Algoritmo de emparejamiento de perfiles en Servicios Web Semánticos

José Javier Samper [*] Eduardo Carrillo[**] Juan José Martínez [*]

Resumen

Este artículo describe las principales características de un algoritmo de emparejamiento de Servicios Web Semánticos . El algoritmo aprovecha al máximo las capacidades proporcionadas por la ontología de descripción de servicios y constituye una mejora en relación con propuestas existentes. Además se describen los principales componentes relacionados con el proceso de implementación. El sistema desarrollado interactúa con ontologías de descripciones de conceptos desarrolladas en DAML+OIL y descripciones de servicios en DAML-S, usando como repositorio/razonador el sistema Sesame+BOR. Finalmente, se describe la implementación de comparaciones entre parámetros de perfiles de servicio mediante consultas realizadas a la base de conocimiento.

Palabras claves: *Web Semántica, Servicios Web, Servicios Web Semánticos, Emparejamiento*

Abstract

This paper describes the main characteristics of an algorithm for matching Semantic Web Services. The algorithm takes advantage of the capacities provided by ontologies that describe services and constitutes an improvement when compared to existing proposals. In addition, the main elements related with the implementation are also described. The system interacts with ontologies of descriptions of concepts developed in DAML+OIL and descriptions of services in DAML-S, using the reasoner and Sesame+BOR reasoner/repository. Finally, we explain the implementation of comparisons among parameters from service profiles by means of queries to the knowledge base.

Keywords*: S**emantic Web, Web Services, Semantic Web Services, Matchmaking.*

1 Introducción

Dos tecnologías fundamentales para el intercambio de información en la Web del mañana son la Web Semántica y los Servicios Web. La Web Semántica tiene como visión la búsqueda de una Web más inteligente en la que se pueda conseguir una comunicación efectiva entre ordenadores y centra sus esfuerzos principalmente en la búsqueda de descripciones enriquecidas semánticamente para los datos en la Web. En este sentido, se promueven descripciones que incluyan no sólo las estructuras de datos, sino también las relaciones existentes con otros conceptos, las restricciones, reglas que permitan realizar inferencia, etc. Así mismo se promueve la definición y reutilización de vocabularios u ontologías de conceptos que faciliten el procesamiento por parte de las máquinas. Por lo tanto, lenguajes descriptivos como DTDs, XML Esquemas y RDFS no son suficientes para

[*] Universidad de Valencia, Instituto de Robotica, Polígono de la Coma s/n, Paterna, Valencia, España.
[**] Universidad Autónoma de Bucaramanga,. Calle 48 Nro 39-234, Bucaramanga, Colombia.

describir las relaciones complejas existentes en ontologías, por lo que se han propuesto otros lenguajes como DAML+OIL y OWL que tienen mayor expresividad.

Por otra parte, los Servicios Web promueven la interacción entre *aplicaciones*. El servicio es un componente software que puede procesar un documento XML que recibe a través de combinaciones de protocolos de transporte y de aplicación. Los Servicios Web están diseñados para mover documentos XML entre procesos de servicio usando protocolos estándar de Internet.

Dos descripciones XML iguales de un servicio Web tienen un significado u otro dependiendo del contexto donde se encuentren, mientras que las descripciones semánticas cubren esta carencia. El uso de la semántica para describir servicios solventa este problema de los sistemas de emparejamiento basados en UDDI. A partir de las propuestas de la Web semántica se creó la ontología de orden superior DAML-S para descripción semántica de Servicios Web, que más tarde evolucionó en OWL-S (basadas en los lenguajes de marcado semántico DAML y OWL respectivamente) [4], las cuales permiten describir semánticamente las capacidades los Servicios Web, para que agentes software puedan leer de esas descripciones y razonar sobre la forma de interactuar con los servicios que ellas describen.

En la actualidad existen algunas propuestas de integración para emparejamiento de Servicios Web como las presentadas en [8], [18] y [5]. La arquitectura de integración descrita en [5] 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. 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.

Como aparece en [7] una descripción de servicio es una colección consistente por ella misma de restricciones sobre propiedades nombradas de un servicio. Los perfiles describen capacidades de servicio, por tanto pueden describir tanto las capacidades de los servicios ofertados por los proveedores (los anuncios) como lo esperado por los clientes (peticiones). Las peticiones son enviadas a registros de servicios de Web que los emparejan con perfiles de anuncios de otros servicios almacenados en alguna base de datos (repositorio) para identificar qué servicios proveen el mejor emparejamiento.

Por otra parte el proceso de *emparejar* consiste en podar el espacio de posibles emparejamientos entre posibles servicios ofertados y peticiones. Un servicio de emparejamiento requiere tanto metadatos ricos y flexibles como algoritmos de emparejamiento [15]. El emparejamiento entre anuncios y peticiones se considera idóneo cuando ambos son lo suficientemente similares. Un emparejamiento vendrá determinado por los diferentes grados de similitud.

Este artículo se basa en la propuesta de un algoritmo de emparejamiento de Servicios Web descritos semánticamente mediante comparaciones de parámetros que hacen uso de ontologías de conceptos y de servicios, como una extensión a propuestas actuales. Para conseguir este objetivo, el artículo ha sido organizado en la siguiente forma: en la sección 2 se presenta el estado del arte, en la sección 2.1 se describen las ontologías para descripción semántica de servicios y los sistemas de emparejamiento. Posteriormente en la sección 3 se describe el algoritmo propuesto, en la sección 4 la implementación de las consultas y en la sección 5 se describe el análisis de los resultados obtenidos.

2 Estado del arte

2.1 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 Servicio Web 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. OWL-S nos da lo necesario para conseguir la representación semántica de servicios basándose en OWL, que soporta razonamiento sobre inclusión en taxonomías de conceptos y permite 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 tiene limitaciones, es suficientemente expresivo como para permitir la descripción de un gran número de servicios y permitir emparejamientos entre ellos.

2.2. Emparejamiento

En relación con el concepto de emparejamiento existen trabajos provenientes de la Ingeniería del Software, como la propuesta presentada por Zaremski y Wing en [20] y [21], relacionada con la reutilización de software y recuperación de librerías, mediante la especificación y posteriormente emparejamiento de componentes software. En [21] extienden el concepto de *signature matching* [20], para tener en cuenta las restricciones en las entradas y en las salidas en funciones y módulos. Este concepto es llamado *specification matching* y provee no sólo información de tipo estático sino también descripciones de comportamiento dinámico. Los autores distinguen varios tipos de emparejamientos en la especificación de emparejamientos de software.

En [17] Sycara et al. presentan LARKS, 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. Los conceptos usados por las descripciones pueden ser definidas mediante un lenguaje de descripción de conceptos denominado ITL (Information Terminological Language). Este lenguaje común es usado por los agentes mediadores para emparejar agentes de solicitud de servicio con agentes proveedores de servicios que resuelvan los requerimientos del agente que hace la solicitud.

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* entonces más tarde, éste le devolverá como resultado información respecto al servicio que empareje con la descripción dada en el requerimiento. Esta información incluirá las IOPEs (*Inputs, Outputs, Preconditions, Effect*s) las cuales están reflejadas en el perfil del servicio así como información adicional de parámetros de servicio, información del proveedor etc., que el proveedor del servicio habrá comunicado con anterioridad al emparejador.

En la práctica el cliente debería ya conocer las IOPEs y lo que no sabrá es el tipo de proceso, si será atómico o por el contrario se tratará de una composición de servicios en cuyo caso habrá que realizar pasos intermedios que consultarán el modelo de proceso para identificar el flujo de trabajo desde la entrada a la salida (con la posibilidad de que la salida de un proceso intermediario sea usada como entrada en el siguiente proceso atómico en la secuencia dada) hasta que todos los procesos involucrados sean ejecutados. Por tanto los agentes que realizan la petición de servicio, deberán de ser capaces de interpretar el contenido de la ontología de proceso y el grounding para poder comunicarse con un agente proveedor de tal servicio [22] [10].

La idea principal que subyace en la aproximación de emparejamiento viene dada por la búsqueda de un servicio basada en el tópico o categoría y después en el uso de los parámetros de salida y resto de parámetros. El problema tal y como apuntaba Massimo Paolucci en [10] es que el uso de categorías de servicio puede no tener casi significado si las categorías permitidas en la ontología son demasiado amplias. Por otra parte, para dotar de significado se necesitará especificar muchas clases diferentes, una por cada tipo de servicio. El uso de los parámetros de entrada y salida permitirán solventar este problema, permitiendo una definición implícita de los servicios. Esto requerirá el uso de ontologías menos precisas aunque, por el contrario, serán necesarias

descripciones de servicios más esmeradas y un proceso de emparejamiento más complejo. La idea por tanto es soportar ambos tipos de emparejamiento, debido al resurgimiento de grandes ontologías y clasificaciones de servicios.

El modelo y algoritmo de emparejamiento semántico planteado en [13] es el que ha sido utilizado por la mayoría de investigadores como base para sus sistemas. Plantean un sistema de emparejamiento basado en la semántica descrita en el perfil (Profile) de los servicios y en la utilización de los registros UDDI para mantener las descripciones de los servicios, aunque el referente para el resto de investigadores ha sido el algoritmo de emparejamiento utilizado. 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) usando la misma ontología. En este trabajo 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 de las entradas se usa más que nada para resolver las igualdades que se produzcan.

En [23] se expone una extensión del 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 con el mayor grado posible con las que los proveedores introducen en los perfiles (Profiles). Tuvieron en cuenta las precondiciones porque consideran que son unos parámetros importantes en las negociaciones, ya que puede darse el caso de Servicios Web que incluyan precondiciones que negocian el tipo de pago o el tipo de tarjetas de crédito que admiten etc.

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[1], marco en el cual se decidió probar nuestra arquitectura.

2.2.1. Grados de similitud o match

El grado de similitud depende de la relación entre los conceptos (tomada de las ontologías) 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 diferencia entre descripciones de servicios, tomando como base el conjunto de relaciones entre ellos.

La denominación de los grados varía según la literatura [1],[9],[13],[19]:

- *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 [13] incluyen como e*xact 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 ? 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)

[1] En el caso de servicios que ofrezcan información de tráfico gratuita a un usuario, después de la ejecución del servicio, el usuario dispondrá del conocimiento requerido, pero este cambio de estado no es un cambio tangible y por lo tanto no queda muy claro la posibilidad de su uso en el proceso de descubrimiento de un servicio o en la composición, donde los efectos de unos servicios se conviertan más tarde en precondiciones de otros.

- Subsume o Container: cuando los conceptos en la petición incluyen los del anuncio; formalmente, (A ? 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 ? P) ? ?

En [7] introdujeron nuevos tipos de emparejamiento que más tarde fueron adoptados por Li y Horrocks [9] 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, ? (A ? P ? ?).

Para entender el proceso de emparejamiento mencionado, es de suma importancia tener en consideración la definición de "*Open World descriptions*" aportada por Tommaso Di Noia et al. en [12]: *"La ausencia de una característica en la descripción de un anuncio o de una petición no se debe interpretar como una restricción de ésta. En su lugar, debe ser considerada como una característica que se podría refinar más tarde, o dejarla abierta si se considera irrelevante para el usuario"*. Esta definición clarifica la idea de que incluso cuando los servicios anunciados y los requeridos no tienen un emparejamiento exacto, puede ser posible o necesario usarlos en instancias específicas. Por tanto, los emparejamientos parciales también son importantes [19]. Lo anterior es conocido como "emparejamiento flexible" para distinguirlo del exacto considerado el más restrictivo de todos.

En [6] se establecen tres diferentes tipos de emparejamientos dados por las diferentes relaciones entre perfiles de petición y ofertados. Para expresarlo formalmente hacen uso de Lógica Descriptiva (DL). Siendo T una ontología común establecida para la descripción de servicios:

? Implicación: T ?? (P ? A) y T ?? (A ? P)[2]. Cada restricción impuesta por P es completada por A y viceversa. Da lugar al emparejamiento exacto.

? Consistencia: (A ? P) es satisfactoria en T, donde las restricciones no se excluyen mutuamente. En este tipo de emparejamientos es necesario establecer un límite en la distancia entre ambas descripciones, que se medirá teniendo en cuenta dicha ontología. Es decir, ¿cuántos destalles en P tengo que preguntar a la otra parte A?. Emparejamiento potencial.

? Inconsistencia: A ? P es insatisfactoria en T, lo que implica que algunas restricciones de una descripción están en conflicto con las de la otra. En este tipo de emparejamientos cabe preguntarse por el grado de inconsistencia de A en P, es decir ¿Cuántos detalles en P tengo que eliminar para poder aceptar A?. Emparejamiento parcial.

3 Algoritmo de emparejamiento propuesto

Para el desarrollo de este algoritmo se han tomado como base los siguientes interrogantes:

? ¿Qué parámetros de la clase Profile podrían ser usados para la búsqueda de servicios?

? ¿Qué grados de emparejamiento se deberían utilizar en las comparaciones de los parámetros funcionales?

[2] Extensión del concepto de equivalencia dado por la visión simplista de tratar únicamente la equivalencia sintáctica.

A partir del análisis de estos requisitos se ha desarrollado el algoritmo que se describirá en los siguientes apartados.

3.1 Algoritmo Base

El algoritmo consiste en buscar emparejamientos de valores semánticos que fueron utilizados para describir cada una de las capacidades del servicio, tanto por parte del proveedor como por parte del cliente en su petición. Por este motivo, las peticiones del cliente serán perfiles como los que utilizan los proveedores para anunciarse.

El algoritmo planteado en este artículo toma como base el algoritmo de Paolucci et al., utilizado en su sistema emparejador, pero posee algunas diferencias, las cuales se centran en cuatro aspectos:

1. Filtrado en función del perfil de la petición, de los perfiles de proveedores utilizados para el emparejamiento, para reducir el coste del algoritmo.
2. Uso de parámetros no funcionales[3].
3. Obtención del grado de emparejamiento mediante el uso de nuevos grados.
4. Mejoras en la ordenación de resultados.

3.1.1 Fases del algoritmo:

Las fases del algoritmo propuesto son:

- ***Fase 1***: Filtrar entre todos los anuncios de servicios, aquéllos que pertenecen a la misma categoría que la solicitada por el cliente:

El hecho de utilizar nuevos filtros nos permite aprovechar otras características que posee la clase *Profile* para anunciar Servicios Web y que no fueron explotadas por anteriores algoritmos.
El algoritmo de emparejamiento tiene un coste considerable ya que intenta emparejar la petición con cada uno de los proveedores disponibles, y para hacer esto cada una de sus entradas o salidas se contrasta con todas las de cada proveedor para encontrar la que más se le parezca semánticamente. Por esta razón, incluimos un filtrado anterior al emparejamiento, cuya finalidad es descartar proveedores que no cumplan unas determinadas características definidas también en los perfiles de la ontología.

El filtrado está basado en la categoría de servicio y es de carácter obligatorio en la formulación de la petición de servicio por parte del usuario. Servirá al sistema *emparejador* para obtener del repositorio de perfiles (*profiles)* aquella lista de servicios que pertenezcan a la categoría que busca el cliente.

- ***Fase 2***: De la lista resultante, combinar todas las posibles parejas entre la petición del cliente y cada uno de los anuncios de proveedores:

Esta fase consiste en establecer las distintas combinaciones posibles entre la petición dada por el cliente y los anuncios publicados por los proveedores, que por la fase anterior, pertenecen a la misma categoría de servicio.

---

[3] Son atributos que no son obligatorios. Suelen utilizarse para que los propios proveedores añadan las características especiales que creen que ofrecen sus servicios Web. Se pueden añadir más a los ya definidos gracias a otras propiedades definidas para la clase Profile, pero los principales son: ServiceCategory, QualityRating y GeographicRadius.

- ***Fase 3***: Cada vez que se obtiene una pareja en la fase anterior, aplicar sobre ella los diferentes grados de similitud para los parámetros funcionales y calcular los pesos relativos para:

i. Parámetros funcionales correspondientes a salidas,
ii. parámetros funcionales correspondientes a entradas y
iii. parámetros no funcionales

El cálculo de pesos relativos a las entradas y parámetros no funcionales (ii e iii) se hará siempre que éstos hayan sido especificados por el cliente y si el grado de similitud de una pareja en cuanto a sus salidas es positivo, es decir, si tienen alguna salida en común.
El riesgo de obtención de falsos positivos o negativos será menor dependiendo de la precisión en la asignación de pesos a las parejas "*petición cliente – anuncio proveedor"*

Los dos principales procesos que caracterizan esta fase son:

a) **Obtención del grado de emparejamiento para parámetros funcionales**:

En [13] se establece una función de ranking (*DegreeOfMatch*()) en la que se diferencian cuatro posibles casos además del de fallo. Además se le asigna el mismo peso (exact) no solo a conceptos iguales sino también al caso en que los conceptos de la petición son subclase (relación directa) de los del anuncio.
En [9] además de diferenciar los grados de Paolucci en cuanto a emparejamiento *exacto* y *ser subclase de*, destaca otro posible emparejamiento que ya fue anteriormente expuesto por [7]:

? Grado de *intersection* cuando un anuncio y una petición son compatibles, es decir mantienen algo en común.

Se propone un algoritmo con características similares a los aquí expuestos pero con importantes modificaciones en los grados de emparejamiento ya que en estos trabajos eran demasiado generales, por lo que se propone 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 [13] 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 [13]. 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 ofertado 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 ofertado 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.

El pseudocódigo de nuestro método para la asignación de grados de emparejamiento se puede observar en la Figura 1:

```
DegreeOfMatch(outR,outA)
        {
                if Iguales(outA, outR)
                        return EXACTO
                else if SubClassOf(outR,outA)
                        return CSUBCLASEP
                else if SubClassOf(outA,outR)
                        return PSUBCLASEC
                else if Subsumes(outA,outR)
                        return PSUBSUMEC
                else if Subsumes(outR,outA)
                        return CSUBSUMEP
                else if Hermanos(outR,outA)
                        return CHERMANOP
                else
                        return FALLO
```

Fig. 1 Método de asignación de grados

b) Coincidencia exacta en parámetros no funcionales.

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 servicio web, por lo que uno de los objetivos fue cubrir este vacío. El algoritmo emparejador explota al máximo las posibilidades de *Profile*, con el fin de hacer uso de todas las capacidades de los servicios, descritas semánticamente.

A continuación veamos 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 Servicios Web es la misma que la pedida 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. Es un parámetro como el anterior, es decir, solo se puede hacer un emparejamiento sintáctico debido a que esta propiedad de Profile también está definida como un *XML Schema String*.

Hay que destacar que los anteriores parámetros no funcionales van a incidir en el peso de distinta manera. Por ejemplo se ha querido premiar a aquellos parámetros de tipo semántico, de tal forma que si su comparación es positiva cada uno de ellos acumulará 5 puntos en el cómputo de peso relativo a este tipo de parámetros, mientras que si tratamos con los emparejamientos de *serviceName* y *ActorName* (sintácticos) aportan 3 y 2 puntos respectivamente.

- *Fase 4*: Teniendo en cuenta los pesos anteriormente calculados, ejecutar el proceso de ordenación para insertar en el lugar correcto de la lista de servicios resultado de la consulta. El servicio que encabeza la lista será aquél que se considere el más óptimo.

Las parejas obtenidas deberán ser ordenadas en función de su peso. La inserción de parejas en la lista ordenada se realiza cada vez que se construye una pareja, de tal forma que se va comparando el peso actual de la pareja recién encontrada con los pesos de cada una de las parejas ya almacenadas.

El mecanismo de inserción ordenada consiste en tomar la pareja recién encontrada y comparar su peso con el de la primera de las parejas que aparece en la lista y que será aquélla que hasta el momento poseía un mejor peso.

El hecho de que ciertos parámetros tengan mayor relevancia que otros para la búsqueda del servicio más adecuado, obliga a que la variable peso no sea manejada como algo general sino que ésta sea utilizada independientemente para cada tipo de emparejamiento. De tal forma que se acumulan pesos para salidas, entradas o parámetros no funcionales. En este último caso se considera un único valor resultado de la distinta aportación de la similitud entre parámetros dependiendo de si ésta es de tipo semántico o sintáctico.

La ordenación propuesta da más importancia al peso de las salidas, igual que en el método *sortRule()* de [13], ya que lo más importante es que el cliente obtenga lo que quiere, que debe ser lo que le proporciona el proveedor mediante las salidas (*outputs*) del servicio. A continuación, no se considera el peso de las entradas sino que se compara antes la suma de los pesos obtenidos con los parámetros de tipo no funcional que son: la proximidad en función de la región o radio geográfico, calidad de servicio, y coincidencia sintáctica del nombre del servicio y del nombre del proveedor (ver Figura 2).

Posteriormente, se compara el peso de las entradas, ya que éstos se consideran parámetros menos importantes para poder encontrar el Servicio Web que aporte lo que el cliente busca, son solamente valores de entrada antes de ejecutar el servicio y obtener de esta manera el beneficio, producto o información que el cliente espera en realidad, es decir, las salidas.

Por último, para deshacer un posible empate en este punto, se considerará el número de veces que se ha obtenido el grado de similitud *FALLO*. Debemos de tener en cuenta, que el hecho de obtener un fallo no significa el descarte de un determinado servicio, ya que en realidad, el fallo será obtenido en la búsqueda de una pareja para un parámetro dado, pero no para todos los miembros del conjunto. Por lo que en el cómputo global para ese conjunto de parámetros su valor podrá no ser nulo. De esta forma, habrá situaciones en las que para un conjunto de parámetros de la petición de un servicio, solo unos pocos consigan emparejarse con los parámetros de un determinado servicio ofertado, mientras que el número de parejas entre el conjunto de la petición y

el conjunto de parámetros de otro servicio ofertado podría ser mayor, y sin embargo obtener el mismo cómputo global, ya que se considerará la suma de todos, y en ella habrá distinta ponderación dependiendo del grado de similitud obtenido. De esta forma se ha querido premiar el hecho de que el peso esté más repartido entre todos los parámetros y no esté concentrado en unos pocos.

El proceso de **ordenación** se inicia con la comparación entre la pareja recién encontrada y la de la cabeza de la lista. Si el resultado es *false*, querrá decir que el peso no es mejor, por lo que no se habrá encontrado una mejor solución, y por tanto deberá compararse con el resto de la lista e insertarse en el lugar adecuado. En este caso, no se habrá conseguido encontrar una solución mejor, pero el registro de esta nueva pareja servirá para realizar un sistema tolerante a fallos, ya que en el caso de un fallo en el uso del servicio obtenido como mejor solución, se podrá recurrir al uso de información provista por el elemento siguiente de la lista.

```
ORDENACION(peso1,peso2)
{
      si peso1.pesosalidas > peso2.pesosalidas
            devuelve true
      sino si peso1.pesosalidas < peso2.pesosalidas
            devuelve false
      si peso1.sumaotrosparametros > peso2.sumaotrosparametros
            devuelve true
      sino
            si peso1.pesoentradas > peso2.pesoentradas
                        devuelve true
            sino si peso1.pesoentradas <= peso2.pesoentradas
                        devuelve false
            sino si peso1.numfallos < peso2.numfallos
                        devuelve true
            sino
                        devuelve false
}
```

Fig. 2 Proceso de ordenación

3.2. Descripción del proceso completo y diagrama de actividades

El emparejador actúa como un consumidor en un esquema básico productor/consumidor. Permanece dormido en *Match* (Figura 3) mientras no detecte que se haya insertado un elemento nuevo en la lista de URIs de peticiones. Cuando esto suceda, comienza el proceso de emparejamiento que se inicia con la extracción de la lista de peticiones (URIs), de aquélla que encabece la lista, el *matchmaker* o emparejador accede al *Profile* del cliente y extrae mediante un *parser*, los conceptos que definen cada uno de los parámetros. Una vez extraídos tanto los parámetros funcionales como los no funcionales, comienza el proceso de emparejamiento (Match) que se describe a continuación. (Ver Figura 3).

El proceso consiste en establecer todas las posibles parejas petición-anuncio entre la petición cursada por el cliente y cada uno de los anuncios tomados de la lista de perfiles de Servicios Web que resultó del filtrado por el parámetro categoría de servicio. Cada vez que se establece una pareja, se le aplica el método *match* (figura 4) que consiste en averiguar cada uno de los pesos parciales relativos a los parámetros de salidas, entradas y parámetros no funcionales, que nos indiquen el grado de similitud semántica entre anuncio y petición. Tanto en la comparación de las salidas como en las entradas (parámetros funcionales) se van comparando cada uno de los parámetros de la petición con los parámetros de su pareja anunciante, haciendo uso de los diferentes grados de similitud.

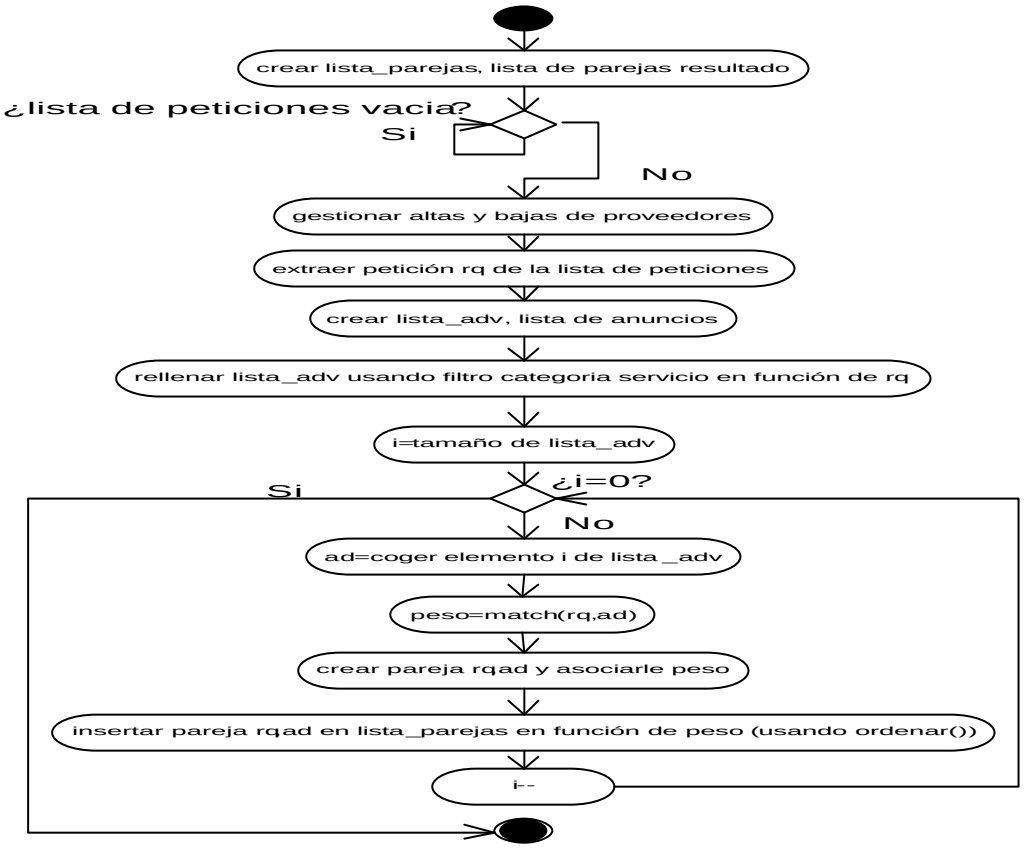

Fig. 3. Diagrama de actividades del método Match

El proceso de cálculo de pesos se realiza de la siguiente forma: (Ver Figuras 4 y 5)

Partiendo de que el sistema ha cargado inicialmente en un repositorio la ontología de conceptos asociada a la categoría de servicio definida en la petición y en los Servicios Web y una vez obtenido un *output* del anuncio y un *output* de la petición, se averigua cuál es el grado de emparejamiento, para lo cual se sigue el siguiente proceso:

- Ejecución de cada una de las consultas (*DegreeOfMatch*, figura 1), de mayor a menor grado, esperando a que alguna de las consultas devuelva resultado positivo, o, si no se consigue resultados en las consultas, se llegará al grado de emparejamiento "*Fail*" que representará que no hay emparejamiento entre los parámetros dados. Cada uno de los grados se ha cuantificado de tal forma que el grado obtenido se suma a la variable que mantiene la suma de todos los grados de similitud de los *outputs*.

Al terminar ese proceso para todos los *outputs* definidos por el usuario, y emparejados con todos los *outputs* de los Servicios Web mantenidos en el repositorio, los valores obtenidos como resultado son almacenados en la variable **peso** (*peso.salidas*).

En este punto y como mejora de anteriores propuestas, se comprueba si el peso obtenido tiene valor nulo en cuyo caso se abandona el proceso comparativo con el resto de parámetros. Si no fuera así se realizará el proceso anterior pero esta vez para los parámetros funcionales de entradas si los hubiera. El valor resultante será almacenado en *peso.entradas.*

Por último, una vez ya emparejados todos los Servicios Web en sus parámetros funcionales de entrada y salida, el siguiente paso es comprobar si estos servicios tienen definidos los parámetros no funcionales que haya especificado el usuario en su petición.

Mediante la implementación de consultas se comparan los parámetros no funcionales de radio geográfico (método *RadioGeoMatch*), calidad del servicio (método *CalidadServicioMatch*) ambos con una aportación de 5 y nombre de servicio (método *NombreServMatch*) y proveedor (método *NombreProvMatch*) con valores de 3 y 2 respectivamente. El resultado global de estas comparaciones es almacenado en la variable *peso.otros.* Al final, cada pareja tendrá asociado un peso que es el que se tomará como medida de idoneidad para la petición cursada.

A continuación y tal como ya hemos comentado, se inserta esta pareja de forma ordenada, haciendo uso del método *ordenacion()* (figura 2), en la lista de parejas resultado. Cada elemento de la lista contendrá tres valores que se corresponden con las URI´s tanto de la petición como del anuncio, así como el peso de la pareja.

4 Implementación

El algoritmo fue implementado dentro de un sistema multiagente de descubrimiento de Servicios Web de información de tráfico, y fue desarrollado en el grupo Lisitt (Laboratorio Integrado de Sistemas Inteligentes y Tecnologías de la Información en Tráfico) del Instituto de Robótica de la Universidad de Valencia. Para su desarrollo se utilizaron los siguientes componentes:

- Como repositorio se utilizó Sesame [2], debido a que permite añadir y eliminar información escrita en RDF en los repositorios, y puede almacenar esta información en cualquier base de datos. Sesame soporta como lenguajes de consulta a RQL, RDQL y SeRQL (Sesame RDF Query Language) [16], para acceder al conocimiento. Permite la interoperabilidad con un razonador de lógica descriptiva, el cual debe ser otro de los participantes en cualquier sistema emparejador.

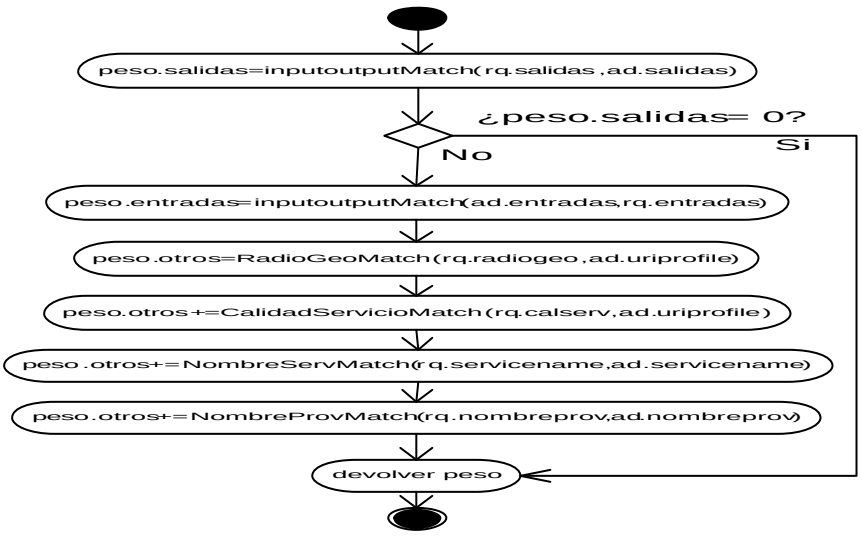

Fig. 4. Cálculo de pesos del método match

- Como razonador se utilizó BOR [3], 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 DAM+OIL (con algunas restricciones), y con ontologías escritas en la especificación OWL Lite. Además se puede incorporar a la aplicación Sesame, para dar soporte de ontologías DAM+OIL y poder realizar razonamiento y todas las funciones descritas en la información almacenada en los repositorios.
- Para dar soporte a DAML+OIL se utilizó el plugin de Sesame llamado SeBOR (Sesame+BOR) [14] que acepta modelos de datos semánticos de DAM+OIL en Sesame.
- El sistema de agentes que integra los diferentes componentes fue desarrollado utilizando la metodología FIPA y se implementó en lenguaje Java.

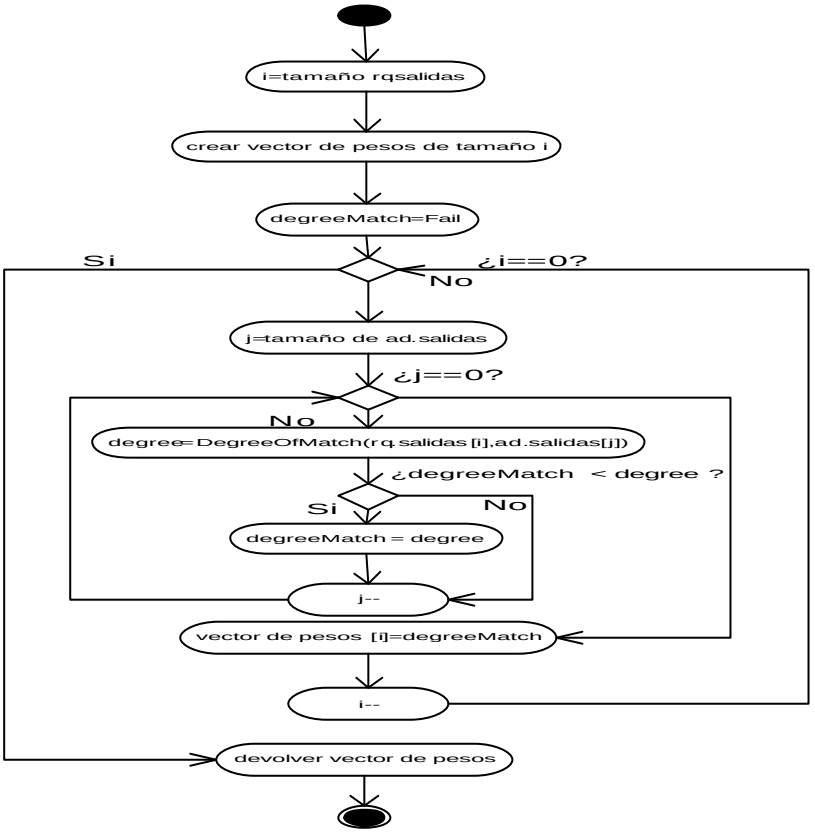

Fig. 5 Diagrama de actividades del método InputOutputMatch(), para cálculo de "peso.salidas"

4.1 Consultas para los grados de similitud en parámetros funcionales.

En las fases de diseño y posteriormente implementación se especificaron las consultas necesarias para emparejar cada uno de los grados de similitud en la fase de emparejamiento de parámetros funcionales, así como aquéllas utilizadas para comprobar el emparejamiento de los parámetros no funcionales que se emplean en el algoritmo.

A partir del estudio realizado de los lenguajes disponibles se utilizó SeRQL, debido a que era el que mejor se integraba con el razonador y repositorio seleccionado.

Dentro de las consultas implementadas para obtención de grados de similitud destaca la siguiente:

***ChermanoP*:** Esta consulta consiste en tomar las restricciones del concepto del cliente, para después navegar por las restricciones de los hermanos de esta clase, para encontrar restricciones que emparejen. En la Figura 6 puede ser observada la relación de este tipo y en la Figura 7 su implementación.

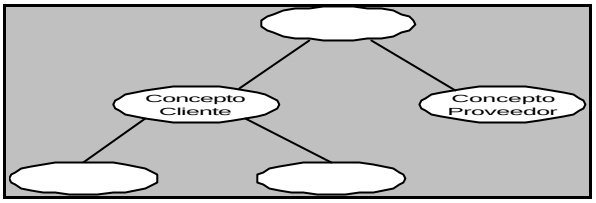

Fig. 6. Consulta por conceptos hermanos

```
Select distinct X, Y
From {X} <daml:subClassOf> {Padre},
     {Y} <daml:subClassOf> {Padre},
     {X} <rdfs:subClassOf> {Rest},
     {Y} <rdfs:subClassOf> {Rest2},
     {Rest} <rdf:type> {<daml:Restriction>},
     {Rest2} <rdf:type> {<daml:Restriction>},
     {Rest} <daml:onProperty> {Prop1},
     {Rest2} <daml:onProperty> {Prop2},
     {Rest} Z {A},
     {Rest2} Z {B}
Where X = <URI del concepto semántico del proveedor>
and Y = <URI del concepto semántico del cliente>
and Prop1=Prop2
and A = B
```

Fig. 7. Implementación de la consulta por conceptos hermanos

En el caso de que se obtenga como respuesta las URIs proporcionadas, se puede concluir que las clases son hermanas y coinciden en alguna de las restricciones.

4.2 Consultas para parámetros no funcionales

En relación con los parámetros no funcionales se implementaron consultas relacionadas con categoría de servicio, radio geográfico, calidad de servicio, nombre de servicio y nombre de proveedor. Por ejemplo, la consulta de categoría de servicio devuelve una lista de todas las URI's de los perfiles de Servicios Web que tienen como categoría de servicio la misma que pide el cliente, descartando así muchos de los perfiles que mantiene el repositorio. Esta lista de URI's de Servicios Web corresponde a los servicios que pasan a las siguientes fases del emparejamiento, liberando así al sistema de la carga de tener que aplicar el resto de filtros a todos los perfiles mantenidos en el repositorio y que no son relevantes para lo que busca el cliente, ahorrando mucho tiempo de proceso al sistema.

5 Pruebas funcionales

Se han realizado diferentes pruebas en las que se ha comprobado la mejor eficiencia del algoritmo de emparejamiento propuesto frente al algoritmo de Paolucci et al[4]. En estas pruebas se mostrará que, gracias a la mejor utilización de los parámetros de la clase *Profile* de la ontología de OWL-S / DAML-S, el algoritmo propuesto ofrece mejores resultados, en cuanto a precisión en las búsquedas de servicios, que el algoritmo proporcionado por Paolucci et al., y a su vez, los tiempos

[4] Implementación del pseudocódigo publicado en [13].

de respuesta no sufren cambios de consideración salvo en situaciones extremas (casos peor y mejor). Para realizar las pruebas se construyeron diferentes perfiles de servicios muy parecidos entre sí, y ante diferentes peticiones del cliente se estudiaron los distintos resultados dados por los dos algoritmos tras su ejecución.

En la Figura 8, se puede observar dónde se encuentran localizados en la taxonomía los principales conceptos utilizados para especificar los distintos parámetros de los servicios.

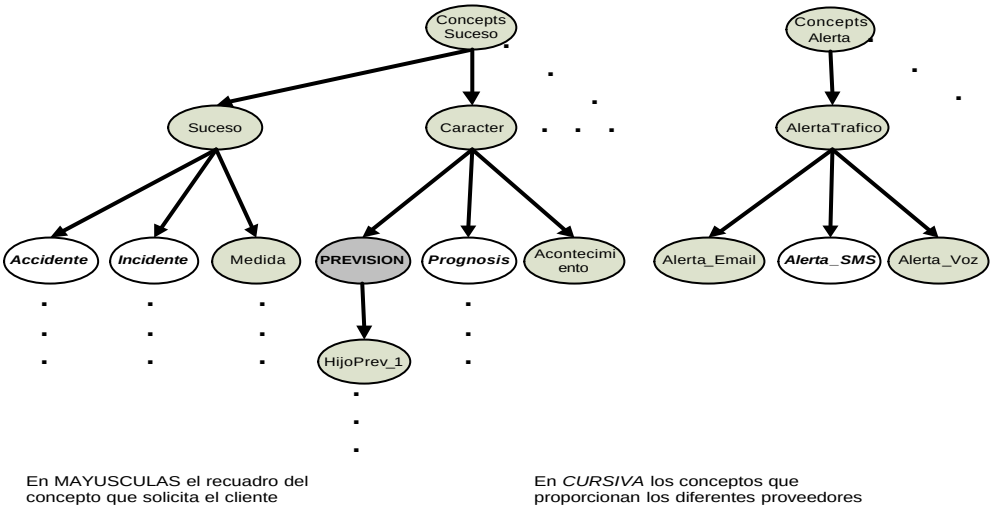

Fig. 8 Jerarquía de conceptos.

5.1 Definición de los casos de prueba.

Para comparar los dos algoritmos, a continuación se explica en qué ha consistido uno de los casos de prueba denominado: ***"Relaciones de parentesco fraternales".***

Paolucci et al. no distinguen el grado *fraternal*, por lo que las pruebas han ido dirigidas a comprobar que aunque el servicio que más se asemeja al buscado posee parámetros que pertenecen a esta relación, el algoritmo estudiado será incapaz de obtener un resultado y por tanto, devolver un perfil correspondiente a un servicio. Se comprobará que el algoritmo propuesto sí que localiza un proveedor de un concepto hermano al solicitado por el cliente. En la tabla 1 se detallan los perfiles de servicios utilizados, especificando sus parámetros mediante conceptos pertenecientes a la jerarquía establecida (Figura 8)[5]. En dicha jerarquía es posible distinguir los diferentes tipos de sucesos o eventos de tráfico, clasificados en Accidentes, Incidentes y Medidas adoptadas por la entidad competente, así como también el carácter de estos: Previsiones de tráfico, Prognosis y Acontecimientos[6]. También aparecen de forma explícita los tres tipos de servicios de alertas o notificaciones principales (por correo electrónico, SMS y uso de la voz por vía telefónica).

InfovozProfile	TraficoCatProfile
Inputs: Usuario: *XMLSchema.xsd#string* Password: *XMLSchema.xsd#string*	**Inputs:** Incidencia: *Sucesos.daml#Tipo_Incidente* Ordenación: *TraficoCatProcess.daml#Orden*

[5] Por criterios de simplicidad y claridad, no han sido expuestos todos los conceptos y jerarquías utilizadas en el emparejamiento de servicios, por ejemplo, las jerarquías de conceptos de topónimos (Geografía) o Vías.

[6] Previsiones de tráfico son sucesos desconocidos por anticipado, pero que posiblemente ocurrirán con un alto grado de probabilidad. Prognosis son sucesos conocidos por anticipado que se pueden planificar. Acontecimientos son sucesos materializados de forma no prevista.

Carretera: *Vias.daml#Via* Kminicio: *XMLSchema.xsd#decimal* Kmfin: *XMLSchema.xsd#decimal* Sentido: *Relaciones_sucesos.daml#Sentido* Dias: *Time.daml#DayofWeek* Hora_ini: *Time.daml#Time* Hora_fin: *Time.daml#Time* **Outputs:** Confirmacion: *XMLSchema.xsd#String* Alerta: *Cruta2.daml#Alerta_SMS* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Excelente*	Demarcacion*: Geografia.daml#Provincia* Comarca: *Geografia.daml#Comarca* **Outputs:***Incidenciaout: Sucesos.daml#Incidente* **Radio Geográfico:** *Geografia.daml#Catalunya* **Calidad de Servicio:** *Concepts.daml#Excelente*
TraficoProfile **Inputs:** Incidencia: *Sucesos.daml#Tipo_Incidente* Provincia: *Geografia.daml#Provincia* Comunidades: *Geografia.daml#Comunidad_Autonoma* MostrarInc: *Concepts.daml#MostrarType* **Outputs:** Incidencia: *Sucesos.daml#Incidente* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Excelente*	PrognosisProfile **Inputs:** Dia: *Time.daml#Date* **Outputs:** Prognosis: *Sucesos.daml#Prognosis* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*
FechaAccidenteProfile **Inputs:** Fecha: *Time.daml#Date* **Outputs:** Salida: *Sucesos.daml#Accidente* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*	

Tabla 1: Perfiles de servicios de proveedores usados en la prueba.

En la tabla 2 se puede observar el escenario propuesto para este caso de prueba (perfil requerido por el cliente y perfiles de los servicios ofertados por los proveedores), así como los resultados obtenidos tras la ejecución de los algoritmos.

CLIENTE	PROVEEDORES	PUNTUACIÓN (O/I/Otros)	
		Propuesto	Paolucci
ClientePrevision Profile **INPUTS:** Dia: *Time.daml#Date* **OUTPUTS:**Prevision: *Sucesos.daml#Prevision* **Radio Geográfico:** *Geografia.daml#Espanya* **Calidad de Servicio:** *Concepts.daml#Bueno*	InfovozProfile	0/0	0/0
	TraficoCatProfile	0/0	0/0
	TraficoProfile	0/0	0/0
	PrognosisProfile	**1/6**	0/0
	FechaAccidenteProfile	0/6	**0/6**

Tabla 2: Resultados escenario

Al no distinguir Paolucci et al. el grado fraternal o de hermanos, puede ocurrir que lo que solicite el cliente no se encuentre disponible mediante la búsqueda por antecedentes ni predecesores del concepto, por lo que en estos casos, es conveniente tener en cuenta en las taxonomías de conceptos, el grado de similitud entre nodos hermanos. En este caso de prueba, se observa su valor. Mientras que en nuestra propuesta el sistema ha dado como resultado el perfil correspondiente a *PrognosisProfil*e en el de Paolucci se elige el correspondiente a *FechaAccidenteProfile*.

5.2 Comparativa de los tiempos de respuesta.

Al igual que en las pruebas anteriores, hemos utilizado 2 prototipos, uno de ellos basado en nuestra propuesta de algoritmo de emparejamiento y el otro siguiendo el pseudocódigo propuesto en [13]. Las pruebas se han realizado variando el número de perfiles de servicios anunciados (1, 3, 5, 8, 10, 15, 20, 25, 30 y 33). El tiempo obtenido está mostrado en milisegundos.

Caso 1: Todos los anuncios disponibles pertenecen a la misma categoría de servicio.
Éste es el peor caso para el algoritmo propuesto. Esto es debido a que todos los perfiles de proveedores de servicios insertados en el repositorio pertenecen a la misma categoría de servicio que el servicio buscado por el cliente. En este caso el filtro de categoría de servicio no descarta perfiles, por lo que el emparejamiento se realiza en ambos prototipos con el mismo número de perfiles. Como nuestro método para calcular el grado de emparejamiento es más costoso debido a que hace más comparaciones, el prototipo basado en el algoritmo de Paolucci obtiene el resultado del emparejamiento en menos tiempo. Véase el gráfico de la figura 9.

Caso 2: El 20% de los anuncios pertenecen a la misma categoría de servicio buscada por el cliente.
Este caso muestra una situación que se aproxima más a la realidad. En ella no todos los perfiles del repositorio pertenecen a la categoría de servicio a la que pertenece el buscado por el cliente. En este caso hay un 20% de los perfiles que pertenecen a la misma categoría. Se puede observar mediante el gráfico de la figura 10 como nuestra propuesta obtiene tiempos muy similares a la de Paolucci et al.

Caso 3: Sólo hay 1 servicio que pertenezca a la categoría de servicio buscada por el cliente.
Éste es el mejor caso para nuestra propuesta, ya que sólo hay un perfil de servicio en el repositorio que coincida con la categoría buscada por el cliente. Mientras nuestra propuesta sólo hace el emparejamiento con este perfil el otro algoritmo debe comparar con todos los disponibles, por lo que nuestra propuesta obtienes tiempos considerablemente mejores. Obsérvese el gráfico de la Figura 11.

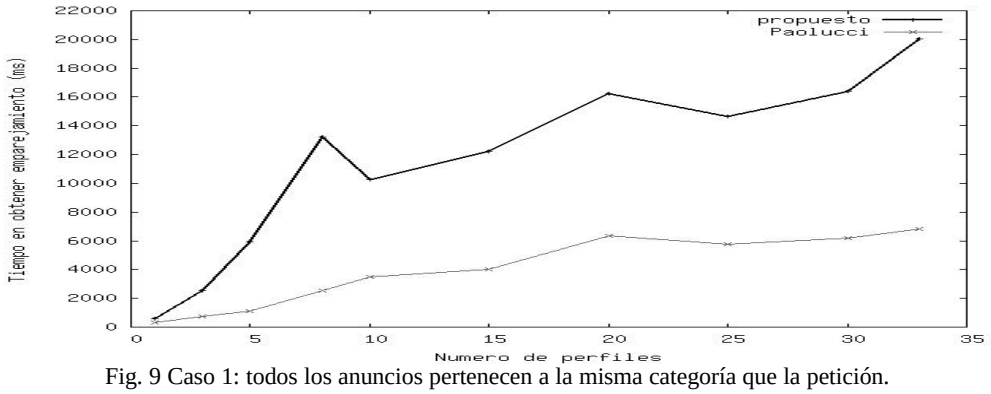

Fig. 9 Caso 1: todos los anuncios pertenecen a la misma categoría que la petición.

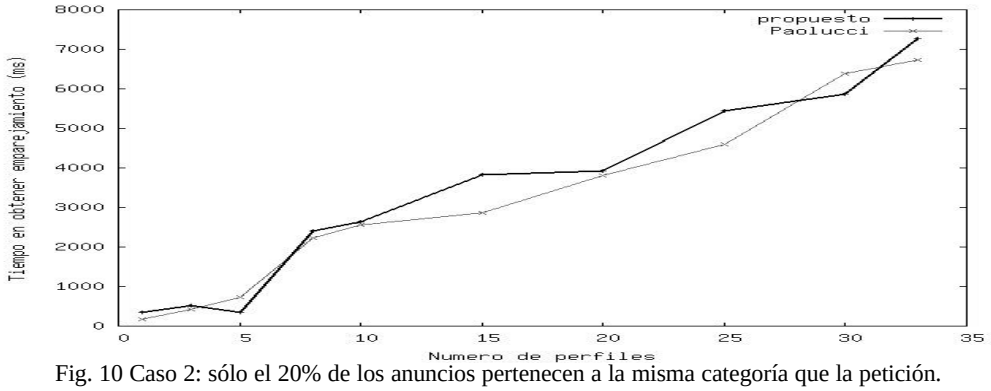

Fig. 10 Caso 2: sólo el 20% de los anuncios pertenecen a la misma categoría que la petición.

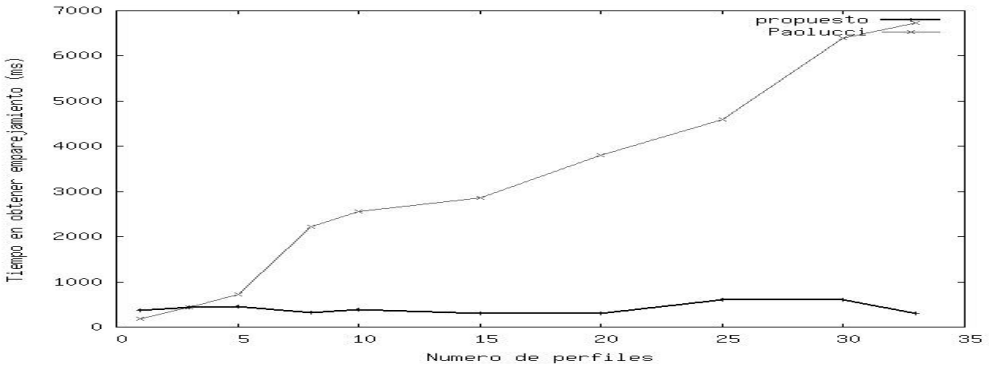

Fig. 11 Caso 3: sólo uno de los anuncios pertenece a la misma categoría.

6 Análisis de Resultados

A partir de las pruebas efectuadas se pudo observar como hecho más importante, que utilizando el algoritmo fue posible encontrar servicios basados en similitudes de tipo hermano, las cuales no habían sido estudiadas anteriormente. Aunque este tipo de similitud pudo no haberse tenido en cuenta debido a que en el caso de encontrarse similitudes entre nodos hermanos, posiblemente sería recomendable desde el punto de vista de diseño crear un nodo padre intermedio que agrupase las características comunes de los hijos, nuestro algoritmo las tiene en cuenta partiendo de la premisa de que es posible que este tipo de similitudes no hayan sido consideradas por el proveedor que definió semánticamente estas relaciones, bien sea por consideraciones de rendimiento (para evitar un grado de profundidad demasiado exhaustivo) o simplemente por no haber detectado este tipo de relaciones en la fase de diseño, con lo que el descubrimiento de servicios basado en algoritmos convencionales dejaría de producir resultados que están incluidos en el rango requerido de respuesta.

Por otra parte, la elección de ontologías en las que no hay conceptos con definiciones completas (condiciones necesarias y suficientes para ser miembro de una clase) puede dar lugar a este tipo de situaciones, ya que los razonadores serán incapaces de establecer clasificaciones de manera automática. En estos casos, el peso de la construcción de una buena y apropiada ontología recaerá en el desarrollador de ésta. Por el contrario, el diseño de ontologías con conceptos definidos permitirá solventar este tipo de problema aunque incrementará notablemente el coste computacional, por lo que la decisión de incorporar este grado de similitud se ve ampliamente respaldada.

Otras decisiones como el establecimiento de filtrados anteriores al algoritmo así como el uso de otros parámetros no funcionales, mejoraron considerablemente el coste y la elección del servicio más idóneo respectivamente.

Es importante destacar ciertas mejoras introducidas en el algoritmo que también supusieron un ahorro en el coste computacional como la finalización del proceso comparativo, con un determinado anuncio, cuando no hay emparejamientos de parámetros de salida, y la posibilidad de hacer uso del resto de parejas (petición-anuncio) almacenadas tras el resultado de la búsqueda, para establecer un sistema tolerante a fallos.

Tras las pruebas realizadas, podemos afirmar que el algoritmo propuesto, al contener muchas más consultas (comparaciones) debido a la inclusión de los nuevos grados y accesos al repositorio donde están las ontologías y los perfiles, tiene un mayor coste temporal, el cual es únicamente importante en casos extremos en los que exista un número muy elevado de anuncios de proveedores que pertenezcan a la misma categoría de servicio que el de la petición del cliente. Sin embargo, en situaciones próximas a la realidad, donde aproximadamente el 20 % de los anuncios

pertenecerán a la misma categoría de servicio, el coste temporal es similar al del algoritmo de Paolucci et al. e incluso lo mejora en escenarios donde el número de anuncios pertenecientes a la categoría del de la petición es muy bajo. Por ello, podemos concluir que se ha conseguido aumentar la precisión en las búsquedas sin perjudicar el coste temporal en situaciones consideradas normales.

7 Conclusiones

En este artículo se ha presentado un resumen del estado del arte de sistemas de emparejamiento y diferentes algoritmos integrados en ellos. El principal punto de investigación han sido los diferentes grados de emparejamiento que permitieran flexibilidad, uso de parámetros no funcionales así como los diversos filtros usados, algunos de los cuales no habían sido considerados en anteriores trabajos de otros autores. La principal contribución al mundo de los Servicios Web Semánticos fue no sólo reflejar el aspecto teórico, sino también el aspecto práctico mediante la construcción de un sistema capaz de hacer uso del algoritmo de emparejamiento propuesto.

En esta propuesta se decidió ampliar el rango de grados de similitud de anteriores propuestas, porque se consideraba que alguno de los grados de similitud definidos eran demasiado generales. Éste era el caso del grado "*exact*" definido por Paolucci et al., que incluye los casos en los que el concepto definido por el cliente en su petición es exactamente el mismo o es subclase del concepto que definió el proveedor en su perfil. En nuestro algoritmo distinguimos dos subgrados, uno, el de mayor valor, que será aquel en el que tanto proveedor y cliente utilizan el mismo concepto para definir un parámetro funcional, y un segundo subgrado que será aquél en el que el concepto semántico descrito por el cliente es subclase directa del que define el proveedor. Adicionalmente se ha descrito un nuevo grado de similitud no considerado por Paolucci et al, consistente en la relación entre "hermanos" que se puede dar con los dos conceptos dados. Este grado consiste en que dados dos conceptos, se les considera hermanos si ambos son subclase directa de otro concepto y poseen una restricción común sobre alguna de sus propiedades.

El algoritmo fue implementado como un componente dentro de un sistema multiagente para publicación y descubrimiento de servicios. Es independiente del lenguaje de especificación de ontologías que se utilice para describir, tanto las descripciones de los Servicios Web como las diferentes ontologías utilizadas para calcular el grado de similitud que se tenga. Por el momento, el sistema funciona únicamente con el lenguaje de especificación DAML+OIL para las ontologías de conceptos y DAML-S para la descripción de los Servicios Web. Se decidió no utilizar el lenguaje OWL como lenguaje de marcado semántico debido a que en el momento de iniciar la implementación del prototipo este lenguaje aún no era soportado por el razonador y repositorio seleccionados. Sin embargo, se han creado descripciones de los servicios tanto en DAML-S 0.9 como en OWL-S 1.0 porque se prevé que próximamente Sesame dé soporte a OWL, por lo que este sistema podrá migrarse aplicando el mismo algoritmo de emparejamiento desarrollado.

Referencias

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

[2] J. Broekstra; A. Kampman and F. Van Harmelen. Sesame: a Generic Architecture for Storing and Querying RDF and RDF Schema. En: 1st Int. Semantic Web Conference, Italy, 06- 2002.

[3] K. Simov and S. Jordanov. BOR: a Pragmatic DAML+OIL Reasoner; Deliverable 40, On-To-Knowledge project, Junio 2002.

[4] A. Ankolenkar et al. The DAML Services Coalition. DAML-S: Web Service Description for the semantic Web. En: The First International Semantic Web Conference (ISWC), 2002.

[5] Matchmaker by CMU. http://www-2.cs.cmu.edu/~softagents/daml_Mmaker/daml-s_matchmaker.htm Última visita 07-11- 2004.

[6] S. Colucci et al. Logic Based Approach to Web Services Discovery and Matchmaking. En: Modeling E-services Workshop at 5th International Conference on Electronic Commerce, (ICEC'03). Pittsburgh, October 3, 2003.
[7] J. Gonzalez-Castillo; D. Trastour and C. Bartolini. "Description Logics for MatchMaking of Services"; HP Technical Reports. October 30 th , 2001
[8] 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.
[9] 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.
[10] D. Martin, M. Paolucci and S. McIlraith. List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2002Mar/0009.html
[11] D. Martin. List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2003May/0028.html
[12] T. Di Noia et al. A System for Principled Matchmaking in an Electronic Marketplace. En: WWW2003.
[13] M. Paolucci et al. "Semantic Matching of Web Services Capabilities". In The First International Semantic Web Conference (ISWC), 2002.
[14] S. Jordanov and K. Simov, BOR: a Pragmatic DAML+OIL Reasoner, System Documentation Overview, Installation, Integration, http://www.ontotext.com/BOR, 21.11.2002
[15] T. R. Payne ; M. Paolucci and K. Sycara "Advertising and Matching DAML-S Service Descriptions". En: Semantic Web Working Symposium (SWWS), 2001.
[16] B.V. Aduna, Sirma AI Ltd. Chapter 6. The SeRQL query language, rev. 1.1 from User Guide for Sesame, http://www.openrdf.org/doc/users/ch06.html., Última visita noviembre 2004.
[17] K. Sycara et al. Dynamic Service Matchmaking Among Agents in Open Information Environments. En: Journal ACM SIGMOD Record, 1999.
[18] 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.
[19] I. Constantinescu, B. Faltings. World Wide Web Consortium Efficient Matchmaking and Directory Services. Technical Report No IC/2002/77, Noviembre, 2002.
[20] A. Zaremski and M. Wing. Signature matching: a Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology, 1995
[21] A. Zaremski and M. Wing. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology (TOSEM), 1997.
[22] T. R. Payne and C. Abela, List of Web Services. En: http://lists.w3.org/Archives/Public/www-ws/2002Jan/0008.html, Enero 2002.
[23] R. Ferraz; S. Labidi and B. Wanghon. "A semantic matching method for clustering traders in B2B Systems", 1er Latin American Web Congress (LA-WEB 2003), 0-7695-2058-8/03.