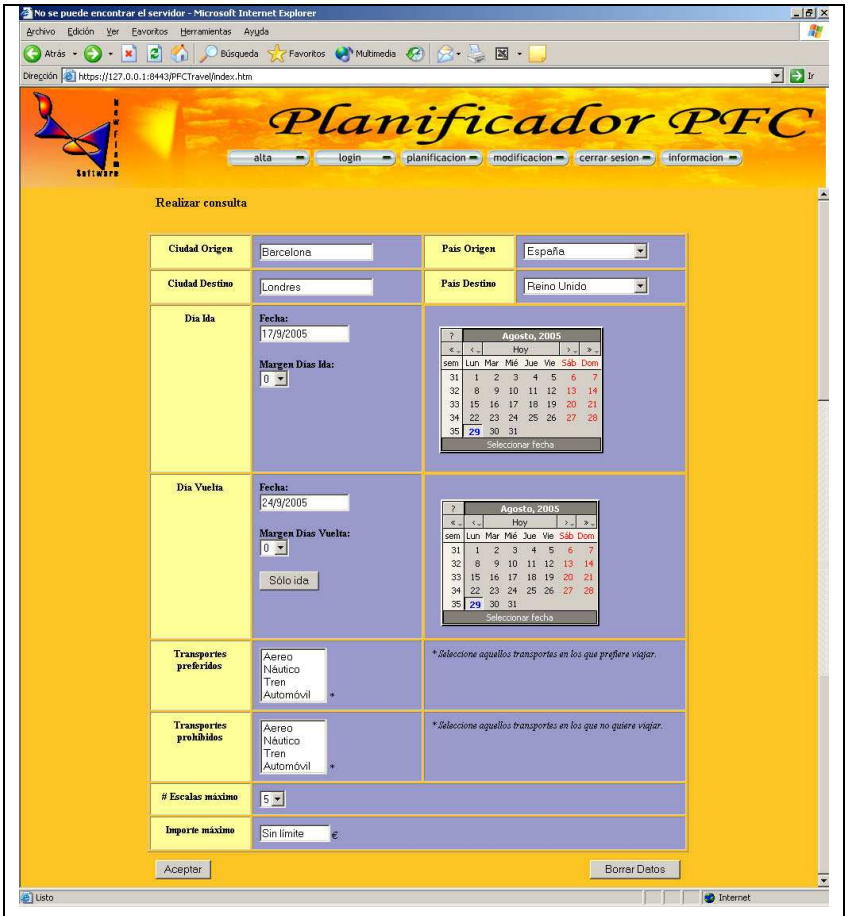
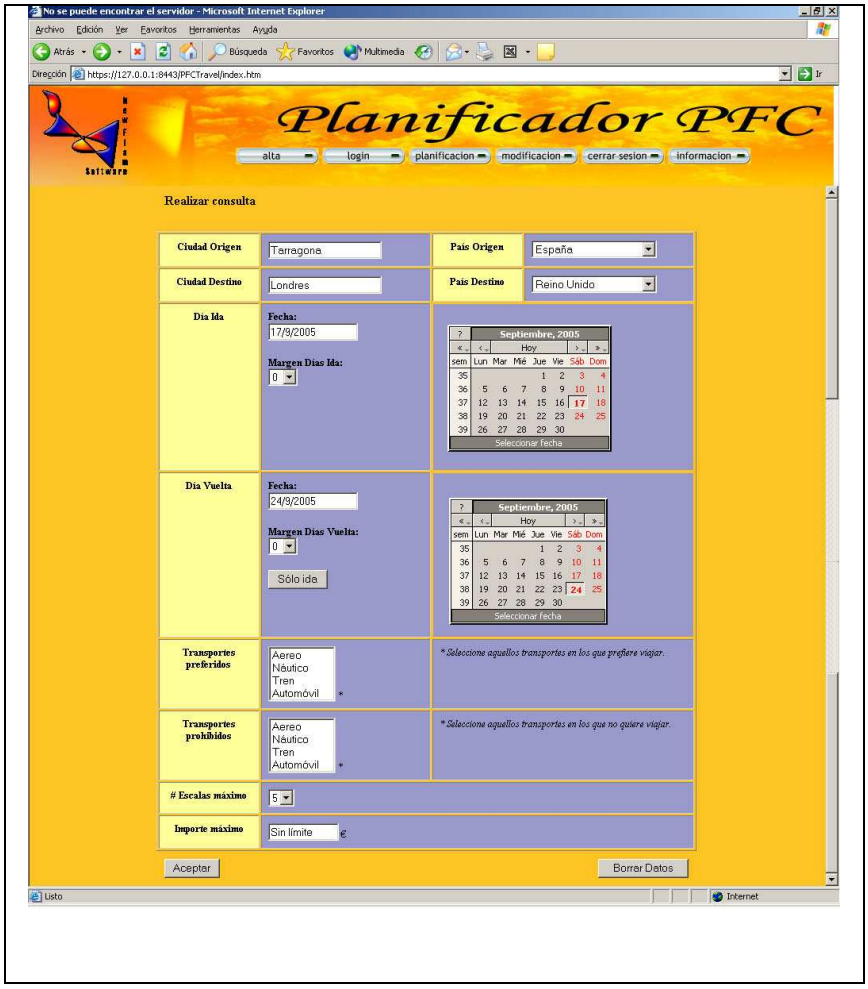


Ilustración 61: Planificación de un trayecto simple o directo

7.3.6 Trayecto compuesto

Realizaremos la búsqueda de Tarragona ←→ Londres, sabemos que existe un vuelo directo Barcelona ←→ Londres como se ha determinado en la prueba anterior, pero ¿será nuestro sistema capaz de encontrar ese camino?



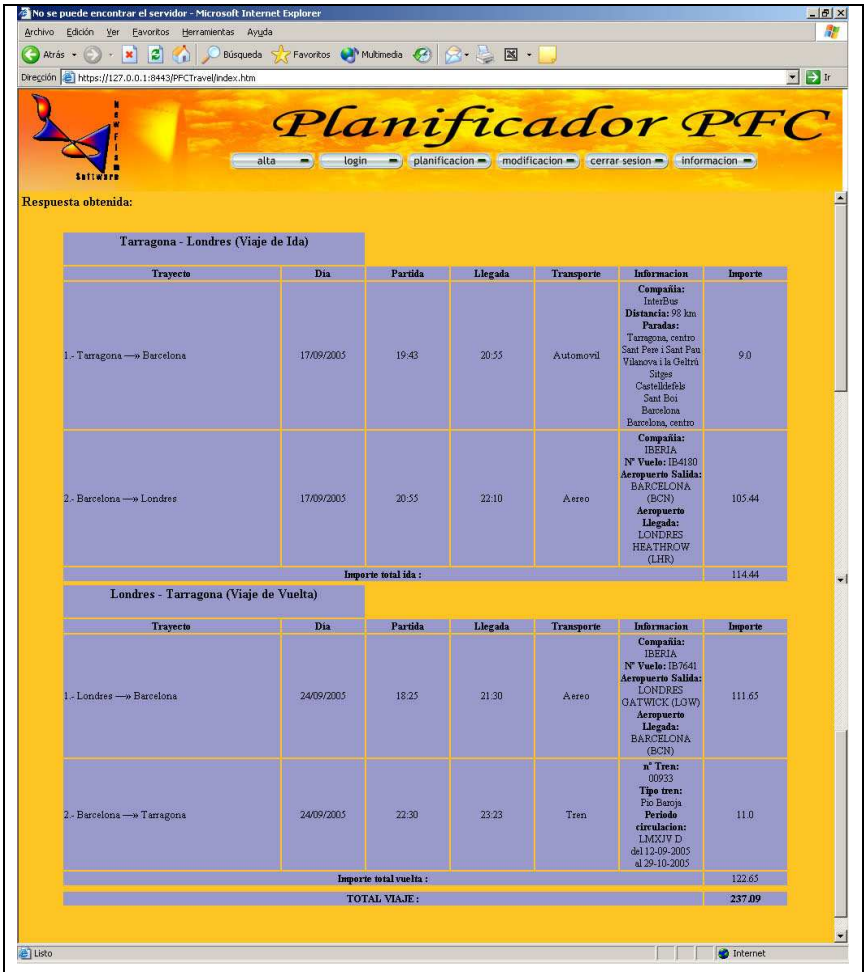


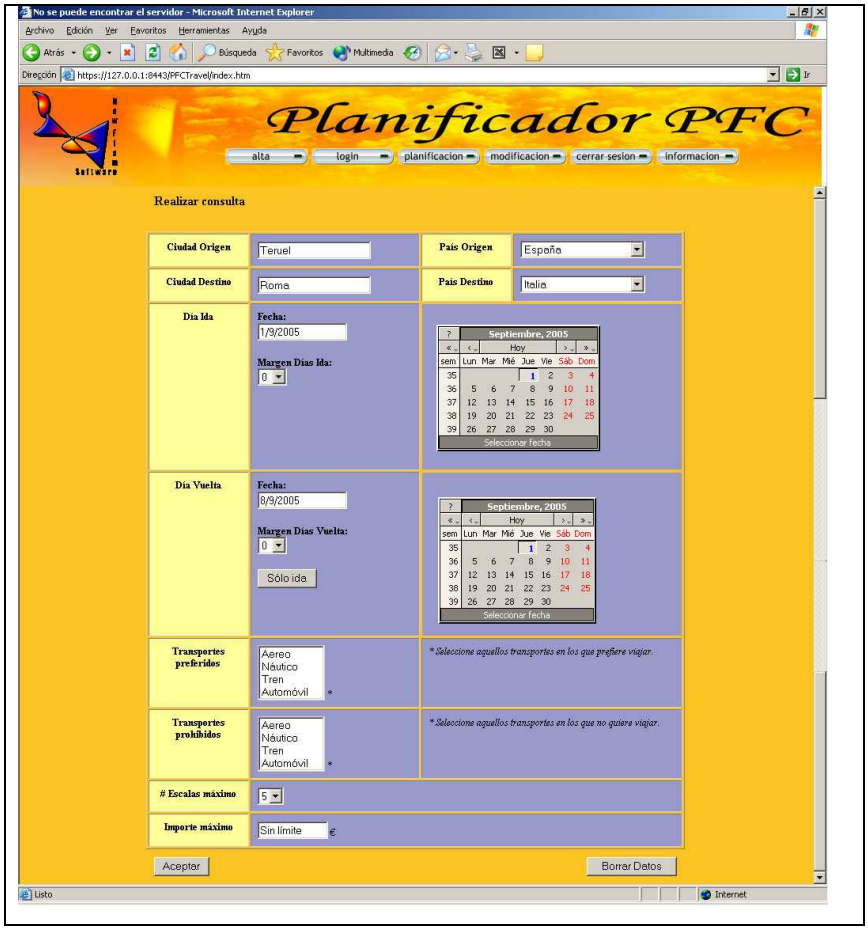
Ilustración 62: Planificación trayecto compuesto o con escalas

Efectivamente. Nuestros agentes han conseguido realizar esta petición exitosamente de nuevo. El rendimiento ha sido sensiblemente más bajo respecto a tiempo de respuesta, debido a la gran cantidad de posibilidades evaluadas, a pesar de la estrategia heurística usada.

7.3.7 Viaje Ida y Vuelta, nivel 2 y 3

En este ejemplo buscaremos el trayecto Teruel ←→ Roma. Como podemos observar en la siguiente ilustración, el sistema realiza una búsqueda mediante el algoritmo de Generalización – Especificación de nivel 3 en la ida, pues busca a través de Barcelona que pertenece al mismo país, pero distinta comunidad autónoma. Esto es debido a que no ha encontrado vuelos de ida desde algún sitio más cercano.

Sin embargo, el trayecto de vuelta lo ha resuelto en un nivel 2, gracias a que ha encontrado un vuelo Roma ➔ Zaragoza.



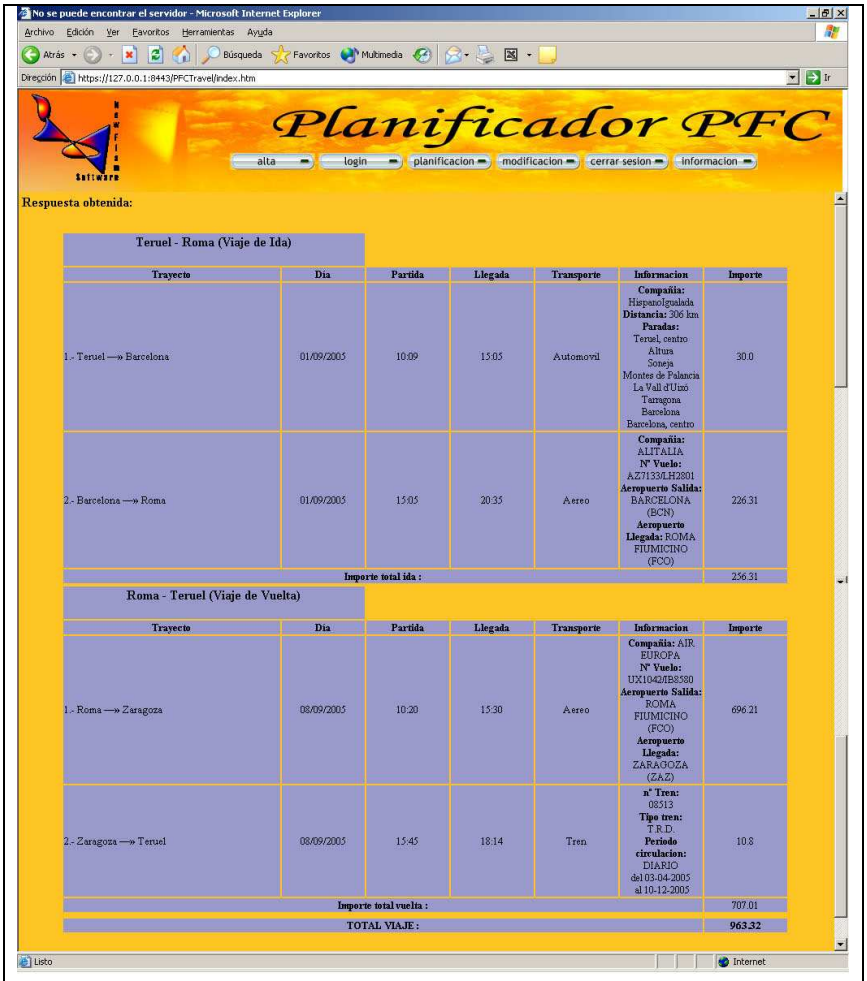


Ilustración 63: Planificación trayecto compuesto nivel 2 y 3

7.3.8 Viaje Ida

Vemos en la siguiente ilustración que el resultado de ejecutar el mismo ejemplo con solo viaje de ida es también satisfactorio.

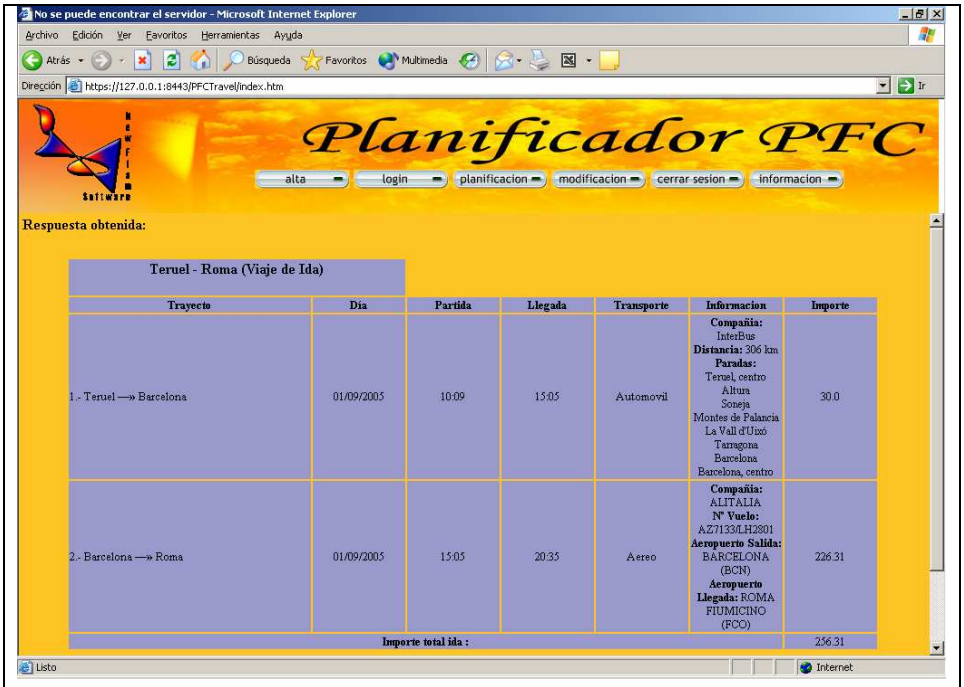


Ilustración 64: Planificación trayecto solo de ida

7.3.9 Limitando número de escalas

Vemos como limitando el número de escalas permitidas a 0, es decir, aceptando únicamente propuestas directas, el sistema nos indica que no se puede hacer con ese número de escalas. Deberíamos subirlo para obtener un resultado favorable.

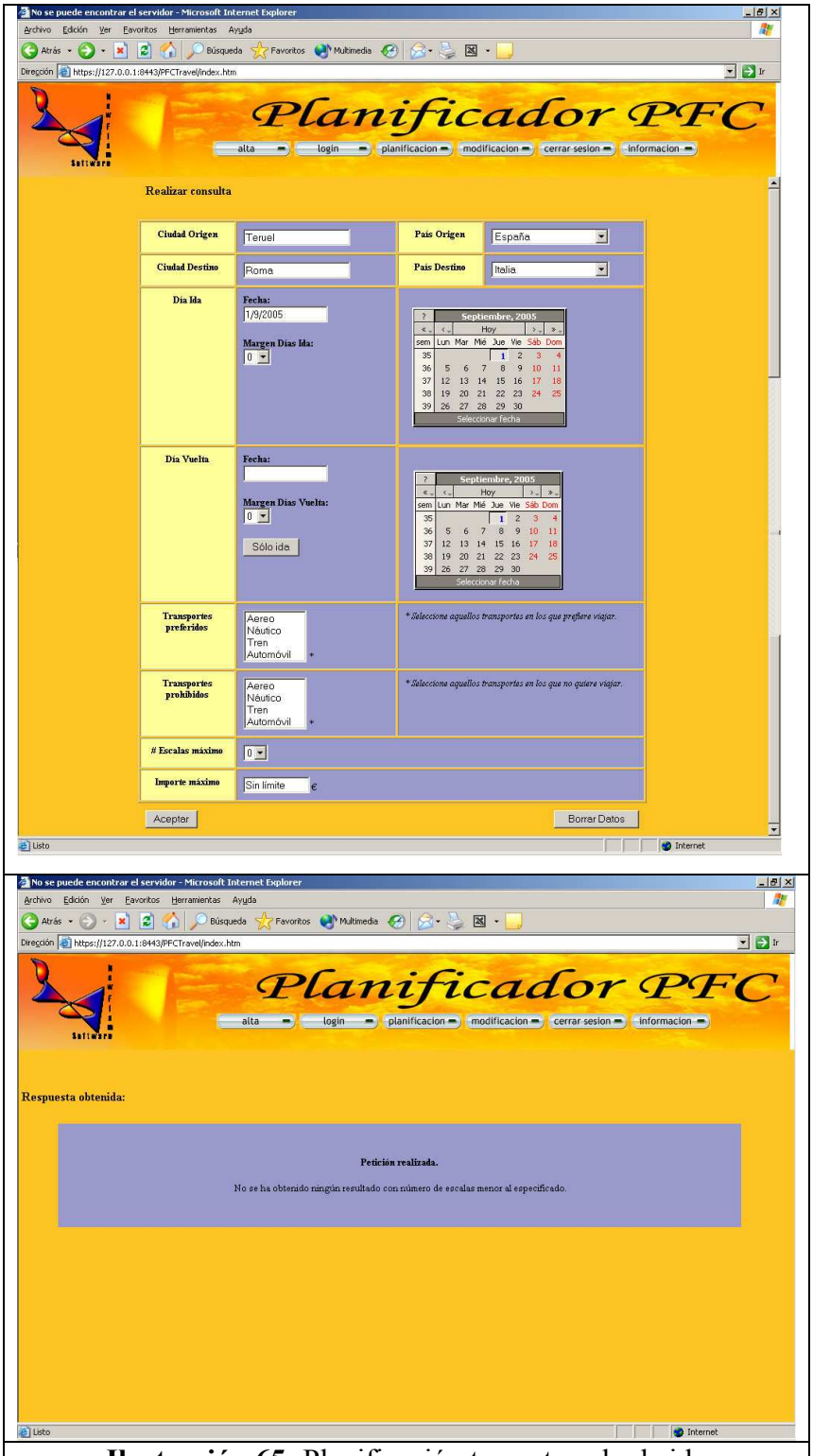


Ilustración 65: Planificación trayecto solo de ida

Con este juego de pruebas hemos visto como la funcionalidad de la aplicación es coherente y eficaz, respondiendo correctamente ante cualquier situación normal o excepcional para la cual haya sido programada.

8. Estado del Arte

8.1 Aplicaciones precedentes similares

Si bien es posible encontrar algunos trabajos previos orientados a la creación de rutas; la filosofía del nuestro dista mucho de aquellos, pues uno de los grandes objetivos que nos hemos marcado desde un principio, es innovar en el uso de diversas tecnologías que trabajen unidas al servicio de los Agentes.

Por tanto, se puede considerar que partimos de cero, no es un proyecto continuista sino precursor de un nuevo paradigma en el uso de Agentes, abriendo un gran abanico de nuevas posibilidades para proyectos venideros.

De todas maneras, compararemos nuestro sistema a otros SMA que hayan buscado funcionalidades similares.

8.1.1 SMA PFCTravel Vs MAPWeb-ETourism

David Camacho, Ricardo Aler, Daniel Borrajo y José M. Molina del Departamento de Informática de la Universidad Carlos III de Madrid, diseñaron una aplicación orientada a satisfacer peticiones del usuario en cuanto a rutas turísticas se refería, podía interactuar con diversos recursos Web, como compañías aéreas, de taxis, de trenes o cadenas hoteleras. Comparte nuestros principios de planificación por un lado y de recolección, interpretación y filtración de información procedente de Internet por otro. Se sustenta sobre una arquitectura SMA alternativa denominada SkeletonAgent, la cual se basa en otras dos previas conocidas como CooperA y ABC^2.

8.1.1.1 Arquitectura SkeletonAgent

Es una arquitectura tipo ‘Agenda’ lo que significa que se anotan en una agenda común las principales metas restantes de cada Agente. Lo que permite combinar acciones de diferentes fuentes, como pueden ser metas internas de unos Agentes, peticiones de otros, posibles cambios en el entorno, etc.

Cada tarea que se debe llevar a cabo, se divide en varias subtareas y se anotan en la Agenda, entonces el sistema decide cómo se puede solucionar una de ellas, comprueba si hay varias posibilidades y asigna, por ejemplo, a dos o tres agentes la resolución de la misma, según sus habilidades. De esta manera pueden proporcionar resultados de manera paralela y después poder elegir el mejor. Una vez se completan, se van eliminando progresivamente de la Agenda.

El control del ciclo de los agentes se hace de la siguiente manera:

- El módulo de control de ciclo, comprueba la agenda hasta que se introduce una tarea principal. Cuando ésta llega, es considerada como la meta principal, aunque puede descomponerse a lo largo del tiempo en varias subtareas, como también pueden inscribirse nuevas tareas totalmente diferentes de otros Agentes.
- Cada tarea tiene una prioridad dinámica asociada, por la cual se van ordenando continuamente para determinar el orden de resolución de cada una.
- Cuando se selecciona una de ellas (la de máxima prioridad cada vez), se evalúa y se decide cuáles son las habilidades necesarias para completarla.
- Llegados a este punto, pueden suceder diversas cosas:
    - Si la tarea está lista para ejecutarse (es decir, no necesita información adicional de otras tareas) puede ocurrir:
        - Si es posible, se ejecuta directamente.
        - En caso contrario, se descompone en subtareas, que se inscriben en la Agenda.
    - Si no está lista, se inserta de nuevo en la Agenda.

Esta arquitectura requiere funciones heurísticas para determinar qué tarea realizar en cada momento; las cuales están incluidas en el módulo '*Heuristics'* (Como vemos en la siguiente ilustración).

Es un sistema basado en habilidades (*Skills*), donde cada agente presenta las suyas, lo cual posibilita al sistema seleccionar cuáles son los más apropiados para desarrollar una tarea determinada.

Al igual que JADE, dispone de servicio de páginas amarillas y de páginas blancas, así como también de un módulo de comunicaciones.

Existe un Agente no incluido en JADE llamado *CoachAgent* encargado de controlar un grupo de Agentes, garantizando la estabilidad y la fluidez de las operaciones de los mismos.

También dispone de una Base de Conocimiento (*Knowledge Base*), que puede ser accedida por las habilidades de cada agente para obtener información acerca del dominio en el que se investiga.

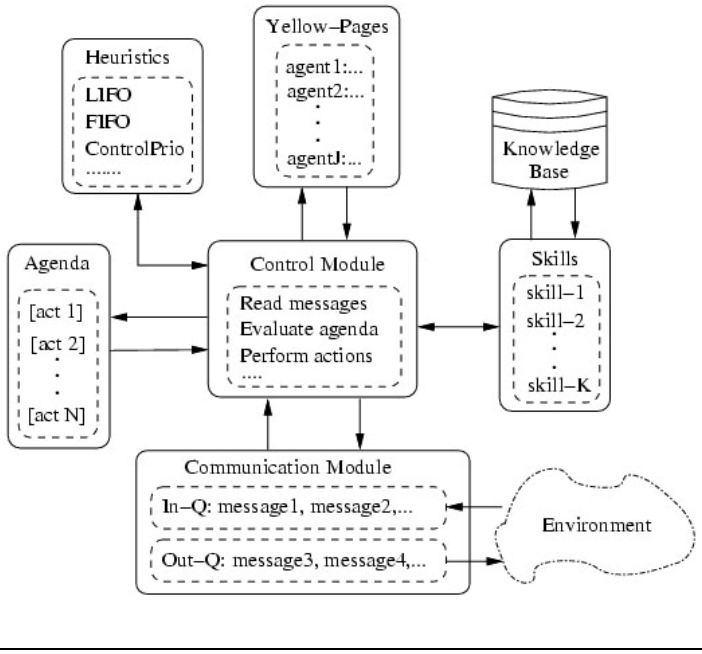


Ilustración 66: Arquitectura SkeletonAgent

8.1.1.2 Diseño MAPWeb-ETourism SMA Vs PFCTravel

Esta propuesta, a diferencia de la nuestra, se descompone en tres niveles de agentes:

- **UserAgents**: Representa el puente entre el usuario y el sistema. Dotado de habilidades para obtener la descripción del usuario y posteriormente para mostrar los resultados.
  - Correspondería a nuestro *ClienteAgente*.
- **PlannerAgents**: Capaces de resolver problemas de planificación usando la información recogida de la Web.

- En nuestro sistema correspondería al dúo *BuscadorAgente* + *FiltradorAgente*. Pues nosotros hemos añadido funcionalidades específicas a cada uno de ellos, y se han agrupado divididas de manera coherente para realizar cada uno sus funciones independientemente del otro.

- **WebAgents**: Capaces de proporcionar información recogida de la Web.
  - Nosotros tenemos los *BDAgente* e *InternetAgente*, que además ya tienen embebida una primera etapa de filtración de resultados. Enviando sólo el más adecuado a *FiltradorAgente*, el cual queda liberado de carga inútil de trabajo, a la vez que agilizamos las comunicaciones al enviar menos información.
  - Al igual que nosotros, cada Agente se ocupa de un recurso en concreto.

8.1.1.2 Planificación MAPWeb-ETourism SMA Vs PFCTravel

En cuanto al algoritmo de planificación, parece que consigue los mismos resultados que nuestro sistema. Mientras que el nuestro se basa en el algoritmo de Generalización – Especificación, no se especifica cómo trabaja en su aplicación, aunque sabemos que usa técnicas clásicas de planificación.

De todas maneras presenta un ejemplo donde se ve el funcionamiento del mismo, como podemos ver en la siguiente ilustración.

Travel problem example to go from Salzburg to Segovia by airplane or train.

Leg	Stage	Date	Transport	Restrictions	Nº Transfers
1	Salzburg → Madrid	May 11th	Plane or train	none	0 or 1
2	2 nights stay	May 11-13th	none	< 100 euros	-
3	Madrid → Segovia	May 13th	Plane or train	none	0 or 1
4	3 nights stay	May 13-16th	none	< 90 euros	-
5	Segovia → Salzburg	May 16th	Plane or train	none	0 or 1

Ilustración 67: Ejemplo planificación MAPWeb-ETourism

8.1.1.3 Conclusiones

El sistema comentado tiene algunos puntos en común con el nuestro, pero quedaría relegado a un nivel más clásico en cuanto al uso de tecnologías, como queda patente en la falta de un modelo distribuido en su diseño, lo cual limita en gran medida la posible ampliación del mismo, al estar atado a un solo ente computacional.

También podríamos encasillar su metodología de planificación en este mismo escalón, usando técnicas clásicas, como se ha comentado anteriormente. Realmente, sería muy interesante combinar estas técnicas con nuestro algoritmo de Generalización – Especificación.

8.1.2 Otros sistemas de acceso a información Web

Existen algunas aplicaciones que también acceden a recursos en Internet, adquieren, interpretan y filtran información procedente de estos, al igual que nuestro proyecto. Vamos a comentarlos brevemente, únicamente para darlos a conocer.

- **Ariadne:** Este sistema incluye un conjunto de herramientas para construir aplicaciones que permiten que los recursos Web parezcan bases de datos.
- **Heracles:** Es un framework concebido para desarrollar distintos sistemas de asistencia de información. Usa un conjunto de Agentes (Ariadne, Theseus, Electric Elves), así como también una red dinámica de propagación limitada por reglas jerárquicas, con la que integra las diferentes fuentes de recursos.
  Se conocen dos sistemas de asistencia creados hasta el momento:
  - *The Travel Planning Assistant*, especializado en dar asistencia a turistas para planificar sus viajes.
  - *The WorldIngo Assitant*, encargado de recoger información sobre el tiempo, noticias, vacaciones, mapas, aeropuertos, etc. para un usuario específico situado en un determinado lugar.
- **WebPlan:** Es un asistente para búsquedas por Internet bajo un dominio específico basado en planificación. El sistema planificador existente llamado CAPlan ha sido extendido de varias modos para tratar, por ejemplo, con información incompleta o interacción con usuarios. Este sistema está orientado a encontrar software para PC en Internet.

8.2 Trabajo futuro

Tras haber leído este documento, vemos como quedan una infinidad de puertas abiertas que pueden ser explotadas para ampliar la aplicación por doquier.

Aparecen nuevas líneas de trabajo, debido a la elección de las tecnologías que han trabajado conjuntamente para alcanzar los objetivos propuestos, que pueden ser aplicadas a otros dominios totalmente diferentes gracias a la gran difusión de las arquitecturas distribuidas, y gracias también a la gran cantidad de información sobre cualquier campo esparcida por Internet.

Centrándonos en nuestro proyecto, podrían ampliarse algunas funcionalidades muy relacionadas con la temática tratada. Por ejemplo, una vez presentados los resultados, podría incluirse la opción de realizar la reserva y compra de billetes a través de Internet, conjuntando una pasarela de pago con un Agente encargado de dicho propósito.

Recogemos varios datos de los usuarios que quedan registrados en nuestro sistema, aunque se podrían ampliar para, posteriormente realizar un estudio mediante técnicas de Data Mining, y así adaptar las propuestas y productos buscados a cada segmento de población.

Los principales objetivos de cara al futuro consisten en una mejora de rendimiento en cuanto a coste computacional al realizar las planificaciones requeridas, así como también a la ampliación del número de posibilidades evaluables, para así tener un mayor número de alternativas disponibles.

Para conseguir un aumento en la velocidad de respuesta, se puede limitar el número de localidades a ser evaluadas mediante el uso de funciones heurísticas, para escoger aquellas más prometedoras. De alguna manera lo hacemos al priorizar localidades con buenas comunicaciones, provistas de aeropuerto, etc. Aunque todavía deberíamos poder prever cuáles son las potencialmente más productivas, al igual que podría hacerlo un ser humano. Por ejemplo, en el algoritmo de Generalización – Especificación, cabe la posibilidad de mejorar varios aspectos que repercuten directamente en la eficiencia del mismo y, en consecuencia, del resto de elementos relacionados. Se podría considerar la distancia entre los resultados generados en cada pasada y la localidad para la cual

generalizamos. De esta manera, combinaríamos nuestro algoritmo con técnicas de planificación clásicas dotándolo de mayor potencial. No sería si quiera necesario tener un grafo de distancias; en su lugar, se podrían realizar consultas directamente sobre un Agente encargado de calcular estas distancias entre dos localidades dadas. De hecho, nuestro *ViaMichelinAgente* es capaz de hacerlo; por tanto sólo habría que realizar algunas pequeñas modificaciones.

En el mismo contexto, se podría incluir la posibilidad de incluir habilidades de planificación orientadas a casos, que tengan en cuenta situaciones o consultas anteriores para resolver una nueva petición; evitándonos realizar una búsqueda a ciegas en cada una de ellas.

Otro punto que quedaría muy abierto, sería la combinación con otro proyecto encargado de analizar automáticamente las páginas Web a las que accedemos, extraer el modo de realizar la consulta y, seguidamente, obtener los resultados, analizarlos y localizar la información que necesitamos. Podría seguir la metodología que hemos usado nosotros 'a mano' en cada caso de análisis de un nuevo recurso, y que hemos explicado con detalle en apartados anteriores. De esta manera, automatizaríamos el punto que más mantenimiento requiere de la aplicación y que, aún no siendo excesivo, siempre es conveniente automatizar el mayor número de tareas de cara a futuras ampliaciones, dotando al sistema de mayores oportunidades de escalabilidad.

El resto de mejoras se englobarían como ampliación de las fuentes que disponemos para obtener la información. Lo vemos en dos dimensiones:

- Por un lado, el incluir el resto de localidades a nivel Europeo en nuestra Base de Datos CorbaBD.
- Por otro, incluir más *BDAgente* e *InternetAgente* con recursos complementarios a los ya existentes, para ampliar las posibilidades de planificación. Por ejemplo, como se ha comentado en un capítulo anterior, tenemos una alternativa ya evaluada que es la de Edreams.es; realmente interesante.

8.3 Conclusiones

Hoy en día, las compañías son conscientes que para mantener la competitividad en su segmento, han de proporcionar una manera de hacer llegar su información al gran público. Un elemento de masas como es Internet, deviene un escaparate perfecto para los productos de cada compañía. En este punto fundamental nos hemos basado para crear una plataforma capaz de recurrir a esta información para solucionar una serie de problemas que plantea el usuario dentro de un dominio concreto; en nuestro caso, hablamos de planificación de viajes mediante medios de transporte.

Hay que tener en cuenta que Internet es un medio dinámico que evoluciona y cambia constantemente a una velocidad relativamente alta. Es por ello que el sistema encargado de recolectar información de la Web debe ser lo suficiente flexible y adaptable para poder mantener el ritmo evolutivo de la Red de Redes.

Es por ello que se ha creado una arquitectura donde la inclusión de nuevos recursos se hace de manera inmediata, sin modificar una sola línea del resto de elementos que componen el sistema. Tan solo debemos crear e introducir en la plataforma un nuevo Agente encargado de recolectar la información proporcionada por ese recurso.

En caso que un recurso ya no sea válido, podemos eliminar el Agente de la plataforma para modificarlo, o bien descartarlo definitivamente. Cosa que se hace sin el más mínimo trauma, siendo una operación limpia, rápida y segura.

Se ha creado un sistema que combina un método de planificación novedoso con un modelo de extracción de información de Internet y Bases de Datos, encargado de resolver problemas complejos planteados por un usuario, en forma de peticiones de rutas entre dos puntos, para responder con resultados que los enlacen usando medios de transporte de manera encadenada para conseguir completar la planificación. Para ello, puede descomponer el problema en porciones más pequeñas que irá resolviendo una a una hasta alcanzar el objetivo deseado.

Las diferentes tecnologías usadas, confieren a la aplicación un aire distribuido que proporciona una escalabilidad fuera de dudas; permitiendo que ampliemos la información consultada mediante la inserción de un mayor número de agentes en la

plataforma, sin que esto sea motivo de degeneración del sistema en cuanto a rendimiento.

9. **Bibliografía**

En este apartado recogemos las principales fuentes de información que se han consultado para la realización de nuestro proyecto.

1. What is an ontology and why we need it, http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
2. Jade - Java Agent DEvelopment Framework, http://sharon.cselt.it/projects/jade/community-3rdpartysw.htm
3. Core Java Technologies Technical Tips, http://java.sun.com/developer/JDCTechTips/2004/tt0210.html#2
4. jGuru, http://www.jguru.com/faq/view.jsp?EID=267915
5. Httpclient under weblogic 7.0 Vs. HttpClient, http://mail-archives.apache.org/mod_mbox/jakarta-httpclient-dev/200305.mbox/%3CBAY2-DAV73mnbWakSlk00020232@hotmail.com%3E
6. Session Handling in MIDP (Para Moviles), http://developers.sun.com/techtopics/mobility/midp/articles/sessions/
7. Guía de pueblos,municipios y localidades de España, http://www.guiadepueblos.com/
8. Aena.es, http://www.aena.es
9. viajes.wanadoo.es, http://viajes.wanadoo.es/
10. ViaMichelin : guias turisticas, mapas carreteras, callejero, http://www.viamichelin.es
11. RENFE, http://www.renfe.es
12. PFCs de GruSMA, http://www.etse.urv.es/recerca/banzai/toni/MAS/PFCs.html

13. CURSO PRÁCTICO DE CORBA EN GNU/LINUX, http://acs.barrapunto.org/articulos/trunk/LinuxActual/CursoCORBA/entrega1/entrega1.html
14. Wooldridge, M., An introduction to multiagent systems, John Wiley Ed., 2002. ISBN 0-471-49691-X.
15. Turist@, Agentes al servicio del turista mediante telefonía móvil. PFC de Ingeniería en Informática de la Universidad Rovira y Virgili (URV). Junio de 2005.

10. **Anexo**

Como anexo, se adjunta el código fuente de nuestra aplicación y un estudio sobre JADE desde el punto de vista de FrameWorks; donde aprenderemos mucho sobre su funcionamiento interno.