ÁÁAAAAAAAAAAAAAAAAAAAA ARTÍCULO

Emparejamiento de Servicios Web Semánticos y su aplicación en el dominio de tráfico rodado.

J. Javier Samper, Francisco Matas

Instituto de Robótica, Universitat de València
Polígono de la Coma s/n,
46980 Paterna, Valencia, Spain.
{jsamper,pacoma}@robotica.uv.es

Eduardo Carrillo

Universidad UNAB
Calle 48 Nro. 39-234,
Bucaramanga, Colombia,
ecarrill@unab.edu.co

Resumen

En este artículo se describe el sistema multiagente basado en estándares FIPA desarrollado para facilitar, asistir y optimizar los procesos de anuncio, descubrimiento e invocación de servicios web semánticos relacionados con la información de tráfico rodado. La función principal del sistema es proporcionar a sus clientes el servicio web que más se aproxime al que estos estén buscando, basándose para ello en su descripción semántica mediante lenguaje DAML-S / OWL-S. Las decisiones tomadas para encontrar el servicio buscado se apoyan en ontologías DAML / OWL y una extensión de algoritmos de emparejamiento existentes. El sistema fue ideado en principio para ser usado en el área de tráfico pero puede ser utilizado en cualquier otro dominio si se hace uso de las ontologías adecuadas.

Palabras clave: JADE, DAML, OWL, DAML-S, OWL-S, sistemas multiagente, servicios web, web semántica, XML, emparejamiento, matchmaking, matchmaker, emparejador, web, FIPA, WSDL, UDDI.

1. Introducción

La web inicial estaba caracterizada por la interacción con documentos, generalmente disponibles a través de portales web, pero dentro de su evolución han surgido los servicios web (SW) como un mecanismo que permite interacción entre aplicaciones. El servicio web es un componente software que puede procesar un documento XML recibido a través de alguna combinación de protocolos de transporte y de aplicación estándar en Internet. En este contexto es muy importante facilitar el proceso de descubrimiento de un servicio que cumpla unas características determinadas, y suele estar basado en la búsqueda de descripciones XML de los servicios disponibles. Es así como se hace por ejemplo en UDDI [32]. Pero debido a la falta de expresividad de XML, dos descripciones iguales de un SW tienen un significado u otro dependiendo del contexto en el que se encuentren. Para solucionar esta carencia se creó, a partir de las propuestas de la web semántica, la ontología de orden superior DAML-S,

que más tarde evolucionó a OWL-S (basadas en los lenguajes de marcado semántico DAML+OIL y OWL (Ontology Web Language) respectivamente [7] [35] para descripción semántica de las capacidades de los servicios web. Estas ontologías se utilizan para que agentes software puedan leer esas descripciones y razonar sobre la forma de interactuar con los servicios que describen, además de decidir cuáles se acercan más a sus necesidades. En este artículo describimos nuestro prototipo de emparejador semántico de servicios web y su integración en una plataforma multiagente basada en estándares FIPA [11]. En la actualidad existen algunas propuestas de integración para emparejamiento de SW como las presentadas en [19], [9] y [6]. La arquitectura de integración descrita en [6] emplea técnicas de recuperación de información, inteligencia artificial e ingeniería de software para procesar la semejanza sintáctica y semántica entre descripciones de capacidad de servicios descritos en DAML-S. En nuestro caso también se utiliza DAML-S porque en el momento en que fue desarrollado el sistema no había razonadores que funcionasen con OWL-S. La arquitectura propuesta en este trabajo se basa en el uso de sistemas multiagente, en el que un agente denominado *matchmaker* o emparejador es el encargado de encontrar (de manera similar a un sistema de páginas amarillas) el servicio, simple o complejo que ha sido expuesto por un proveedor y que satisface las necesidades del cliente a partir de los parámetros definidos por éste.

En el punto 2 se mostrará el dominio del problema a resolver, en el punto 3 se introducirán los conceptos básicos para entender el funcionamiento de la solución propuesta para resolverlo, en el punto 4 se presentarán los principales actores de nuestro sistema de emparejamiento, en el punto 5 se dará una visión general del funcionamiento del sistema, en el punto 6 se descompondrá el sistema en capas par facilitar su comprensión, en el punto 7 se detallará el despliegue del sistema, en el punto 8 se comentarán las pruebas realizadas, en el punto 9 se mostrará un caso de uso del sistema y, por último, en el punto 10 se darán algunas conclusiones a las que se ha llegado y se hablará sobre una posible continuidad del trabajo.

2. Dominio del problema a resolver

La difusión de información de tráfico está marcada por la ausencia de vocabularios comunes que puedan ser elegidos como núcleo o punto de partida para los desarrolladores. Esto hace que el proceso de tratar o compartir información entre distintos desarrolladores, incluso pertenecientes a la misma empresa, sea complejo, y aún más si cabe entre diferentes administraciones y países. Otro problema importante es que la información está distribuida entre fuentes muy diversas, como pueden ser bases de datos y portales web. Además, este volumen de información distribuida es enorme, por lo que será necesario tratarla de forma eficiente. El objetivo de nuestro prototipo de sistema fue gestionar esta información de tráfico y ofrecerla al usuario de la forma más eficiente, y para alcanzarlo, en primer lugar, se estableció un vocabulario común sobre los conceptos relacionados con la información de tráfico, conseguido mediante ontologías. En segundo lugar se buscó una forma de homogeneizar el acceso a toda esta información heterogénea, disponible en fuentes tan diversas, y se optó por utilizar SW que sirviesen de intermediarios. En tercer lugar se debía encontrar un modo de decidir qué SW satisfacía en mejor medida las necesidades de un usuario. Ni WSDL [37] ni UDDI [32] son de ayuda para la localización de servicios en función de lo que estos ofrecen, por lo que se optó por describir semánticamente estos servicios utilizando DAML-S / OWL-S y se desarrolló un emparejador de estos servicios web semánticos capaz de decidir cuál de ellos se aproximaba más al buscado por el cliente. Este emparejamiento se basó en la proximidad semántica (significado) de los conceptos buscados por el usuario y los ofrecidos por estos servicios. Además, todo esto se integró en una plataforma multiagente basada en estándares FIPA aprovechando el *middleware* ofrecido por este tipo de plataformas y su capacidad de interacción con otras que cumplan este mismo estándar.

3. Estado del arte

3.1. Ontologías para descripción semántica de servicios

OWL-S (o DAML-S) define 3 ontologías para la descripción, invocación y ejecución de servicios. La ontología *Service Profile* es usada para describir y anunciar el SW requerido por el usuario y ofertado por el proveedor respectivamente. La ontología *Service Model* es usada para definir el modelo de proceso y ejecución del servicio y nos sirve por tanto para saber cómo funciona, y *Service Grounding* es usada para describir cómo acceder al servicio, indicándonos cómo puede ser usado. Tanto DAML-S como OWL-S nos dan lo necesario para conseguir la representación semántica de servicios basándose en DAML+OIL y OWL res-

pectivamente, los cuales soportan razonamiento sobre inclusión en taxonomías de conceptos y permiten la relación entre conceptos pudiendo expresar por ejemplo que X es parte de Y, o de forma más general, que existe una relación entre X e Y. Aunque tienen limitaciones, son lo suficientemente expresivos como para permitir la descripción de un gran número de servicios y realizar emparejamientos entre ellos.

3.2. Concepto de Servicio Web Semántico

La Web no solamente proporciona acceso a contenidos sino que también ofrece interacción y servicios (comprar un libro, reservar una plaza en un vuelo, hacer una transferencia bancaria, simular una hipoteca). Los servicios web semánticos (SWS) son una línea importante de la Web Semántica, que propone describir no sólo información sino definir ontologías de funcionalidad y procedimientos para describir SW: sus entradas y salidas, las condiciones necesarias para que se puedan ejecutar, los efectos que producen, o los pasos a seguir cuando se trata de un servicio compuesto. Estas descripciones procesables por máquinas permitirían automatizar el descubrimiento, la composición, y la ejecución de servicios, así como la comunicación entre unos y otros.

Los SWS proponen extender las tecnologías de SW tradicionales, en vías de consolidación, con ontologías y semántica que permitan la selección, integración e invocación dinámica de servicios, dotándoles así mismo de la capacidad de reconfigurarse dinámicamente para adaptarse a los cambios. Los SWS son un nuevo paradigma de la computación, definido generalmente como el surgimiento de las descripciones de SW con anotaciones de Web Semántica, para facilitar la automatización del descubrimiento, composición, invocación, y monitorización de los servicios en un ambiente no regulado y caótico como es la Web [22]. A partir de los esfuerzos de estandarización del W3C y a través de su grupo de interés Semantic Web Services Interest Group [25] se está fomentando la integración de la tecnología de la Web Semántica y el trabajo de otros grupos sobre SW realizado en W3C. Una de las iniciativas más importantes surgidas a partir de este trabajo ha tomado como base la labor de investigación realizada por la coalición DAML-S (DAML for Services), surgida en el año 2001, la cual propuso una ontología para la descripción semántica de servicios basada en DAML+OIL (DAML-S), cuya última versión (0.9) fue lanzada en mayo de 2003. A partir de noviembre de 2003 se desarrolló OWL-S, basado en el lenguaje de marcado ontológico OWL . Actualmente esta iniciativa es el resultado de la unión de otras dos: Semantic Web Services Initiative (SWSI) [24] y Semantic Web Services Language (SWSL) Committee [26]. OWL-S fue enviado a W3C para su reconocimiento en julio de 2004.

3.3. Emparejamiento

El *matchmaking* o emparejamiento es un proceso por el cual se emparejan características requeridas por un participante y las ofrecidas por otro con el objetivo de que la entidad encargada de realizar el emparejamiento pueda decidir cuál de ellos se aproxima más a lo requerido. Para ello deberá conocer los servicios ofrecidos por proveedores y sus características, así como los requerimientos expresados por el cliente. Por tanto el proceso general que se sigue en el emparejamiento de servicios ofertados y requeridos se basa en que cuando un agente envía una petición al *matchmaker* (actor encargado de realizar el emparejamiento) más tarde, éste le devolverá como resultado información respecto al servicio que empareje con la descripción dada en el requerimiento. El modelo y algoritmo de emparejamiento semántico planteado en [28] es el que ha sido utilizado por la mayoría de investigadores como fundamento para sus sistemas.

3.4. Grados de similitud o match

El grado de similitud depende de la relación (tomada de las ontologías) entre los conceptos que se están comparando, y generalmente se reduce a la mínima distancia entre ellos en el árbol taxonómico. La función principal del algoritmo de emparejamiento que usamos consiste en determinar el grado de similitud o distancia semántica entre descripciones de servicios, tomando como base el conjunto de relaciones entre ellos.

La denominación de los grados varía según la literatura [2],[18],[28],[13]:

- ***Exact***: cuando los conceptos tanto en la petición como en el anuncio son equivalentes.
- ***Subclass of***: determinado por la relación *ser subclase de*. Cuando los conceptos en la petición son subclase (relación directa) de los del anuncio. (En [28] incluyen como *exact matching* a este tipo de emparejamiento).
- ***Subsumption***: la cual puede ser de dos tipos:

- ***Plug-in o Contained***: cuando los conceptos en el anuncio A incluyen los de la petición P. Formalmente, (P $\subseteq$ A). En este caso, la petición puede ser satisfecha debido a que los conceptos del anuncio, son más generales que los de la petición, y por tanto existe la posibilidad de que el cliente pueda cumplir sus objetivos. Sin embargo, no se considera que exista una relación de subclase directa (grado anterior)
- ***Subsume o Container***: cuando los conceptos en la petición incluyen los del anuncio; formalmente, (A $\subseteq$ P). Este tipo de emparejamiento no satisface completamente la petición pero puede ser considerado como una solución parcial válida ya que puede permitir al cliente que realizó la petición ir alcanzando parcialmente sus objetivos o metas.

- ***Fail ,nul o Disjoint***: cuando no hay relación de inclusión entre los conceptos; formalmente, (A $\cap$ P)$\subseteq \perp$.

Los autores de [14] introdujeron nuevos tipos de emparejamiento que más tarde fueron adoptados por Li y Horrocks [18] como extensión a los anteriormente expuestos:

- *Intersection u Overlap.* Si la intersección de un anuncio A y una petición P se satisface, es decir son compatibles; formalmente, $\neg(A \bigcap P \subseteq \perp)$.

3.5. Diferentes sistemas e investigaciones relacionadas

La base de toda la investigación relacionada con el emparejamiento semántico de servicios, recae principalmente en los estudios de Ingeniería de Software llevados a cabo por Zaremski y Wing en [38] y [39].

Sycara et al. desarrollaron LARKS [27], [29], en el cual los servicios son vistos como *frames* y sus *slots input*, *output*, *inConstraints* y *outConstraints* pueden ser usados para describir los atributos esenciales de un servicio. El proceso de emparejamiento en LARKS realiza tanto emparejamiento sintáctico como semántico, y se compone de un conjunto de filtros (independientes entre sí) que progresivamente restringen el número de anuncios candidatos a ser emparejados.

En [19] se describe una aproximación basada en ontologías que emplea las características de una taxonomía de proceso. El algoritmo que empareja utiliza la relación semántica codificada en la ontología de proceso al emparejar el modelo del proceso del servicio con requerimientos.

La aproximación usada por [14] para emparejamiento de servicios es un algoritmo basado en el árbol del subsumción dado por un razonador de lógicas descriptivas (DL). Las descripciones del servicio son especificadas en DAML+OIL y puesto que DAML+OIL se basa en DL, el razonador de DL es el corazón del algoritmo de emparejamiento. Los diferentes tipos de emparejamiento para un servicio S se definen como:

- conceptos equivalentes a S,
- subconceptos de S,
- superconceptos de S que son incluidos por el concepto en la ontología de descripción del servicio,
- los subconceptos de cualquier superconcepto directo de S cuya intersección con S sea satisfactoria.

El equipo *Intelligent Software Agents Group* de la Carnegie Mellon University (CMU) diseñó una arquitectura basada en ATLAS (Agent Transaction Language for Advertising Services), lenguaje basado en DAML, para emparejamiento de servicios [30]. Éste utiliza dos filtros: emparejamiento de atributos funcionales para determinar la aplicabilidad de los anuncios y emparejamiento de funcionalidades de servicio, para determinar si el servicio anunciado empareja el servicio requerido.

De nuevo en CMU [6], el *DAML-S Matchmaker* emplea técnicas de recuperación de datos, de IA, y de Ingeniería de Software para procesar la semejanza sintáctica y semántica entre descripciones de capacidad de servicios. Estas investigaciones son las que más se han tomado como referencia para otros trabajos como [28], ya comentado anteriormente, cuyo modelo y algoritmo de emparejamiento semántico ha sido utilizado por la mayoría de investigadores como base para sus sistemas, y que plantea un sistema de emparejamiento basado en la semántica descrita en el *profile* de los servicios y en la utilización de los registros UDDI para mantener las descripciones de estos servicios. Estas investigaciones hacen uso de la ontología perfil del servicio y de DAML-S como lenguaje de especificación para las descripciones del servicio. Se asume que los servicios de la red anuncian sus interfaces (inputs/outputs)(IO's) usando la misma

ontología. En este proyecto [28] se aborda la importancia para la clasificación del emparejamiento de las salidas. Un emparejamiento entre un anuncio y una petición de servicio consiste en emparejar todas las salidas de la petición con las del anuncio; y todas las entradas del anuncio con las de la petición. El motivo por el que la prioridad de los parámetros de salida (*outputs*) es mayor, es debido a que es más importante el emparejamiento de lo que el cliente espera obtener como resultado, es decir, la salida del servicio en cuestión. Una de las ideas más importantes que aportan Paolucci et al. en su algoritmo para que tenga un mejor comportamiento, es la de que tanto clientes como proveedores empleen las mismas ontologías de conceptos para dar valor semántico a cada uno de los IO's de sus *profiles*. De esta manera, se facilita la tarea al motor de inferencia o razonador, porque sólo necesita que se carguen las ontologías de conceptos que utilizaron para construir los *profiles* para poder medir el grado de similitud que tengan los IO's que definiesen.
Han sido varios los investigadores que han basado su sistema de emparejamiento en este algoritmo, añadiéndole algunas funcionalidades: Charlie Abela [2] desarrolló un sistema de emparejamiento basado en el algoritmo que desarrollaron Paolucci et al. Este autor hace uso de un algoritmo que usa dos operaciones de filtrado para realizar el emparejamiento. El primer filtro limita el espacio de búsqueda a las restricciones definidas por el usuario en la petición (búsqueda por *ServiceProvider, ServiceCategory* o *ServiceName*). El segundo filtro es un emparejamiento de las salidas que el usuario define en la petición, permitiéndose el uso de un solo filtro o ambos. En el resultado de un emparejamiento pueden darse varios servicios y por tanto se deberá crear un rango para especificar cual de ellos es el que más se ajusta a las necesidades del cliente. A veces será necesaria la intervención de éste para poder elegir aquél que considere más apropiado.

Otra aproximación fue desarrollada por *Information Management Group* en la *University of Manchester* [18]. El prototipo descrito de su algoritmo de *matchmaking* también se basa en las descripciones de DAML-S. Utiliza el razonador *Racer Descriptions Logics* para realizar emparejamientos semánticos entre los servicios, y utiliza JADE como plataforma de agentes. Como en el diseño de DALM-S Matchmaker, las peticiones del servicio en DAML-S se emparejan con los anuncios del servicio. Sin embargo, este algoritmo de emparejamiento no divide el procedimiento en varias partes; en su lugar intenta encontrar un emparejamiento semántico directamente de los perfiles de servicios especificados. En esta aproximación los perfiles son tratados como entidades enteras, y sus componentes no son tratados por separado. Una vez más al resultado del emparejamiento se le asocia un cierto grado de similitud.

Los brasileños Ricardo Ferraz, Sofiane Labidi y Bernardo Wanghon [23] extendieron el algoritmo de emparejamiento de Paolucci et al. En este caso el cliente define las precondiciones que piensa que debe tener el servicio para que emparejen en el mayor grado posible con las que los proveedores introducen en los *profiles*, realizando el mismo algoritmo de emparejamiento que el de los IO's. Incluyeron las precondiciones porque consideran que son unos parámetros importantes a tener en cuenta en las negociaciones. Por ejemplo puede darse el caso de SW que incluyan precondiciones que negocian el tipo de pago, o el tipo de tarjetas de crédito que admiten. Sin embargo este tipo de emparejamiento no se ha tenido en cuenta en nuestra propuesta, por la peculiaridad de los servicios de información de tráfico, marco en el cual se decidió probar nuestra arquitectura.

3.6. Ontologías en tráfico

Actualmente, existen vocabularios o lenguajes que describen conceptos y estructuras de datos relacionados con tráfico, pero son sólo descripciones sintácticas, carentes de semántica. Por lo tanto, aunque existen numerosos trabajos y desarrollos que hacen uso de lenguajes de marcado XML (difusión de información, intercambio de información, modelado de tráfico), tras un proceso previo de revisión bibliográfica, no se encontraron vocabularios que incluyeran elementos semánticos para información de tráfico vial. Surgió por tanto la necesidad de crear especificaciones semánticas (ontologías) y hacer uso de ellas mediante la construcción de diferentes prototipos.

¿Por qué se necesitan ontologías de tráfico? El tratamiento informático actual de la información de tráfico es muy limitado y susceptible de ser mejorado desde distintos puntos de vista. El esquema de representación debe cumplir los siguientes aspectos:

- ser interpretable por el ordenador y fácilmente intercambiable entre aplicaciones.
- adherirse en sus aspectos sintácticos a los estándares de representación de información existentes.
- permitir la obtención de conocimiento no explícito a priori.

Si un usuario u operador del sistema necesita conocer

información específica sobre los accidentes acontecidos en una determinada carretera o localización, posiblemente quiera saber detalles sobre los vehículos implicados, tipos de vías etc., y por tanto este tipo de consulta involucre no sólo una base o fuente de conocimiento sino varias, de ahí la importancia del uso de ontologías, algunas de cuyas características son la distribución y la posibilidad de inferir conocimiento no explícito a priori, es decir, podemos ir todavía más lejos, y preguntar por ciertos elementos que en principio no han sido establecidos en este dominio, y que formarán parte de una especificación en otro lugar que no tiene porque ser físicamente el mismo. Otra característica de las ontologías que se puede explotar es la capacidad de éstas para definir sin ambigüedad los conceptos. Gracias a ésta, un concepto podrá ser definido de forma unívoca, de tal forma que aunque por ejemplo, distintas organizaciones determinen diferentes significados para un mismo término, el receptor de la información siempre obtendrá el significado correcto correspondiente.
La construcción del modelo consiste en tres fases:

- ***adquisición del conocimiento:*** identificación de las clases o términos básicos y sus propiedades,
- ***definición:*** identificación de las relaciones entre las clases y,
- ***especificación de restricciones:*** identificación de las restricciones que limitarán la manera en la que las descripciones pueden ser formadas.

Este proceso será iterativo, con sucesivos refinamientos.

Para abordar el problema del desarrollo de una ontología de tráfico rodado se ha planteado la definición de subdominios que se relacionarán entre sí. En este sentido los subdominios de relevancia que han sido definidos son los siguientes:

- Clasificación de Vías (Autopista Libre, Autopista de Peaje, Conexión, Acceso, Ronda etc.)
- Clasificación de Vehículos (Ciclomotor, Turismo, Camión etc.)
- Localización (Área, Punto, Tramo, Métrica, Sentido etc.)
- Geografía (Poblaciones, Países etc.)
- Sucesos (Accidentes, Incidentes, Medidas etc.)
- Personas (Peatón, Conductor, Titular de Vehículo etc.)
- Rutas (Ruta Urbana, Ruta Interurbana etc.)

A su vez hay que destacar la reutilización de vocabularios ya definidos como FOAF [33]y GEO [34] y la ampliación de otros como Time [3].

Una vez construidas las diferentes subontologías de tráfico se fusionan en una ontología de conceptos de tráfico llamada (Traffic Ontology) que es la que reúne todos los conceptos principales y sus relaciones.

Los idiomas utilizados para la construcción de las ontologías fueron el castellano y el inglés. Para lograr el objetivo de ontología multilenguaje se pueden usar varias técnicas como describir el mismo conocimiento en varias ontologías, cada una en una lengua distinta, manteniendo toda la estructura y semántica descrita en la ontología original, y haciendo uso del operador de equivalencia entre términos y relaciones. A través de un requerimiento de idioma en inglés y a partir de una instancia en castellano, seremos capaces de obtener una definición en el idioma requerido. Si un cliente interacciona con el sistema, determinando que el idioma elegido es el inglés, la petición especificará este requerimiento, de tal forma que la información resultado será traducida (en realidad recuperada en castellano, pero documentada en inglés). Otro problema distinto será el hecho de recuperar información desde servicios o web Sites de otros paises o administraciones. La solución a este problema pasa por describir estas Web sites como servicios web específicos, donde el uso de ontologías en dicho idioma ayudarán tanto en la búsqueda como en el resto de tareas. Otra posibilidad es hacer uso de atributos XML:lang en los comentarios, que posibiliten en una misma ontología la descripción de sus términos en varios idiomas. Sin embargo, esta última técnica no funciona en algunos razonadores como RACER debido a la limitación de su lenguaje de requerimientos para acceder a determinados campos como rdf:comment de la descripción del concepto.

4. Emparejador de Servicios Web de Información de Tráfico (ESWIT)

ESWIT es el prototipo desarrollado para introducir la semántica en el emparejamiento de SW. Se optó por una aproximación basada en agentes distribuidos, y se seleccionó FIPA como el estándar a seguir. Como entorno de desarrollo se utilizó JADE [31] (Java Agent

Development Framework), por cumplir lo propuesto por FIPA en su modelo de referencia de administración de agentes [12], donde se establecen los elementos básicos de los que debe constar un sistema multiagente. Debido a que Internet es un entorno abierto donde los orígenes de la información, los enlaces de comunicación y los mismos agentes pueden aparecer y desaparecer en cualquier momento, existe la necesidad de ayudar a los agentes solicitadores (agentes que buscan un servicio determinado) a encontrar a los agentes proveedores. A los agentes que ayudan a localizar a otros agentes se les denomina *middle agents* (intermediarios). Los autores de [17] identifican diferentes tipos de *middle agents* en Internet, como son los *matchmakers* (servicios de páginas amarillas), o *brokers*.
En este caso se optó por una aproximación basada en matchmakers.

Siguiendo el modelo de intermediación *matchmaker* se identificaron durante la fase de análisis los siguientes actores, que serían los componentes básicos de nuestro sistema:

- ***Cliente o Usuario:*** es la persona que usa el sistema.
- ***Agente cliente:*** agente con el rol de cliente. Representa al usuario dentro de la plataforma, y proporciona al agente emparejador la descripción del servicio que busca.
- ***Proveedor:*** empresa, organización o persona que anuncie un servicio en la plataforma mediante un agente proveedor.
- ***Agente proveedor***: agente con el rol de proveedor de servicio web. Representa al proveedor dentro de la plataforma.
- ***Agente emparejador:*** agente que tiene el rol de *matchmaker*, es el encargado de emparejar en función de su proximidad semántica (grado de similitud), descripciones de servicios recibidas de clientes con descripciones de servicios anunciados por los proveedores.
- ***Facilitador de directorio (DF):*** agente incluido en las plataformas que cumplen el modelo FIPA. Aunque en esta especificación es opcional, ha sido utilizado en nuestro sistema para facilitar las tareas de administración de nuestros agentes, como un componente de tipo páginas amarillas.
- ***Servicio web:*** servicio web externo a la plataforma, el cual es representado dentro de ella por un agente proveedor (puede haber más de uno).
- ***Emparejador de servicios:*** en función de la descripción de sus capacidades, basado en semántica y utilizado por el agente emparejador.

5. Visión general del sistema e interacciones entre los distintos agentes que lo componen

Las fases que resumen el funcionamiento del sistema son las siguientes:

1. Cuando se anuncia un nuevo servicio, en primer lugar su agente proveedor, representante de este servicio dentro de la plataforma, se registra en el DF. A continuación el DF avisa al agente emparejador y éste solicita al nuevo proveedor la URI (Uniform Resource Identifier) [36] de la descripción semántica del servicio que ofrece.
2. Al lanzar un usuario un agente cliente le pasa como parámetro la URI de la descripción semántica del servicio que busca en la plataforma. Este agente cliente se pone en contacto con el agente emparejador y le proporciona la URI de la descripción de este servicio. El agente emparejador interroga al razonador y contrasta la descripción del servicio buscado con todas las de los servicios disponibles, decidiendo qué descripción es la que más se ajusta al servicio buscado, e informando al agente cliente de la URI de la descripción de este servicio.
3. Una vez el agente cliente conoce la descripción que más se parece semánticamente al servicio que busca, genera, a partir de ella, la interfaz gráfica para recoger los datos de entrada necesarios para poder hacer la correcta invocación del servicio que describe.
4. Una vez completado el formulario con los datos de entrada requeridos, el agente cliente lo invoca directamente con esos datos y recoge el resultado que devuelve.
5. Llegados a este punto el agente cliente vuelve a ponerse en contacto con el usuario y le muestra el resultado de la invocación del servicio.

Para hacer posible estas interacciones es necesario el establecimiento de comunicaciones entre los distintos agentes que forman parte del sistema. Las principales comunicaciones establecidas entre estos agentes

se detallan en los siguientes diagramas AUML [10] de protocolo, todos ellos con actos comunicativos estandarizados por FIPA. Estos ejemplos de comunicación entre agentes son los que más se dan durante el funcionamiento de nuestro sistema.

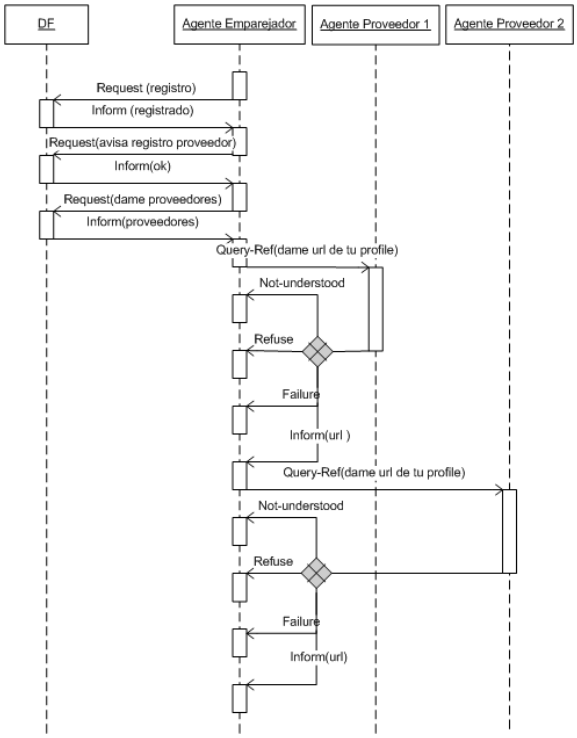

Figura 1. Emparejador lanzado cuando ya hay proveedores activos

La figura 1 representa al caso en el que el agente emparejador es lanzado cuando ya hay agentes proveedores activos en la plataforma.

1. En primer lugar el agente emparejador se registra en el facilitador de directorio.

2. Después de recibir la notificación de que el registro se ha realizado sin problemas solicita al facilitador de directorio ser avisado cada vez que un agente del tipo proveedor se da de alta.

3. Una vez solicitado el aviso pregunta al facilitador de directorio por todos los agentes de tipo proveedor que ya están registrados.

4. Cuando recibe la lista de agentes de tipo proveedor ya registrados, el agente emparejador solicita a cada uno de ellos la URI del *profile* que describe el servicio al que representa.

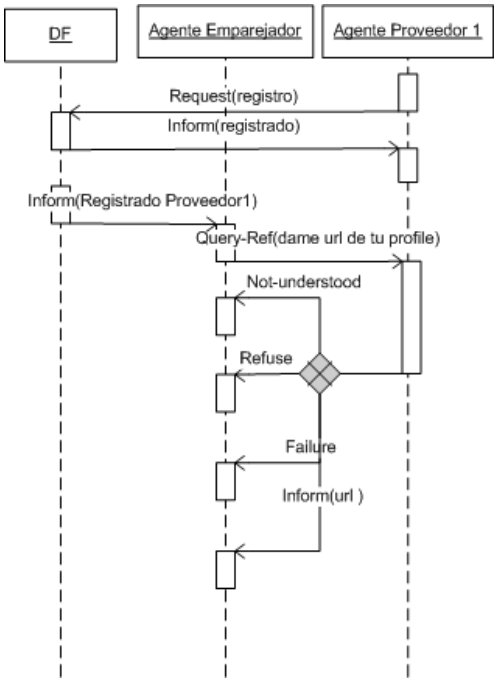

Figura 2. Proveedores lanzados después de emparejador

En la figura 2 se muestra el caso en el que los agentes proveedores se lanzan después de ser iniciado el agente emparejador.

1. Cuando un agente de tipo proveedor aparece en la plataforma lo primero que hace es registrarse en el facilitador de directorio.

2. Como vimos en el diagrama anterior el agente emparejador solicita ser avisado cuando agentes de tipo proveedor se registren, por lo que el facilitador de directorio notificará al agente emparejador este nuevo registro.

3. El agente emparejador pedirá entonces al nuevo agente proveedor que le dé la URI del *profile* que describe el servicio al que representa.

4. El agente proveedor proporciona al agente emparejador la URI solicitada en el paso anterior.

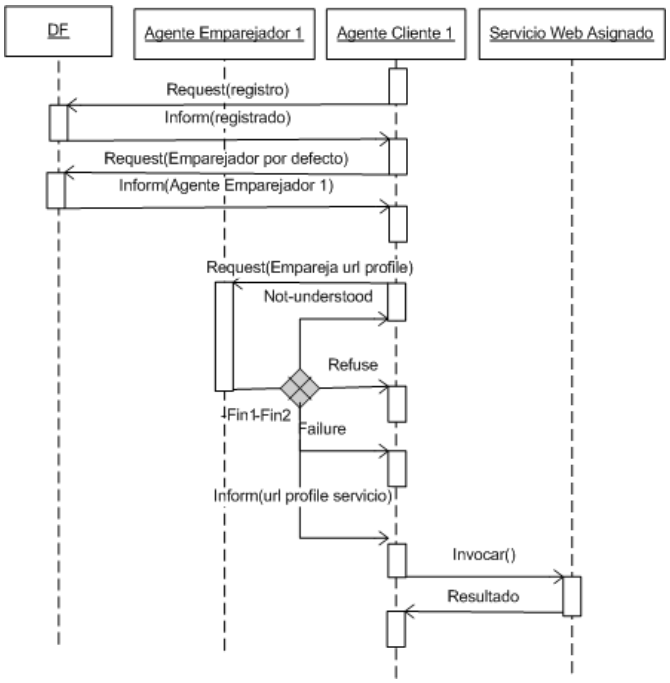

Figura 3. Cliente lanzado cuando hay agente emparejador y proveedores en el sistema

En la figura 3 se puede observar el caso en el que el cliente es lanzado cuando ya hay un agente emparejador y agentes proveedores en el sistema.

1. Cuando el cliente es lanzado en la plataforma se registra en el facilitador de directorio y le pregunta cuál es el agente emparejador por defecto.
2. Una vez el cliente conoce qué agente emparejador debe utilizar le pide que empareje la URI del profile que describe el servicio que busca con las URIs de todos los servicios disponibles.
3. Cuando el agente cliente recibe la URI del servicio que más se parece semánticamente al que busca lo invoca y obtiene el resultado.

6. Descomposición en capas del sistema

El sistema desarrollado puede descomponerse en la estructura de capas mostrada en la figura 4. Se puede observar cómo la *Plataforma Multiagente* está presente en todas las capas aunque no todos los componentes del sistema forman parte de ella. El facilitador de directorio no ha sido incluido en esta estructura de capas por ser propio de JADE. El sistema está formado por agentes encargados del emparejamiento de servicios, agentes que representan a clientes dentro de la plataforma y agentes que representan a proveedores de servicios.

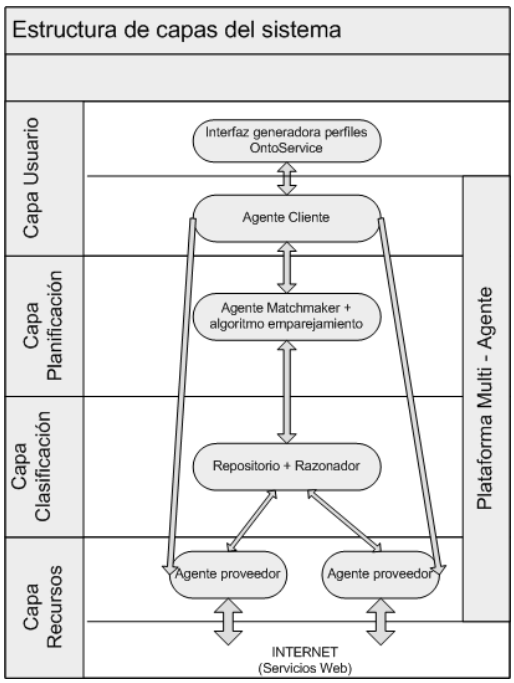

Figura 4. Estructura de capas del sistema.

Trataremos ahora de exponer con mayor detalle cada una de estas capas y los diferentes elementos que las componen.

6.1. Capa Usuario

6.1.1. OntoService

La primera necesidad que surgió fue la de encontrar una herramienta que combinase la visualización de ontologías con la creación de perfiles de búsqueda, por lo que se decidió implementar una herramienta, a la que se le llamó OntoService [15], basada en la integración de capacidades para definición de perfiles de servicios Web semánticos con visualización y verificación de consistencia de los conceptos sobre los cuales interaccionaría un determinado servicio. En la figura 5 se puede observar la interfaz gráfica principal de OntoService.

En relación con herramientas para descripción de SWS las principales propuestas que fueron encon-

tradas son A-Match [4] y OWL-S Editor [5] de Carnegie-Mellon University.

La herramienta implementada Ontoservice tiene dos funcionalidades independientes, a la vez que complementarias. Por un lado es un visualizador/verificador de ontologías, las cuales pueden estar descritas indiferentemente en el lenguaje DAML+OIL o bien en OWL, y por otro, es una herramienta de búsqueda de servicios a partir de un perfil de búsqueda generado automáticamente.

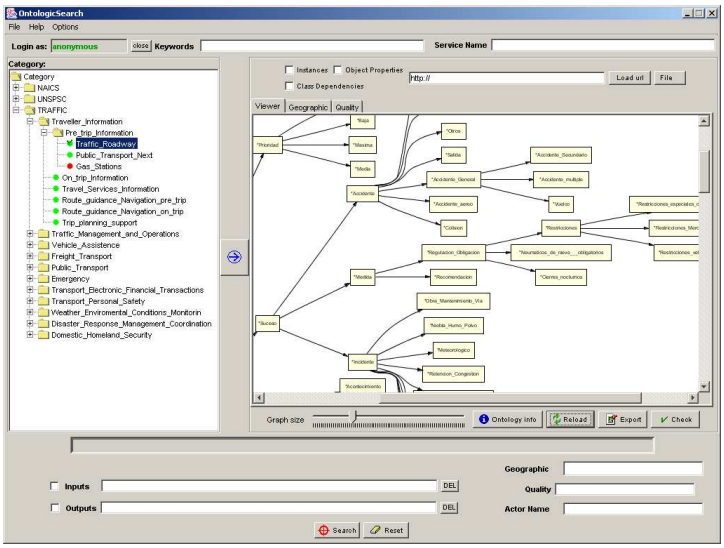

Figura 5. Interfaz gráfica de OntoService.

Se genera el fichero *profile* a partir de una plantilla, ya que cualquier requerimiento de búsqueda se basará exclusivamente en la información contenida en él. En esta fase de funcionamiento del sistema sólo es necesario conocer la descripción del servicio (*profile*), aún no hace falta saber cómo usarlo o acceder a él. El usuario puede almacenar esta plantilla en el disco duro o bien, dado que la herramienta puede integrarse en una plataforma de agentes, realizar el envío del perfil generado a un agente encargado de llevar a cabo la búsqueda del SW e incluso lanzar la ejecución de este agente.

Utilizamos por tanto Ontoservice para que el cliente defina el servicio que busca, genere el perfil que lo describe y lance un agente cliente dentro de la plataforma con este nuevo perfil generado.

Debido a los recursos necesarios para el correcto funcionamiento de OntoService se optó por no integrarlo dentro de la plataforma y separarlo del agente cliente.

6.1.2. Agente Cliente

Se decidió crear un agente cliente que recibiese una URI de descripción del servicio generada por ejemplo por OntoService en función de los datos introducidos por el usuario, pudiendo ser OntoService el que lanzase a este agente dentro del sistema. El agente cliente es por tanto el representante del cliente dentro de la plataforma. Indica al agente emparejador la URI de la descripción del servicio que busca. Una vez obtenida la URI de la descripción semántica del SW que satisfaga sus necesidades, extrae de su archivo grounding asociado, la información necesaria para realizar su invocación. Se consiguió implementar un cliente genérico para invocar cualquier servicio, sin necesidad de recompilar la interfaz gráfica cada vez. Además, se invoca al servicio dinámicamente, sin necesidad de tener un *stub* para cada posible servicio. Esto es necesario porque los servicios aparecen y desaparecen dinámicamente en la plataforma y sería inviable tener preparado un cliente para cada uno de los posibles servicios que en algún momento determinado puedan estar disponibles. Después del emparejamiento se accede al archivo *grounding* asociado al *profile* con el que el cliente haya sido emparejado, para extraer de él la URI de la descripción *wsdl* [37] asociada al servicio. Una vez obtenido el archivo *wsdl* se procede a la invocación, y el resultado es mostrado al cliente en una interfaz diseñada para ello.

6.2. Capa Planificación

6.2.1. Agente emparejador (*matchmaker*)

Este agente emparejador puede servir como alternativa o complemento a UDDI y tiene como principal característica el hecho de estar basado en semántica. Para facilitar la construcción de este agente se desarrolló un emparejador de descripciones semánticas usado desde nuestro agente emparejador. Este emparejador se encarga de recibir URIs de descripciones de peticiones de servicios y contrastar su contenido con las descripciones de los servicios anunciados para, de esta manera, identificar cuál de ellos se acerca más a las necesidades del cliente. A cada pareja obtenida se le asigna un peso que indica la proximidad semántica entre los dos conceptos que se relacionan. Este peso se obtiene interrogando a un razonador sobre la jerarquía definida por una ontología de conceptos que define el vocabulario común a utilizar. Las peticiones de clientes y los nuevos anuncios o bajas de servicios ofrecidos son gestionados dinámicamente. Tanto las ontologías como las descripciones de servicios utilizadas están orientadas a servicios de tráfico, pero el

emparejador puede trabajar con cualquier ontología sin necesidad de hacer modificaciones.

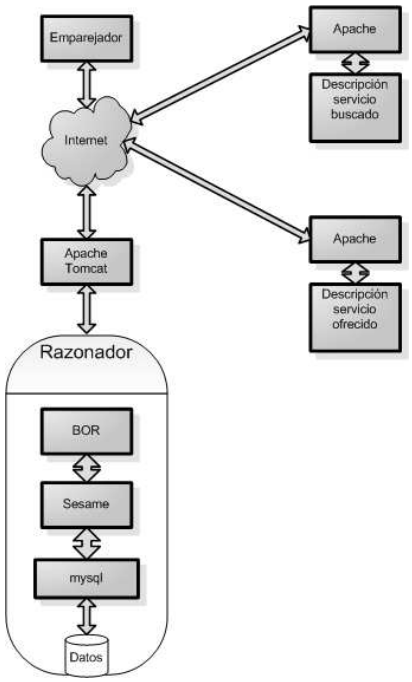

Figura 6. Diagrama de despliegue del emparejador semántico.

6.2.2. Algoritmo utilizado por el emparejador

Nuestro emparejador de servicios (*matchmaker*) está basado en el algoritmo descrito en el artículo [28], que es la base de la mayoría de los últimos trabajos sobre este tema. Su despliegue puede observarse en la figura 6. Este algoritmo, para realizar el emparejamiento, utiliza principalmente la información contenida en los profiles DAML-S/OWL-S de las definiciones de peticiones de clientes y de anuncios de proveedores. Aprovecha al máximo las capacidades proporcionadas por la ontología de descripción de servicios y constituye una mejora en relación con propuestas existentes. La principal aportación han sido los diferentes grados de emparejamiento que permitieran emparejamientos flexibles, la utilización de parámetros no funcionales, así como los diversos filtros usados, algunos de los cuales no habían sido considerados en trabajos de otros autores. Las diferencias más importantes entre nuestro algoritmo y los de trabajos anteriores se pueden observar en los grados de emparejamiento ya que en estos eran demasiado generales, por lo que se propuso una versión compuesta por siete grados de emparejamiento que detallamos a continuación, en orden descendente de importancia:

- ***Exacto:*** los conceptos definidos por el cliente y por el proveedor son los mismos.
- ***CsubclaseP:*** dentro del árbol de la taxonomía de conceptos la distancia entre el concepto demandado por el cliente y el ofrecido por el proveedor es igual a 1, siendo por tanto, el descrito por el cliente, subclase directa del que proporcionó el proveedor. En este caso el proveedor ofrece un concepto más general que el que pide el cliente. El concepto del cliente es más restrictivo pero está incluido en el que proporciona el proveedor.
- ***PsubclaseC:*** el concepto descrito por el proveedor es subclase directa del que pide el cliente, es decir, el proveedor ofrece un concepto más restrictivo que el demandado por el cliente.
- ***PsubsumeC:*** el concepto descrito por el cliente se encuentra dentro del subárbol de conceptos descendiente del definido por el proveedor. Sería equivalente al *Plugin* definido en [28] aunque se diferencia de él en que no permite que el cliente especifique el nivel de profundidad máximo hasta el que llegará la búsqueda. Se ha optado por no permitir al cliente especificar esto porque consideramos que conceptos a distancias mayores o iguales a 3 niveles no tienen prácticamente relación semántica, debido al modo en que se construyen las jerarquías de conceptos. Por ello se ha decidido fijar la profundidad máxima en dos niveles, por lo que sólo se comprueba si el concepto definido por el cliente es como máximo ”nieto”del concepto del proveedor. De no hacerlo así, aumentaría el riesgo de falsos positivos y por tanto, podríamos obtener servicios como válidos, cuando la relación semántica entre el concepto que se provee y el que pide el cliente es tan lejano semánticamente que no responde a las expectativas de éste.
- ***CsubsumeP:*** el concepto descrito por el proveedor se encuentra dentro del subárbol de conceptos descendiente del definido por el cliente, es decir, es el caso inverso al anterior, y es equivalente al grado *Subsume* definido en [28]. Como en el caso anterior, aquí también se limita a dos niveles de profundidad la comprobación de si el concepto demandado por el cliente incluye (subsume) al ofrecido por el proveedor.
- ***ChermanoP:*** el concepto proporcionado por el cliente y el del proveedor tienen restricciones en común en alguna propiedad que ambos poseen, y además, tanto el concepto ofrecido por

proveedor como el demandado por el cliente son hijos del mismo padre, es decir, son conceptos ”hermanos”.

- ***Fail: cuando no se cumple*** ningún caso de los anteriores, es decir el concepto del proveedor y el de cliente no tienen relación alguna.

El motivo de este orden para los casos de subclase, es decir, porque *CsubclaseP* es mejor grado que *PsubclaseC*, es debido a que consideramos más importante que el proveedor ofrezca un producto menos restrictivo que el cliente, ya que en este caso puede suceder que también ofrezca lo que solicita el cliente. En caso contrario, si un proveedor ofrece algo más restrictivo que lo que solicitó el cliente, puede que a éste no le interese. El caso del orden entre *PsubsumeC* y *CsubsumeP* es análogo solo que considerando una mayor distancia en el árbol de la taxonomía de conceptos especificados en la ontología.

También realizamos un primer filtrado en función del parámetro que describe la calidad del servicio buscado por el cliente para reducir el espacio de búsqueda.

Además el algoritmo de Paolucci et al. no aprovecha otras propiedades de la descripción del perfil (*profile*), como son los parámetros no funcionales, que también aportan información semántica del SW, por lo que uno de los objetivos fue cubrir este vacío. Nuestro algoritmo emparejador intenta aprovechar al máximo las posibilidades de *profile*, con el fin de hacer uso de todas las capacidades de los servicios, descritas semánticamente. A continuación se describe qué filtros se han desarrollado para los diferentes parámetros no funcionales:

- ***filtro para región:*** con él se comprobará si el radio geográfico dado por el cliente es el mismo que el proporcionado por el proveedor.
- ***filtro para calidad de servicio:*** consiste en comprobar en el repositorio si la calidad que aportan los SW es la misma que la solicitada por el cliente.
- ***filtro para nombre de servicio:*** en esta consulta se comprueba si el cliente encuentra algún servicio en particular con el nombre que proporcionó. Este emparejamiento es sintáctico, ya que, tal y como está definida la propiedad *serviceName* en la subontología *Profile*, tiene como rango de valores el *datatype String* del *XML Schema*, con el que solo se pueden hacer comparaciones sintácticas.
- ***filtro para nombre del proveedor:*** permite comprobar si el servicio web lo provee la empresa en la que está interesado el cliente. Al igual que en el anterior, solamente se pueden hacer comparaciones sintácticas.

Estos filtros son utilizados a la hora de asignar pesos a las parejas obtenidas en el emparejamiento, lo que ayuda a aumentar la precisión a la hora de elegir el servicio más parecido al buscado.

6.3. Capa Clasificación

La decisión de qué servicio se asemeja más semánticamente al servicio buscado se realiza consultando un razonador que infiere sobre ontologías que acumulan conocimiento sobre el dominio del problema. En este proceso intervienen los siguientes elementos que constituyen esta capa:

6.3.1. Razonador

Como razonador se utilizó BOR [20], el cual está basado en descripciones lógicas y tiene soporte para inferencias sobre instancias y sobre conceptos. Este razonador puede ser usado tanto con ontologías escritas en DAML+OIL (con algunas restricciones), como con ontologías escritas en la especificación OWL Lite. Además se puede utilizar como plugin por Sesame [21].

6.3.2. Repositorio

Como repositorio (almacén de ontologías) se utilizó Sesame, debido a que permite añadir y eliminar información escrita en RDF , y puede almacenar esta información en cualquier base de datos. Sesame soporta tres lenguajes de consulta, RQL, RDQL y SeRQL (Sesame RDF Query Language), para acceder al conocimiento. Permite la interoperabilidad con un razonador de DL. Sesame tiene como característica destacable el poder ser usado con cualquier tipo de base de datos, ya sea relacional, orientada a objeto o de almacenamiento de tripletas. En nuestro caso se utiliza Mysql. Para poder utilizar archivos DAML+OIL se utilizó el plugin de Sesame llamado SeBOR (Sesame+BOR) [20].

6.3.3. Lenguaje de consultas

Las consultas utilizadas para extraer información útil para el emparejamiento fueron realizadas mediante SeRQL .

6.4. Capa de Recursos

Esta capa está representada exclusivamente por los agentes de tipo proveedor y los servicios web a los que representan.

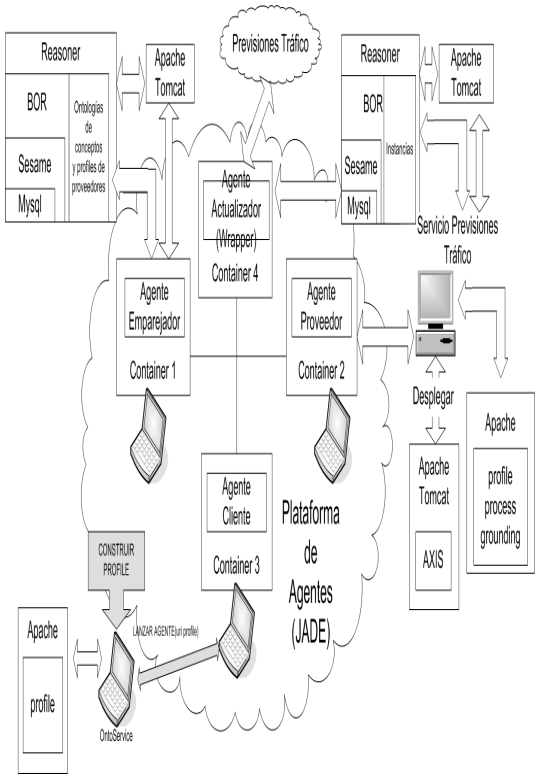

Figura 7. Diagrama de despliegue del sistema.

6.4.1. Agente Proveedor

Es el más simple del sistema. El proveedor que quiera anunciar un servicio dentro de la plataforma debe lanzar este agente pasándole como parámetro la URI de la descripción de ese servicio. Además de registrarse en el DF como tipo proveedor, sólo debe planificar un comportamiento que se encargue de enviar la URI del profile del servicio que represente al agente emparejador cada vez que éste se lo solicite, siempre siguiendo los actos comunicativos de los diagramas de protocolo. En la subcapa inferior de las dos que forman la capa de recursos estarían situados los SW anunciados por los agentes proveedores dentro de la plataforma. Estos SW no están por tanto integrados dentro del sistema multiagente, son servicios web tradicionales.

6.4.2. Servicio Web utilizado en las pruebas

En un escenario de funcionamiento real, el sistema debería tener una serie de SW registrados y poder decidir, en función de los perfiles de servicio que los describiesen, cuál de ellos es el que mejor satisface al cliente. Para comprobar el correcto funcionamiento del ciclo completo de anuncio, descubrimiento e invocación, hubo que crear un SW de prueba, ya que, hasta donde llegó la revisión bibliográfica, no encontramos ninguno relacionado con el área información de tráfico. Para ello se optó por adaptar la información referente a previsiones de tráfico de la web de la Dirección General de Tráfico (DGT) para ser accedida a través de un SW creado por nosotros.

El servicio que implementamos tenía dos componentes:

- ***Agente Actualizador:*** La captura de datos de la Web de la DGT y su tratamiento para poder ser utilizados desde nuestro SW podría haberse hecho de muchas formas, pero nosotros optamos por que un agente lanzado también en la plataforma se encargase de este trabajo. En este agente de tipo *wrapper* se definió una tarea con un temporizador, inicializado al ser lanzado, el cual periódicamente extrae información de la web y la deposita en un repositorio SEBOR. La extracción de la información se hace con las librerías WebL [8] y un script generado para ellas que convierte los datos extraidos de la web a formato DAML+OIL.
- ***Servicio Previsión del estado del tràfico:*** Para generar la estructura básica de nuestro servicio web se utilizaron las herramientas de Apache AXIS [1] y se completó con llamadas al código usado para calcular la previsión a partir de los datos almacenados en el repositorio SEBOR. El contenedor de servlets usado por Axis fue Apache Tomcat.

7. Despliegue del sistema

El diagrama de despliegue del sistema con el servicio web creado por nosotros puede observarse en la figura 7.

Se puede apreciar cómo la plataforma de agentes está distribuida, existiendo varios contenedores situados en distintos ordenadores que en conjunto forman el sistema multiagente. También puede observarse cómo la aplicación OntoService, utilizada para ayudar a la creación de profiles de servicios buscados por clientes e iniciar agentes cliente, está fuera de la plataforma de agentes JADE, siendo por tanto un programa externo a ella. Lo mismo sucede con el servicio web de información sobre previsiones de tráfico, que también está fuera de la plataforma y es accesible a través de un servidor de aplicaciones Apache Tomcat. Los archivos *profile*, *process*, *grounding*, *service* y *wsdl* utilizados para describir servicios son accesibles por Web gracias a servidores web Apache, al igual que los archivos *profile* que describen servicios buscados por clientes. El razonador es accedido por el emparejador y por el servicio web también a través de Apache Tomcat. El agente actualizador está dentro de la plataforma y cada cierto tiempo extrae información de la página Web de la DGT, pero no interviene en las tareas de anuncio, descubrimiento e invocación de servicios web.

8. Pruebas realizadas

Para comparar nuestro algoritmo con el que nos sirvió como base [28] se implementaron ambos y se comparó su funcionamiento dentro del sistema. Paolucci et al. no distinguen grados *fraternales*, y la inclusión de comprobaciones de conceptos hermanos era una de las mejoras que añadimos, por esta razón se realizaron pruebas encaminadas a mostrar la precisión de los dos algoritmos a la hora de encontrar conceptos similares a los buscados por el cliente. Se pudo comprobar que nuestra versión tenía en algunos casos una mayor precisión, debido justamente al hecho de considerar conceptos con grado de similitud fraternal. El segundo tipo de pruebas realizadas estuvieron encaminadas a averiguar el tiempo de respuesta del sistema utilizando los dos algoritmos. En este caso se apreció el aumento de rendimiento gracias al uso de los diversos filtros añadidos a nuestra versión. Aunque nuestro algoritmo en el peor de los casos es más costoso, debido a las comparaciones necesarias para tener en cuenta las relaciones *fraternales* en la taxonomía de conceptos, si consideramos un caso típico en el que sólo el 20 % de los anuncios pertenezcan a la misma categoría de la petición, la respuesta de ambos es similar teniendo en cuenta tal y como se indicó anteriormente, que la precisión del nuestro es mayor. La gráfica de la respuesta temporal correspondiente a este caso puede observarse en la figura 8. Para más información sobre las pruebas realizadas puede consultarse [16].

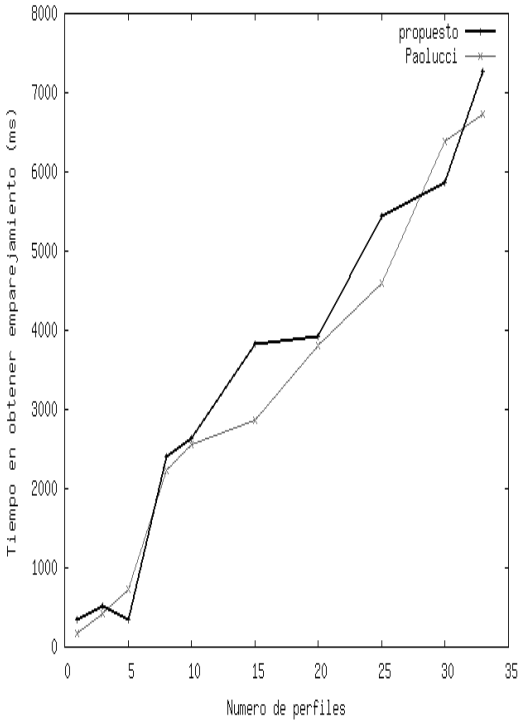

Figura 8. Pruebas de tiempo en la obtención de emparejamiento.

Por último, un estudio de factibilidad en el que se recojan las opciones y mejoras de este sistema con respecto a los actuales sistemas de publicación y búsqueda de información de tráfico, como la Web, Wap, etc., permitirá obtener como resultado ciertos valores que nos posibiliten establecer una valoración objetiva de las ventajas de este nuevo sistema en la vida real y su posible aplicación a través de las diferentes Administraciones.

9. Caso de uso simple

A continuación se detalla una de las pruebas del sistema realizada para comprobar que el prototipo desarrollado era funcional. Las pruebas se realizaron con un *profile* que describía al *servicio previsión* y otro de un servicio ficticio llamado *TraficoCatProfile*. El *profile* que describía al servicio previsión fue generado con la herramienta OntoService, y el agente cliente fue lanzado desde esta herramienta pasándole como parámetro la URI del *profile* generado.

Descripción del caso de uso:

En primer lugar, se lanzó el contenedor principal de la plataforma, y después el agente Sniffer de JADE para espiar las comunicaciones entre los agentes que iban a ir siendo iniciados. Posteriormente se lanzó el agente emparejador (al que llamamos *Matchmaker*) en otro contenedor (en este caso llamado *Container-1*). Una vez lanzado el agente *matchmaker* se inició el proveedor que representaba en la plataforma al servicio previsión. Después se lanzó otro contenedor con un agente proveedor que representaba al servicio ficticio. El último agente que se inició fue el agente cliente, al que se le pasó como parámetro la URI de un profile más parecido semánticamente a nuestro servicio previsión. En la figura 9, que muestra la interfaz gráfica del Sniffer, se pueden observar las interacciones entre los distintos agentes de la plataforma.

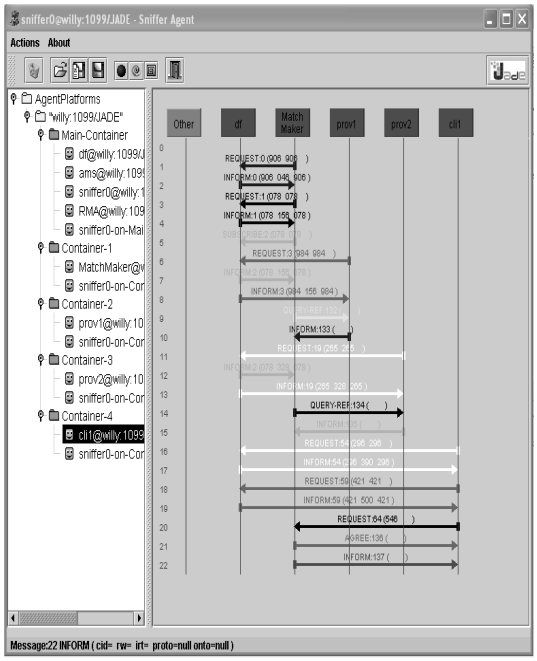

Figura 9. Agente Sniffer durante la prueba.

El emparejador, tras contrastar el *profile* recibido del cliente con todos los que tenía en su repositorio, decidió que el correspondiente a nuestro servicio previsión era el que más se le parecía semánticamente. En ese momento el agente emparejador notificó al agente cliente la URI del servicio previsión y el agente extrajo de este profile, ayudándose de un analizador *(parser)*, después que el usuario pulsase sobre el botón generar, la información necesaria para construir dinámicamente una interfaz gráfica para que el usuario introdujese los parámetros necesarios para realizar la invocación. En la figura 10 se puede observar cómo los parámetros eran introducidos en la interfaz generada dinámicamente.

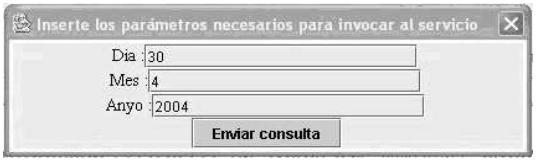

Figura 10. Cuadro de diálogo generado dinámicamente.

En ese momento el cliente obtuvo con el *parser* el fichero *wsdl* enlazado con el archivo *grounding* del servicio, lanzó el invocador dinámico y recogió los resultados de la invocación para mostrarlas en su interfaz gráfica, como se puede observar en la figura 11.

Figura 11. Interfaz gráfica mostrando el resultado de la invocación.

10. Conclusiones y trabajo futuro

Se ha analizado, diseñado, implementado y probado un sistema multiagente basado en la interacción con SWS que constituye una aportación académica al desarrollo de la ingeniería de la web del futuro, la *Web Semántica*. Teniendo en cuenta la gran inmadurez y falta de estabilidad y control de compatibilidad hacia atrás de las herramientas disponibles actualmente para los desarrollos de *Web Semántica* en forma de razonadores, bibliotecas, etc., se ha tenido gran dificultad para conseguir una solución estable. Aunque se ha logrado un prototipo funcional, éste está aún muy lejos de lo deseable para una posible explotación comercial.

La plataforma final permite que agentes clientes soliciten a agentes emparejadores la búsqueda de SW acordes con las capacidades semánticas que están buscando, para después poder invocarlos. Para conseguir esto se ha propuesto un algoritmo de emparejamiento semántico basado en las características que puede definir un servicio web en el *profile* de su descripción semántica, el cual puede haber sido especificado mediante la herramienta OntoService. En un futuro se pretende que el sistema utilice la recomendación de OWL aportada por el W3C [35], por lo que si BOR no es adaptado para poder utilizarla sería necesario buscar una alternativa a este razonador.

Actualmente existe una percepción general por parte de la comunidad involucrada en este tipo de sistemas en que los Servicios Web pueden crear nuevas oportunidades de negocio y generar beneficios a largo plazo. Aunque un elevado número opina que estos sistemas aún no están preparados para entornos de misión crítica y producción. En general, coinciden en resaltar la importancia de los Servicios Web en la comunicación de las aplicaciones y Sistemas de Información de la empresa. Otro aspecto importante a destacar es la conformidad de estos hacia un conjunto de estándares abiertos aceptados por la industria. Los entornos de desarrollo serán fundamentales para proyectos de desarrollo de Servicios Web y Web Semántica. Lo dicho anteriormente es totalmente extrapolable a cualquier proveedor de información de tráfico rodado, como lo puedan ser las propias Administraciones.

Agradecimientos

Esta investigación ha sido financiada por el proyecto CICYT del Ministerio de Ciencia y Tecnología, de España, cuya referencia es TRA2004-06276 / MODAL ('Desarrollo de un sistema de intercambio de información entre camiones y dispositivos externos para control de mercancías mediante el uso de una infraestructura conceptual basada en ontologías. ')

Referencias

[1] Apache. Apache Axis. En: http://ws.apache.org/axis/.

[2] C.Abela; M.Montebello. DAML enabled Web Services and Agents in the Semantic Web. En: WS–RSD'02, Erfurt Germany, October 2002.

[3] Ceccaroni, L. and Ribiere,. Experiences in Modeling Agentcities Utility-Ontologies with a Collaborative Approach. En: Ontologies in Agent Systems Workshop, Autonomous Agents and Multi-Agent Systems Conference 2002, Bologna, Italy, July 2002.

[4] CMU. A-Match. En: http://www-2.cs.cmu.edu/ softagents/a-match/index.html.

[5] CMU. Editor OWL-S. En: http://www.daml.ri.cmu.edu/tools/details.html.

[6] CMU. Matchmaker by CMU. En: http://www-2.cs.cmu.edu/ softagents/.

[7] The DAML Services Coalition. Daml-s:semantic markup for web sevices, 2002.

[8] Compaq. Compaq's Web Language. En: http://research.compaq.com/SRC/WebL/.

[9] D. Trastour; C. Bartolini and J. Gonzalez-Castillo. A Semantic Web Approach to Service Description for Matchmaking of Services. En: 1st Semantic Web Working Symposium, CA. 2001.

[10] FIPA. Agent UML. En: http://www.auml.org/.

[11] Foundation for Intelligent Physical Agents. Fipa modeling: Interaction diagrams. Technical report, Foundation for Intelligent Physical Agents, Geneva, Switzerland, Julio 2003. preliminar, primera propuesta.

[12] Foundation for Intelligent Physical Agents. Fipa agent management specification. Technical report, Foundation for Intelligent Physical Agents, Geneva, Switzerland, Marzo 2004. Standard.

[13] I. Constantinescu, B. Faltings. World Wide Web Consortium Efficient Matchmaking and Directory Services. En:Technical Report No IC/2002/77, Noviembre, 2002.

[14] J. Gonzalez-Castillo; D. Trastour and C. Bartolini. Description Logics for MatchMaking of Services. En: HP Technical Reports. October 30 th , 2001.

[15] J. J. Samper, A Cervera, E. Carrillo. ONTO-SERVICE: Una Herramienta para la Edición de Perfiles y Visualización de Ontologías en Servicios Web Semánticos. WebMedia/LA-Web 2004, the Joint Conference the 2nd Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web. Octubre 12-15, Ribeirao Preto, Brasil. Pag 304-306, ISBN 85-7669-010-1.

[16] J. Samper, E. Carrillo,J. Martínez. Algoritmo de emparejamiento de perfiles en servicios web semánticos. En: Revista Colombiana de Computación Volumen 6 número 1, ISSN 1657-2831, Junio, 2005.

[17] M. Williamson. K. Decker, K. Sycara. Middle-agents for the internet. *En: Proc. 15th, IJCAI, Nagoya, Japan.*, pages 578–583, August 1997.

[18] L. Lei and I. Horrocks. A software Framework For Matchmaking Based on Semantic Web Technology. En: Twelfth International World Wide Conference (WWW2003), pages 331-339, ACM, 2003.

[19] M. Klein, and A. Bernstein. Searching for Services on the Semantic Web using Process Ontologies. En: The First Semantic Web Working Symposium (SWWS-1). Stanford, 2001.

[20] Ontotext. BOR - a Pragmatic DAML+OIL Reasoner. En: http://www.ontotext.com/bor/.

[21] Openrdf. Sesame. En: http://www.openrdf.org/.

[22] Payne, T. Lassila, O. Semantic Web Services. En: IEEE Intelligent Systems. Special Issue on Semantic Web Services. 2004.

[23] R. Ferraz; S. Labidi and B. Wanghon. A semantic matching method for clustering traders in B2B Systems.

[24] Semantic Web Services Initiative (SWSI) . En: http://www.swsi.org/.

[25] Semantic Web Services Interest Group. . En: http://www.w3c.org/2002/ws/swsig/.

[26] Semantic Web Services Language (SWSL) Committee,. http://www.daml.org/services/swsl/.

[27] Sycara, K., Klusch, M., Widoff, S., Lu, J., . Dynamic Service Matchmaking Among Agents in Open Information Environments. En: Journal ACM SIGMOD Record, 1999.

[28] Paolucci, Massimo Kawamura, Takahiro Payne, Terry R. Sycara, Katia. Semantic matching of web services capabilities. *En: Proceddings of the 1st International Semantic Web Conference (ISWC2002)*, September 2002.

[29] Sycara Katia, Widoff Seth, Klusch Matthias and Lu Jianguo, . LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. En: In Autonomous Agents and Multi-Agent Systems, 5, 173-203, 2002.

[30] T. R. Payne ; M. Paolucci and K. Sycara. Advertising and Matching DAML-S Service Descriptions. En: Semantic Web Working Symposium (SWWS), 2001.

[31] Tilab. Java Agent Development Framework. En: http://jade.tilab.com.

[32] Uddi.org. UDDI Universal Description Discovery and Integration protocol. En: http://www.uddi.org.

[33] W3C. FOAF Vocabulary Specification. En: http://xmlns.com/foaf/0.1/.

[34] W3C. GEO Vocabulary Specification. En: http://www.w3.org/2003/01/geo/.

[35] W3C. OWL Web Ontology Language. En: http://www.w3.org/TR/owl-features/.

[36] W3C. URI Uniform Resource Identificator. En: http://www.w3.org/Addressing/.

[37] W3C. WSDL Web Services Description Language. En: http://www.w3.org/TR/wsdl.

[38] Zaremski, Amy Moormann and Wing Jeannette M. . Signature matching: a Tool for Using Software Libraries. 1995.

[39] Zaremski, Amy Moormann and Wing Jeannette M. Specification Matching of Software Components. 1997.