Desarrollo de una aplicación de gestión de servicios turísticos para la plataforma THOMAS

Ingeniería Informática

Autor:
Jaume Jordán Prunera

Directores:
Dra. Estefanía Argente Villaplana
Dr. Vicente Julián Inglada

Septiembre, 2009

Índice general

Índice de figuras

Índice de tablas

A mi familia

Agradecimientos

A mis directores, Estefanía y Vicente, por su interés y apoyo en la realización de este proyecto y por compartir sus conocimientos conmigo.

A Elena y Natalia, por su ayuda y paciencia resolviendo los problemas encontrados con THOMAS y por su disposición a aclarar todas mis dudas sobre la implementación.

Capítulo 1

Motivación y Objetivos

1.1. Motivación

En la actualidad, los sistemas multiagente (SMA) se enfrentan al reto de tomar decisiones de forma autónoma y flexible. Además, también deben ser capaces de integrar otros agentes ajenos al sistema dentro de una "sociedad". Desde esta perspectiva, para integrar la "sociedad" los SMA tienen que satisfacer una serie de requisitos como son: distribución, evolución constante, flexibilidad de entrada y salida para los miembros, gestión de la estructura organizativa que define la sociedad, ejecución de los agentes en distintos dispositivos, entre otros. Todos estos requisitos definen un conjunto de características que pueden abordarse a través del paradigma de sistemas abiertos y organizaciones virtuales.

Desde el punto de vista de las organizaciones, este paradigma es una solución para gestionar la coordinación y controlar el comportamiento de los agentes en los SMA abiertos. Así pues, el modelado de la organización permite describir la composición estructural (i.e. roles, grupos de agentes, patrones de interacción, relaciones entre roles), el comportamiento funcional (i.e. tareas de agentes, planes o servicios) y la normativa de regulación (para controlar el comportamiento de los agentes).

Por otra parte, las arquitecturas orientadas a servicios (SOA) ofrecen un marco adecuado para la flexibilidad e independencia necesaria en los SMA abiertos, ya que permiten una alta escalabilidad y proporcionan una forma estándar de exposición e invocación de servicios. Las SOA se basan en la utilización de servicios accesibles a través de una red y que pueden ser combinados y reutilizados. Además, la interoperabilidad de los servicios es muy importante en este tipo de arquitecturas, y el uso de servicios

web es muy común.

Así pues, la necesidad de un SMA abierto basado en organizaciones que cumpla con los requisitos de flexibilidad y escalabilidad inherentes a las nuevas tecnologías de la información, hace surgir la idea de fusionar los SMA con los servicios web. Dentro de esta nueva concepción de los SMA se encuentra la propuesta THOMAS [10] [3] [22], una plataforma recientemente desarrollada que ofrece una gran versatilidad para gestionar organizaciones virtuales con agentes provenientes de distintos contextos, y en la que se pueden crear y usar una gran diversidad de servicios web.

El presente proyecto ha sido impulsado principalmente por la necesidad de validar la plataforma THOMAS. La creación reciente de ésta requiere la validación de su funcionamiento. Para ello, es recomendable realizar distintas pruebas a dicha plataforma y desarrollar aplicaciones basadas en ella.

Por otra parte, también es necesario disponer de ejemplos en los que se muestre el funcionamiento de la plataforma THOMAS. Así pues, este proyecto aporta una aplicación de gestión de servicios turísticos usando la plataforma. Con esta aplicación se puede ejecutar un conjunto de escenarios aplicables al contexto de los servicios que ofrece la plataforma. Además, en esta aplicación se trabaja concretamente en la gestión de servicios turísticos. Con ello se pretende mostrar una alternativa aplicable a la industria de las nuevas tecnologías de la información.

1.2. Objetivos

El principal objetivo de este proyecto es la creación de una aplicación de gestión de servicios turísticos para la plataforma THOMAS. Para la consecución de dicho objetivo, se desarrollarán los siguientes sub-objetivos:

- Estudio de la plataforma THOMAS. Análisis y comprensión del funcionamiento de la plataforma y del uso de sus meta-servicios. También se investigará su implementación con el fin de obtener los conocimientos necesarios para poder desarrollar la aplicación de gestión de servicios turísticos.

- Diseño de la aplicación de gestión de servicios turísticos. Se diseñarán agentes proveedores y clientes de servicios relacionados con el turismo. Esto implica un diseño apropiado para la correcta ejecución de los meta-servicios de la plataforma THOMAS y de los servicios de gestión turística. Además, se diseñará la interfaz adecuada para mostrar la información de la organización virtual, accediendo a los datos de interés del OMS y del SF.

- Implementación de la aplicación de gestión de servicios turísticos. Se crearán los agentes proveedores y clientes adaptados a la plataforma THOMAS y también su sistema de comunicación con los agentes intermediarios OMS y SF utilizando las especificaciones de la plataforma. Además se generarán distintos servicios web adicionales pertenecientes al ámbito turístico, que serán usados y proveídos por los agentes desarrollados.

- Validación de la plataforma THOMAS a través de la aplicación desarrollada. Se realizará mediante la ejecución de distintos escenarios en los que se invocará un conjunto de servicios web, tanto de la plataforma como externos a ésta referentes a la gestión turística.

1.3. Estructura del trabajo

En esta sección se explica de forma resumida la estructuración en capítulos de este trabajo. A parte de este capítulo de introducción, el resto de capítulos están organizados de la siguiente manera:

- En el capítulo 2 se explican los conceptos y características de los SMA abiertos y se relacionan con los servicios web.
- En el capítulo 3 se introduce la plataforma THOMAS al lector, explicando sus componentes y aspectos teóricos así como las cuestiones de su implementación.
- En el capítulo 4 se detalla el diseño y la implementación de la aplicación desarrollada. Primeramente se explica el caso de estudio sobre el que se basa la aplicación de gestión de servicios turísticos. A continuación, se explica la interfaz gráfica de usuario y los detalles de la implementación del sistema. Finalmente, se realiza una validación de la plataforma THOMAS y de la aplicación desarrollada.
- En el capítulo 5 se muestran las conclusiones de este proyecto y los trabajos futuros.

Capítulo 2

Estado del Arte

En el campo de las nuevas tecnologías de la información, los sistemas multiagente (SMA) han adquirido mucho protagonismo por parte de una notable cantidad de proyectos de investigación dedicados a dichos sistemas. Sus ventajas frente a otros tipos de sistemas software aportan importantes avances y nuevos paradigmas para la resolución de problemas de forma distinta a los sistemas tradicionales.

En este capítulo se introducirán los conceptos de sistemas multiagente, agentes software y sistemas multiagente abiertos. También se relacionarán los servicios web con los sistemas multiagente dado el creciente interés en el uso combinado de ambas tecnologías.

2.1. Sistemas Multiagente Abiertos

El concepto de sistema multiagente surgió a partir de la Inteligencia Artificial Distribuida [16]. Es necesario aclarar que aunque se tratan de sistemas distribuidos no son como los tradicionales. La principal diferencia entre ellos es que en un SMA cada agente persigue sus propios objetivos, en cambio, en un sistema distribuido clásico todos los nodos pretenden alcanzar un objetivo común. Por otra parte, en muchas ocasiones se considera los SMA una parte de la Inteligencia Artificial. Esto es debido a la gran relación existente, ya que la Inteligencia Artificial pretende emular el comportamiento y razonamiento humano, y los SMA son la plataforma para simular sociedades mediante agentes software "inteligentes".

Por lo tanto, los SMA se pueden definir como un conjunto de agentes autónomos que

trabajan juntos para resolver problemas. Estos agentes tienen parte de la información o de la capacidad para resolver el problema en cuestión, por lo que la resolución se debe realizar de forma cooperativa mediante algún tipo de comunicación. Los datos suelen estar descentralizados y la computación es asíncrona. Además, los agentes pueden decidir las tareas a realizar y quién debe realizarlas.

También es necesario aclarar el concepto de agente software, ya que es la base de cualquier SMA. Sin embargo, no existe una definición concisa y aceptada ampliamente por la comunidad científica. Una de las primeras definiciones bastante citada es: "un agente se define como una entidad cuyo estado es visto como un conjunto de componentes mentales, tales como creencias, capacidades, elecciones y acuerdos" [19]. Otra de las más citadas y un poco más actual es la siguiente: "un agente es un sistema informático situado en un entorno y que es capaz de realizar acciones de forma autónoma para conseguir sus objetivos de diseño" [24]. Estas definiciones son un ejemplo de la discusión que supone el concepto de agente, sin embargo, la siguiente definición junto con las anteriores ayuda a comprender lo que significa un agente software: "El concepto de agente caracteriza a una entidad software con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas computacionales y es capaz de realizar de forma inteligente y autónoma distintos objetivos intercambiando información con el entorno, o con otros agentes humanos o computacionales" [9]. Se puede encontrar información más detallada sobre la discusión acerca del concepto de agente en [14].

Los SMA ofrecen muchas ventajas frente a la utilización de otros tipos de tecnología software. A continuación se comentarán algunas de ellas:

- Se trata de una tecnología que incorpora aspectos de Ingeniería del Software, Inteligencia Artificial y telecomunicaciones.
- Tienen menor coste ya que los agentes facilitan la reusabilidad y requieren menor tiempo de desarrollo que otros sistemas convencionales.
- Mejoran la funcionalidad y la calidad frente a los sistemas actuales, que son rígidos e inestables. La flexibilidad y adaptabilidad de los SMA es mucho mayor.
- El mantenimiento se reduce al facilitar la transformación y la evolución. La funcionalidad puede ampliarse o cambiarse modificando el comportamiento, las

estrategias o los objetivos de los agentes. También se pueden incluir nuevos agentes y nuevo conocimiento.

- Son fácilmente integrables con otras tecnologías (web, bases de datos, etc.).
- Facilitan la tarea de los ingenieros porque pueden utilizarse patrones de agentes. Con lo cual, se puede prestar toda la atención en definir el comportamiento del agente en vez de a la codificación convencional.

A pesar de la corta historia de los SMA, actualmente existen multitud de aplicaciones en distintos campos. En el ámbito industrial su utilización ha sido muy amplia, así pues, existen aplicaciones de control y planificación de la fabricación, diseño de productos, control de tráfico aéreo, gestión de electricidad, ubicación de contenedores, control de líneas de producción, etc. Además, también hay aplicaciones en otros ámbitos como la medicina, recuperación de la información, comercio electrónico y telecomunicaciones. Esto demuestra el interés generalizado en el uso de los SMA dadas las ventajas que ofrecen, y probablemente en los próximos años su desarrollo y aplicación irá en aumento.

Desde los inicios de los SMA, se han desarrollado muchas plataformas para gestionar la interconexión y ejecución de los agentes dentro de un sistema. La tendencia general de la mayoría de las plataformas es seguir los estándares FIPA[1], aunque existen algunas que tienen su propia arquitectura de agentes. Dos de las plataformas más conocidas que siguen el estándar FIPA son *Java Agent DEvelopment*[2] (JADE), desarrollada por CSELT S.p.A (actualmente Telecom Italia o TILab), y *FIPA Open Source*[3] (FIPA-OS), desarrollada por Nortel Networks. Otras plataformas también utilizadas en distintos ámbitos son: *ABLE*[4] de IBM, *Comtec*[5] y *APRIL*[6]. A continuación se describirá brevemente la plataforma JADE dada su amplia utilización y por ser la base de la plataforma THOMAS.

JADE es una de las plataformas de SMA más conocidas y utilizadas en la actualidad. Como se ha comentado anteriormente, sigue el estándar de FIPA y está desarro-

[1] http://www.fipa.org/
[2] http://jade.tilab.com/
[3] http://www.fipa.org/
[4] http://www.alphaworks.ibm.com/tech/able
[5] http://ias.comtec.co.jp/ap/
[6] http://www.nar.fujitsulabs.com/app/

llada en lenguaje Java. Esta plataforma ofrece una librería para crear agentes que se pueden comunicar entre sí mediante el lenguaje FIPA-ACL (***A**gent **C**ommunication **L**anguage*) y servicios estándar de gestión de agentes FIPA. También dispone de una interfaz gráfica que permite gestionar directamente los agentes, enviar mensajes, monitorizar la ejecución, etc. Además, JADE proporciona un conjunto de agentes ya implementados que pueden ser utilizados para distintos objetivos y que están integrados en la plataforma:

- ***D**irectory **F**acilitator* (DF): proporciona los servicios de directorio.
- ***A**gent **M**anagement **S**ystem* (AMS): ejerce de supervisor de acceso y uso de la plataforma. Proporciona un servicio de páginas blancas y de ciclo de vida, manteniendo una lista de los identificadores de los agentes (AID) y su estado.
- Agente *Sniffer*: es una herramienta que ayuda a la depuración de las interacciones entre los agentes de la plataforma. La función que realiza consiste en interceptar el flujo de mensajes entre los agentes y mostrarlos gráficamente para el usuario.
- Agente *Introspector*: permite el control del ciclo de vida de los agentes en ejecución y los mensajes que intercambian.

Para la implementación de los agentes JADE se debe heredar la clase *jade.core.Agent*, la cual dispone de métodos para iniciar el agente, enviar y recibir mensajes, etc. Además, la plataforma dispone de la clase *jade.core.Behaviour* que proporciona métodos para definir los comportamientos de los agentes.

Por otra parte, los sistemas abiertos son aquellos que permiten la entrada de nuevos componentes durante la ejecución del sistema que pueden no haber sido conocidos en la fase de diseño [11]. Por lo tanto, los agentes que participen en un sistema de este tipo pueden usar distintos protocolos o estar desarrollados con distintos lenguajes o arquitecturas. Esto proporciona mucha flexibilidad y heterogeneidad al sistema. No obstante, independientemente del diseño o desarrollo que haya tenido un componente, se unirá al sistema adquiriendo un determinado rol para el cual se habrán establecido un conjunto de normas y permisos que regulen su comportamiento.

Los SMA abiertos permiten trabajar en entornos dinámicos en los que los agentes pueden entrar o abandonar el sistema de forma continua [25]. Así pues, en la fase de

diseño no se puede saber cuántos agentes estarán presentes en el sistema. Por lo tanto, para el diseño se debe considerar esta dinamicidad y también la heterogeneidad de los distintos agentes que pueden entrar en la organización. A consecuencia de esto, es necesario establecer control y seguridad ya que pueden entrar agentes poco fiables o con objetivos contrarios a la organización. Otra parte muy compleja de los SMA abiertos es el desarrollo de las comunicaciones, debido principalmente a la heterogeneidad de los componentes que pueden entrar en el sistema.

Se pueden encontrar ejemplos de SMA abiertos como las aplicaciones de *e-commerce* y los sistemas de agentes de información [6]. En estos casos, los agentes adoptan temporalmente roles de comprador o de tipos similares. También existen algunos trabajos que abordan la problemática de los sistemas abiertos mediante la utilización de agentes internos que representan a los agentes externos que solicitan participar en la organización [8]. Con lo cual, los agentes externos no participan de forma directa en la organización y así se pueden evitar muchos problemas de seguridad y de control.

2.2. Servicios Web y Sistemas Multiagente

A consecuencia del espectacular crecimiento de Internet han surgido nuevas tecnologías como los servicios web, los cuales ofrecen distintas funcionalidades aplicables a muchos campos de la computación. Los servicios web se pueden definir como un conjunto de tecnologías, protocolos y estándares combinados para interoperar en la web o intercambiar datos entre aplicaciones. También pueden ser definidos, desde otro punto de vista [14], como aplicaciones auto-contenidas y modulares que pueden ser descritas, publicadas, localizadas e invocadas en una red, normalmente la web.

Para los servicios web existen distintas especificaciones estándar, algunas frecuentemente usadas son: SOAP (***S**imple **O**bject **A**ccess **P**rotocol*) como protocolo de comunicación, WSDL (***W**eb **S**ervice **D**escription **L**anguage*) como lenguaje de descripción de servicios, y UDDI (***U**niversal **D**iscovery **D**escription and **I**ntegration*) como directorio para registro de descripciones de servicios. Los lenguajes de descripción de servicios más utilizados actualmente son WSDL y OWL-S. A continuación se comentan algunos detalles de ambos lenguajes.

El lenguaje WSDL [4] está basado en XML y ha sido desarrollado por IBM y Mi-

crosoft para describir servicios web. Este lenguaje intenta separar los servicios de los formatos de datos y de los protocolos concretos que se utilicen en la implementación. Así pues, define asociaciones entre las descripciones abstractas y sus implementaciones específicas. Sin embargo, WSDL no permite descripciones semánticas ya que se centra más en los *mappings* de los servicios. Además, aunque incluye tipos de entrada y salida, no soporta definición de restricciones. Por lo tanto, al tener esta falta de expresividad su uso más adecuado es la descripción del acceso a los servicios.

OWL-S [7] es un lenguaje de ontologías basado en XML. En concreto es una ontología en OWL para describir servicios web. La intención de OWL-S consiste en hacer interpretables los servicios web por parte de los programas. Esto supone una descripción del servicio web con información suficiente para que de forma automatizada se puedan descubrir, invocar, componer y monitorizar la ejecución de servicios. Al ser una ontología OWL, este lenguaje tiene las aportaciones de los contenidos web descritos en OWL. Concretamente tiene una semántica bien definida, con lo que permite definir objetos y relaciones entre ellos, incluyendo clase, relaciones de subclase, restricciones de cardinalidad, etc. Además, incluye el formato de tipos de XML. La ontología tiene como elemento principal la clase *Service* y consta de tres partes o sub-ontologías:

- *ServiceProfile*: describe qué hace el servicio. Es como una entrada de páginas amarillas para un servicio. Especifica la funcionalidad que proporciona, las entradas y salidas; y las precondiciones y efectos.
- *ServiceModel*: especifica el modelo de procesos del servicio, es decir, cómo se ejecuta. Su función es facilitar la composición, ejecución y monitorización automática de servicios.
- *ServiceGrounding*: describe los detalles de cómo se accede al servicio. Normalmente se especifica un protocolo de comunicación, formatos de los mensajes y otros detalles de implementación de las comunicaciones.

Cabe destacar que un servicio sólo puede tener un *ServiceModel* pero puede tener varios *ServiceProfiles* y *ServiceGroundings*.

En la actualidad, las arquitecturas orientadas a servicios ofrecen un marco ideal para aportar flexibilidad e independencia a los SMA abiertos. Además, el área de la

computación orientada a servicios y los SMA se están acercando cada vez más. Desde el punto de vista de los servicios web, éstos pueden ser una buena solución para las tareas que se realicen cuando son necesitados de forma estática. El problema surge cuando aparece la necesidad de trabajar en un entorno cambiante donde aparecen nuevos servicios y deben ser descubiertos, compuestos o adaptados a diferentes ontologías. Este problema puede ser resuelto mediante la combinación de los servicios web y los SMA, ya que estos últimos ofrecen inteligencia y capacidad organizativa, además de la automatización del descubrimiento y composición de servicios. Por lo tanto, como ya se ha comentado, los SMA y la computación orientada a servicios se adaptan perfectamente debido al uso de ontologías, modelos de proceso, coreografía y directorios de descripciones.

La idea de integrar organizaciones y agentes como proveedores de servicios ya ha sido estudiada por diversos grupos de investigación. La causa de esto es el creciente interés de la industria en el uso de servicios web estándar para crear sistemas complejos y reutilizables. Así pues, el principal esfuerzo se concentra en integrar los agentes con los servicios web permitiendo redirección, agregación, integración o propósitos administrativos en los servicios [12].

En base a esta idea existen dos lineas de investigación: integración directa de servicios web y agentes mediante intercambio de mensajes, y la consideración de agentes como *matchmakers* para descubrimiento y composición de servicios.

Un *matchmaker* [14] es un agente que empareja solicitantes de servicios con proveedores mediante la equiparación de solicitudes con servicios anunciados por agentes. A diferencia de los mediadores y *brokers*, un *matchmaker* simplemente devuelve una lista (valorada) ordenada de agentes que proporcionan el servicio solicitado.

La integración directa de servicios web y agentes se ha estudiado desde tres puntos de vista distintos:

- Utilización de entidades intermediarias entre los agentes y los servicios web. Dentro de esta linea se encuentran la arquitectura *Web Service Integration Gateway Service* (WSIG) [13] y AgentWeb Gateway [18].
- Agentes con interfaz de comunicación directa con los servicios web. Con este

punto de vista se pueden encontrar WSDL2JADE [23] y WS2JADE [15].

- Permitir usar los servicios de los agentes a los clientes de servicios web. Web Service Agent Integration (WSAI) [7] es un ejemplo de este planteamiento.

La consideración de agentes como *matchmakers* es la línea de investigación principal para la idea de integrar agentes y servicios web [21]. En [2] se presenta una propuesta que complementa los métodos existentes considerando los tipos de interacciones y roles que pueden ser usados por los servicios. Otras propuestas [17] usan los datos de la experiencia de los agentes para evaluar sus expectativas sobre un proveedor de servicios y toman decisiones usando su propio criterio y su estado actual. En [20] se presenta un protocolo que consiste en dos complejas tareas de razonamiento: descubrimiento y mediación.

El proceso de *matchmaking* se considera ocasionalmente como una consulta a un conjunto de datos estáticos disponibles, sin embargo, esta visión es demasiado simple. Por otra parte, los protocolos de negociación son otro mecanismo usado normalmente. En este caso, los agentes participantes negocian sobre las propiedades de los servicios que ofrecen y solicitan para conseguir acuerdos y contratos entre ellos [5].

Finalmente, cabe destacar que debido a la falta de productos que ofrecieran una correcta integración entre las tecnologías de los servicios web y los sistemas multiagente nació la propuesta THOMAS. Esta plataforma ofrece un sistema que fusiona ambas tecnologías de forma satisfactoria y permite gran flexibilidad en su utilización. En el siguiente capítulo se presenta la plataforma THOMAS y sus características principales.

[7] http://wsai.sourceforge.net/

Capítulo 3

THOMAS

En el presente capítulo se describe la plataforma THOMAS con la intención de dar al lector una visión general de ésta. Así pues, se pretende proporcionar los conocimientos necesarios para entender la aplicación de gestión de servicios turísticos desarrollada sobre dicha plataforma. Por lo tanto, se comentan tanto los aspectos teóricos como los técnicos.

3.1. Descripción general

THOMAS [10] [3] [22] (Me**TH**ods, Techniques and Tools for **O**pen **M**ulti-**A**gent **S**ystems) es una plataforma de sistemas multiagente abiertos basada en organizaciones y servicios web. Ha sido desarrollada por el Grupo de Tecnología Informática - Inteligencia Artificial del Departamento de Sistemas Informáticos y Computación, de la Universidad Politécnica de Valencia.

La arquitectura de THOMAS consiste básicamente en un conjunto de servicios modulares. Aunque está basada en la arquitectura FIPA, THOMAS expande las capacidades de dicha arquitectura para gestionar organizaciones. Por lo tanto, se ha incluido un nuevo módulo para conseguir este objetivo, con la redefinición del ***D****irectory* ***F****acilitator* de FIPA. Con lo cual, puede manejar los servicios de una manera más elaborada siguiendo las guías para las arquitecturas orientadas a servicios (***S****ervice* ***O****riented* ***A****rchitectures, SOA*). Los servicios son lo más importante de la plataforma THOMAS ya que para tener acceso a la infraestructura, los agentes disponen de un conjunto de servicios incluidos en diferentes módulos o componentes. Los principales componentes son:

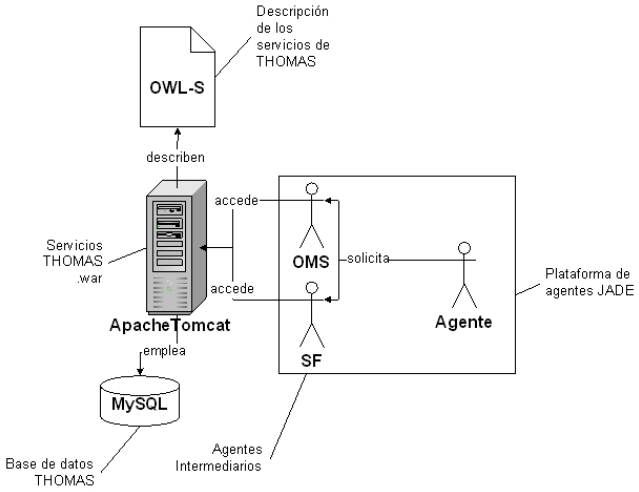

Figura 3.1: Estructura general de la plataforma THOMAS

- Agente intermediario SF (***S**ervice **F**acilitator*). Ofrece servicios simples y complejos para los agentes activos y las organizaciones. Su funcionalidad básica es ofrecer un servicio de páginas amarillas para búsqueda y otro de páginas verdes como descriptor de servicios.
- Agente intermediario OMS (***O**rganization **M**anagement **S**ystem*). Es el responsable de la gestión de las organizaciones y de sus entidades. Con lo cual, permite crear y gestionar cualquier organización.
- Kernel de la plataforma (***P**latform **K**ernel, PK*). Se encarga de los servicios básicos de gestión de una plataforma de agentes.

En los siguientes apartados serán descritos detalladamente estos componentes de la plataforma THOMAS. Por otra parte, en la figura 3.1 se puede observar la estructura general de la plataforma.

3.2. Agente intermediario SF

El SF es un mecanismo mediante el cual los agentes y las organizaciones pueden ofrecer y encontrar servicios. Provee un soporte en el que las entidades autónomas pueden registrar descripciones de servicios como entradas de directorio. También actúa

como pasarela de acceso a la plataforma THOMAS. Esto es manejado de forma transparente pero teniendo en cuenta la seguridad y la gestión de derechos de acceso.

Este agente intermediario puede encontrar servicios mediante la búsqueda de un perfil dado o de un objetivo que puede ser conseguido ejecutando el servicio. Además, también actúa como gestor de páginas amarillas, con lo que puede encontrar qué entidades proveen un servicio dado.

Un servicio representa la interacción entre dos entidades que se comunican entre sí. Éste ofrece las capacidades para satisfacer un objetivo dado y suele tener algunas precondiciones que necesitan cumplirse para su ejecución. A parte de los parámetros funcionales del servicio, también pueden existir algunos en la descripción de éste que no lo sean, como por ejemplo: calidad del servicio y protocolos de seguridad.

En el caso de la plataforma THOMAS, se utilizan los protocolos de comunicación FIPA (***F****oundation for* ***I****ntelligent* ***P****hysical* ***A****gents*), los cuales establecen los mecanismos para estandarizar las interacciones. Por lo tanto, cada servicio tiene asociado un protocolo y en los casos en los que es necesario la ejecución de una cadena de protocolos quedan marcados como "complejos". Por otra parte, teniendo en cuenta que THOMAS trabaja con servicios semánticos, la ontología usada en el servicio es muy importante. Así pues, cuando se accede a la descripción de un servicio, cualquier entidad tiene la información necesaria para interactuar con el mismo y hacer una aplicación que lo utilice. Además, la descripción se puede utilizar para servicios precompilados, donde el modelo de proceso se compone de una secuencia de servicios elementales que serán ejecutados, en vez de los procesos internos de ese servicio.

En general, un servicio puede ser ofrecido por más de un proveedor en el sistema, por lo tanto, cada servicio tiene asignada una lista de proveedores. Todos los proveedores pueden ofrecer copias exactas del servicio, es decir, comparten una implementación común. Por otra parte, pueden compartir sólo la interfaz y cada uno implementar el servicio de una forma. Esto se consigue en THOMAS teniendo el perfil del servicio separado del proceso.

Un servicio está definido como una tupla (*sID, goal, prof, proc, ground, ont*):

- sID: identificador único del servicio.

- goal: es el objetivo del servicio y provee un primer nivel de abstracción para la composición del servicio.

- prof: es el perfil del servicio, la descripción de éste en referencia a sus entradas, salidas, precondiciones y efectos. También engloba los atributos no funcionales de una forma legible para aquellos agentes que están buscando información. Este tipo de representación incluye una descripción de lo que consigue el servicio, las restricciones de su aplicabilidad, la calidad y los requisitos que deben satisfacer los clientes para usarlo.

- proc: describe cómo tiene que usar el servicio un cliente. Especifica el contenido semántico para usar el servicio, las situaciones en que es obtenido y cuando es requerido, así como los procesos paso a paso para obtener esos resultados. En conclusión, especifica cómo llamar al servicio y qué pasa cuando es ejecutado.

- ground: describe detalladamente cómo un agente puede acceder al servicio. El *grounding* especifica un protocolo de comunicación, el formato de los mensajes, el puerto de contacto y otros detalles. Está implementado usando OWL-S estándar extendido con los protocolos de FIPA.

- ont: es la ontología que da significado a todos los elementos del servicio. OWL-DL es el lenguaje utilizado.

Esta proposición está basada en la especificación para servicios web semánticos OWL-S, extendida cuando es necesario para aumentar su funcionalidad. Los objetivos, precondiciones y efectos (o postcondiciones) son fórmulas lógicas.

La tupla definida anteriormente está implementada en dos partes: el *servicio abstracto*, general para todos los proveedores; y el *servicio concreto*, con los detalles de la implementación. Así pues, los servicios son almacenados en el sistema divididos en estas dos partes: el *perfil del servicio* (que representa la especificación del servicio abstracto) y un conjunto de *procesos del servicio* (que detallan el servicio concreto). Por lo tanto, los servicios en THOMAS están implementados como se detalla en la siguiente tupla, en la que los elementos son especificaciones extendidas OWL-S:

<ServiceID, Providers, ServGoal, ServProfile>

Providers ::=<ProvIDList, ServImpID, ServProcess, ServGround>$^+$

ProvIDList ::= ProviderID$^+$

donde:

- *Providers* es un conjunto de tuplas compuestas por una lista de identificadores de proveedores (*ProvIDList*), el identificador de la implementación del servicio (*ServImpID*), el modelo de especificación del proceso del servicio (*ServProcess*), y su instanciación particular (*ServGround*).
- *ProvIDList* mantiene un lista de identificadores de proveedores del servicio.

El SF dispone de un conjunto de servicios estándar (meta-servicios) para gestionar los servicios proveídos por las organizaciones o por agentes individuales. Estos meta-servicios también tienen que ser usados por el resto de los componentes de THOMAS (OMS y PK) para publicitar los suyos. Los meta-servicios pueden ser clasificados en tres tipos:

- **Registro**: permiten añadir, modificar y eliminar servicios del directorio SF.
- **Alcance**: se utilizan para gestionar la asociación entre los proveedores y sus servicios.
- **Descubrimiento**: su funcionalidad consiste en la búsqueda y composición de servicios como respuesta a las peticiones del usuario. Pueden ser tan complejos que pueden ser delegados a componentes especializados.

En la tabla 3.1 se puede encontrar la relación de los meta-servicios del SF.

3.3. Agente intermediario OMS

El agente intermediario OMS (***O**rganization **M**anagement **S**ystem*) se encarga de la gestión de las organizaciones. Esto incluye la especificación y la administración de

Tipo	Meta-servicio	Descripción
Registro	RegisterProfile	Crea una nueva descripción de servicio (perfil)
	RegisterProcess	Crea una implementación particular (proceso) para un servicio
	ModifyProfile	Modifica un perfil de servicio existente
	MofifyProcess	Modifica un proceso de servicio existente
	DeregisterProfile	Elimina una descripción de servicio
Alcance	RemoveProvider	Elimina un proveedor de un proceso de servicio
Descubrimiento	SearchService	Busca un servicio (o composición de servicios) que satisface los requisitos del usuario
	GetProfile	Obtiene la descripción (perfil) de un servicio específico
	GetProcess	Obtiene la implementación (proceso) de un servicio específico

Tabla 3.1: Meta-servicios del agente intermediario SF

los componentes estructurales (roles, unidades y normas) y de los componentes de ejecución (agentes participantes, roles que éstos juegan y unidades organizativas activas).

Las organizaciones están estructuradas mediante *unidades organizativas* que representan grupos de entidades (agentes u otras unidades). Los componentes que forman una unidad persiguen un objetivo común. Las unidades organizativas tienen una topología interna que impone control y restricciones a las relaciones entre los agentes.

Existe una unidad llamada "virtual" en la plataforma THOMAS que ha sido definida para representar el "mundo" del sistema en el que los agentes participan por defecto. Las organizaciones son creadas dentro de esta unidad y éstas, a su vez, pueden estar compuestas de más unidades. Cabe destacar que los roles se definen dentro de cada unidad y representan la funcionalidad requerida para alcanzar el objetivo de la unidad. Además pueden tener normas asociadas para controlar las acciones de los roles (i.e. qué servicios pueden solicitar u ofrecer los agentes que están jugando un rol determinado; permisos para acceder a determinados recursos). En conclusión, los agentes pueden adoptar roles dinámicamente dentro de las unidades, con lo cual el OMS controla todo

este proceso y cuáles son las entidades que juegan un rol en cada instante.

El agente intermediario OMS usa la siguiente información:

- ***UnitList***: guarda las unidades existentes junto con sus objetivos, topología y unidad superior.
- ***RoleList***: almacena la lista de roles en cada unidad y sus atributos (accesibilidad, visibilidad, posición y herencia). El atributo *accesibilidad* indica si un rol puede ser adoptado por un agente. *Visibilidad* indica si los agentes pueden obtener información sobre el rol. *Posición* concreta si es un supervisor, un subordinado o miembro de la unidad. La *herencia* indica el rol padre.
- ***NormList***: guarda las normas definidas en el sistema.
- ***EntityPlayList***: describe la asociación <*entidad*, *unidad*, *rol*>. Es decir, qué roles han sido adoptados por una entidad (agente) dentro de cada unidad.

Los servicios que ofrece el OMS se clasifican como estructurales y dinámicos. Los servicios estructurales son los que modifican la estructura y la normativa de la organización. Por su parte, los dinámicos permiten a los agentes entrar o abandonar la organización de forma dinámica, así como la adopción de roles. Se puede observar un lista completa de los servicios del OMS en la tabla 3.2.

3.4. Kernel de la plataforma

El Kernel de la plataforma es el encargado de proporcionar los servicios usuales requeridos en una plataforma multiagente. Así pues, es el responsable de gestionar el ciclo de vida de los agentes incluidos en las distintas organizaciones y también permite tener un canal de comunicación para facilitar la interacción entre entidades. Además, el Kernel también ofrece conectividad segura y los mecanismos necesarios para proporcionar interconectividad entre dispositivos.

Los servicios ofrecidos deben ser heredados de FIPA con algunas modificaciones. Los servicios del Kernel necesitados en una infraestructura THOMAS se clasifican en cuatro tipos: (i) *Registro*: permiten añadir, modificar y eliminar agentes nativos de la

Tipo	Subtipo	Meta-servicio	Descripción
Estructural	Registro	RegisterRole	Crea un nuevo rol dentro de una unidad
		RegisterNorm	Incluye una nueva norma dentro de una unidad
		RegisterUnit	Crea una nueva unidad dentro de una organización
		DeregisterRole	Elimina un rol de una unidad
		DeregisterNorm	Elimina una norma específica
		DeregisterUnit	Elimina una unidad de una organización
	Información	InformAgentRole	Indica los roles adoptados por un agente
		InformMembers	Indica las entidades que son miembros de una unidad
		QuantityMembers	Provee el número de miembros actuales en una unidad
		InformUnit	Provee la descripción de una unidad
		InformUnitRoles	Indica los roles definidos dentro de una unidad
		InformRoleProfiles	Indica los perfiles asociados a un rol
		InformRoleNorms	Indica las normas dirigidas a un rol
Dinámico	Compuesto	AcquireRole	Solicita la adopción de un rol concreto dentro de una unidad
		LeaveRole	Solicita el abandono de un rol
		Expulse	Fuerza a un agente a abandonar un rol específico

Tabla 3.2: Meta-servicios del agente intermediario OMS

Tipo	Servicio	Descripción
Registro	Register	Registra un nuevo agente en la plataforma
	Deregister	Elimina el registro de un agente
	Update register	Modifica la información de un registro de agente
Descubrimiento	Agent Search	Solicita información sobre un agente registrado
	Get Description	Obtiene la descripción de la plataforma
Gestión	Suspend	Suspende la ejecución de un agente concreto
	Activation	Activa la ejecución de un agente concreto suspendido
Comunicación	Send	Envía un mensaje a cualquier agente de la plataforma o de fuera de ella

Tabla 3.3: Servicios del Kernel de la plataforma (PK)

plataforma; (ii) *Descubrimiento*: servicios para obtener alguna información sobre los agentes activos de la plataforma; (iii) *Gestión*: servicios para controlar el estado de activación de los agentes de la plataforma; (iv) *Comunicación*: servicios para comunicar agentes en la plataforma y fuera de ella. La relación completa de los servicios del Kernel se puede observar en la tabla 3.3.

Puesto que la propuesta THOMAS no persigue desarrollar un nuevo kernel para plataformas multiagente, los servicios requeridos a nivel de kernel pueden ser proporcionados por distintas plataformas conocidas. Así pues, se debe elegir una plataforma que cumpla los servicios mínimos comentados anteriormente y además tiene que ofrecer un mecanismo de comunicación que proporcione compatibilidad con FIPA. La plataforma elegida por la propuesta THOMAS para facilitar todo esto es JADE.

3.5. Implementación de la plataforma THOMAS

El marco de ejecución de THOMAS [22] está basado en JADE[1] (***J**ava **A**gent **DE**velopment Framework*), que es una plataforma de código abierto y libre para el

[1]http://jade.tilab.com/

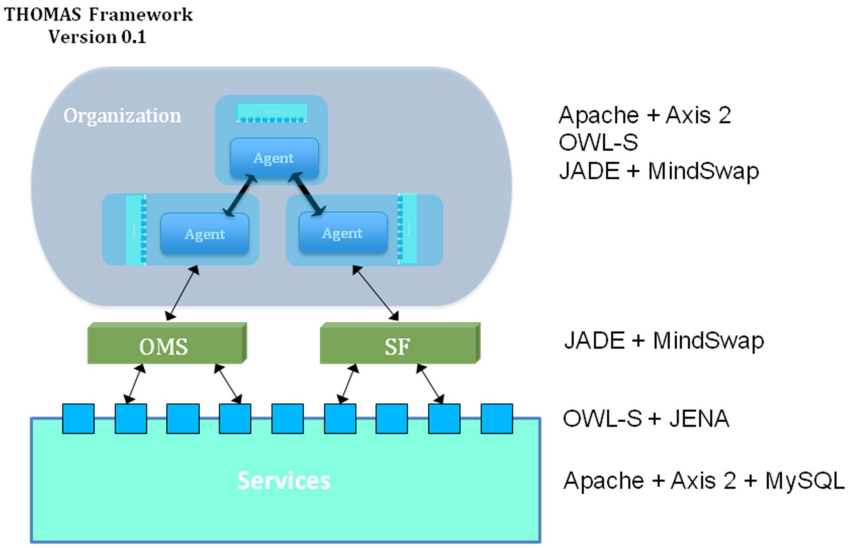

Figura 3.2: Estructura de la implementación de la plataforma THOMAS

desarrollo de sistemas multiagente, implementada completamente en lenguaje Java. JADE simplifica la creación de sistemas multiagente ya que cumple con las especificaciones FIPA[2] (***F**oundation for **I**ntelligent **P**hysical **A**gents*) y proporciona un conjunto de herramientas gráficas que ayudan en las fases de desarrollo y depuración. Además, la plataforma permite la distribución entre distintas máquinas, incluso usando distintos sistemas operativos, y la configuración puede ser controlada con una interfaz de usuario remota.

Los servicios de THOMAS están implementados como servicios web semánticos. Se usa Apache Axis2/Java[3] como motor de los servicios web y cada servicio tiene una descripción semántica en OWL-S[4][4] y su correspondiente en WSDL[7]. En el documento WSDL, las operaciones y los mensajes están descritos de forma abstracta y después se concreta un protocolo de red y el formato de los mensajes. El documento OWL-S detalla las propiedades y capacidades de un servicio web de forma interpretable por el ordenador. Esta descripción facilita la automatización de las tareas del servicio web, incluyendo descubrimiento, ejecución, composición e interoperabilidad. En la figura 3.2 se puede observar la estructura de la implementación de la plataforma THOMAS.

Todos los meta-servicios del SF y del OMS están registrados en el SF, y también

[2]http://www.fipa.org/
[3]http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html
[4]http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

los servicios de agentes y organizaciones. Para cada servicio registrado en el SF deben existir dos documentos OWL-S, uno con la descripción del perfil y otro con la del proceso y el *grounding*. La razón de tener dos documentos es para evitar redundancia, ya que en el caso que exista más de un proveedor con distintas implementaciones, el servicio concreto tendría un "perfil" y varios "procesos" (uno por cada implementación distinta).

Los componentes OMS y SF están implementados como agentes JADE. La lógica de estos agentes se ha desarrollado usando la API OWL-S de Mindswap[5], que a su vez, utiliza una API de Java para programar el acceso a la lectura, ejecución y escritura de las descripciones de servicios OWL-S. Cuando el agente intermediario OMS o SF recibe un mensaje de petición FIPA de un cliente, utiliza la API OWL-S para acceder a la descripción del servicio en OWL y ejecuta el correspondiente servicio web. La ejecución de un servicio implica el acceso a la información del proceso incluida en la descripción del servicio.

El SF debe ser capaz de registrar y gestionar los servicios proveídos por agentes externos. Para que estos servicios sean entendidos por una máquina, la información semántica se añade con descripciones OWL-S y la ontología especificada en OWL para los parámetros. Para manejar toda la información semántica en OWL se usa JENA[6]. Entre otras cosas, JENA ofrece una interfaz para guardar la información de forma persistente en una base de datos. Además, también se usa SPARQL[7] como lenguaje de consultas, con el cual se toma la descripción de lo que la aplicación quiere, en forma de consulta, y se obtiene la información solicitada. En la figura 3.3 se puede observar la interacción entre el SF, un agente cliente y los servicios web.

Como se ha comentado en la sección correspondiente al OMS, los meta-servicios que ofrece este componente de la plataforma se ocupan de la gestión de las organizaciones y también dispone de servicios informativos. La implementación de estos meta-servicios considera el mantenimiento de la estructura organizativa y las normas que regulan el acceso. Así pues, cuando se hace una petición de un servicio al OMS el proceso seguido es el siguiente:

[5]http://www.mindswap.org/2004/owl-s/api/
[6]http://jena.sourceforge.net/ontology/index.html
[7]http://www.w3.org/TR/rdf-sparql-query/

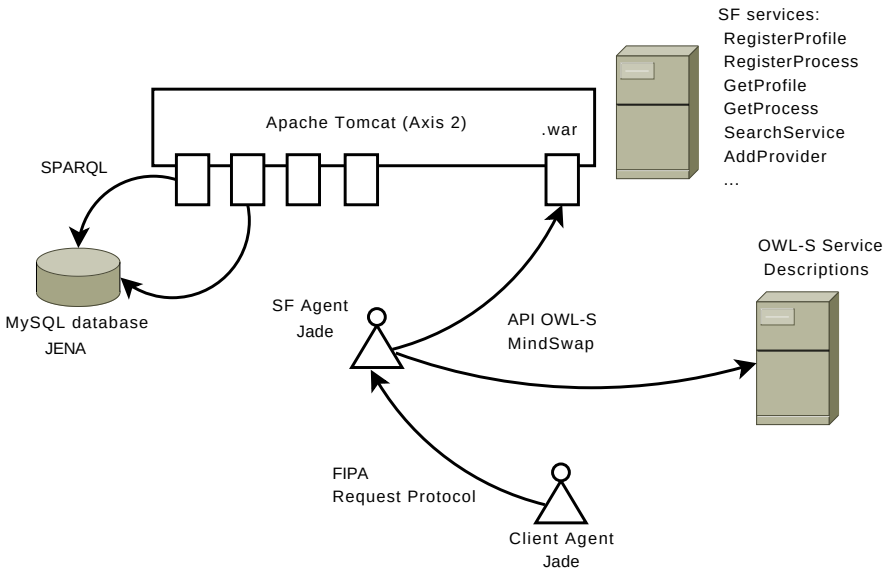

Figura 3.3: Interacción del SF con un agente cliente y los servicios web

1. Validación de las entradas del meta-servicio.
2. Análisis del contexto normativo para determinar si el agente cliente tiene una posición en la organización que le permita hacer la petición.
3. Si corresponde, el OMS provee el meta-servicio pedido.
4. Si el estado organizativo cambia como resultado del servicio ejecutado, se actualiza adecuadamente.

Para la comprobación de la compatibilidad del objetivo del meta-servicio que se ejecuta con las normas existentes, y para la actualización posterior a los cambios sobre la organización que puede provocar un meta-servicio, existen dos procesos. Éstos están implementados como servicios internos del OMS y son ejecutados por un proceso general de gestión de la normativa.

Capítulo 4

Diseño e Implementación de la Aplicación

En este capítulo se explicarán detalladamente todos los aspectos de la aplicación desarrollada. En primer lugar, se mostrarán de forma teórica las implicaciones de la ejecución de los meta-servicios de la plataforma THOMAS y de los servicios turísticos implementados. Todo ello por medio de posibles escenarios que se pueden presentar durante el funcionamiento de la aplicación. Seguidamente, se explicará la implementación del sistema desde el aspecto de la interfaz gráfica de usuario y de los detalles de la implementación de la aplicación. Así pues, se comentará la interfaz gráfica de usuario detallando los elementos por los que está formada y sus posibles usos. Más adelante, se detallarán aspectos técnicos de la implementación para poder tener una idea de cómo se han desarrollado los procesos internos de la aplicación a nivel de código fuente. Finalmente, se hará una validación de la plataforma THOMAS a través de la aplicación desarrollada, comprobando y mostrando al lector que efectivamente se pueden reproducir los escenarios y que se cumple el objetivo de ejecutar servicios de gestión turística.

4.1. Caso de Estudio

Con el objetivo de mostrar las funcionalidades de la plataforma THOMAS, se ha diseñado una aplicación para gestionar servicios turísticos. Esta aplicación muestra un ejemplo de agencia de viajes donde existen clientes (particulares, empresas o agencias de viajes) y proveedores (cadenas de hoteles y compañías aéreas). Con lo cual, los servicios que se pueden ofrecer y solicitar quedan acotados al sector turístico. Cada proveedor puede ofrecer los servicios libremente respetando las restricciones que im-

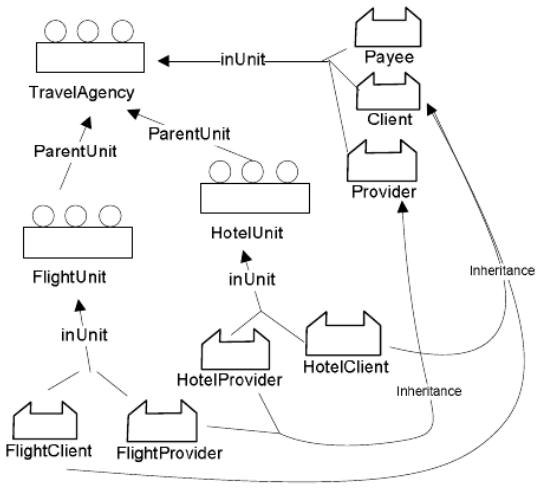

Figura 4.1: Estructura de la organización *Travel Agency*

pone la plataforma.

Así pues, es necesario crear una organización virtual llamada *Travel Agency* (que será una unidad dentro de la unidad principal "virtual" de la plataforma) donde agrupar todos los roles relacionados con la agencia de viajes. En la organización *Travel Agency* hay tres roles disponibles: *Customer*, *Provider* y *Payee*. El rol *Customer* es el que solicita servicios, como búsqueda y reserva de hoteles y vuelos, siendo el rol *Provider* el que ofrece estos servicios. El otro rol presente en la organización es *Payee*, el cual representa a una entidad financiera que se encarga de ofrecer un mecanismo de pago. Sin embargo, este es un rol interno de la organización y no se tiene en cuenta en la aplicación desarrollada. Por otra parte, dentro de la organización *Travel Agency* hay dos unidades llamadas *HotelUnit* y *FlightUnit*, que están dedicadas a agrupar agentes interesados en ofrecer o solicitar servicios relacionados con hoteles y vuelos respectivamente. En este caso, dentro de *HotelUnit* existen dos roles, *HotelCustomer* y *HotelProvider*, que son una especialización de los roles generales *Customer* y *Provider* de la *Travel Agency*. De la misma forma, en la unidad FlightUnit están presentes los roles *FlightCustomer* y *FlightProvider*. La representación de toda la estructura de la organización se puede observar en la figura 4.1 y en las tablas 4.1 y 4.2.

Por simplicidad todos los roles y las unidades se registran al iniciar la aplicación. Esto se hace introduciéndolos directamente en la base de datos de la plataforma. La razón es tener todos los roles y unidades disponibles cuando se inicia el sistema para ahorrar al usuario todo el proceso de registro. Además, cuando se inicia la aplicación

Nombre de unidad	**Unidad superior**	**Objetivo**	**Tipo**
Virtual	——	——	Flat
TravelAgency	Virtual	ReserveTravel	Congregation
HotelUnit	TravelAgency	ReserveHotel	Flat
FlightUnit	TravelAgency	ReserveFlight	Flat

Tabla 4.1: Contenido inicial de la *UnitList*

Rol	**Unidad**	**Visibilidad**	**Posición**	**Acceso**	**Herencia**
Customer	TravelAgency	Public	Member	External	——
Provider	TravelAgency	Public	Member	External	——
Payee	TravelAgency	Private	Member	External	——
HotelCustomer	HotelUnit	Public	Member	External	Customer
HotelProvider	HotelUnit	Public	Member	External	Provider
FlightCustomer	FlightUnit	Public	Member	External	Customer
FlightProvider	FlightUnit	Public	Member	External	Provider

Tabla 4.2: Contenido inicial de la *RoleList*

ya están presentes los agentes proveedores y cliente, con lo cual, se puede considerar que la organización ya está creada. De todas formas, en cualquier momento se pueden registrar o eliminar roles y unidades mediante las opciones disponibles en las barras de menú de los agentes.

Para la aplicación desarrollada, se han implementado un conjunto de servicios turísticos con el objetivo de demostrar la viabilidad de la plataforma THOMAS para la gestión de cualquier tipo de servicio web en el entorno de un sistema multiagente. Estos servicios de gestión turística intentan reflejar la realidad de lo que ofrecen las agencias de viajes actuales, como es la búsqueda y reserva de hoteles y vuelos. Los servicios desarrollados son: *Search Hotel*, *Reserve Hotel*, *Search Flight* y *Reserve Flight*. Estos servicios están destinados a ser proveídos por los agentes proveedores presentes en la aplicación, es decir, *HotelProvider* y *FlightProvider*. En las tablas 4.3 y 4.4 se encuentra toda la información referente a ellos: descripción, roles que deben poseer el cliente y el proveedor, entradas y salidas.

Para entender el funcionamiento de la aplicación desarrollada es necesario aclarar

HotelUnit

Servicio: SearchHotel	**Descripción:** Buscar hoteles en una ciudad
Rol Cliente: HotelCustomer	**Rol Proveedor:** HotelProvider
Entradas:	**Salidas:**
Country:string	HotelChain:string
City:string	HotelName:string
Category:integer	RoomRate:float
	Adress:string
Servicio: ReserveHotel	**Descripción:** Reservar hoteles
Rol Cliente: HotelCustomer	**Rol Proveedor:** HotelProvider
Entradas:	**Salidas:**
Company:string	RsvTicket:Reserve
Location:string	Price:float
Date:time	IBAN:string
NumRsv:integer	
Nights:integer	

Tabla 4.3: Servicios de la organización *Travel Agency*, unidad *HotelUnit*

FlightUnit

Servicio: SearchFlight	**Descripción:** Buscar vuelos
Rol Cliente: FlightCustomer	**Rol Proveedor:** FlightProvider
Entradas:	**Salidas:**
CountryFrom:string	FlightCompany:string
CityFrom:string	FlightType:string
CountryTo:string	Price:float
CityTo:string	HourDept:time
Date:time	HourArr:time
	NumConnect:integer
Servicio: ReserveFlight	**Descripción:** Reservar vuelos
Rol Cliente: FlightCustomer	**Rol Proveedor:** FlightProvider
Entradas:	**Salidas:**
Company:string	RsvTicket:Reserve
Location:string	Price:float
Date:time	IBAN:string
NumRsv:integer	
CityTo:string	
HourDept:time	
HourArr:time	

Tabla 4.4: Servicios de la organización *Travel Agency*, unidad *FlightUnit*

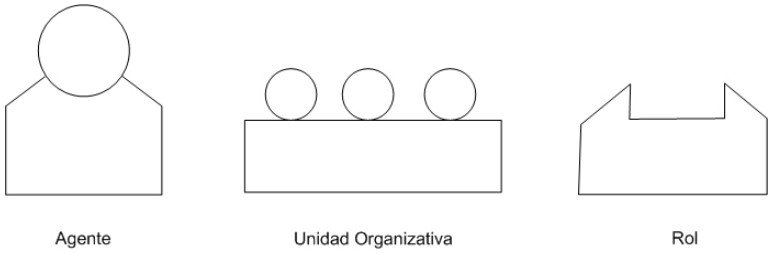

Figura 4.2: Notación de los elementos utilizados en los diagramas

todos los aspectos relacionados con la plataforma y los meta-servicios de que dispone. A lo largo de esta sección se explicarán de forma teórica los distintos escenarios que se pueden ejecutar en la aplicación con el fin de mostrar la funcionalidad de ésta y su adecuación a la plataforma THOMAS. En las imágenes que representan los escenarios se utiliza la notación que se puede observar en la figura 4.2. Para obtener información adicional acerca de esta notación se puede consultar [1].

Seguidamente, se irán comentando los escenarios más comunes que se pueden dar en la aplicación. Se ha optado por mostrar ejemplos de forma sencilla, sin realizar demasiadas acciones ya que la cantidad de combinaciones entre meta-servicios y de situaciones que se pueden dar es muy amplia. Los escenarios que se explicarán son los siguientes:

1. Registro de un agente
2. Registro de un proveedor de hoteles
3. Registro de un proceso
4. Registro de un cliente
5. Petición de servicio
6. Creación de una nueva unidad
7. Creación de nuevos roles
8. Expulsión de un agente
9. Registro de una norma

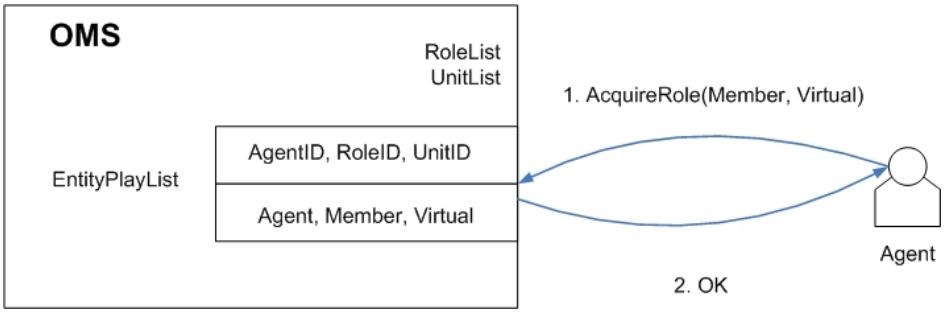

Figura 4.3: Escenario de registro de un agente

4.1.1. Registro de un agente

El primer paso que debe hacer un agente antes de poder utilizar cualquier servicio y empezar a interactuar dentro de la plataforma es entrar a formar parte de ella. Para ello es necesario adquirir el rol *member* dentro de la unidad llamada *virtual*. En la plataforma THOMAS se ha establecido este rol en esta unidad para representar el "mundo" del sistema en el que existen los agentes por defecto. Además, dentro de esta unidad se crean todas las unidades organizativas ya que así lo establece el OMS.

Así pues, para registrar un agente en la plataforma se utiliza el meta-servicio *Acquire Role* del OMS. La solicitud que debe hacer cualquier agente interesado en formar parte del sistema es un *Acquire Role* para obtener el rol *member* dentro de la organización *virtual*. Esta petición se realiza enviando un mensaje FIPA-Request al OMS solicitando el meta-servicio *Acquire Role* con los parámetros *member* y *virtual*. El OMS se encarga de controlar que se cumplan todas las restricciones (existencia de los identificadores del rol y de la unidad, compatibilidad de roles, etc) y registra al agente como miembro de la plataforma THOMAS. La representación de este escenario se encuentra en la figura 4.3.

En la aplicación desarrollada, los agentes que forman parte del ejemplo se registran automáticamente al lanzar la aplicación, puesto que se inician tanto los agentes intermediarios de la plataforma como éstos. Se ha realizado de forma automática ya que es lógico que los agentes que empiezan con el ejemplo se encuentren dentro del sistema. Sin embargo, no se elimina la posibilidad de que llegue cualquier agente externo y pueda entrar en la plataforma, ya que simplemente debería llamar al servicio *Acquire Role* para solicitar adquirir el rol *member* en la unidad *virtual*.

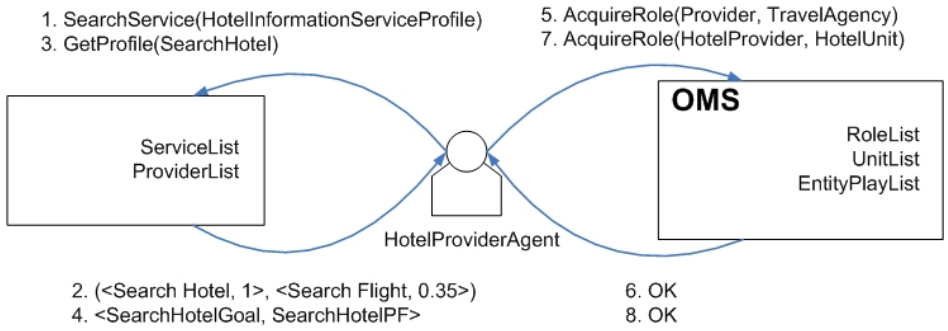

Figura 4.4: Escenario de registro de un proveedor de hoteles

4.1.2. Registro de un proveedor de hoteles

En este escenario se representa cómo se registra un proveedor de hoteles (figura 4.4). Debido a la estructura de la *TravelAgency*, para incluir a un agente como proveedor de hoteles es necesario adquirir dos roles: *Provider* de la *TravelAgency* y *HotelProvider* de la *HotelUnit*. El primero de ellos representa a un proveedor en la unidad general de la agencia de viajes. Por su parte, el rol HotelProvider es una especialización del primero y está destinado a representar agentes que ofrezcan servicios concretos de gestión de hoteles.

El primer paso en este escenario consiste en averiguar si existe alguna descripción de servicio con un perfil muy parecido al que se pretende ofrecer. Así pues, se utiliza el meta-servicio del SF *Search Service* para obtener esta información (mensaje 1). En este caso, existen dos servicios registrados que coinciden con el perfil que pretende ofrecer el agente, clasificados según un rango específico (mensaje 2). Se elige el que más se acerca al perfil y se solicita un *Get Profile* (mensaje 3) para obtener más información acerca del servicio elegido (*Search Hotel*). La información devuelta por el SF (mensaje 4) incluye el objetivo del servicio *Search Hotel* y su perfil, de donde se puede ver que para proveer este servicio hay que estar en posesión de los roles *Provider* de la *TravelAgency* y *HotelProvider* de la unidad *HotelUnit*. Con lo cual, se hace la petición *Acquire Role* al OMS (mensaje 5) para obtener el rol de proveedor en la agencia de viajes. El OMS responde positivamente (mensaje 6), con lo que ha tenido éxito la operación. Seguidamente, se hace otra llamada al meta-servicio *Acquire Role* (mensaje 7) para obtener el otro rol necesario para proveer el servicio *Search Hotel*. De nuevo, el OMS responde positivamente (mensaje 8). Con lo cual, ya se han adquirido los roles necesarios para ser proveedor del servicio *Search Hotel*.

4.1.3. Registro de un proceso

En este escenario (figura 4.5) se muestran los pasos que debe seguir un proveedor para registrar un proceso o implementación de un servicio que desea proveer. Previamente a la ejecución de los meta-servicios que se ejecutan en el escenario, hay una serie de acciones que se han debido realizar, como por ejemplo adquirir los roles necesarios para poder ser proveedor de servicios dentro de la organización determinada.

Lo primero que debe hacer cualquier proveedor es comprobar si existe registrado algún perfil en la organización que describa el servicio que desea proveer. Con lo cual, hará una llamada al meta-servicio *Search Service* (mensaje 1) incluyendo como argumento de entrada el propósito del servicio en cuestión. En este caso no existe ningún perfil registrado de las características deseadas puesto que el SF nos devuelve el mensaje vacío (mensaje 2). De todas formas, se podría dar otra ocasión en la que ya hubiera un perfil registrado. Por ejemplo, al finalizar este escenario puede llegar un nuevo proveedor del mismo servicio y no tendría que registrar de nuevo el perfil, porque ya lo ha hecho un proveedor previamente. En nuestro caso, dado que el SF indica al agente proveedor que no existe ningún perfil con esas características, el siguiente paso será registrar el perfil en cuestión con la llamada al meta-servicio del SF *Register Profile* (mensaje 3). Si la operación tiene éxito (mensaje 4), ya se podrá proceder a registrar la implementación que ofrece el proveedor y la organización ya dispondrá de un agente que proporcione el servicio. Con lo cual, en el mensaje 5 se hace la llamada al meta-servicio *Register Process* que permite registrar la implementación del servicio. Cabe destacar que con esta acción el agente que registra el proceso se convierte automáticamente en proveedor del mismo. Por lo tanto, cualquier agente cliente podrá hacer la petición para utilizarlo. Como se puede ver en el mensaje 6, la operación tiene éxito y el agente *HotelProvider* se ha convertido en proveedor del servicio *Search Hotel* con su implementación particular.

4.1.4. Registro de un cliente

Cuando un agente ya se ha registrado en la plataforma, si se trata de un cliente, el siguiente paso consiste en buscar los servicios en los cuales esté interesado. Seguidamente, lo más probable es que necesite adquirir un rol concreto en la organización donde se encuentre el proveedor del servicio que desea ejecutar. Con lo cual, en este escenario se explica un ejemplo concreto de cómo llega a registrarse un agente como cliente de una organización.

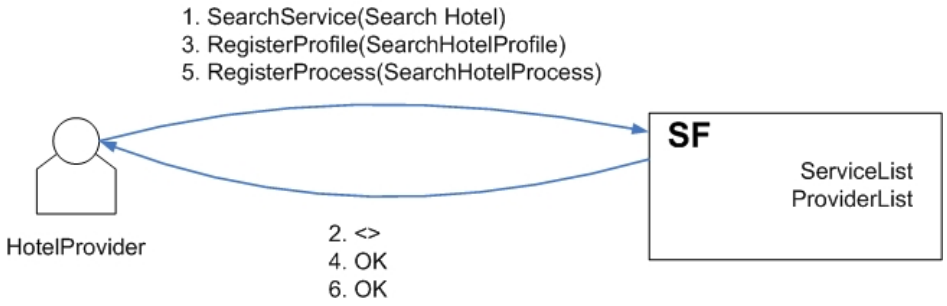

Figura 4.5: Escenario de registro de un proceso

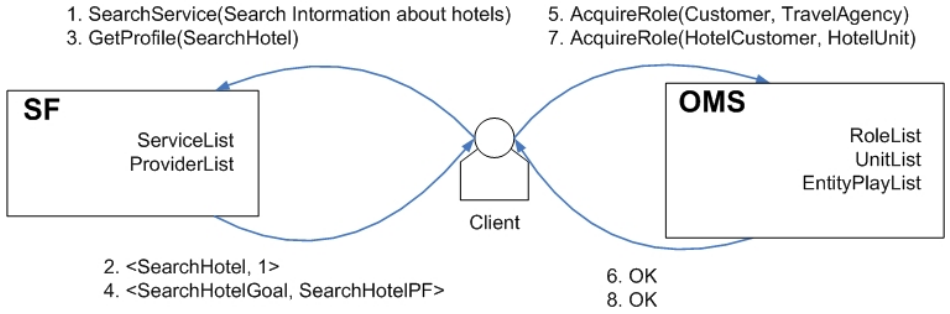

Figura 4.6: Escenario de registro de un cliente

Como se puede observar en la figura 4.6, el primer paso que realiza el agente es buscar el servicio que le interesa mediante la llamada al meta-servicio *Search Service* con el propósito deseado (mensaje 1). Teniendo en cuenta que la búsqueda del servicio ha tenido éxito (mensaje 2), se invoca un *Get Profile* (mensaje 3) para obtener el perfil del servicio. Una vez conseguido este perfil (mensaje 4), se puede consultar el rol o roles que debe poseer un agente para poder ejecutar el servicio. Así pues, los roles necesarios en este escenario para invocar el servicio deseado por el cliente son *Customer* de la unidad *TravelAgency* y *HotelCustomer* de la unidad *HotelUnit*. Por lo tanto, se hacen las llamadas pertinentes al meta-servicio *AcquireRole* (mensajes 5 y 7) para obtener los roles necesarios, que en este caso, se han podido adquirir sin ningún problema (mensajes 6 y 8).

En conclusión, el agente se ha convertido en cliente de las unidades *TravelAgency* y *HotelUnit*, con lo que tiene acceso a la ejecución de los servicios pertenecientes a estos ámbitos.

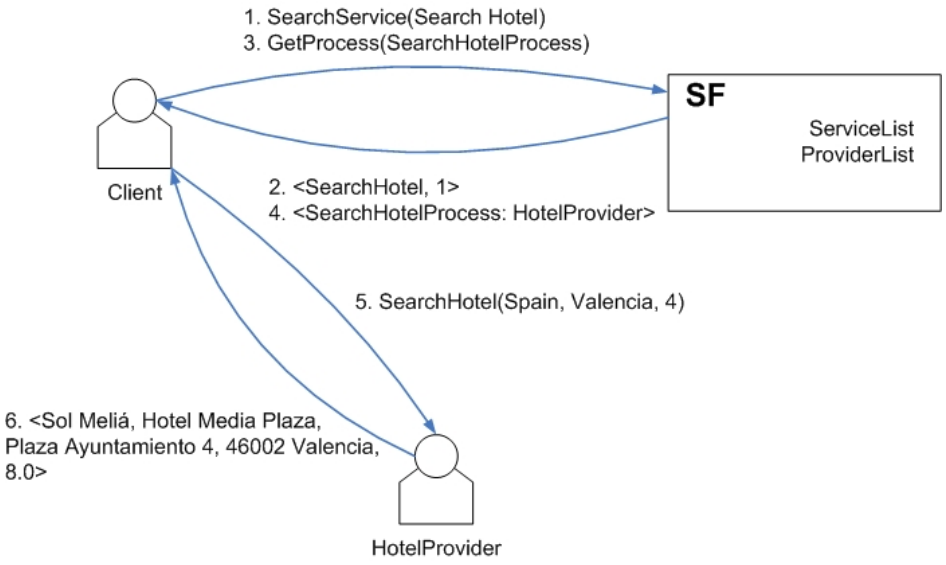

Figura 4.7: Escenario de petición de servicio

4.1.5. Petición de servicio

El siguiente escenario (figura 4.7) es uno de los más relevantes ya que se trata de la petición y ejecución de un servicio de gestión turística, lo que es uno de los principales objetivos de la aplicación.

El primer paso para ejecutar un servicio es encontrar un perfil que se adapte a las necesidades del cliente. Para ello se utiliza el meta-servicio del SF *Search Service* especificando el propósito del servicio que quiere ejecutar el cliente (mensaje 1). Una vez obtenido un perfil adecuado a las peticiones del cliente (mensaje 2), se hace una llamada al meta-servicio *Get Process* (mensaje 3) para obtener los proveedores de la implementación del servicio. Previamente, un agente proveedor había registrado un proceso, con lo cual, la respuesta tiene éxito devolviendo el proveedor del servicio que se quiere ejecutar (mensaje 4). Así pues, el agente cliente ya puede hacer la petición de servicio al proveedor, facilitando las entradas correspondientes (mensaje 5). Cabe destacar que el agente cliente debe estar en posesión de los roles requeridos para poder ejecutar el servicio, como se ha visto en escenarios anteriores. Finalmente, el agente proveedor le devuelve al cliente el resultado de la ejecución del servicio (mensaje 6).

Figura 4.8: Escenario de registro de una nueva unidad

4.1.6. Creación de una nueva unidad

Uno de los logros más importantes de la plataforma THOMAS es su gestión organizativa. Así pues, el meta-servicio básico para registrar nuevas unidades organizativas es *Register Unit*. Normalmente, la finalidad de registrar una nueva unidad es agrupar un conjunto de roles con un objetivo común. En este escenario se muestra de forma sencilla el registro de una nueva unidad, aunque podría haber un proceso previo mucho mayor para llegar a registrarla.

En la figura 4.8 se puede ver el ejemplo de este escenario. Antes de la ejecución del mismo, se ha tenido que crear la unidad *TravelAgency* dentro de la unidad principal *virtual*, que ya está creada por defecto dentro de la plataforma. Con lo cual, se puede hacer la llamada al meta-servicio *Register Unit* con los argumentos adecuados (mensaje 1). El OMS responde al agente solicitante del meta-servicio indicando el éxito de la ejecución (mensaje 2). En este caso, se registra la unidad *RestaurantUnit* dentro de *TravelAgency* con la finalidad de albergar los roles relacionados con la gestión de servicios sobre búsqueda y reserva de restaurantes. En el siguiente escenario se verá la continuación de este con el registro de algunos roles dentro de la unidad creada.

4.1.7. Creación de nuevos roles

Este sencillo escenario (figura 4.9) se puede considerar una continuación del anterior, ya que después del registro de una nueva unidad lo más lógico es introducir nuevos roles dentro de ella. Así pues, previamente al escenario que se explica a continuación, la unidad *RestaurantUnit* ya ha sido registrada.

Una vez introducidos dentro del contexto previo al escenario, el agente puede hacer la llamada al meta-servicio del OMS *Register Role*. En el primer caso, los argumentos de la llamada son para registrar un rol (*RestaurantCustomer*) perteneciente a la

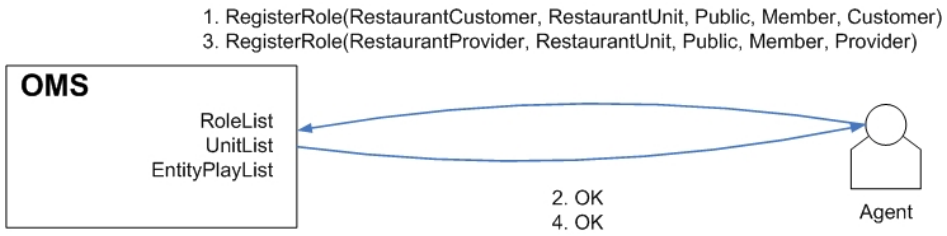

Figura 4.9: Escenario de registro de nuevos roles

unidad *RestaurantUnit* dedicado a los clientes de ésta, especializando el rol *Customer* de la *TravelAgency*. El mensaje 3 es también una llamada a *Register Role*, pero en este caso se registra igualmente otro rol dentro de *RestaurantUnit* para los proveedores, *RestaurantProvider*, que especializa a *Provider*. En ambas llamadas el OMS responde con un mensaje avisando de que se ha tenido éxito en el registro de los roles (mensajes 2 y 4).

4.1.8. Expulsión de un agente

En un entorno dinámico y cambiante en los que puede trabajar la plataforma THOMAS, siempre pueden existir agentes con objetivos contrarios a la organización o directamente clasificables como maliciosos. Para estos casos, se hace necesario disponer de herramientas para poder eliminar las amenazas y garantizar la seguridad del sistema. Así pues, el OMS ofrece un meta-servicio de expulsión de agentes.

En este escenario (figura 4.10) se representa cómo un agente de gestión de pagos (*PayeeAgent*) advierte un comportamiento fraudulento por parte del agente *Agent*. Antes de que ocurrieran los hechos, el agente malicioso había adquirido el rol de cliente en la agencia de viajes. Por lo tanto, *PayeeAgent* hace una llamada al meta-servicio *Expulse* del OMS para expulsar a *Agent*, es decir, para que se le quite el rol *Customer* de la *TravelAgency* (mensaje 1). El meta-servicio *Expulse* se ejecuta con éxito y *Agent* pierde el rol, con lo cual queda expulsado de la organización y ya no puede utilizar ningún meta-servicio o servicio.

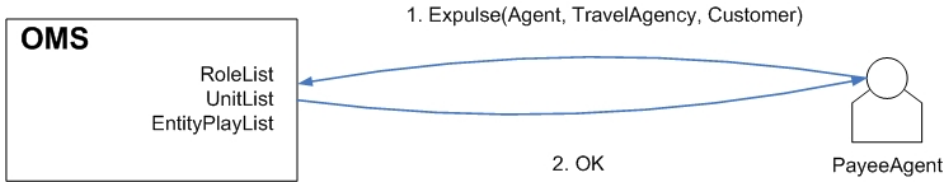

Figura 4.10: Escenario de expulsión de un agente malicioso

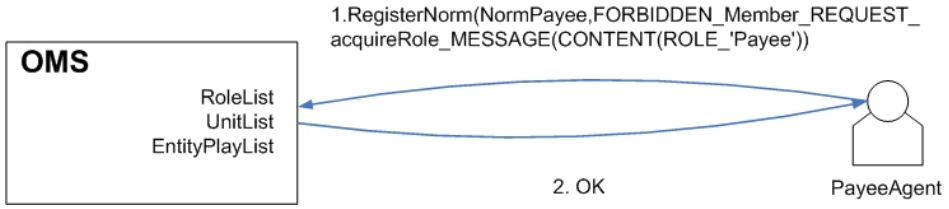

Figura 4.11: Escenario de registro de una norma

4.1.9. Registro de una norma

En la plataforma THOMAS se ha incluido un sistema de normas mediante el cual se pueden controlar las acciones de los agentes y establecer restricciones. Con lo cual, existe un meta-servicio que ofrece el OMS para registrar normas.

En el escenario de la figura 4.11 hay un agente de gestión de pagos llamado *Payee-Agent*. En este caso, el agente ha adquirido el rol *Payee* dentro de la organización *TravelAgency*, y se quiere establecer que sólo haya un gestor de pagos. Con lo cual, registra una norma que no permite a ningún agente adquirir el rol *Payee*. Como se puede ver en el mensaje 1, se especifican dos argumentos: un identificador de la norma y la norma en cuestión. Concretamente, la norma que se registra prohíbe cualquier mensaje de tipo *Request* en el que haya una petición del meta-servicio *Acquire Role* con el rol *Payee* como argumento. Puesto que el meta-servicio *Register Norm* del OMS tiene éxito, se obtiene un mensaje *Inform* notificándolo (mensaje 2).

4.2. Implementación del Sistema

En esta sección se explicará cómo se ha implementado el sistema. Primeramente, se detallará la interfaz gráfica de usuario que se ha diseñado y sus funcionalidades. Por otra parte, se comentará la implementación de todo el sistema desde un punto de vista más técnico, entrando en ciertos detalles de código fuente.

4.2.1. Interfaz Gráfica de Usuario

En la aplicación desarrollada se ha diseñado e implementado una interfaz gráfica de usuario para los distintos componentes existentes. Puesto que el lenguaje elegido para implementar la aplicación ha sido Java, se ha utilizado el paquete *javax.swing*[1] para el desarrollo de la interfaz gráfica de usuario. Este popular paquete de Java ofrece un conjunto de métodos y herramientas para la creación de componentes gráficos para el tipo de interfaz deseada. En esta sección se explican detalladamente las distintas partes de la interfaz en la que se han implementado los siguientes elementos:

- Interfaz para agentes de tipo cliente
- Interfaz para agentes de tipo proveedor
- Interfaces para servicios y meta-servicios
- Visor del agente intermediario OMS
- Visor del agente intermediario SF

4.2.1.1. Interfaces para agentes, servicios y meta-servicios

El esquema de diseño que se ha seguido para los agentes de tipo cliente y proveedor ha sido similar, dado que la mayoría de las funciones que realizan son las mismas. Las diferencias sólo residen en algunos servicios distintos que puede ejecutar cada tipo de agente. Cabe destacar que aunque estas interfaces están diseñadas concretamente para cada tipo de agente, no dejan de ser agentes generales de la plataforma THOMAS, con lo cual podrían ejecutar cualquier tipo de servicio. Teniendo en cuenta esto, ambas interfaces disponen de una barra de menú idéntica donde se puede llamar a cualquier

[1] http://java.sun.com/javase/6/docs/api/javax/swing/package-summary.html

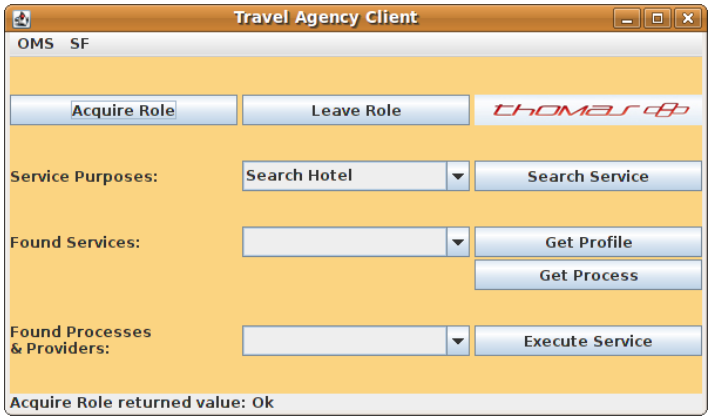

Figura 4.12: Interfaz gráfica de usuario para agentes de tipo cliente

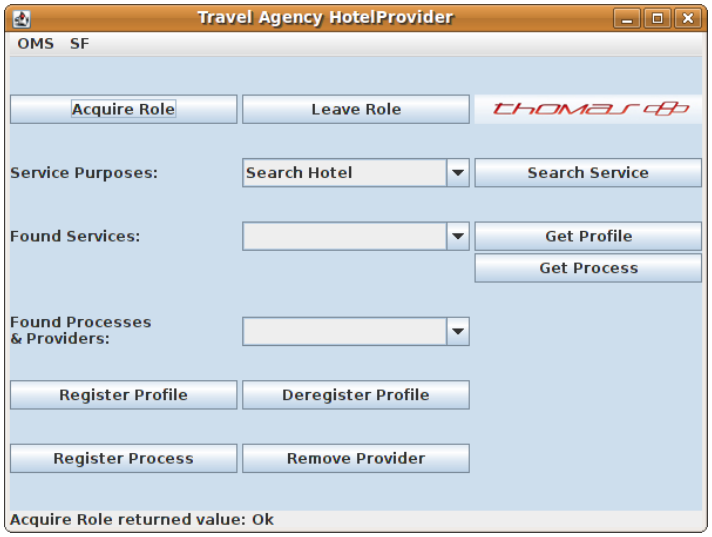

Figura 4.13: Interfaz gráfica de usuario para agentes de tipo proveedor

servicio disponible de la plataforma THOMAS. Con este diseño se consigue más generalidad en la aplicación y mayor libertad para el usuario.

En las figuras 4.12 y 4.13 se puede observar la interfaz para los agentes de tipo cliente y la interfaz para los agentes de tipo proveedor respectivamente. Como se puede ver, ambas constan de una barra de menú en la parte superior, un conjunto de botones y listas desplegables en la parte central, y una barra de estado en la parte inferior.

Primeramente se describirá la barra de menú existente tanto en la interfaz para agentes de tipo cliente como en la de agentes de tipo proveedor. Después se explicará la parte central de ambas interfaces y las diferencias entre ellas, así como las razones de dichas diferencias. Finalmente se comentará la función de la barra de estado.

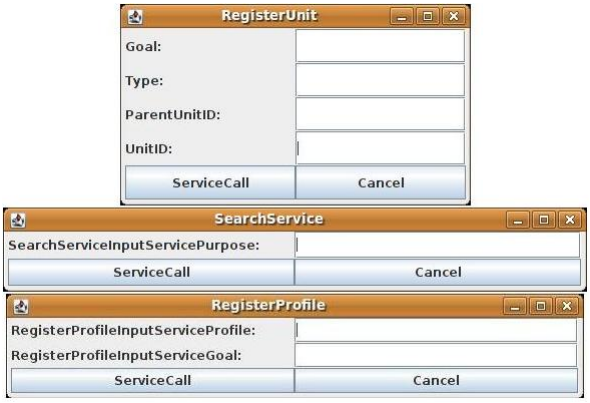

Figura 4.14: Ejemplo de ventanas de ejecución de servicios

La barra de menú está compuesta por dos entradas: *OMS* y *SF*. La primera de las entradas recoge todos los servicios del OMS que se pueden invocar, y la segunda engloba los servicios relativos al SF.

Las entradas de la barra de menú dan acceso a una serie de submenús que al final llegan a alguna opción cuyo nombre coincide con un servicio de la plataforma. Al pulsar sobre esta opción se abrirá la ventana correspondiente al servicio solicitado, en la que serán proporcionadas por el usuario las entradas requeridas para el servicio a ejecutar. En este tipo de ventanas siempre existirá un botón para llamar al servicio y otro botón de cancelación. Se pueden observar distintos ejemplos concretos de estas ventanas en la figura 4.14.

La entrada de menú **OMS** consta de tres submenús: ***Structural Services***, ***Informative Services*** y ***Dynamical Services***.

El submenú ***Structural Services*** está compuesto, a su vez, por dos submenús. El primero es *Register Services* (figura 4.15) y en él se encuentran las opciones *Register Role*, *Register Unit* y *Register Norm*, las cuales dan acceso a la ventana concreta donde se especifican las entradas del servicio al que se pretende llamar. El segundo submenú de *Structural Services* es *Deregister Services* (figura 4.16), el cual engloba las opciones contrarias al anterior submenú, es decir, *Deregister Role*, *Deregister Unit* y *Deregister Norm*. Al igual que con los otros servicios, al pulsar en cualquiera de estas opciones se abre una nueva ventana con las entradas que se deben introducir correspondientes al servicio que se desea ejecutar.

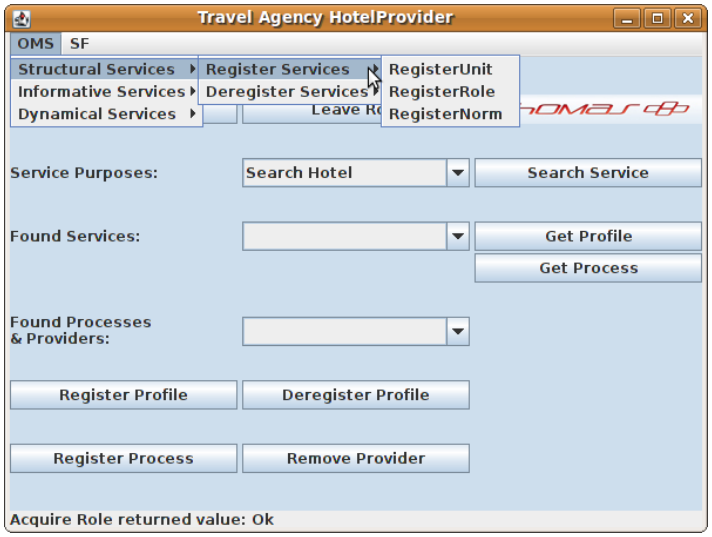

Figura 4.15: Submenú *Register Services* del submenú *Structural Services*

Por otra parte, el submenú ***Informative Services*** (figura 4.17) da acceso directo a las opciones de los servicios informativos, éstos son: *Inform Agent Role*, *Inform Members*, *Quantity Members*, *Inform Unit*, *Inform Unit Roles*, *Inform Role Profiles* y *Inform Role Norms*. Al pulsar sobre cualquiera de estas opciones se abre la ventana correspondiente al servicio con el mismo nombre, esta ventana cuenta con los campos concretos de entrada que deben ser rellenados por el usuario.

El último submenú de OMS es ***Dynamical Services*** (figura 4.18). Éste reúne las opciones *Acquire Role*, *Leave Role* y *Expulse*, que dan acceso a las ventanas concretas para cada servicio, con las entradas adecuadas a éste.

El menú ***SF*** está compuesto por dos submenús: ***Register Services*** y ***Discovery Services***.

El primer submenú del SF, ***Register Services*** (figura 4.19), está referido a los meta-servicios del SF encargados del registro de servicios. Por ello, engloba las siguientes opciones: *Register Profile*, *Register Process*, *Modify Profile*, *Modify Process* y *Deregister Profile*. Cada una de estas opciones da acceso a la ventana correspondiente (ejemplos en la figura 4.14) con las entradas del meta-servicio solicitado que debe rellenar el usuario.

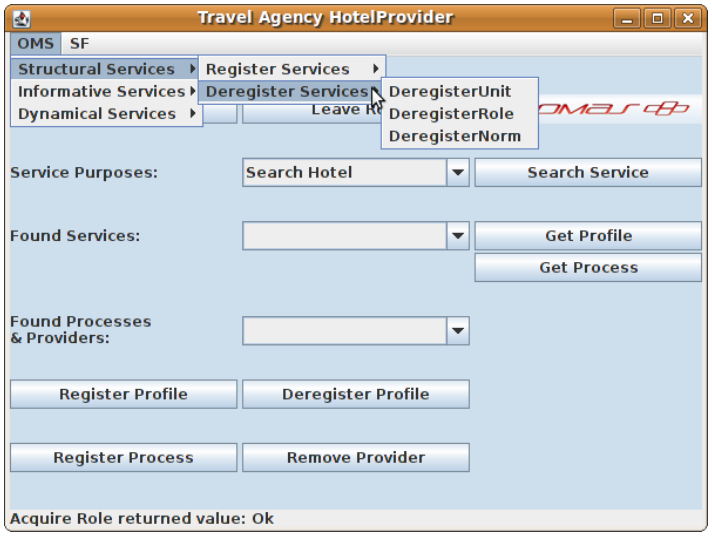

Figura 4.16: Submenú *Deregister Services* del submenú *Structural Services*

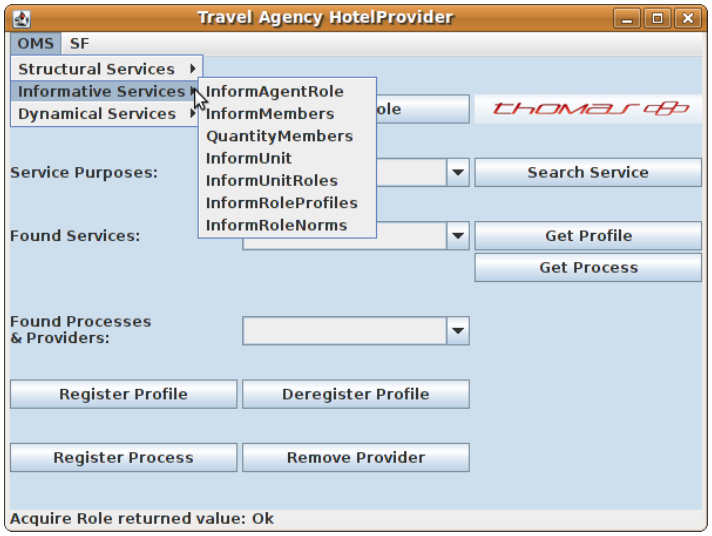

Figura 4.17: Submenú *Informative Services*

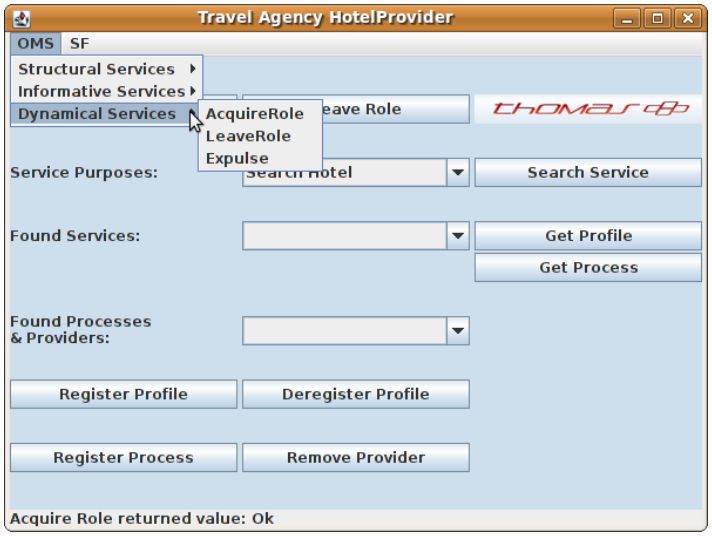

Figura 4.18: Submenú *Dynamical Services*

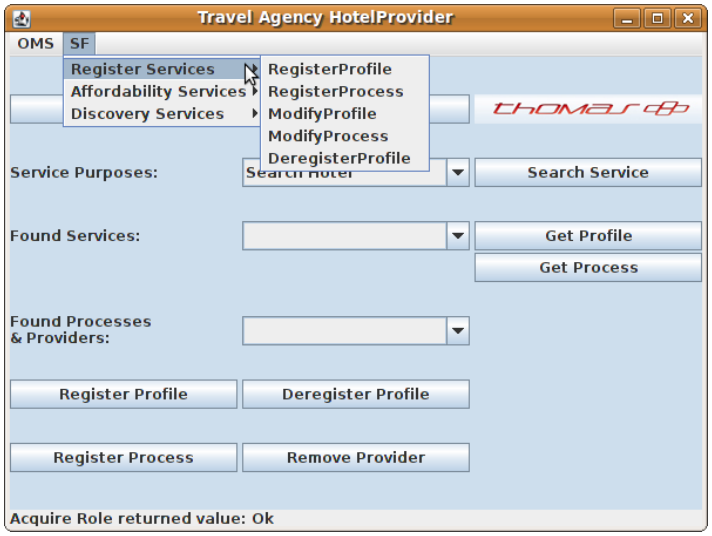

Figura 4.19: Submenú *Register Services*

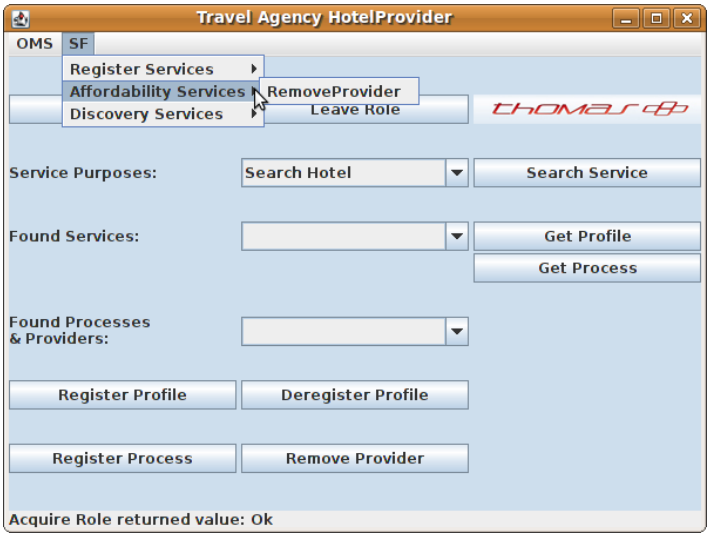

Figura 4.20: Submenú *Affordability Services*

Affordability Services (figura 4.20) es el submenú del SF que contiene la opción *Remove Provider*. Con esta opción se accede a la ventana correspondiente del meta-servicio *Remove Provider* en la que se facilitan los argumentos necesarios para la ejecución mediante el botón oportuno en la parte inferior.

El último submenú del SF es ***Discovery Services*** (figura 4.21) y engloba los meta-servicios de búsqueda y de obtención de perfiles y procesos. Con lo cual, las opciones que tiene son: *Search Service*, *Get Profile* y *Get Process*. Al igual que en el resto de elementos de los submenús, seleccionando alguna de las opciones se accede a la ventana correspondiente (ejemplos en la figura 4.14) para solicitar el servicio, facilitando las entradas de éste.

La parte central de las interfaces para agentes de tipo cliente y proveedor (figuras 4.12 y 4.13) tiene un diseño común en la zona superior, es decir, los mismos botones y listas desplegables con funcionalidades idénticas. Sin embargo, la zona inferior de la interfaz para agentes de tipo proveedor no tiene el botón *Execute Service* pero dispone de un conjunto de botones referidos a la gestión de perfiles y procesos. La razón de todo esto es que un agente de tipo proveedor no está pensado para solicitar servicios de otros agentes (sin tener en cuenta los meta-servicios de la plataforma), y por otra parte, este tipo de agentes son los que registran las descripciones de servicio (perfiles) y las implementaciones (procesos). Con lo que los agentes de tipo cliente se limitan a buscar y solicitar los servicios a agentes proveedores. Aun así, aunque estas interfaces

Figura 4.21: Submenú *Discovery Services*

están pensadas para cada tipo de agente, se puede dar el caso en la realidad que un agente pueda ser proveedor y cliente al mismo tiempo. Por ello, mediante las barras de menú ambas interfaces tienen acceso a todos los meta-servicios de la plataforma, con lo que podrían actuar de la forma deseada. La intención de este diseño ha sido hacer más intuitivo y comprensible al usuario el funcionamiento de la plataforma THOMAS y de sus meta-servicios.

Una vez aclaradas las razones del diseño de las interfaces, se pasará a describir la parte común de ambas zonas centrales de éstas, y más adelante se explicarán las partes que difieren entre los dos diseños.

Las interfaces desarrolladas están encabezadas por dos botones: ***Acquire Role*** y ***Leave Role***, que corresponden a los meta-servicios del OMS con el mismo nombre. Pulsando cualquiera de éstos aparece una ventana con dos listas desplegables referentes a las unidades y los roles registrados (para *Acquire Role*, figura 4.22) o adquiridos por el agente (para *Leave Role*, figura 4.23) en el instante en que se ha accedido a la ventana. Dependiendo de la unidad seleccionada en la primera lista desplegable, aparecen en la segunda lista los roles disponibles o adquiridos (según sea *Acquire Role* o *Leave Role*) dentro de esa unidad. Así pues, cuando se pulse el botón para adquirir o dejar ese rol, se leerá la actual selección de las listas desplegables y se solicitará el servicio correspondiente con dichas entradas.

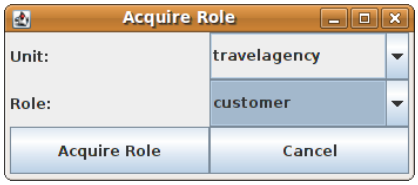

Figura 4.22: Ventana del servicio *Acquire Role*

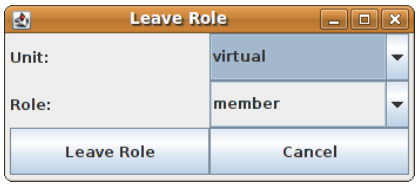

Figura 4.23: Ventana del servicio *Leave Role*

Continuando con la descripción de las interfaces mostradas en las figuras 4.12 y 4.13, la primera lista desplegable está referida a los propósitos de los servicios que se desean buscar, como su etiqueta (***Service Purposes***) indica. Así pues, cuando se pulsa sobre el botón adyacente *Search Service*, se hace una llamada al meta-servicio del mismo nombre con el propósito que esté seleccionado en la lista desplegable. Se han puesto de forma predeterminada los propósitos de los servicios que se han implementado para la aplicación, relacionados con la gestión turística. Con esto se pretende hacer más ágil la búsqueda de servicios proporcionando los objetivos que deben ser buscados en esta aplicación, sin embargo, si se desea buscar distintos propósitos de servicios se puede utilizar la opción *Search Service* de la barra de menú, la cual permite introducir el propósito deseado. Instantes después de hacer la llamada al meta-servicio *Search Service* aparecerá en pantalla una pequeña ventana informando si se ha encontrado o no el servicio buscado y también en la barra de estado de la interfaz del agente. Un servicio será encontrado si existe un perfil registrado en el SF. En caso de que la búsqueda haya tenido éxito, se mostrará el servicio encontrado en la lista desplegable de la siguiente línea, etiquetada como *Found Services*.

La segunda lista desplegable que aparece en las interfaces está etiquetada como ***Found Services*** y su función es contener los servicios que se han encontrado mediante una llamada a *Search Service*. Así pues, una vez se haya encontrado algún servicio, podrá ser seleccionado en la lista desplegable para poder ejecutar un *Get Profile* o un *Get Process* mediante los botones etiquetados con dichos nombres. Si se intenta hacer una llamada a cualquiera de estos dos meta-servicios mientras no se haya encontrado un servicio previamente, la aplicación nos indicará que se debe realizar

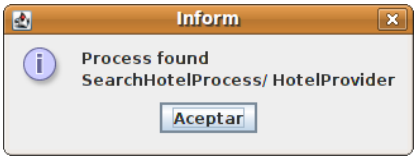

Figura 4.24: Ventana informativa después del éxito de encontrar una implementación del servicio *Search Hotel* con *Get Process*

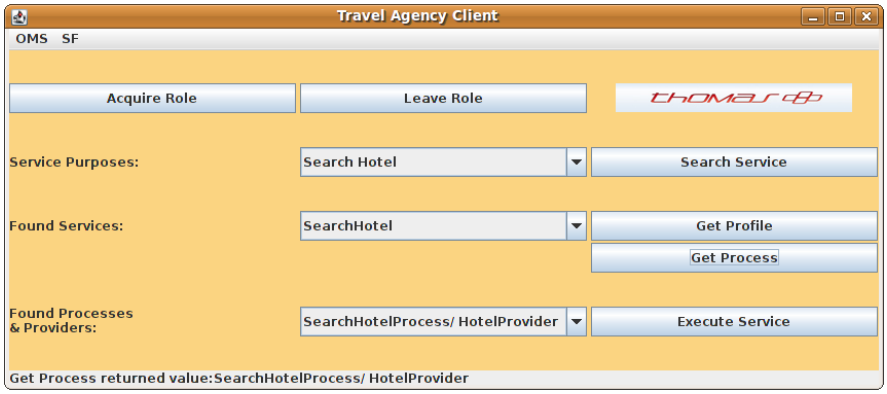

Figura 4.25: Interfaz de agentes de tipo cliente después de buscar un servicio y proveedores

una búsqueda primero. Por lo tanto, teniendo seleccionado un servicio que se haya encontrado, se puede ejecutar un *Get Profile* o *Get Process*. En el caso del botón *Get Profile*, se hará la llamada a este meta-servicio del SF y nos devolverá el perfil del servicio. Si se pulsa el botón *Get Process*, se solicitará este meta-servicio pasando como argumento el servicio que haya seleccionado en la lista desplegable *Found Services*. La respuesta de este meta-servicio dependerá de si el perfil del servicio seleccionado tiene registrado algún proceso, es decir, alguna implementación del servicio. Tanto en el caso de que exista algún proceso registrado como en el caso de que no sea así, el usuario será informado mediante un mensaje en una pequeña ventana y en la barra de estado. Así pues, si existe alguna implementación registrada se nos informará de ello (figura 4.24) y seguidamente se incluirá en la tercera lista desplegable de la interfaz gráfica del agente, etiquetada como *Found Processes & Providers*. Concretamente, se añadirá el nombre del proceso que está registrado y el nombre del agente proveedor de esa implementación separados por una barra.

En la figura 4.25 se puede observar en la interfaz para agentes de tipo cliente las listas desplegables rellenadas al haber realizado todo el proceso hasta encontrar el servicio deseado y su proveedor. En este caso, se ha buscado con *Search Service* el servicio

Search Hotel, teniendo éxito en la operación ya que hay un proveedor que ha registrado el perfil. Seguidamente se ha seleccionado la opción *Search Hotel* de la lista desplegable *Found Services* que se ve en la figura y se ha ejecutado un *Get Process* para obtener, si existen, los proveedores de dicho servicio. Puesto que *HotelProvider* ha registrado una implementación (proceso) del servicio *Search Hotel*, este agente es un proveedor de dicho servicio, y al ejecutar el meta-servicio *Get Process* obtenemos el nombre del proceso (SearchHotelProcess) y el citado proveedor. Así pues, al tratarse de un agente cliente se podría solicitar el servicio al proveedor encontrado. Por el contrario, si se trata de un agente proveedor, todo este proceso sería sólo para obtener la información de si hay proveedor de un servicio concreto.

La tercera lista desplegable de las interfaces está etiquetada como ***Found Processes & Providers***. Como ya se ha comentado, el contenido de ésta serán los nombres de los procesos junto con los nombres de los proveedores de éstos que hayan sido encontrados mediante la ejecución del meta-servicio *Get Process*. La función de esta lista desplegable difiere según la interfaz gráfica para agentes de tipo proveedor respecto a la de agentes de tipo cliente. Si se trata de la interfaz del proveedor, la lista desplegable tiene una funcionalidad meramente informativa, es decir, para mostrar al usuario las implementaciones que existen y los proveedores de un determinado servicio. La utilidad de esto reside en que el propio proveedor pueda disponer de esta información para decidir si puede proveer un servicio y si ya existen proveedores del mismo. En el caso del cliente, además de servir para tener información acerca de los procesos registrados y sus proveedores, se podrá usar el botón *Execute Service* a la derecha de la lista desplegable, con lo que se abrirá una nueva ventana con las entradas de la implementación de servicio que se encuentre seleccionada en la lista desplegable. Mediante esta ventana, el usuario podrá rellenar las entradas y ejecutar el servicio, que en este caso tratará sobre gestión turística. Las ventanas que pueden aparecer dependen del servicio turístico que se haya buscado y son los siguientes: *Search Hotel* (figura 4.26), *Search Flight* (figura 4.27), *Reserve Hotel* (figura 4.28) y *Reserve Flight* (figura 4.29).

Además, en los servicios *Reserve Hotel* y *Reserve Flight*, si se ha ejecutado con éxito un *Search Hotel* o un *Search Flight* respectivamente, se rellenan automáticamente las entradas que corresponden con la búsqueda anterior. La intención de este proceso es ahorrar tiempo al usuario ya que lo más lógico es primero buscar un hotel o un vuelo y seguidamente reservarlo. Así pues, los campos que se rellenan automáticamente en el *Reserve Hotel* son: *Hotel Chain*, *Hotel Name*, *Room Rate* y *Address*, ya que se conocen

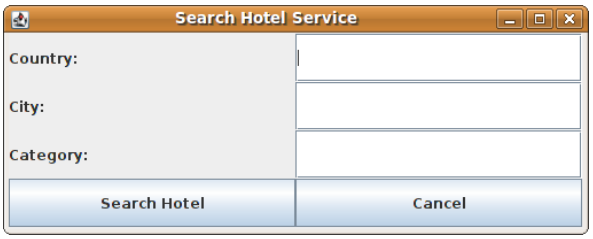

Figura 4.26: Ventana del servicio *Search Hotel*

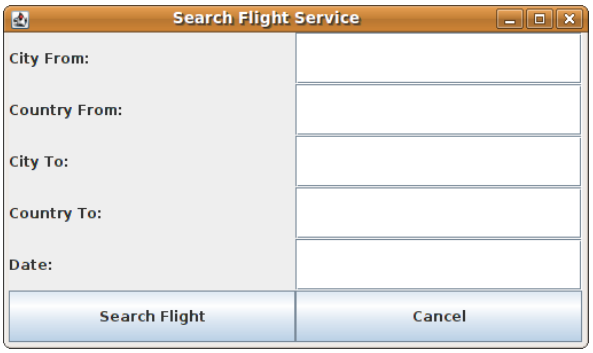

Figura 4.27: Ventana del servicio *Search Flight*

Figura 4.28: Ventana del servicio *Reserve Hotel*

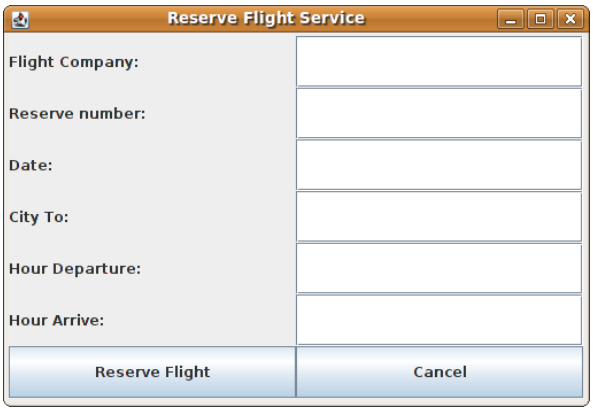

Figura 4.29: Ventana del servicio *Reserve Flight*

Figura 4.30: Ventana de advertencia de error de introducción, causada por tener el campo *Country* vacío

del servicio *Search Hotel* ejecutado con anterioridad. En el caso de *Reserve Flight*, los campos completados si se ha hecho previamente un *Search Flight* son: *Flight Company*, *Date*, *City To*, *Hour Departure* y *Hour Arrive*.

Por otra parte, en todas estas ventanas para la ejecución de los servicios turísticos se comprueba que las entradas a introducir por el usuario estén correctamente rellenadas siguiendo el formato correcto en cada caso. Si no es así, cuando se intenta hacer la llamada al servicio mediante el botón correspondiente, la aplicación avisa de la incorrección (figuras 4.30, 4.31 y 4.32) y el usuario debe enmendar el error para poder seguir con la ejecución de dicho servicio. Si la llamada al servicio tiene éxito se devuelven los resultados en una ventana informativa (figuras 4.33 y 4.34, notificando el resultado de un *Search Hotel* y de un *Reserve Hotel* respectivamente) y también se notifica que el resultado es satisfactorio a través de la barra de estado.

A continuación, se explicarán los botones que forman la parte inferior de la interfaz de agentes de tipo proveedor puesto que todos los elementos comunes con la interfaz de agentes de tipo cliente ya han sido detallados. Estos botones exclusivos para los agentes de tipo proveedor están relacionados con la gestión de perfiles y procesos de servicios. La razón de que los agentes de tipo cliente no posean en su interfaz estos

Figura 4.31: Ventana de advertencia de error de formato, causada por una incorrección en el campo *Category* que debe ser un entero

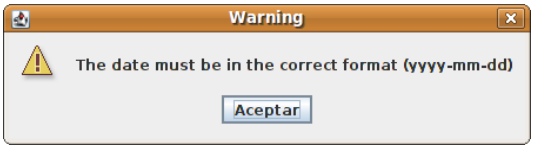

Figura 4.32: Ventana de advertencia de error de formato, causada por una incorrección en el campo *Date* que debe ser de tipo fecha (aaaa-mm-dd)

Figura 4.33: Ventana informativa después del éxito de la ejecución del servicio *Search Hotel*

Figura 4.34: Ventana informativa después del éxito de la ejecución del servicio *Reserve Hotel*

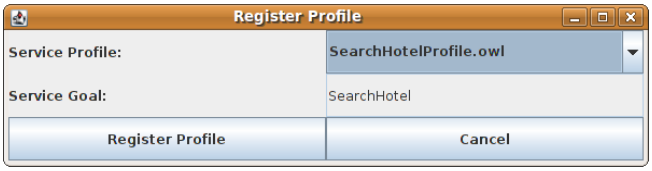

Figura 4.35: Ventana de ejecución del servicio *Register Profile*

botones reside en que no están pensados para proveer servicios, sólo para solicitarlos, y con lo cual no tiene sentido poder acceder a registrar perfiles y procesos. A pesar de esto, como ya se ha comentado anteriormente con otros casos, mediante las barras de menú el agente de tipo cliente podría utilizar todos los servicios relacionados con la gestión de perfiles y procesos, ya que se podría dar la circunstancia de que un cliente fuera a su vez proveedor de algún servicio.

El primero de los botones de la parte inferior de la interfaz para agentes de tipo proveedor es ***Register Profile***. Al pulsar sobre éste aparece una ventana (figura 4.35) en la que se puede seleccionar el perfil que se desea registrar en el SF. La selección se realiza mediante la lista desplegable etiquetada como *Service Profile* en la que existen un conjunto de opciones con los perfiles que puede registrar el agente con el que se trabaja. La caja de texto etiquetada como *Service Purpose* contiene el propósito del servicio que se registrará con el perfil seleccionado, así pues, este propósito es el que se utilizará para la búsqueda del servicio mediante *Search Service*. Además, se cambia automáticamente según el perfil seleccionado en la lista desplegable superior, por lo tanto no es editable por el usuario.

A pesar de que sólo se permite el registro de ciertos perfiles mediante el botón de la interfaz, el usuario siempre puede utilizar las opciones de la barra de menú para ejecutar el meta-servicio *Register Profile* con los argumentos deseados. Sin embargo, esta opción es más desaconsejable ya que se pueden provocar fallos por la inexperiencia del usuario o por la mala introducción de algún argumento. De esta forma, la aplicación ofrece robustez, desde el punto de vista de su utilización más intuitiva, y flexibilidad al poder usar los meta-servicios con los argumentos que desee un usuario avanzado.

Una vez se ha pulsado el botón *Register Profile* de la ventana del meta-servicio con el mismo nombre, se ejecuta la llamada con los argumentos seleccionados. Si la llamada tiene éxito, la aplicación avisará al usuario con un mensaje informativo (figura 4.36) y también mediante la barra de estado. Por el contrario, si no se ha podido

Figura 4.36: Ventana informativa después del éxito de la ejecución del meta-servicio *Register Profile*

Figura 4.37: Ventana de ejecución del meta-servicio *Deregister Profile*

registrar el perfil, la aplicación notificará este hecho de igual forma, es decir, mediante un mensaje informativo correspondiente e información pertinente en la barra de estado.

A la derecha del botón *Register Profile* se encuentra el botón ***Deregister Profile***. La misión de este último es dar acceso a la ejecución del meta-servicio de su mismo nombre. Así pues, al pulsarlo se accede a la ventana de la figura 4.37 en la que se encuentran los perfiles registrados en el SF. En caso de no existir ningún perfil de servicio registrado aparecería un mensaje de aviso como el de la figura 4.38 informando al usuario. Una vez se ha abierto la ventana con los perfiles registrados, el usuario debe seleccionar el que está interesado en eliminar del registro del SF. Así pues, se pulsaría sobre el botón *Deregister Profile* para hacer la llamada al meta-servicio del mismo nombre con el perfil seleccionado como argumento. Como es habitual en la aplicación, se informará al usuario del éxito o no de la operación mediante una ventana informativa y a través de la barra de estado.

El primer botón presente en la última línea de botones de la interfaz para agentes de tipo proveedor es ***Register Process***. El uso de éste provoca la ejecución de una ventana (figura 4.39) donde se puede elegir el identificador del servicio del que se desea registrar una implementación. De forma similar a la ventana de *Register Profile*, se

Figura 4.38: Ventana de aviso de que no hay ningún perfil registrado

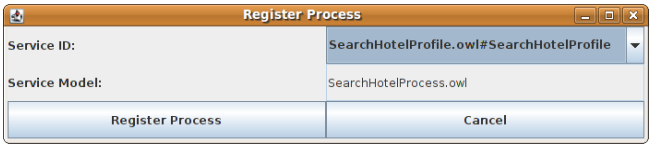

Figura 4.39: Ventana de ejecución del servicio *Register Process*

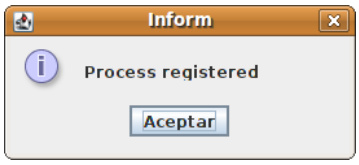

Figura 4.40: Ventana informativa después del éxito de la ejecución del meta-servicio *Register Process*

dispone de una lista desplegable con la que se selecciona el identificador del servicio, y dependiendo de esta selección, el cuadro de texto inferior cambiará su contenido con el modelo de implementación correspondiente, con lo cual, no es editable por el usuario. Además, los identificadores de servicio que aparecerán como opciones para registrar el proceso serán solamente aquellos cuyo perfil haya sido registrado previamente. Cuando se pulse sobre el botón *Register Process* se hará la llamada a este meta-servicio con los argumentos seleccionados en la ventana. Tanto si tiene éxito como si no, la aplicación nos avisará mediante una ventana informativa (éxito en la ejecución, figura 4.40) y también se notificará en la barra de estado. La intención de este diseño es la misma que en la ventana de *Register Profile*, facilitar al usuario el uso de la aplicación para que no se cometan equivocaciones en la introducción de argumentos. Al igual que en otros casos, si se desea hacer un registro de un proceso distinto a los que se ofrecen por defecto sólo hay que acceder al meta-servicio *Register Process* desde la barra de menú y facilitar los argumentos deseados.

Para poder eliminar un proveedor de una implementación de servicio, existe el meta-servicio del SF ***Remove Provider***. Con lo cual, el botón destinado a este meta-servicio tiene el mismo nombre y está situado a la derecha de *Register Process*. Cuando se pulsa, si no hay ningún proceso registrado aparecerá una ventana de aviso como la de la figura 4.41. Por el contrario, si ya se ha registrado algún proceso se mostrará una ventana como la de la figura 4.42 en la que se podrá elegir cualquiera de los procesos registrados en el SF. Una vez elegido el proceso que se desea eliminar del registro del SF se debe pulsar el botón *Remove Provider* de la ventana para hacer la llamada al meta-servicio homónimo. Como es habitual en la aplicación, se informará al usuario de

Figura 4.41: Ventana de aviso de que no hay ningún proceso registrado

Figura 4.42: Ventana de ejecución del meta-servicio *Remove Provider*

si la operación ha tenido éxito mediante una ventana informativa y la barra de estado. Cabe destacar que si el proceso que se elimina era el último que proporcionaba el servicio de un perfil determinado, dicho perfil también será eliminado del registro del SF. Por lo tanto, al eliminar el último proveedor de una implementación de servicio se provoca la eliminación del perfil que le corresponde. Es necesario aclarar que este es un mecanismo automático presente en la plataforma THOMAS y que no tiene nada que ver con la aplicación desarrollada.

Finalmente, la **barra de estado**, tanto en la interfaz para agentes de tipo cliente como en la de los agentes de tipo proveedor, sirve para informar al usuario de qué servicio se está ejecutando o del resultado que ha devuelto. Es decir, tal y como se ha visto a lo largo de esta sección, ofrece la utilidad típica en este tipo de componente para una aplicación.

4.2.1.2. Visores del OMS y del SF

Para comprender el funcionamiento de la plataforma THOMAS y poder observar la información que tiene registrada internamente, se han implementado dos visores para los agentes intermediarios OMS y SF. Es necesario aclarar que cualquier aplicación real desarrollada sobre la plataforma sólo tendría acceso a esta información por medio de los meta-servicios informativos que ofrece el OMS. Sin embargo, en este caso se ha optado por acceder directamente a la base de datos de THOMAS para obtener la información de forma rápida y directa, y no provocar un tráfico de mensajes que podría saturar el canal de comunicación. Así pues, el propósito de estos visores es mostrar al usuario los datos que se consideran interesantes para observar cómo funciona la plataforma THOMAS y la gestión de los meta-servicios que realizan los agentes inter-

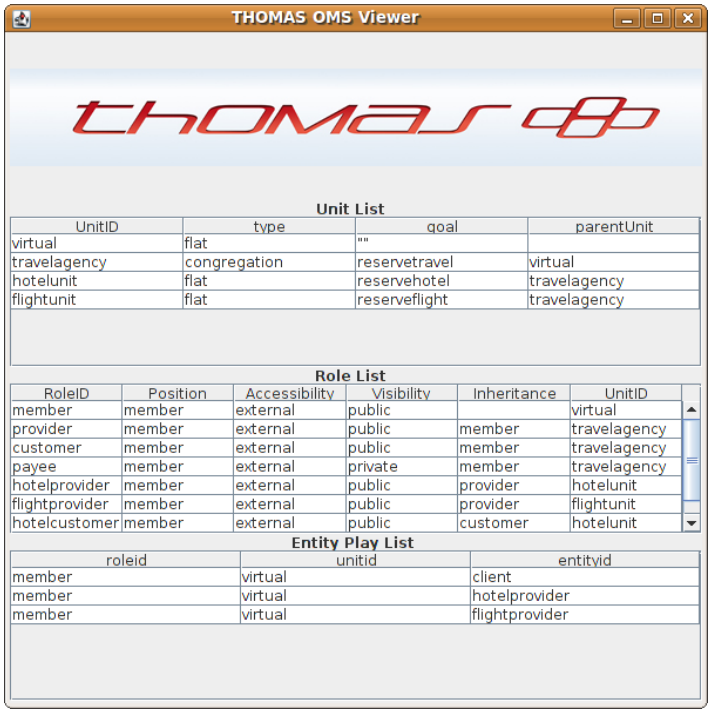

Figura 4.43: Visor del agente intermediario OMS

mediarios OMS y SF. Cabe destacar que ambos visores están directamente vinculados con el agente intermediario al que representan. Por ello obtienen los datos directamente.

El visor del OMS está compuesto por tres tablas principales, tal y como se puede observar en la figura 4.43. La primera tabla es *Unit List*, es decir, la lista de unidades registradas en el OMS. Se pueden ver las organizaciones que hay presentes en la plataforma junto con sus atributos: tipo, objetivo y unidad superior. La segunda tabla contiene la lista de roles, *Role List*. De igual forma que en la tabla anterior, se muestran los roles registrados en el OMS junto con sus atributos, que son: posición, acceso, visibilidad, herencia y unidad. Finalmente, la tercera tabla muestra la *Entity Play List*, es decir, la lista de los roles que poseen los agentes en la plataforma. Así pues, los campos que componen esta tabla son: identificador de rol, identificador de unidad e identificador de entidad o agente.

Con el objetivo de mostrar al usuario los perfiles y procesos registrados en el SF se ha creado un visor para este agente intermediario. En la figura 4.44 se puede observar el visor del SF al inicio de la ejecución del sistema. Por esta razón se encuentra vacío ya que no se ha registrado ningún perfil ni proceso todavía. El visor está compuesto principalmente por dos tablas: *Profiles List* y *Processes List*. Tal y como su nombre indica

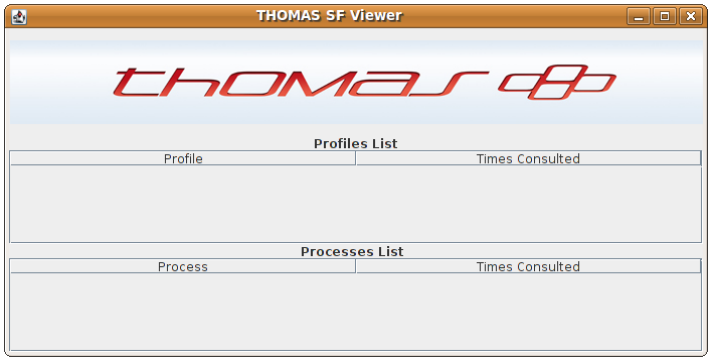

Figura 4.44: Visor del agente intermediario SF

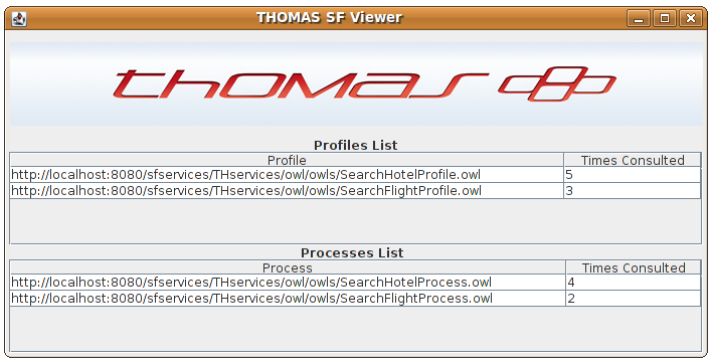

Figura 4.45: Visor del agente intermediario SF con perfiles y procesos consultados

la primera de ellas es una lista de los perfiles registrados y la segunda otra lista con los procesos registrados. Además, ambas tablas disponen de una columna etiquetada como *Times Consulted*, que es un contador de las veces que se ha consultado el perfil o proceso en cuestión. Se entiende una consulta de un perfil la ejecución del meta-servicio *Get Profile* sobre ese perfil. Por su parte, una consulta a un proceso se hace mediante el meta-servicio *Get Process*. En la figura 4.45 se puede ver el visor del SF con algunos perfiles y procesos registrados que han sido consultados.

4.2.2. Implementación de la Aplicación

Más allá de la interfaz gráfica de usuario que se ha explicado anteriormente, es necesario detallar el desarrollo de la aplicación desde el punto de vista de su funcionamiento interno. A continuación, se comenta de forma general el conjunto de métodos implementados.

4.2.2.1. Implementación de los agentes y sus interfaces

Todo el funcionamiento de la aplicación va estrechamente ligado a la interfaz gráfica, ya que a través de los componentes de que dispone el usuario se ejecutan las funciones de la aplicación. El código implementado se divide en distintas clases de Java agrupando los métodos necesarios. Se han desarrollado clases auxiliares para la creación de las interfaces gráficas necesarias según el caso, como las ventanas que aparecen para ejecutar los distintos servicios o las propias interfaces para los agentes de tipo cliente y proveedor. El objetivo de estas clases auxiliares es dividir el código en distintas partes para no concentrarlo todo en una misma clase. Así pues, las principales clases que se han implementado son la de agentes de tipo cliente (*ClientAgent*) y la de agentes de tipo proveedor (*ProviderAgent*). Ambas clases son bastante similares ya que comparten muchos métodos y variables, aunque existen diferencias debidas a la naturaleza de cada tipo de agente y también relacionadas con las distintas interfaces gráficas de que disponen.

Tanto la clase *ClientAgent* como la *ProviderAgent*, desarrolladas para incluir agentes de tipo cliente y proveedor en la aplicación, extienden a la clase *jade.core.Agent*. Con lo cual, en el método *setup()* es donde se concentra la acción. Concretamente, en la implementación realizada se ubican todos los *Action Listeners* de la interfaz gráfica de usuario dentro de este método. Es necesario aclarar que un *Action Listener* se ejecuta cuando se produce un evento sobre el componente en el cual se ha creado. Se han incluido en el *setup()* porque este método es el encargado de configurar las acciones del agente y se ejecuta cuando éste es iniciado. Así pues, cuando entre en ejecución el agente después de su creación, los *Action Listeners* se iniciarán y quedarán a la espera del evento que los active para ejecutar el código que alberguen. El funcionamiento básico de la mayoría de los *Action Listeners* que se han añadido a los botones de la interfaz, tanto en el cliente como en el proveedor, consiste en crear y mostrar una nueva ventana correspondiente al servicio o meta-servicio que se desea ejecutar. En otros casos, como el botón *Search Service*, se obtiene el valor de la lista desplegable correspondiente y se hace directamente una llamada al servicio con este parámetro. Volviendo a los *Action Listeners* que muestran una nueva ventana, tras la creación de ésta se hace una llamada al servicio o meta-servicio cuando el usuario pulse el botón pertinente y haya rellenado todos los campos de entrada correctamente. En estas ventanas nuevas que aparecen, también existe un *Action Listener* en el botón de llamada al servicio o meta-servicio para controlar cuándo se utiliza, y que los campos estén rellenados adecuadamente.

Además, existe otro *Action Listener* para controlar el botón de cancelación, que en caso de ser pulsado cierra la ventana automáticamente.

En el caso de los menús de la parte superior de los agentes de tipo cliente y de tipo proveedor, la dinámica es bastante similar a los botones de la parte central de las interfaces gráficas. También se han añadido *Action Listeners* para las opciones de los menús en los que se crean nuevas ventanas con los meta-servicios o servicios a ejecutar. Dentro de estas nuevas ventanas también existen *Action Listeners* para los botones de ejecución del servicio o de cancelación. La particularidad de estas ventanas reside en que sus etiquetas y cajas de texto, donde se introducen las entradas de los servicios o meta-servicios, son creadas de forma genérica leyendo la información de las entradas del servicio, con lo que se adapta el tamaño de la ventana y el número de entradas al servicio determinado. Además, para ofrecer mayor flexibilidad, no se hacen comprobaciones sobre cómo se rellenan las entradas del servicio o meta-servicio. Por lo tanto, es responsabilidad del usuario la correcta introducción de los argumentos para no provocar fallos inesperados en el sistema.

4.2.2.2. Implementación de las comunicaciones

Uno de los métodos más importantes en la aplicación es *execute_call(String call, AID receiver)*. Está presente en los agentes de tipo cliente y en los de tipo proveedor, aunque existen ciertas diferencias entre la implementación concreta del método en uno u otro agente. Este método recibe dos argumentos: una cadena de caracteres donde está la llamada al servicio con los parámetros incluidos, y un AID (Identificador de Agente) con el receptor de la petición de servicio. La llamada al servicio que se recibe como argumento será el contenido del mensaje *FIPA_Request*, que es como se gestionan las peticiones de servicio en la plataforma THOMAS. Por otra parte, si el AID recibido como segundo argumento es distinto de *null*, el receptor del mensaje de petición será este AID. Por el contrario, si este argumento es null, se buscará dentro de la llamada si se trata de un servicio dirigido al OMS o SF y se pondrá como receptor el que corresponda.

Una vez se ha enviado el mensaje de petición, se añade un comportamiento cíclico al agente para esperar una respuesta de tipo *Inform* del receptor del mensaje que se ha enviado. Si se recibe un mensaje que cumple con estas características, se comprue-

ba además que sea la respuesta *Inform* del meta-servicio o servicio que se solicitó y se tratan los datos recibidos en el contenido. En caso de no ser la respuesta al servicio solicitado, se bloquea el comportamiento hasta recibir otro mensaje. Así pues, si la respuesta se corresponde con el servicio que se había solicitado se procesan los datos dependiendo de éste. Además, se elimina el comportamiento para no recoger más mensajes ya que se ha recibido el que se esperaba. En este punto es donde difiere la implementación del método *execute_call()* en el agente cliente con respecto a la del agente proveedor. Esta diferencia reside en que algunos servicios o meta-servicios sólo los pueden ejecutar los agentes de tipo cliente y no los de tipo proveedor y viceversa. Los servicios que son ejecutados solamente por los agentes de tipo cliente son los pertenecientes a la gestión turística incluidos en la aplicación, estos son: *Search Hotel*, *Search Flight*, *Reserve Hotel* y *Reserve Flight*. Por su parte, los meta-servicios que sólo invocan los agentes de tipo proveedor son los de gestión de registro de descripciones de servicio e implementaciones: *Register Profile*, *Modify Profile*, *Deregister Profile*, *Register Process* y *Modify Process*. Finalmente, los servicios que se ejecutan tanto en los agentes de tipo cliente como en los de tipo proveedor son: *Acquire Role*, *Leave Role*, *Search Service*, *Get Profile* y *Get Process*. Sin embargo, como ya se ha comentado anteriormente, cabe destacar que la aplicación desarrollada permite mucha flexibilidad, es decir, se puede ejecutar cualquier meta-servicio o servicio disponible a través de las barras de menú. Aun así, la aplicación está pensada sobretodo para ejecutar los servicios o meta-servicios que aparecen en las propias interfaces. Por ello, sólo se tratan los datos devueltos de estos servicios o meta-servicios que se hayan ejecutado. Los resultados de los servicios o meta-servicios para los que no están pensados los agentes y se ejecuten a través de las barras de menú, pueden ser consultados en la consola de ejecución.

En todos los servicios o meta-servicios de los que se recibe la respuesta *Inform*, se tratan los datos de manera muy similar. El proceso de estos datos recibidos consiste en separar en distintos campos el resultado del servicio o meta-servicio, que se recibe en el contenido del mensaje. Con estos campos se sabe, según el caso, si la petición del servicio ha tenido éxito o no, y el resultado concreto. Así pues, se muestra al usuario a través de la interfaz gráfica mediante ventanas de mensajes informativos y también con la barra de estado.

Los agentes de tipo proveedor poseen una clase *ProviderResponder* para responder a las peticiones de servicio. Al iniciar un agente proveedor, es decir, al principio del método *setup()*, se añade este comportamiento. Este protocolo es una extensión de

la clase *jade.proto.SimpleAchieveREResponder*. En el constructor de *ProviderResponder* se hace una llamada a la clase por encima especificando que los mensajes que se deben atrapar serán de tipo *FIPA_Request*. Seguidamente, se dispone de dos métodos: *prepareResponse(ACLMessage msg)* y *prepareResultNotification(ACLMessage inmsg, ACLMessage outmsg)*. El primero de ellos recibe como argumento un mensaje en lenguaje FIPA-ACL del tipo *FIPA_Request*, como se ha especificado en el constructor. Este mensaje recibido se procesará extrayendo sus *tokens* para separar y clasificar la información que contiene. Una vez se han obtenido los datos necesarios, se comprueba que el proceso de servicio que envía el agente solicitante en el mensaje coincida con alguno de los procesos que tiene registrados el agente proveedor. En caso negativo se responderá a la solicitud declinando la petición del cliente con un *Refuse*. Si por el contrario el proveedor tiene registrado el proceso que se pide, se enviará al solicitante una respuesta afirmativa con *Agree* y se ejecutará el segundo método de la clase, *prepareResultNotification*. Así pues, se lee el servicio y se obtiene su proceso. A continuación, es necesario clasificar las entradas que ha facilitado el cliente e introducirlas en un mapa de valores. Finalmente, se ejecutará el servicio con los valores extraídos y se recogerá el resultado, el cual se podrá enviar al cliente mediante un mensaje *Inform*.

4.2.2.3. Implementación de los visores del OMS y del SF

Como se ha explicado en el apartado correspondiente a los visores del OMS y del SF, éstos están formados por un conjunto de tablas con la información que se quiere mostrar. A continuación se explicará brevemente en qué consiste la implementación.

El visor del OMS está compuesto por tres tablas: *Unit List*, *Role List* y *Entity Play List*. Cada una de éstas se corresponde con una tabla de la base de datos de THOMAS. Con lo cual, para obtener esta información hay que hacer una conexión y extraer las tablas que interesen por medio de comandos de SQL. Así pues, se crea una instancia y se inicia una conexión a la base de datos y seguidamente se ejecutan llamadas "select" del lenguaje de SQL con los elementos que se desean extraer. Una vez obtenida la información en un *ResultSet* de SQL, se transforma para introducirla en un *DefaultTableModel* de Java y poder mostrar todos los datos a través de la interfaz. Hay que destacar que para poder recargar los datos continuamente, el visor del OMS es iniciado por el mismo OMS en su método *setup()*. Por otra parte, después de ejecutar cualquier meta-servicio que haya pedido algún cliente, se llama al método *reLoad()* del visor del

OMS para que vuelva a hacer la lectura de la base de datos con los cambios que se hayan podido producir.

Por su parte, el visor del SF tiene un diseño similar al del OMS pero con sus particularidades. Está compuesto por dos tablas: *Profiles List* y *Processes List*. En la primera de ellas se muestra la lista de los perfiles registrados y en la segunda la lista de los procesos registrados. Además, cada perfil o proceso registrado dispone de un contador (*Times Consulted*) que indica el número de veces que ha sido consultado. *Get Profile* se utiliza para consultar un perfil y *Get Process* para consultar un proceso. Al igual que en el visor del OMS, los datos mostrados en el visor del SF se obtienen directamente de la base de datos de THOMAS. Para ello se consultan las listas de perfiles y procesos registrados. En el caso de los datos que se muestran en el visor del OMS, la información se obtiene mediante comandos de SQL directamente sobre la base de datos de THOMAS. Sin embargo, para los datos mostrados en el visor del SF se hace necesario hacer consultas a las tablas de JENA, que son las que guardan los perfiles y procesos registrados. Para ello se utiliza el método necesario para obtener datos de JENA. Una vez obtenidos los perfiles o procesos registrados, se transforma el ResultSet de JENA a cadenas de caracteres. Seguidamente, se muestran en el visor del SF de la misma forma que en el visor del OMS, es decir, con *DefaultTableModel*. Hay que destacar que las consultas a JENA para obtener los perfiles y los procesos son distintas entre sí.

Por otra parte, para obtener los contadores *Times Consulted* de los perfiles y procesos registrados, se dispone de dos vectores (de la clase *java.util.Vector*) de enteros destinados a almacenar estos contadores y otros dos vectores de cadenas de caracteres para guardar los perfiles y procesos asociados a dichos contadores. Así pues, cuando se hace una lectura de perfiles registrados (igualmente para los procesos registrados) se crean los contadores necesarios en caso de no existir previamente y se mantienen los que ya se tenían. Se pueden guardar correctamente los contadores ya que los perfiles y procesos registrados mantienen el orden en que se han registrado cuando se consultan. Estos contadores son incrementados por el agente intermediario SF. Cuando éste recibe una petición de meta-servicio, revisa si se trata de un *Get Profile* o de un *Get Process*. En caso de tratarse del primero, el SF busca en el vector de perfiles registrados que tiene almacenado el visor para encontrar si algún perfil coincide con el que realiza la petición de *Get Profile*. En caso afirmativo se incrementa el contador correspondiente a ese perfil. Así pues, si se trata de un *Get Process* el SF actúa de la misma forma, consultando los procesos registrados que almacena el visor e incrementando el contador

en caso que coincida el proceso consultado con alguno de los registrados.

El tipo de lectura que realiza el visor del SF sobre las tablas de JENA para obtener los perfiles y procesos registrados también se utiliza en los agentes proveedores. Se hace una lectura para obtener los perfiles registrados en el SF cuando se pulsa sobre el botón *Deregister Profile* de la interfaz para agentes de tipo proveedor. Del mismo modo, se hace una lectura similar de los procesos registrados en el SF cuando se pulsa el botón *Remove Provider* de la citada interfaz. Con estas lecturas se obtienen, en caso de existir, los perfiles o procesos registrados en el SF. Con lo cual, se puede facilitar la tarea del usuario mostrando una lista de los perfiles o procesos que puede eliminar del registro del SF, o informarle de que no existe ninguno si se da el caso. Cabe destacar que esta funcionalidad presente en los agentes de tipo proveedor podría no existir o estar implementada de distinta forma en una aplicación real. La razón de ello es que un agente no debe tener acceso a la base de datos directamente. Sin embargo, se podría implementar haciendo llamadas a los meta-servicios *Search Service* y *Get Process*, pero supondría un flujo de mensajes entre agentes que ralentizaría el proceso. En la aplicación desarrollada se ha incluido esta funcionalidad para dar más robustez al sistema y evitar errores de introducción de datos por parte del usuario.

4.2.2.4. Implementación de los servicios turísticos

Una de los objetivos de la plataforma THOMAS es poder ejecutar servicios de cualquier tipo. En este caso, la aplicación desarrollada trata sobre la gestión turística, por lo tanto, se han implementado un conjunto de servicios relacionados con el sector para mostrar su funcionamiento. Como ya se ha comentado anteriormente, los servicios desarrollados son: *Search Hotel*, *Reserve Hotel*, *Search Flight* y *Reserve Flight*.

El desarrollo de los servicios turísticos se ha llevado a cabo imitando a los meta-servicios ya desarrollados y siguiendo las directrices de la plataforma THOMAS. Con lo cual, se han empleado básicamente documentos WSDL y OWL-S.

Por una parte, con los documentos WSDL se definen las entradas y las salidas del servicio junto con otros parámetros necesarios. Así pues, utilizando los *plugins* adecuados en Eclipse (existe un tutorial sobre esto en la web de Eclipse WTP[2]), se puede usar

[2]http://www.eclipse.org/webtools/community/tutorials/BottomUpAxis2WebService/bu_tutorial.html

el fichero WSDL con los datos del servicio para generar automáticamente un conjunto de clases de Java. A continuación, se puede incluir código fuente en lenguaje Java para implementar los procesos que seguirá el servicio antes de devolver una respuesta. Finalmente, se deben empaquetar las clases en un archivo *war* que será colocado en el directorio *webapps* del servidor *Apache Tomcat*.

Por otra parte, es necesario incluir una descripción del servicio en lenguaje OWL-S. Para escribir los documentos necesarios de perfil y proceso en cada servicio se ha utilizado como modelo los existentes en la plataforma THOMAS. En estos documentos es necesario especificar la URL donde se encontrará el servicio y las entradas y salidas del mismo. Una vez escritos deben ser colocados en el directorio *webapps* del servidor *Apache Tomcat* dentro de las subcarpetas que corresponda.

Es necesario subrayar que los servicios de gestión turística desarrollados para esta aplicación tienen la finalidad de servir de ejemplo y de comprobar que realmente se pueden ejecutar en la plataforma THOMAS. Por ello, el proceso interno que realizan y las respuestas que devuelven no son fruto de ninguna búsqueda real. Sin embargo, se han implementado sencillas comprobaciones para que según las entradas facilitadas devuelvan un resultado distinto. Con ello se comprueba que reciben correctamente los argumentos y que pueden devolver distintos resultados.

4.3. Validación de la plataforma THOMAS

Una vez desarrollada la aplicación de gestión de servicios turísticos, se puede analizar la validez de la plataforma THOMAS sobre la que se ha basado este trabajo. Para ello, se mostrará al lector la ejecución de los escenarios explicados en el caso de estudio (sección 4.1) mediante la aplicación desarrollada. Esto permitirá conocer el conjunto de funcionalidades que se utilizan en la aplicación para reproducir los escenarios y comprobar el correcto funcionamiento de la plataforma THOMAS. El objetivo es demostrar la viabilidad del desarrollo de software sobre dicha plataforma y validar la ejecución de los servicios y meta-servicios.

4.3.1. Registro de los agentes como miembros de la plataforma THOMAS

La aplicación de gestión de servicios turísticos desarrollada está íntimamente ligada a la plataforma THOMAS. Esto es debido a que tanto los agentes intermediarios de THOMAS como los agentes de la aplicación comparten el mismo marco de ejecución en la plataforma JADE. Además, los visores del OMS y del SF que se han implementado para mostrar información al usuario dependen directamente del OMS y del SF. Con lo cual, se ha añadido código fuente en estos agentes intermediarios. Esto significa que la plataforma no es independiente de la aplicación, aunque en un caso real y típico, no existirían los visores y la independencia sería total.

Puesto que el marco de ejecución de los agentes de la aplicación y de la plataforma THOMAS es compartido, se lanzan todos al mismo tiempo. La aplicación dispone de un agente cliente (*Client*) y dos agentes proveedores: *HotelProvider* y *FlightProvider*. Como ya se ha comentado anteriormente, la primera acción que realizan los agentes de la aplicación es registrarse en la plataforma THOMAS. Para ello, hacen una llamada al meta-servicio *Acquire Role* para obtener el rol *member* dentro de la unidad *virtual*. Cabe destacar que esta adquisición del rol se hace automáticamente ya que es básico pertenecer a THOMAS para poder empezar a utilizar sus meta-servicios. En la figura 4.46 se observa que en la barra de estado del agente cliente se está haciendo una petición a *Acquire Role* al iniciar la aplicación. Esta es la imagen que ofrecen los agentes durante un breve periodo de tiempo al inicio de su ejecución.

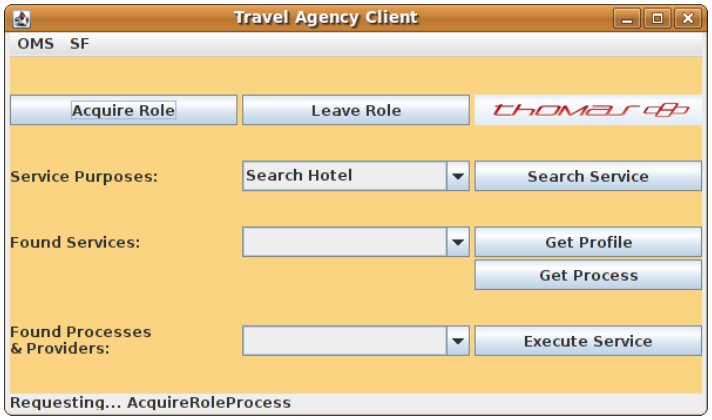

Figura 4.46: Agente cliente solicitando *Acquire Role* al inicio de la ejecución

Así pues, cuando los tres agentes hayan podido adquirir con éxito el rol de miembros en la organización virtual de THOMAS, se verá esta información en la tabla *Entity Play List* del visor del OMS (figura 4.47). Con este resultado, se puede ver que el escenario de registro de agentes en la plataforma THOMAS (figura 4.3) se ha podido ejecutar sin problema en todos los agentes presentes en la aplicación.

4.3.2. Registro de proveedores y servicios

Antes de que el cliente pueda utilizar algún servicio turístico, éstos deben ser registrados por parte de los agentes proveedores presentes en el sistema. Para poder proveer un servicio turístico en la aplicación se deben seguir unos pasos determinados. Básicamente, es necesario registrar el perfil del servicio y después el proceso o implementación del mismo. Con el registro del perfil se consigue que cualquier agente interesado en el servicio pueda buscarlo mediante el meta-servicio *Search Service* del SF. Por otra parte, el registro del proceso supone ser el proveedor del servicio con la implementación registrada. Esto significa que a partir de ese momento, cualquier agente puede saber quién ofrece el servicio y enviarle una petición para utilizarlo. A continuación se detalla cómo seguir estos pasos en la aplicación.

Primeramente se necesita registrar el perfil correspondiente al servicio que se desea proveer. Antes de hacerlo es obligatorio adquirir el rol o roles necesarios para ser proveedor del servicio. Con lo cual, se utilizará el botón *Acquire Role* de la interfaz que da acceso a la ventana en la que se elige el rol a adquirir. En este caso se obtendrá el rol *Provider* de la *TravelAgency* y después, dependiendo de si se trata del agente

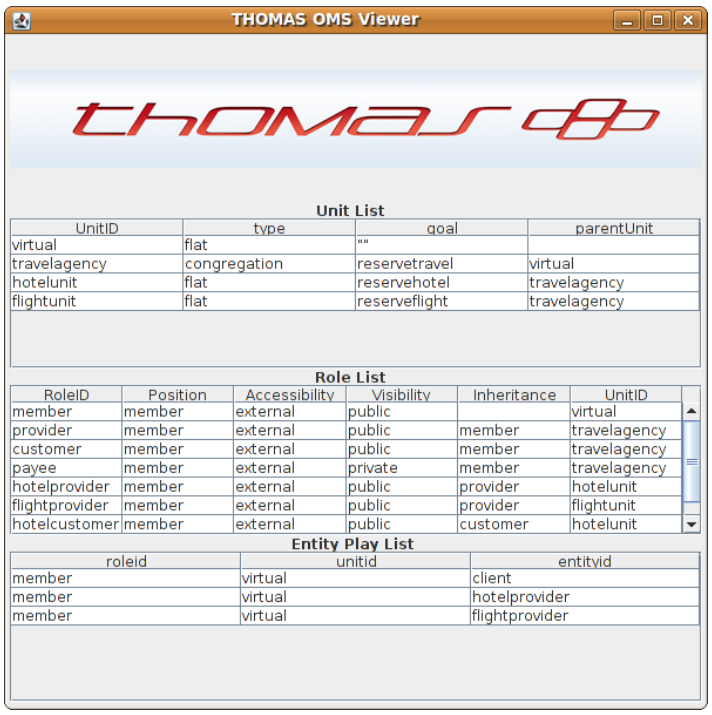

Figura 4.47: Visor del agente intermediario OMS después de que los agentes hayan adquirido el rol *member* en *virtual*

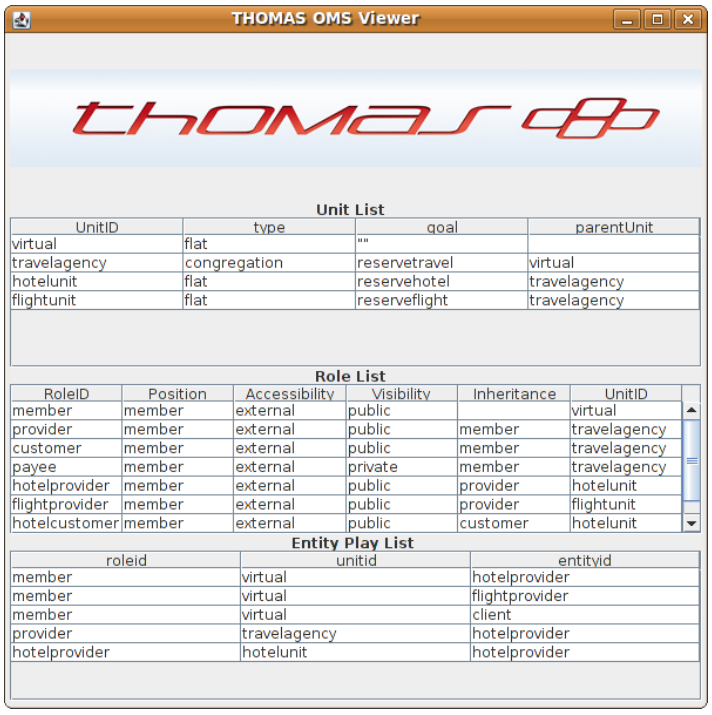

Figura 4.48: Visor del agente intermediario OMS después de que el agente *HotelProvider* haya adquirido los roles *Provider* de la *TravelAgency* y *HotelProvider* de la *HotelUnit*

Figura 4.49: Mensaje de advertencia de necesidad de registrar un perfil antes que un proceso

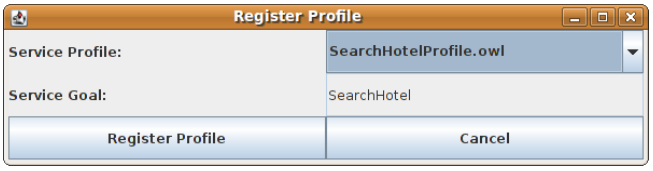

Figura 4.50: Ventana de ejecución del servicio *Register Profile*

HotelProvider o *FlightProvider*, se adquirirá el rol *HotelProvider* de la *HotelUnit* o el rol *FlightProvider* de la *FlightUnit*. Por lo tanto, si tomamos como ejemplo el agente *HotelProvider*, se puede ver en la tabla *Entity Play List* del visor del OMS de la figura 4.48 que se han obtenido los roles necesarios. Así pues, en este punto se podría dar por concluido el escenario de registro de un proveedor de hoteles (figura 4.4), ya que se han adquirido los roles que acreditan al agente como tal.

Cuando ya se han obtenido los roles, se puede registrar el perfil del servicio a proveer. Para ello se utiliza el botón *Register Profile* del agente proveedor en cuestión. Cabe destacar que si se intenta registrar un proceso antes de haber registrado un perfil, la aplicación nos avisará de que no es posible (con un mensaje como el de la figura 4.49). Esto ocurre porque se controla que sólo se puede registrar un proceso cuyo perfil ya haya sido registrado. Si el agente proveedor ya ha registrado algún perfil, se dará la opción de registrar el proceso correspondiente al perfil ya registrado, pero no otros sin registrar. Con lo cual, al pulsar el botón *Register Profile* aparecerá la ventana de la figura 4.50 con los perfiles que se pueden registrar según el agente proveedor. En el caso del *HotelProvider* serán los servicios relativos a la búsqueda y reserva de hoteles, y en el caso del *FlightProvider* estarán disponibles los servicios relativos a la búsqueda y reserva de vuelos. Así pues, se elegirá el perfil que se desee registrar y se pulsará el botón correspondiente en la ventana. Si la operación tiene éxito, la aplicación lo notificará adecuadamente y aparecerá en el visor del SF el perfil que se acaba de registrar (ejemplo en la figura 4.51).

Una vez se ha registrado el perfil del servicio, es posible registrar el proceso del mis-

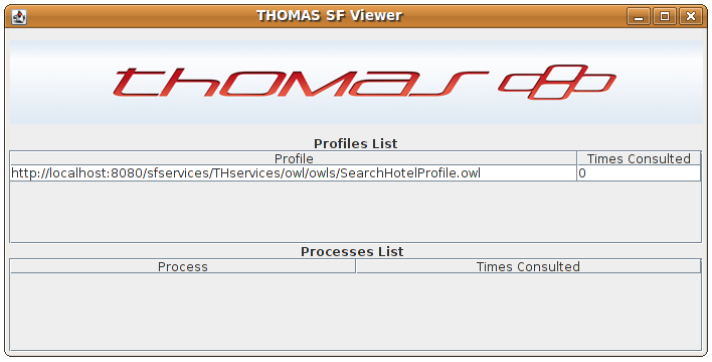

Figura 4.51: Visor del agente intermediario SF con un perfil registrado

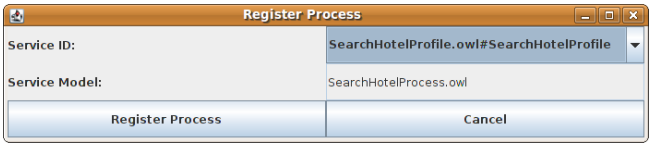

Figura 4.52: Ventana de ejecución del servicio *Register Process*

mo con el agente proveedor correspondiente. Para ello se debe pulsar el botón *Register Process* de la interfaz del agente proveedor y en la ventana que aparece (figura 4.52) seleccionar el perfil registrado previamente. Con lo cual, al pulsar el botón de la ventana se registrará una implementación del perfil seleccionado y el agente se convertirá en proveedor de este servicio. La aplicación informará al usuario de si la operación tiene éxito o no, y en caso afirmativo se podrá ver el proceso registrado en el visor del SF (ejemplo en la figura 4.53). Por lo tanto, se ha conseguido reproducir con éxito el escenario de registro de un proceso (figura 4.5).

A partir de este momento, el agente cliente ya dispone de un servicio que puede utilizar. Para que el resto de servicios turísticos estén disponibles en la aplicación se

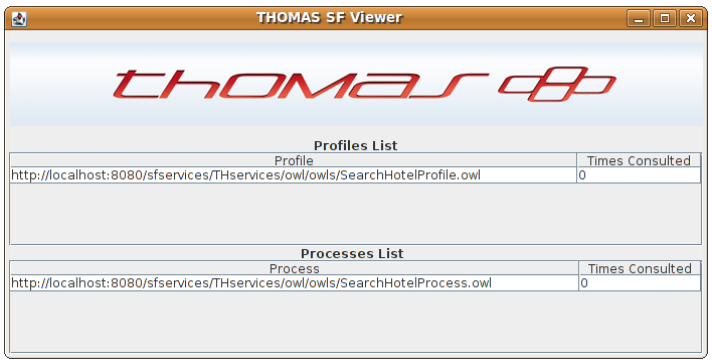

Figura 4.53: Visor del agente intermediario SF con un perfil y un proceso registrados

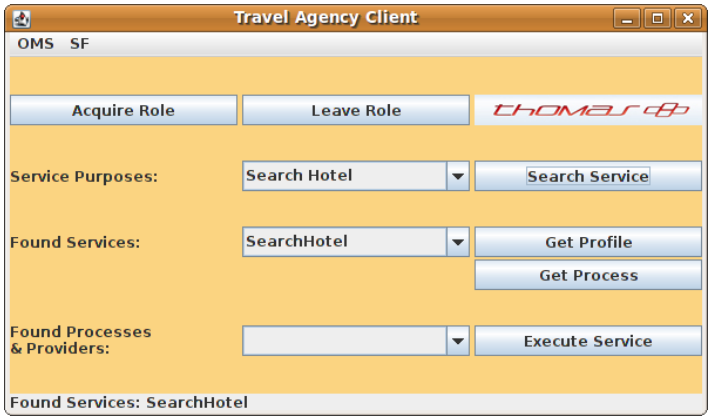

Figura 4.54: Agente cliente después de encontrar el servicio *Search Hotel*

deben seguir los pasos que se acaban de detallar. En resumen, desde el agente proveedor se deben adquirir los roles necesarios, registrar el perfil del servicio y el proceso o implementación del mismo.

4.3.3. Búsqueda y utilización de servicios turísticos

Cuando ya se dispone de servicios turísticos registrados y con proveedores que pueden servirlos, el cliente puede buscarlos para su utilización. A continuación se explicarán los pasos a seguir para encontrar un servicio turístico que desee el cliente y hacer la petición pertinente para su uso.

Lo primero que debe hacer un cliente para utilizar un servicio es buscarlo mediante el meta-servicio *Search Service* del SF. En la interfaz del agente cliente de la aplicación existe una lista desplegable etiquetada como *Services Purposes* en la que se debe elegir el propósito del servicio a buscar. Una vez elegido se puede pulsar el botón *Search Service* con el que se hace la llamada al meta-servicio del mismo nombre. Si existe un perfil de servicio registrado en el SF con el propósito que se buscaba aparecerá en la siguiente lista desplegable etiquetada como *Found Services*. En caso contrario la aplicación avisará de que no se ha encontrado ningún perfil con dichas características. En la figura 4.54 se puede ver el agente cliente después de haber encontrado el servicio *Search Hotel*.

En este momento se pueden realizar dos acciones: obtener el perfil del servicio, o el proceso y proveedor del mismo. El perfil del servicio se obtendrá sin problema con *Get Profile*, ya que si no existiera no se hubiera encontrado con el meta-servicio *Search*

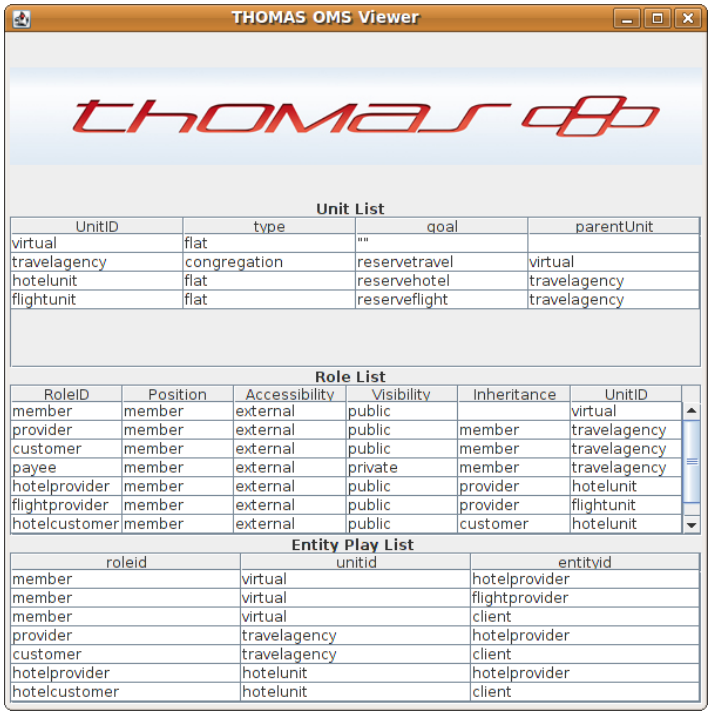

Figura 4.55: Visor del agente intermediario OMS después de que el agente *Client* haya adquirido los roles *Customer* de la *TravelAgency* y *HotelCustomer* de la *HotelUnit*

Service. Por otra parte, si no se ha registrado ningún proceso referente al perfil del servicio, no se encontrará nada al llamar al meta-servicio *Get Process*. En caso que sí exista algún proceso registrado, esta llamada al meta-servicio del SF devolvería el proceso en cuestión y el proveedor del mismo. De todas formas, la acción que se debe realizar primero es obtener el perfil del servicio, mediante el meta-servicio *Get Profile*, con el objetivo de consultar los roles que hay que poseer para convertirse en cliente del mismo. Así pues, para este caso se debe poseer el rol *Customer* de la *TravelAgency* y el rol *HotelCustomer* de la *HotelUnit*. Con lo cual, se hacen las llamadas pertinentes al meta-servicio *Acquire Role* del OMS con el botón del mismo nombre disponible en la interfaz del agente. Una vez obtenidos los roles necesarios (en la figura 4.55 se puede observar el visor del OMS con los roles registrados para el agente *Client*) , ya se puede solicitar la ejecución del servicio. En este punto, el escenario de registro de un cliente (figura 4.6) se ha reproducido con éxito. Así pues, ya se puede pasar al siguiente escenario sobre la petición del servicio. Para ello hay que obtener mediante el meta-servicio *Get Process* el proceso del servicio y el proveedor del mismo, en caso de que se haya registrado. Si hay un proveedor que ha registrado el proceso pertinente, el agente cliente quedará como en la figura 4.56. Se puede observar que en la lista desplegable *Found Processes & Providers* aparece el nombre del proceso del servicio *Search Hotel* y el

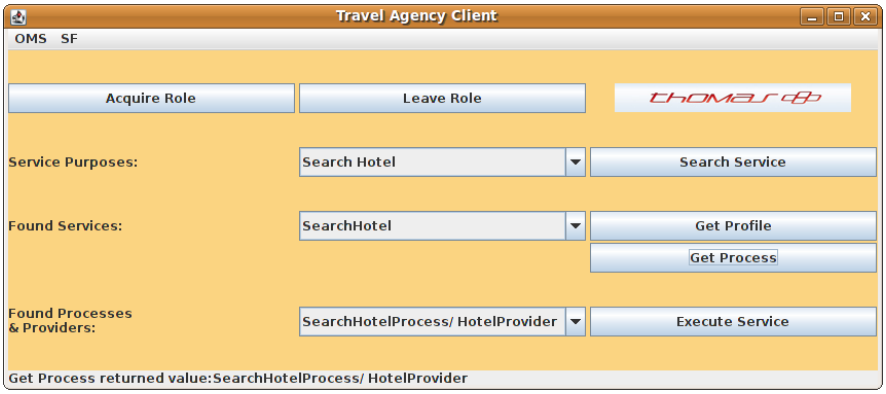

Figura 4.56: Agente cliente después de obtener el proceso y el proveedor del servicio *Search Hotel*

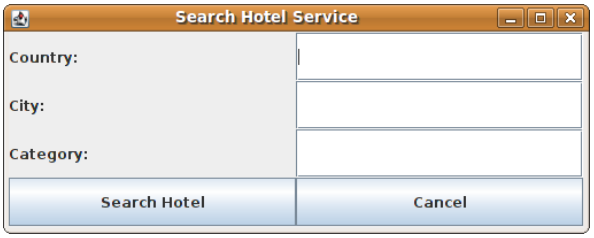

Figura 4.57: Ventana del servicio *Search Hotel*

proveedor del mismo, *HotelProvider*.

Finalmente se ha obtenido el proveedor del servicio y la implementación o proceso que ofrece. Por lo tanto, el agente cliente ya puede hacer la petición de servicio al proveedor utilizando el botón *Execute Service* con la opción correspondiente seleccionada en la lista desplegable. Al apretar el botón, aparece una ventana con las entradas para rellenar del servicio a ejecutar (figura 4.57). Una vez se hayan completado correctamente todas las entradas se podrá hacer la llamada al mismo enviando la petición al proveedor. En la figura 4.58 se puede ver una imagen del agente *Sniffer* de la plataforma JADE. Se ha capturado el flujo de mensajes enviados entre los agentes cliente y proveedor. En total se han enviado tres mensajes: *FIPA_Request* para la petición de servicio del cliente al proveedor, *Agree* del proveedor al cliente aceptando proveer el servicio y el *Inform* del proveedor al cliente notificando el resultado del servicio. Cuando se recibe el resultado de la ejecución del servicio, el agente cliente lo muestra para que el usuario lo pueda ver (figura 4.59). Con lo cual, se ha finalizado el escenario de petición de servicio con éxito (figura 4.7).

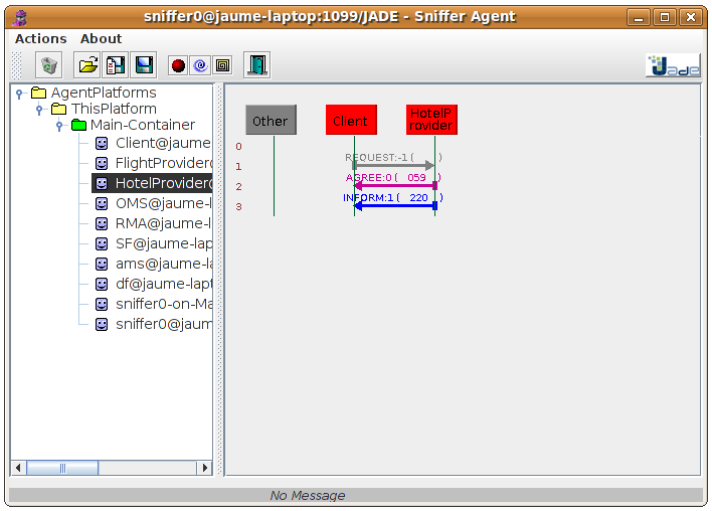

Figura 4.58: Ventana del agente Sniffer de JADE capturando los mensajes enviados entre el cliente y el proveedor *HotelProvider*

Figura 4.59: Ventana informativa después del éxito de la ejecución del servicio *Search Hotel*

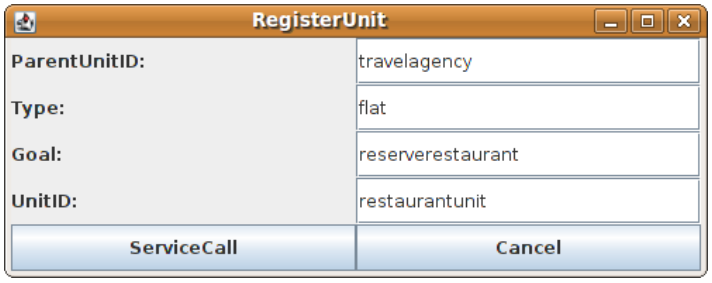

Figura 4.60: Ventana del meta-servicio *Register Unit* con los campos rellenados para registrar la unidad *RestaurantUnit*

4.3.4. Otros meta-servicios

Una vez se han mostrado las funcionalidades típicas y más utilizadas se puede pasar a probar el funcionamiento de otras cuestiones que se pueden encontrar en la aplicación. A continuación, se utilizarán los meta-servicios para registrar y eliminar unidades, roles y normas, así como para expulsar agentes. Puesto que estas funcionalidades no son las más utilizadas en el sistema, no disponen de ningún botón en las interfaces de los agentes. Pero están contempladas en las barras de menú.

Una de las ventajas de la plataforma THOMAS es su gestión de las organizaciones virtuales. Para ello, se ha creado un sistema de registro y mantenimiento de unidades organizativas y roles. En algún momento de la vida del sistema podría ser necesario registrar una nueva unidad con sus correspondientes roles para incluir otra organización en la plataforma. Así pues, se podrían utilizar los meta-servicios *Register Unit* y *Register Role* del OMS. En la aplicación desarrollada se ha incluido la posibilidad de utilizar estos meta-servicios mediante las opciones de la barra de menú.

Para registrar una nueva unidad organizativa se puede usar la barra de menú de cualquiera de los agentes de la aplicación. La opción se encuentra en OMS, *Structural Services*, *Register Services*. Con lo cual, pulsando sobre *Register Unit* se abrirá una ventana con las entradas a facilitar. En esta ventana se podrían rellenar las entradas del meta-servicio como se ve en la figura 4.60. Así pues, ejecutando la llamada al meta-servicio y en caso de que tenga éxito, aparecería reflejada la nueva unidad en el visor del OMS como se puede ver en la figura 4.61. En este punto se habría ejecutado con éxito el escenario de creación de una nueva unidad (figura 4.8) explicado en la sección 4.1.

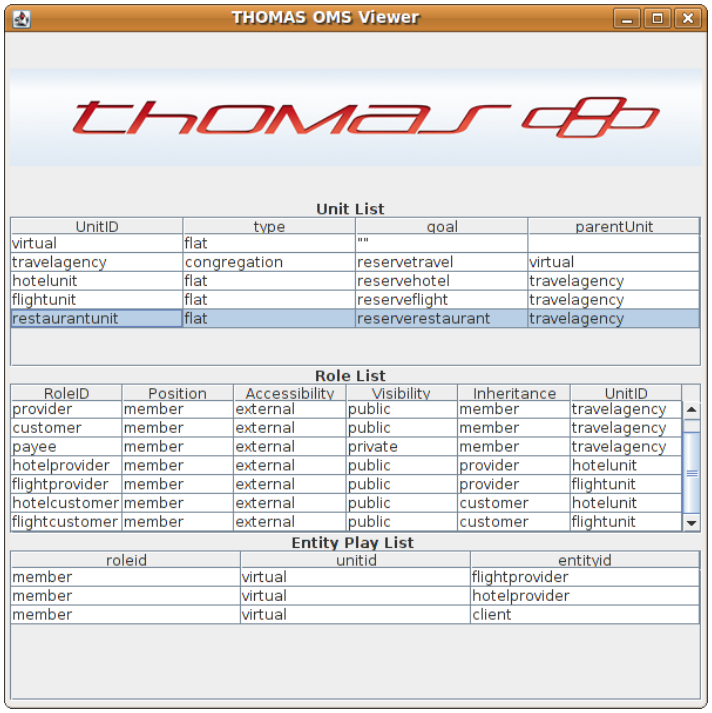

Figura 4.61: Visor del agente intermediario OMS después de registrar la nueva unidad *RestaurantUnit*

Cuando se registra una nueva unidad organizativa en el sistema, lo más lógico es registrar roles dentro de dicha unidad. Así pues, en la barra de menú de la aplicación se dispone de una opción para utilizar el meta-servicio *Register Role*. Dicha opción se encuentra en el mismo submenú que *Register Unit*. La ventana que aparece al pulsar sobre la opción del menú *Register Role* contiene las entradas a rellenar por parte del usuario. En la figura 4.62 se puede ver un ejemplo de los argumentos introducidos para ejecutar el meta-servicio y registrar un nuevo rol *RestaurantProvider* dentro de la unidad *RestaurantUnit* creada previamente. Puesto que el meta-servicio tiene éxito, aparece en el OMS (figura 4.63) el nuevo rol registrado. De la misma forma se podrían registrar el resto de roles necesarios dentro de la unidad. Así pues, se podría dar por concluido el escenario de creación de nuevos roles (figura 4.9).

En algún momento de la ejecución de la aplicación podría ser necesario eliminar alguna de las organizaciones registradas en la plataforma. Con lo cual, existen los meta-servicios *Deregister Role* y *Deregister Unit* para poder realizar estas acciones. Mediante las barras de menú OMS, *Structural Services*, *Deregister Services*, se accede a dichos meta-servicios. Cabe destacar que para eliminar una unidad registrada en el OMS es necesario eliminar previamente todos los roles que pertenezcan a ella. Si no se realiza de

Figura 4.62: Ventana del meta-servicio *Register Role* con los campos rellenados para registrar el rol *RestaurantProvider*

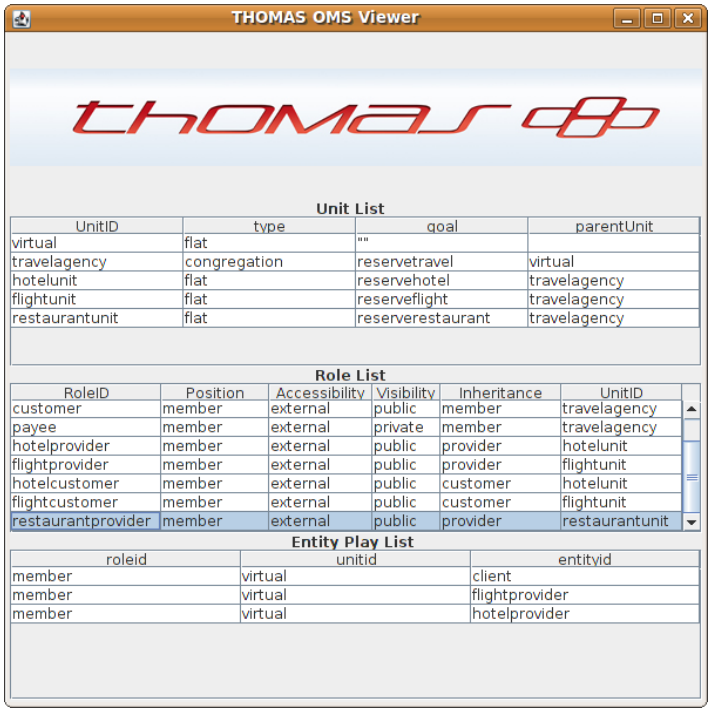

Figura 4.63: Visor del agente intermediario OMS después de registrar el nuevo rol *RestaurantProvider* dentro de la unidad *RestaurantUnit*

Figura 4.64: Ventana del meta-servicio *Deregister Role* con los campos rellenados para eliminar el rol *RestaurantProvider*

Figura 4.65: Ventana del meta-servicio *Deregister Unit* con su único campo rellenado para eliminar la unidad *RestaurantUnit*

este modo, la plataforma THOMAS no permite la eliminación de la unidad. Así pues, se procederá a eliminar los roles de la unidad *RestaurantUnit* mediante el meta-servicio *Deregister Role*. Como se puede ver en la figura 4.64, se rellenan los campos indicando el rol que se desea eliminar y la unidad a la que pertenece.

Una vez se han eliminado todos los roles de la unidad *RestaurantUnit*, se puede proceder a eliminarla mediante el meta-servicio *Deregister Unit* del OMS. Para ello se debe especificar la unidad a eliminar en el único campo que se debe rellenar en dicho meta-servicio (figura 4.65). Cuando se ejecute el servicio con éxito se habrá eliminado por completo la unidad organizativa.

Otra de las ventajas que ofrece la plataforma THOMAS es la posibilidad de expulsar agentes maliciosos del sistema. Esta medida es necesaria ya que en cualquier momento puede entrar algún agente con objetivos contrarios a la organización. Así pues, existe un meta-servicio disponible en el OMS llamado *Expulse* mediante el cual se puede eliminar cualquier rol que posea un agente. Esto ofrece la seguridad necesaria en un sistema dinámico y cambiante. A continuación se simulará el escenario correspondiente a la expulsión de un agente de la aplicación.

Un contexto posible para que se diera el caso necesario de expulsar a un agente sería el intento fraudulento de realizar un pago con cuentas falsas. Sin embargo, en la aplicación desarrollada no se han incluido mecanismos de pago. Con lo cual, se asimi-

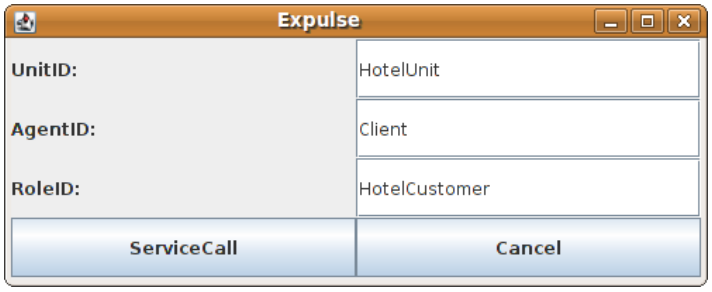

Figura 4.66: Ventana del meta-servicio *Expulse* con los campos rellenados para expulsar a un agente malicioso

lará que el cliente está haciendo demasiadas reservas consecutivas y es sospechoso de ser un agente malicioso. Por lo tanto, el agente proveedor *HotelProvider* realizará la llamada al meta-servicio *Expulse* del OMS para expulsar al cliente del rol *HotelCustomer*. Para ello, hay que acceder a la opción correspondiente de la barra de menú del agente, es decir, OMS, *Dynamical Services*. Una vez se ha pulsado la opción *Expulse*, se tendrá acceso a la ventana de petición del meta-servicio. En este caso, se rellenaría como en la figura 4.66 puesto que se necesita expulsar al agente *Client* de la unidad *HotelUnit* donde ha protagonizado un comportamiento malicioso. La petición del meta-servicio *Expulse* la hace el agente proveedor de hoteles *HotelProvider* ya que es quien ha detectado el comportamiento inadecuado. Finalmente, no hay ningún problema en la ejecución del meta-servicio y el agente *Client* ha perdido el rol *HotelCustomer*, con lo que no podrá utilizar más los servicios de la unidad *HotelUnit*. Así pues, se ha podido reproducir el escenario de expulsión de un agente malicioso (figura 4.10).

La plataforma THOMAS también ofrece un sistema de normas. Cualquier agente registrado en la plataforma puede definir una norma, por ejemplo, para limitar el acceso a ciertos roles. La definición de normas aporta seguridad y control al sistema de organizaciones de la plataforma. Dada la importancia de las normas, se ha incluido el meta-servicio *Register Norm* del OMS para poder registrarlas. Sin embargo, la definición y gestión de normas no es un punto importante en este trabajo. Así pues, con la opción que se encuentra en OMS, *Structural Services*, *Register Services* se tiene acceso a la ventana del meta-servicio *Register Norm*. A modo de ejemplo, se asumirá que el agente que registra la norma ha obtenido el rol *Payee*. Con lo cual, debe limitar el acceso a este rol para ser la única entidad financiera que trabaja en esta organización. Así pues, se rellenarían los campos del meta-servicio *Register Norm* como en la figura 4.67. En este caso, la norma significa que no se aceptará ningún mensaje de tipo

Figura 4.67: Ventana del meta-servicio *Register Norm* con los campos rellenados para registar una nueva norma

Request que contenga una petición *Acquire Role* con el rol *Payee* como argumento. Pulsando el botón correspondiente de la ventana del meta-servicio del OMS se envía la petición pertinente y se obtendrá una respuesta afirmativa si la ejecución tiene éxito.

4.3.5. Valoración final

En toda esta sección se ha mostrado el funcionamiento de la aplicación junto con la plataforma THOMAS. Se han recreado con éxito todos los escenarios del caso de estudio explicando los pasos que se deben seguir en la aplicación. Además, se han mostrado las ejecuciones reales sobre la aplicación para que los usuarios sean capaces de reproducir cualquier situación. Es por ello que se puede afirmar que es completamente viable desarrollar software basado en la plataforma THOMAS. Por lo tanto, con el correcto funcionamiento de los meta-servicios y de la plataforma en general, el objetivo de validar la plataforma THOMAS ha sido cumplido.

Por otra parte, es necesario comentar que la plataforma THOMAS aun está en desarrollo y no se dispone de una versión final. Con lo cual, la versión disponible actualmente es un prototipo. Esto implica que pueden existir fallos y funcionalidades sin implementar. A lo largo del desarrollo de la aplicación de gestión de servicios turísticos, se han encontrado ciertas dificultades por pequeños fallos existentes en la plataforma THOMAS. En la mayoría de los casos, estos fallos se han comunicado y han sido solucionados por el personal de desarrollo de la plataforma THOMAS. Además, a continuación se comentan ciertas mejoras que se han realizado o se están realizando durante el desarrollo de este trabajo:

- Mejora de la instalación de la plataforma. Uno de los primeros problemas con los que se podía encontrar el usuario de la plataforma era su propia instalación. Sin embargo, durante la elaboración de este trabajo, el personal desarrollador

de THOMAS ha mejorado notablemente este aspecto y ya no plantea mucha dificultad.

- Ocultar las URL de los servicios cuando no son necesarias. En muchas ocasiones se hace obligatorio utilizar las URL de los servicios o son mostradas por la plataforma sin necesidad de ello. Por lo tanto, sería conveniente un cambio añadiendo soporte para identificadores de servicio que no incluyan la URL. Actualmente se tiene constancia de que el equipo de desarrollo de la plataforma THOMAS está trabajando en esta cuestión y será una funcionalidad presente en la siguiente versión.

Después de haber observado el comportamiento de la plataforma THOMAS y haberla utilizado durante el desarrollo de este trabajo, han surgido ideas que podrían mejorar el rendimiento y utilización de dicha plataforma. Con lo cual, en el último capítulo se propondrá un conjunto de mejoras para la plataforma THOMAS.

Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

A lo largo de los capítulos anteriores se ha mostrado el trabajo realizado para este proyecto. Primeramente, se ha hecho un estudio de la plataforma THOMAS para adquirir un nivel de comprensión adecuado y tener la capacidad de desarrollar una aplicación que utilice los mecanismos que ofrece la plataforma. Una vez superado este obstáculo se ha planteado un diseño para la aplicación, en el que la principal intención ha sido facilitar el uso de la misma al usuario y mostrar de forma transparente cómo funciona la plataforma THOMAS. Finalmente se ha implementado la aplicación siguiendo las directrices de la plataforma para obtener un buen funcionamiento y demostrar que es viable desarrollar y adaptar software con THOMAS. Con lo cual, se puede afirmar que se han cumplido los objetivos planteados al inicio de este trabajo.

Además, dada la complejidad inherente al desarrollo de software basado en sistemas multiagente y servicios web, se puede afirmar que ha sido posible implementar una aplicación robusta y flexible. Por una parte, la dificultad que entraña tratar con una plataforma de SMA que incluye servicios web a los que se debe acceder de forma concreta, y teniendo en cuenta que se trabaja normalmente con cadenas de caracteres, hay que destacar el buen funcionamiento de la aplicación debido a la acotación realizada y las ayudas proporcionadas al usuario. Es decir, limitando las opciones disponibles en algunos casos para no provocar fallos causados por la inexperiencia del usuario en el funcionamiento del sistema. Por otra parte, dada la limitación de algunos contenidos que se pueden considerar más intuitivos, la aplicación ofrece un conjunto de opciones a través de las barras de menú con las que el usuario avanzado dispone de un control y una flexibilidad total para el uso de todos los meta-servicios que ofrece la plataforma.

Es importante destacar que mediante el desarrollo de la aplicación de gestión de servicios turísticos se ha podido demostrar la validez de la plataforma THOMAS para desarrollar aplicaciones basadas en ella. Con las pruebas realizadas y la propia aplicación queda constancia de que los meta-servicios y mecanismos de THOMAS funcionan correctamente y cumplen sus objetivos.

A modo de resumen, a continuación se muestran las aportaciones de este trabajo:

- Diseño de interfaces gráficas de usuario para agentes de tipo proveedor y cliente. Se ha realizado un estudio para obtener un diseño cómodo e intuitivo para el usuario. La intención es facilitar la comprensión al usuario de los meta-servicios de la plataforma THOMAS y de su utilización general.
- Diseño de visores para los agentes intermediarios OMS y SF de la plataforma THOMAS. Esto ha implicado añadir código fuente a ambos agentes, con lo que se ha dotado a la plataforma de nuevos mecanismos para la visualización de su actividad.
- Implementación de agentes para la aplicación. Se ha implementado la gestión de los eventos que provoca el usuario a través de la interfaz gráfica, así como las funcionalidades para informar y guiar al usuario durante la ejecución. Además, se ha incluido un sistema de comunicación para el envío y la recepción de mensajes entre los agentes.
- Implementación de servicios web de gestión turística. Se han desarrollado cuatro servicios web relacionados con la gestión turística que son utilizados por la misma aplicación.
- Aplicación para la gestión de servicios turísticos. Con el diseño y la implementación de todos los componentes que forman la aplicación se ha obtenido un producto final ejecutable y novedoso.
- Estudio y validación de la plataforma THOMAS. Primeramente se ha analizado cómo es y cómo funciona la plataforma comprendiendo sus mecanismos y meta-servicios. Seguidamente, se ha comprobado el funcionamiento de la plataforma recreando escenarios en los que están implicados todos los meta-servicios de THOMAS. Finalmente, se ha podido demostrar que es viable desarrollar software

sobre dicha plataforma ya que sus mecanismos y meta-servicios funcionan correctamente.

5.2. Trabajos futuros

Para finalizar, después de todo el trabajo realizado, aun se pueden plantear mejoras y nuevos retos a tener en cuenta en el futuro. Se proponen trabajos futuros para la aplicación desarrollada y para la plataforma THOMAS.

Así pues, como trabajos futuros referentes a la aplicación desarrollada sobre gestión de servicios turísticos, se pueden hacer las siguientes propuestas:

- Ampliación de la aplicación para poder incluir agentes situados en ubicaciones remotas. Así pues, cualquier agente que cumpliese con los requisitos mínimos podría entrar a formar parte del sistema y ofrecer sus servicios.
- Implementación de servicios de gestión turística que extraigan información real a través de la web y actúen de forma inteligente. En los servicios de búsqueda se debería establecer una conexión con un gestor apropiado para encontrar hoteles y vuelos reales con disponibilidad. Complementariamente se tendría la opción de hacer una reserva de los hoteles o vuelos encontrados. La búsqueda y reserva se podrían hacer como servicios simples y también juntarlos en un servicio compuesto. Además, habría que tener en cuenta la inclusión de cierta inteligencia en los servicios para poder clasificar y tratar con todos los datos extraídos de las búsquedas.
- Adaptaciones a las próximas versiones de la plataforma THOMAS. Como ya se ha comentado, la plataforma se encuentra en desarrollo. Por lo tanto, se hace necesario adaptar la aplicación a los cambios y nuevas funcionalidades que puedan incluir las nuevas versiones de la plataforma THOMAS.

Por otra parte, desde el punto de vista de la plataforma THOMAS, se pueden proponer los siguientes trabajos y mejoras:

- Comprobación de los roles que posee un agente antes de permitir la ejecución de un servicio. El funcionamiento de la plataforma está pensado para que un agente

pueda utilizar un servicio si posee los roles que se exigen. Este mecanismo de comprobación aun no se encuentra en funcionamiento en la plataforma THOMAS y es bastante necesario.

- Composición de servicios. Sería útil disponer de meta-servicios compuestos por otros meta-servicios simples. De esta forma sería más ágil la actividad con la plataforma. Un ejemplo podría ser la fusión del meta-servicio *Search Service* con *Get Profile* y *Get Process*. Así pues, se podría obtener directamente el perfil y proceso del servicio que se haya encontrado con mayor índice de coincidencia.

- Tolerancia a fallos. Se ha observado durante el desarrollo de este trabajo que la plataforma THOMAS es poco tolerante a fallos o ejecuciones erróneas. Con lo cual, si hay comportamientos inesperados o algún argumento incorrecto en un meta-servicio se puede causar la detención de algún agente intermediario.

- Inteligencia en los servicios web. La plataforma THOMAS podría disponer de algún servicio de ejemplo que tuviera mecanismos que incorporasen inteligencia artificial y búsqueda inteligente en la web. Esto serviría para que cualquier usuario o desarrollador pudiera ver el potencial de la plataforma y lo que se puede realizar.

- Ejecución de agentes remotos. Podría existir algún ejemplo real en el que se puedan ejecutar agentes ubicados en distintas máquinas para demostrar que es viable esta opción.

Bibliografía

[1] E. Argente, V. Julián, and V. Botti. Mas modelling based on organizations. In *9th Int. Workshop on Agent Oriented Software Engineering (AOSE08)*, volume 5386, pages 16–30. Springer-Verlag, 2008.

[2] C. Caceres, A. Fernandez, S. Ossowski, and M. Vasirani. Role-based service description and discovery. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[3] C. Carrascosa, A. Giret, V. Julian, M. Rebollo, E. Argente, and V. Botti. Service oriented multi-agent systems: An open architecture. In *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1–2, 2009.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* http://www.w3.org/TR/wsdl, 2001.

[5] J. Dang and M. Hungs. Concurrent multiple-issue negotiation for internet-based services. In *IEEE Internet Computing*, volume 10 - 6, pages 41–49, 2006.

[6] M. Dastani, V. Dignum, and F. Dignum. Role-assignment in open agent societies. In *2nd Int. Joint Conference on Autonomous Agents and Multi-agent Systems*, pages 489–496, 2003.

[7] M. Dean and G. Schreiber. *OWL Web Ontology Language Reference.* Editors, W3C Recommendation. http://www.w3.org/TR/owl-ref/, 2004.

[8] M. Esteva, J. Rodriguez, C. Sierra, P. Garcia, and J. Arcos. On the formal specification of electronic institutions. In *Agent Mediated Electronic Commerce*, volume 1991, pages 126–147, 2001.

[9] F. Garijo. Tecnología de agentes: Experiencias y perspectivas para el desarrollo de nuevos servicios y aplicaciones. In *Bole.tic*, volume 24, pages 1–9, 2002.

[10] A. Giret, V. Julian, M. Rebollo, E. Argente, C. Carrascosa, and V. Botti. An open architecture for service-oriented virtual organizations. In *Seventh international Workshop on Programming Multi-Agent Systems. PROMAS 2009*, pages 23–33, 2009.

[11] J. Gonzalez-Palacios and M. Luck. Towards compliance of agents in open multi-agent systems. In *Software Engineering for Multi-Agent Systems V, Lecture Notes in Computer Science*. Springer, 2007.

[12] D. Greenwood and M. Calisti. Engineering web service - agent integration. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1918–1925, 2004.

[13] D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. The ieee fipa approach to integrating software agents and web services. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7, ACM, New York, NY, USA, 2007.

[14] Ana Mas. *Agentes Software y Sistemas Multi-Agente. Conceptos, Arquitecturas y Aplicaciones*. Pearson. Prentice Hall, 2005.

[15] T.Ñguyen and R. Kowalczyk. *Ws2jade: Integrating web service with jade agents*. Centre for Intelligent Agents and Multi-Agent Systems, Swinburne University of Technology, 2005.

[16] G. M. P. O'Hare and Nicholas R. Jennings. *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, Inc., New York, NY, 1996.

[17] M. Sensoy, C. Pembe, H. Zirtiloglu, P. Yolum, and A. Bener. Experience-based service provider selection in agent-mediated e-commerce. In *Engineering Applications of Artificial Intelligence*, volume 3, pages 325–335, 2007.

[18] MO. Shafiq, A. Ali, HF. Ahmad, and H. Suguri. Agentweb gateway - a middleware for dynamic integration of multi agent system and web services framework. In *14th IEEE International Workshops on Enabling Technologies (WETICE 2005)*, pages 267–270, Linköping, Sweeden, 2005. IEEE Computer Society.

[19] Y. Shoham. Agent-oriented programming. In *Artificial Intelligence*, volume 60(1), pages 51–92, 1993.

[20] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. In *IEEE Internet Computing*, volume 8 - 3, pages 66–73, 2004.

[21] K. Sycara, S. Widoffand, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. In *Jounal on Autonomous Agents and Multi-Agent Systems*, 1982.

[22] E. Del Val, N. Criado, M. Rebollo, E. Argente, and V. Julian. Service-oriented framework for virtual organizations. In *International Conference on Artificial Intelligence (ICAI)*, volume 1, pages 108–114. CSREA Press, 2009.

[23] LZ. Varga and Á. Hajnal. Engineering web service invocations from agent systems. In *Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691, pages 626–635, Prague, Czech Republic, June 16-18, 2003. Proceedings, Springer, Lecture Notes in Computer Science.

[24] M. Wooldridge. *An introduction to Multiagent Systems.* John Wiley and Sons Ltd, 2002.

[25] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. In *ACM Transactions on Software Engineering and Methodology*, volume 12, pages 317–370, 2003.