UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Tesis para optar al Título de
Magister en Informática

UN ENTORNO DE TRABAJO PARA LA MEDICION DE ALGORITMOS DE DESCUBRIMIENTO SEMÁNTICO DE SERVICIOS WEB

Autor: Gustavo Carlos Núñez Mori

Directores de Tesis: Fernando Carpani
Regina Motz

Montevideo, Uruguay
2012

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE INGENIERIA

ACTA DE DEFENSA

TESIS DE MAGISTER EN INFORMATICA

Fecha:

Lugar: Montevideo, Facultad de Ingeniería - Universidad de la República.

Plan de Estudio:

Aspirante: Gustavo Carlos Núñez Mori

Documento de Identidad: 1.412,406-2

Directores de Tesis: Fernando Carpani, Regina Motz.

Tribunal:

Los miembros del tribunal hacen constar que en el día de la fecha el **Ing. Gustavo Carlos Núñez Mori** ha sido APROBADO/REPROBADO en la defensa de su tesis de **Mágister en Informática** titulada: **"Un entorno de trabajo para la medición de algoritmos de descubrimiento semántico de servicios web"**.

La resolución del tribunal se fundamenta en los puntos detallados a continuación:

Para que conste,

Tabla de contenido

Índice del Figuras

Resumen

El presente trabajo introduce una herramienta de comparación de algoritmos de búsqueda de servicios web. Basado en la utilización de distintas herramientas y bibliotecas de desarrollo, se propone un entorno de trabajo que permite medir el desempeño de algoritmos de búsqueda semántica de servicios web de manera objetiva y reproducible de acuerdo a métricas predefinidas.

Se comienza introduciendo a la motivación de este trabajo basada en el incremento creciente del interés de la comunidad académica y de la industria en la Web semántica y en las definiciones e intentos de estandarización de las distintas representaciones de servicios Web. Se continúa con una breve descripción de los dos modelos fundamentales adoptados para estas representaciones como paso previo a la introducción de los algoritmos de búsqueda de servicios.

Como punto de partida para el estudio del conjunto de algoritmos de búsqueda de servicios web involucrados (algoritmos de descubrimiento de servicios web o *discovery*), se provee una clasificación de los mismos de acuerdo con sus propiedades observables en el contexto que nos concierne.

Con base en publicaciones y trabajos anteriores, y partiendo de la clasificación de algoritmos realizada, se determina un conjunto de

requerimientos básicos a cumplir por el ambiente de comparación, y se modela e implementa una arquitectura adecuada para la construcción del mismo, fundamentando la elección de las distintas herramientas utilizadas.-

Posteriormente se describen los elementos fundamentales de la arquitectura, sus características esenciales y las restricciones que se deben imponer a un algoritmo para poder ser medido y categorizado. Se comentan y ejemplifican los pasos necesarios para poder utilizar un algoritmo en este contexto.

Se aporta un ejemplo específico de funcionamiento de una implementación de un algoritmo de discovery, mostrando los resultados que arrojan las mediciones y la utilidad del entorno de trabajo desarrollado como aporte en la comparación objetiva de distintos algoritmos. El resultado final del trabajo es el desarrollo de un entorno de trabajo que permite la medición de algoritmos y una biblioteca (API) que debe utilizar un algoritmo de discovery para poder ser medido por el entorno de trabajo.

Como corolario se proponen nuevas líneas de trabajo futuro.-

Palabras Clave

Algoritmos, Matchmaking, Web semántica, Discovery, Interoperabilidad, Métricas, Ambiente, Librería.

1. Introducción

En los años recientes la evolución de las comunicaciones, Internet y la proliferación de datos disponibles en diversas áreas de conocimiento humano, han puesto de manifiesto la necesidad de mejorar los criterios de búsqueda de información y servicios para transformar esta información. Es así que con este crecimiento la tecnología de búsqueda y clasificación también ha evolucionado a nuevos paradigmas.

El procesamiento de información proveniente de distintas disciplinas introducen la necesidad de la definición de un conjunto de términos y relaciones entre los mismos que permita ordenar y especificar, mas allá de toda ambigüedad, los conceptos que aparecen en un área de estudio concreta. Es así que surgen estructuras como las ontologías, agrupando los conceptos y sus relaciones de un área de conocimiento o dominio de aplicación.

Las ontologías tienen un rol trascendente en la interoperabilidad de sistemas. Las mismas proveen los elementos para procesar de manera automática la información y el conocimiento compartido, y promueven además la capacidad de compartir información relevante entre sistemas heterogéneos.

En el campo de los servicios web es precisamente donde este fenómeno se manifiesta de manera acentuada. Las propias características de los

servicios web hacen de ellos elementos idóneos para construir sistemas de mayor complejidad mediante una orquestación que sea adecuada a un modelo de negocios.

Pero la construcción de sistemas multiplataforma, interoperables con otros, de creciente complejidad y basados en servicios web introduce al problema de la búsqueda de los servicios web adecuados para tales objetivos. Los mecanismos previos a la irrupción de la web semántica de los que se dispone para buscar y encontrar servicios web no cuentan con elementos para tratar con ontologías ni con toda la semántica asociada a la búsqueda en forma de metadatos. La presencia de éstos metadatos es el elemento novedoso que facilita la interoperabilidad de sistemas débilmente acoplados como los servicios web dotando a los mismos de un significado en un contexto.

Es precisamente en el campo de la búsqueda y clasificación de servicios web semánticos que se circunscribe este trabajo, proveyendo un marco de referencia para la medición objetiva de los mecanismos de búsqueda disponibles.

Sintetizando, se dispone de servicios web para procesar información y de ontologías para organizarla y compartirla. El tercer elemento necesario es un mecanismo de búsqueda de servicios web basado en la información que queremos procesar, en los resultados que queremos obtener y en el dominio de aplicación que nos interesa.

En el contexto de estas necesidades han surgido nuevos algoritmos de búsqueda que tienen en cuenta la semántica que se puede expresar en la publicación de un servicio web. Con estos algoritmos también ha surgido la necesidad de medir cual algoritmo es mas eficiente a partir de métricas específicas y en igualdad de condiciones.

Solo resta disponer de un entorno de trabajo que permita comparar estos nuevos algoritmos y que permita determinar que algoritmo es mejor de acuerdo a necesidades concretas expresadas en forma de métricas.

Que se requiere de un entorno de trabajo como el descrito:

1. Que permita ejecutar algoritmos de búsqueda en un ambiente controlado.
2. Que permita medir automáticamente algoritmos de búsqueda para poder compararlos de acuerdo a los mismos criterios.
3. Que permita ejecutar distintos algoritmos en los mismos escenarios de búsqueda.
4. Que permita reproducir los resultados de medición obtenidos.
5. Que permita administrar publicaciones de servicios web y ontologías.
6. Que permita construir escenarios de búsqueda de publicaciones de servicios web.

7. Que permita administrar distintos algoritmos para poder asociarlos a voluntad a los escenarios escogidos.

El objetivo de este estudio es la construcción de un entorno o marco de trabajo para la medición y comparación de algoritmos de descubrimiento semántico de servicios web.

Para lograr este objetivo se procederá de acuerdo al siguiente enfoque:

1. Estudio y categorización de algoritmos existentes.
2. Estudio de representaciones semánticas de servicios web.
3. Búsqueda de herramientas que permitan una implementación de un ambiente como el descrito y que a su vez permitan manipular representaciones semánticas de servicios web en función de las necesidades básicas encontradas en 1.
4. Validación con la construcción de un algoritmo designado.

2. Conceptos Preliminares

2.1. Búsqueda y contexto

La creciente cantidad de información disponible en la web señala la necesidad de búsquedas eficientes. La disponibilidad de distintos tipos de recursos como imágenes, video, servicios web y otros trae aparejada la necesidad de enriquecer la búsqueda con datos adicionales acerca de los datos buscados. Los procedimientos tradicionales de búsqueda han estado basados en la similitud o distancia (una métrica) dentro de un espacio de búsqueda.

Estos procedimientos de búsqueda no siempre han demostrado ser suficientes cuando las métricas asociadas no tienen en cuenta el contexto y su dominio de aplicación.

A modo de ejemplo considérese el caso de los sinónimos. Asúmase que se desea conocer a que nacionalidad u origen corresponde el gentilicio "*canario*"[1]. Un buscador público dará las siguientes respuestas a esta pregunta (figura 1):

1 Dícese de los nativos de las Islas Canarias

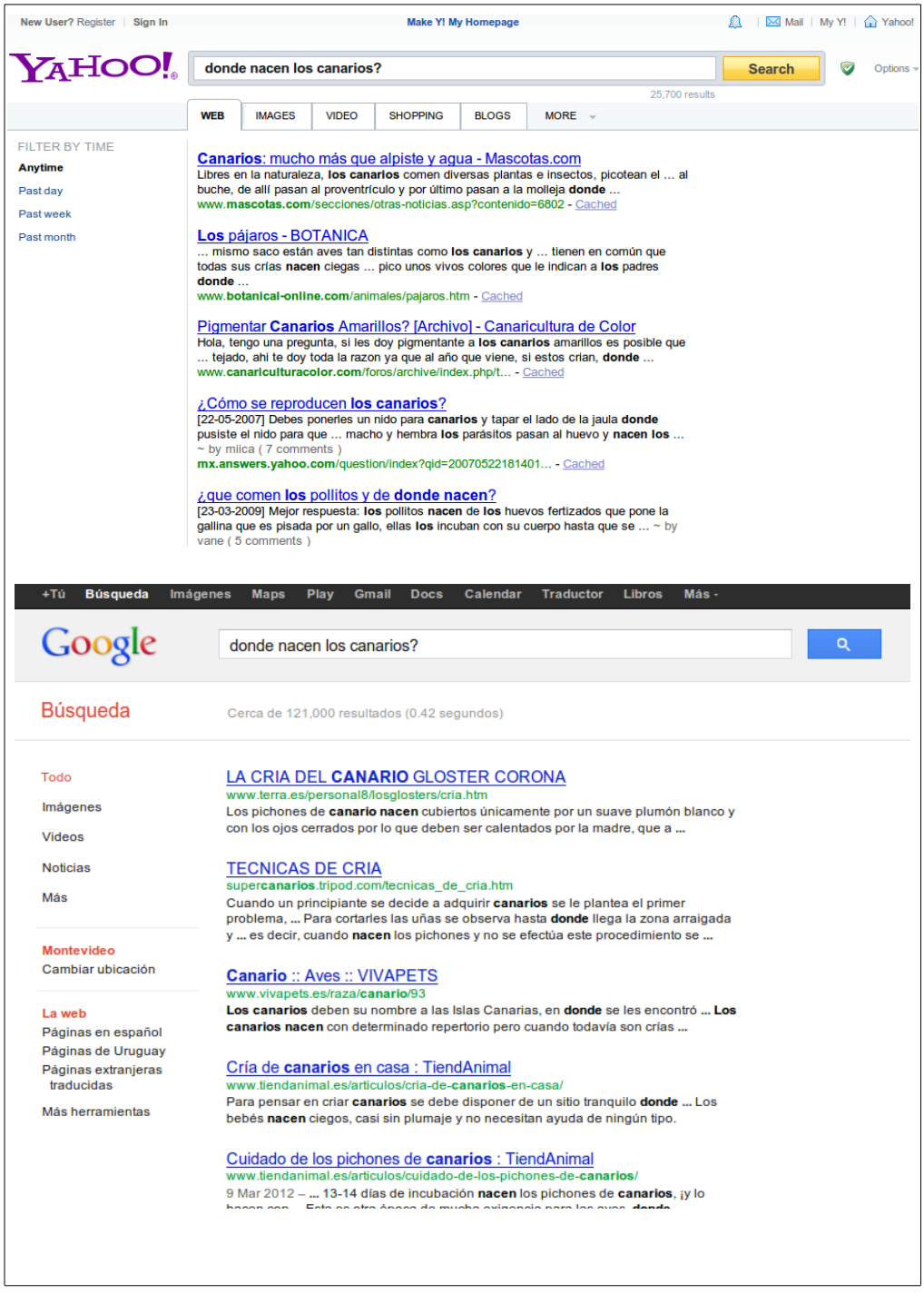


Figura 1: Búsqueda y homónimos.

La incorporación del contexto a estos mecanismos es un cambio que se comienza a dar en la industria, tal como se puede apreciar en [26] y [27].

Una manera de hacer incidir el dominio de aplicación al procedimiento de búsqueda es incluir la semántica asociada al concepto buscado. Es así que ya no nos referimos a términos de búsqueda sino a *conceptos* de búsqueda [27][2].

Las representaciones semánticas de conceptos dan lugar a estructuras de conceptos ligadas entre sí por relaciones que tienen sentido en un contexto determinado. Se introduce así la utilización de ontologías de dominios de búsqueda.

A partir de ontologías pertenecientes a distintas disciplinas, y operando con ellas, se pueden construir nuevas ontologías. El estudio de las herramientas y técnicas formales de operaciones con ontologías cae fuera de nuestro trabajo.

2.2. Servicios Web

Entre los recursos mas importantes pasibles de ser obtenidos en la red se encuentran aquellos que son provistos por los servicios.

Por servicios entendemos no solamente información estática que se obtiene a través de una URL sino agentes que proveen información específica a partir de parámetros que se le proporcionan y que le permiten

2 "*Introducing the Knowledge Graph: things, not strings*"

ejecutar acciones específicas para lo que fueron diseñados, eventualmente proporcionando alguna respuesta.

Un ejemplo simple puede proporcionarlo un portal de Internet que gestiona viajes u otros servicios asociados.

Asúmase que un usuario desea contratar un viaje de ida y vuelta a un destino específico. Podemos añadir restricciones al viaje, como la fecha de salida y retorno, que sea un vuelo directo y que se trate del vuelo mas económico. Supóngase además que se requiere también reservar la estadía en un hotel de determinada categoría por el mismo período y que además se desea alquilar un automóvil para usufructuar durante la estadía.

El portal referido (un sistema computacional) puede requerir de servicios provistos por otros sistemas computacionales, como por ejemplo:

- Un sistema global de reservas de vuelos, que aceptará como parámetros de entrada nuestras demandas de vuelo (origen, destino, fechas de salida y vuelta, precios aceptables, nuestro nombre y numero de tarjeta de crédito). Este sistema deberá devolver un boleto electrónico de avión emitido a nuestro nombre con los servicios requeridos.
- Un sistema de reserva de hoteles que aceptará como parámetros nuestra fecha de ingreso, la duración de nuestra estadía, una

expresión de las características del alojamiento que buscamos, y un número de tarjeta de crédito. Este sistema devolverá un número de voucher para presentar a nuestra llegada a destino.

- Un sistema de alquiler de automóviles, que requerirá nuevamente un número de tarjeta de crédito y el tipo de vehículo requerido. Este sistema deberá devolver un boleto de reserva.

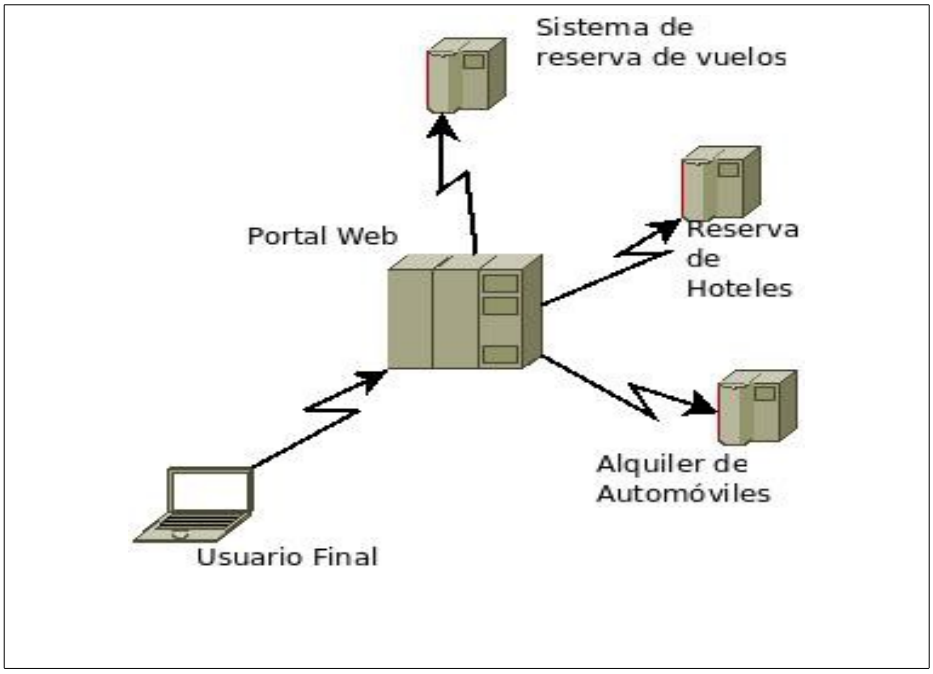


Figura 2: Un portal de viajes trabajando con servicios web.

En la figura 2 observamos que el portal *consume* información generada por otros agentes de software.

Frente a estos agentes se comporta como un *consumidor*, en

contraposición con el rol *productor* de los otros agentes de la figura (sistema de reservas de vuelos, sistema de reservas de hoteles, sistema de alquiler de automóviles) que de algún modo han *publicado* oportunamente en algún lugar que ellos son capaces de proveer esos servicios buscados.

Mas formalmente, según W3C [1] y sin perjuicio de otras definiciones, un web service puede ser definido como "un sistema de software designado para soportar interoperabilidad entre máquinas a través de una red". Según [2], "a diferencia de los sistemas distribuidos existentes los Web Services están adaptados a la web."

Nótese que estas definiciones abarcan desde simples páginas web estáticas (que no son objeto de nuestro estudio) hasta sistemas complejos basados en arquitecturas orientadas a servicios.

De la definición surge además la interoperabilidad de sistemas distribuidos como un elemento clave. Entre las características de los servicios web se destaca su típicamente débil acoplamiento, y que deben ser publicados para su consumo.

Utilizando web services conjuntamente con todas sus tecnologías asociadas (notablemente SOAP, XML, WSDL y UDDI [1], [2], [3], [4], [5], [6] interactuando en distintos niveles) se pueden construir sistemáticamente sistemas interoperables distribuidos a través de la red.

La información generada por un Web Service puede ser consumida por humanos o por otros sistemas de información, siendo este último aspecto el que permite construir aplicaciones distribuidas.

El estudio específico de servicios web cae fuera del alcance del presente trabajo, pudiéndose profundizar en [1] y [2]

2.3. Búsqueda y servicios

Hasta el advenimiento de la Web semántica se ha dispuesto de mecanismos de búsqueda de servicios como *UDDI,* donde precisamente la semántica asociada al dominio de aplicación queda en un segundo plano o no es tomada en cuenta.

Al exponer un servicio web para su búsqueda, se proporcionan detalles de punto de acceso para que los clientes hagan uso de la funcionalidad ofrecida a través de una interfaz. Los requisitos de formatos, comunicaciones, protocolos son también publicados para que un cliente tenga conocimiento de como invocar un servicio.

Para ejemplificar los distintos escenarios e interacciones podemos considerar algunos problemas habituales de la vida diaria.

Sea un sistema de control de la nómina de personal de una empresa que tiene que interactuar con un sistema bancario. Supongamos que los haberes de un empleado a fin de mes se conozcan con el nombre de “haberes” para el sistema de control de nómina de personal, mientras que para el sistema bancario el mismo concepto reciba el nombre de “crédito por remuneración”. En este caso hemos encontrado un concepto compartido entre dos áreas de conocimiento con distintas denominaciones (sinónimos). Por otro lado, y partiendo del mismo ejemplo considérese la suma de créditos y débitos por honorarios de un

empleado (haberes, adelantos recibidos, préstamos otorgados por el empleador, etc.) que para el sistema de control de nómina de personal puede recibir el nombre genérico de “balance”. El mismo nombre de “balance” puede significar otro concepto distinto para el sistema bancario, que incluya por ejemplo débitos automáticos generados por servicios básicos (consumo de electricidad, otros débitos bancarios).

Hemos encontrado entonces un término que describe dos conceptos distintos en los dos sistemas (homónimos). La realidad puede ser incluso bastante mas compleja, donde no exista una relación tan trivial entre vocablos y conceptos.

Para nuestro cerebro eliminar esta ambigüedad es un proceso bastante intuitivo, porque en general contamos con conocimiento de los dominios de aplicación suficientes como para no confundir estos conceptos. Sin embargo, los sistemas computacionales que queremos integrar no necesariamente disponen de ese conocimiento, ni de una representación del mismo, y por lo tanto, para poder interoperar necesitan que se les proporcione las herramientas adecuadas para poder organizar y relacionar estos conceptos.

Los sistemas que automatizan la solución de problemas en distintas áreas de conocimiento podrán interoperar si manejan conceptos en común. Esta noción implica la definición de operaciones de ontologías (intersecciones, uniones, otras) y se extiende hasta la interoperabilidad de los distintos

sistemas.

2.4. La web Semántica

Para W3C [6][3] "es la representación abstracta de datos de la Web, basada en el estándar de RDF y otros estándares a ser definidos, siendo desarrollado por W3C en colaboración con un gran número de investigadores y socios de la industria.“

El concepto nuevo es el de la semántica, los datos ahora tienen un sentido específico vinculado a un contexto y se relacionan entre ellos mediante metadatos descriptivos, que necesariamente acompañan al dato.

La existencia de un contexto donde el dato adquiere un significado introduce a la utilización de ontologías, donde los conceptos y su significado se relacionan con otros conceptos.

Partiendo del ejemplo anterior del portal de Internet que gestiona viajes, consideremos nuevamente los requerimientos del usuario final. Al realizar la reserva de un hotel el usuario final puede especificar la categoría deseada de su alojamiento, por ejemplo, en *estrellas*. Las estrellas componen un mecanismo ampliamente difundido de clasificación de

3 Del original: **“**The Semantic Web is the abstract representation of data on the World Wide Web, based on the RDF standards and other standards to be defined. It is being developed by the W3C, in collaboration with a large number of researchers and industrial partners.
Basic Idea: "The Semantic Web brings to the Web the idea of having data defined and linked in a way that it can be used for more effective discovery, automation, integration, and reuse across various applications."

hospedajes, pero no es el único, existen otros (*diamantes*, clasificación por clases, clasificación por códigos de letras, etc.). La correspondencia entre los distintos sistemas de clasificación no es necesariamente trivial.

El sistema de reservas hoteleras deberá entonces poder lidiar con esta heterogeneidad, siendo necesario para ello un conjunto de relaciones entre los distintos conceptos en las escalas de clasificación.

Aquí es donde se introduce la semántica, es decir, el significado de los distintos conceptos en el contexto adecuado (el contexto o dominio de la hotelería en nuestro caso).

El conjunto de conceptos intervinientes en las distintas clasificaciones, las relaciones entre ellos, así como las reglas que sigue su clasificación y combinación formarán parte de una ontología.

Las ontologías no son meramente repositorios de conceptos y relaciones. Entre las principales funciones de las ontologías está representar conocimiento de áreas específicas, permitir a distintos agentes interactuar con el conocimiento y eventualmente permitir inferir nuevas relaciones generando, por lo tanto, más conocimiento.

En la siguiente representación gráfica se pueden apreciar de manera jerárquica los conceptos fundamentales de la mencionada ontología y algunas de las relaciones que los vinculan.

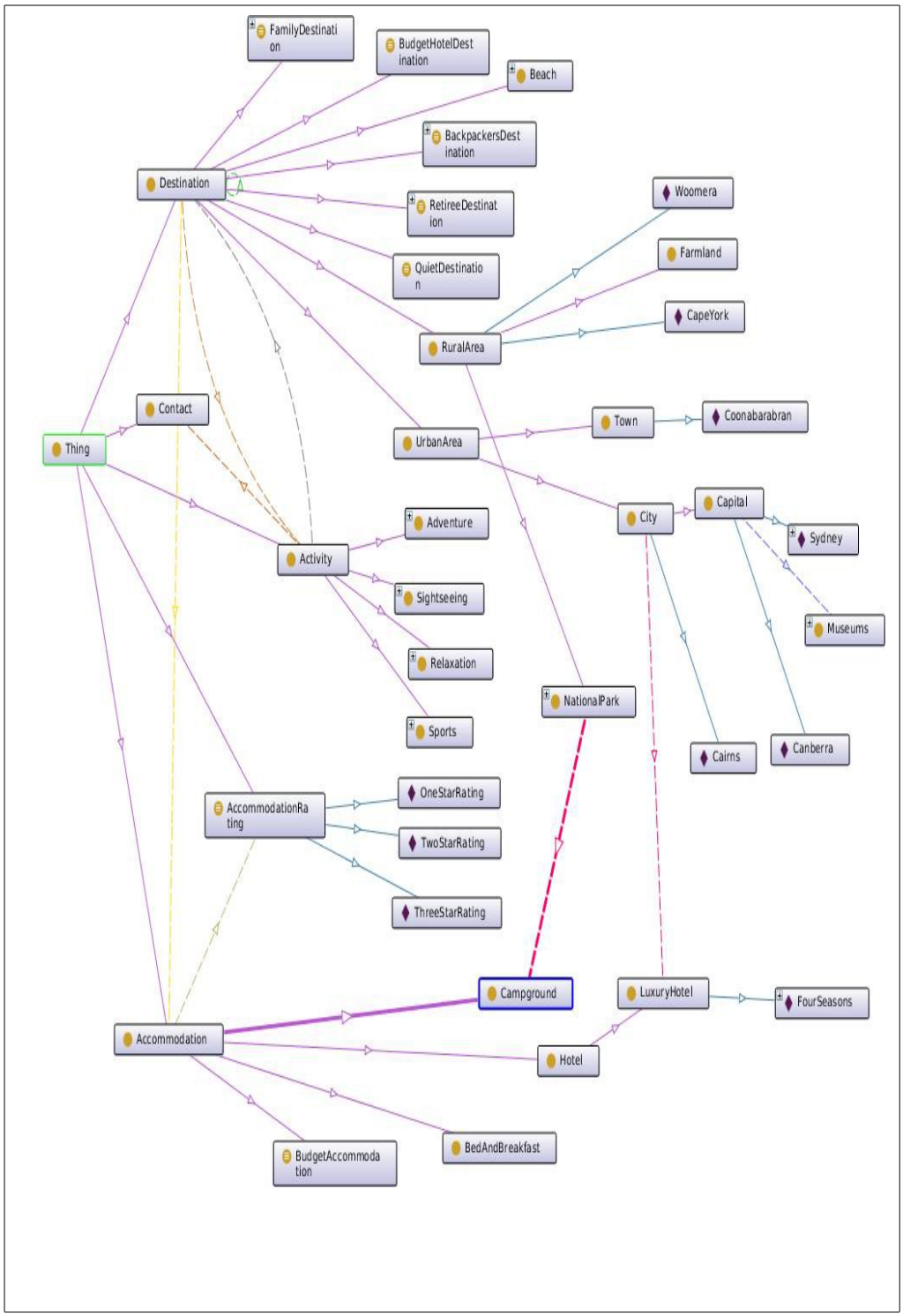


Figura 3: Representación gráfica de una ontología

Referencias a la figura 3:

HasSubclass (color violeta) Es una relación que indica que una clase tiene una especialización. En la figura anterior se pueden apreciar como ejemplos de esta relación:

Destination – HasSubclass--> Beach (lo que también se puede leer como "la playa es un destino")

Activity– HasSubclass--> Adventure

Accomodation – HasSubclass--> Hotel

HasAccomodation (color amarillo) Es la relación que vincula a las clases *Destination* y *Accomodation.*

hasIndividual (color celeste). Relación que indica la pertenencia de individuos a una clase. Ejemplo:

LuxuryHotel – HasIndividual--> FourSeasons

Los individuos clasificados en esta ontología se encuentran representados en la figura por diamantes.

Los conceptos que se encuentran al tope de la jerarquía de clases son *Destination, Contact, Activity, AccomodationRating* y *Accomodation*. A partir del desarrollo de estos conceptos y las relaciones que se detallan en la figura se expresa el conocimiento del área del dominio.

Todos los conceptos agrupados por relaciones, y expresados en un

lenguaje adecuado a nuestras necesidades conforman una ontología en nuestro dominio de aplicación.-

Mas formalmente, de acuerdo con [22], una ontología es "una organización rigurosa y exhaustiva de algún área de conocimiento que es usualmente jerárquica y que contiene todas las entidades relevantes y sus relaciones". Según [23] una ontología es una "especificación explícita de una conceptualización", y según [24] "una ontología define un conjunto de primitivas de representación con las cuales se modela un dominio de conocimientos o de discurso. Las primitivas de representación son típicamente clases (o grupos), atributos (o propiedades), y las relaciones entre ellas (o las relaciones entre los miembros de la clase). Las definiciones de las primitivas de representación incluyen información sobre su significado y las limitaciones en su aplicación lógicamente consistente". Nótese que las definiciones anteriores introducen las ontologías como mecanismos de especificación y comunicación de conocimiento.

Existen ontologías en uso en distintas áreas de conocimiento, entre ellas: Gene Ontology en genética[4], SNOMED[5] y GALEN[6] en Medicina, Dublin Core para publicaciones[7], FOAF usada redes sociales, LKIF Core de uso

4 Genética: Gene Ontology, accesible en http://www.geneontology.org/
5 Medicina: SNOMED, accesible a través de http://www.ihtsdo.org/, International Health Terminology Standards Development Organisation
6 Medicina: GALEN, disponible en http://www.daml.org/ontologies/400
7 Publicaciones: Dublin Core, disponible en http://www.cs.umd.edu/projects/plus/SHOE/onts/dublin.html

legal[8], ONTOCape en el campo de procesos de Ingeniería Química[9], Fynance Ontology en el área financiera[10], solo por citar algunos ejemplos.

El impacto de la utilización de ontologías es el siguiente:

- Necesidad de desarrollo de un lenguaje y de bibliotecas para habilitar el consumo de las ontologías por parte de los sistemas de información.
- Introducción de semántica en la descripción de servicios web, dando paso a los Servicios Web Semánticos, capaces de ser consumidos por agentes inteligentes.
- Razonadores y motores de reglas.
- Introducción de nuevos algoritmos para la búsqueda *semántica* de servicios web.

El primer punto implica la necesidad de un lenguaje para la descripción de ontologías a través de la web (*OWL, Ontology Web Languaje*). OWL está fuertemente orientado a la interoperabilidad, al procesamiento e integración de distintas fuentes de información, de acuerdo con [7], “OWL es diseñado para la utilización de aplicaciones que necesitan procesar el contenido de la información, en lugar de simplemente presentar la información a humanos“. Se requiere además de software que pueda

8 Legal: disponible en http://ontology.leibnizcenter.org/trac/wiki/LKIFCore
9 Ingeniería: ONTOCape, accesible desde http://www.avt.rwth-aachen.de/AVT/index.php?id=730&L=1
10 Finanzas: Fynance Ontology, accesible en http://www.fadyart.com/

manejar este lenguaje y sus variantes de distinto grado de expresividad (OWL-Lite, OWL-DL y OWL-full) o extensiones de reglas derivadas de este lenguaje (SWRL). Material adicional sobre la evolución de los lenguajes de ontologías hasta llegar a OWL y SWRL se puede hallar en [8] y [9].

Entre las distintas herramientas que han surgido destacan, por citar solo algunas, *Jena* y *Protegè.* La primera es una biblioteca completa provista de mecanismos para manejo de ontologías, razonadores y consultas sobre descripciones semánticas desarrollada por HP Labs, la segunda es un editor de ontologías construido sobre Jena, desarrollado por la Universidad de Stanford.

En cuanto al segundo punto, introducción de semántica en la descripción de servicios web, también fue necesaria la introducción de nuevos lenguajes descriptivos para los mismos, notablemente WSMO (descrito en [10], [11], [17]) y OWL-S (tratado en [12]).

En lo relativo a motores de reglas y razonadores, surgen de la necesidad de inferir y clasificar en función de las relaciones, taxonomía, demás atributos de los conceptos existentes en una ontología. Muchos de ellos soportan un protocolo de mensajería (DIG) que permite comunicarse con ellos a través de mensajes enviados por los demás componentes.

Finalmente, se hizo necesaria una nueva generación de algoritmos de

búsqueda que incluyera los elementos de búsqueda semántica, y por lo tanto, capaces de manejar descripciones semánticas de web services y ontologías de dominio relacionadas con los mismos. Denominaremos a estos algoritmos *semánticos.*

2.5. Representación semántica de Web Services

Las ontologías que se presentan a continuación intentan proporcionar un marco conceptual para la descripción semántica de servicios Web.

Para lograr esto es necesario describir el contenido de la Web mediante formalizaciones consensuadas del conocimiento publicado en la misma, a esto último se hace referencia con el término “ontología” [11], [17].

Según [17] “las tecnologías corrientes como SOAP, WSDL y UDDI, agrupadas bajo el término “Servicios Web” resuelven parcialmente el problema de la integración, ya que operan a un nivel puramente sintáctico, pero se necesita semántica para hacer posible el objetivo” (lograr que los datos publicados en la Web sean procesables por computadora). “Lo que se conoce como Servicios Web Semánticos (SWS) es la combinación de tecnología de Web Semántica y servicios Web. Mediante la utilización de ontologías para proporcionar el modelo de datos semántico para las tecnologías de Servicios Web se logran descripciones de los Servicios Web procesables por computadora”.

En esta dirección, existen dos marcadas tendencias para la formalización

de la descripción semántica de servicios web, OWL-S y WSMO.

2.5.1. OWL-S

De acuerdo con [12] es una Ontología de servicios Web basada en OWL, que proporciona a los proveedores de servicios un conjunto de construcciones para describir propiedades y funcionalidad de servicios web de manera no ambigua e interpretable por un agente no humano.

Está diseñada para permitir a agentes de software descubrir, invocar, componer y monitorizar los recursos de la red ofrecidos por servicios que tengan determinadas propiedades buscadas, con un alto grado de automatización.

Las tareas que OWL-S intenta facilitar son:

1. Descubrimiento automático. Se busca la posibilidad de localización de servicios web que proveen determinado tipo de servicio, adhiriendo a un conjunto de restricciones establecidas por el consumidor.
2. Invocación de un Web Service. Es la posibilidad de invocación de un servicio web por medio de un agente, suministrando únicamente una descripción declarativa del servicio, por oposición al hecho cuando el agente es pre programado para invocar un servicio particular.
3. Composición automática de servicios e interoperabilidad. Involucra

la selección, composición e interoperación de servicios web de manera de poder realizar una tarea mas compleja.

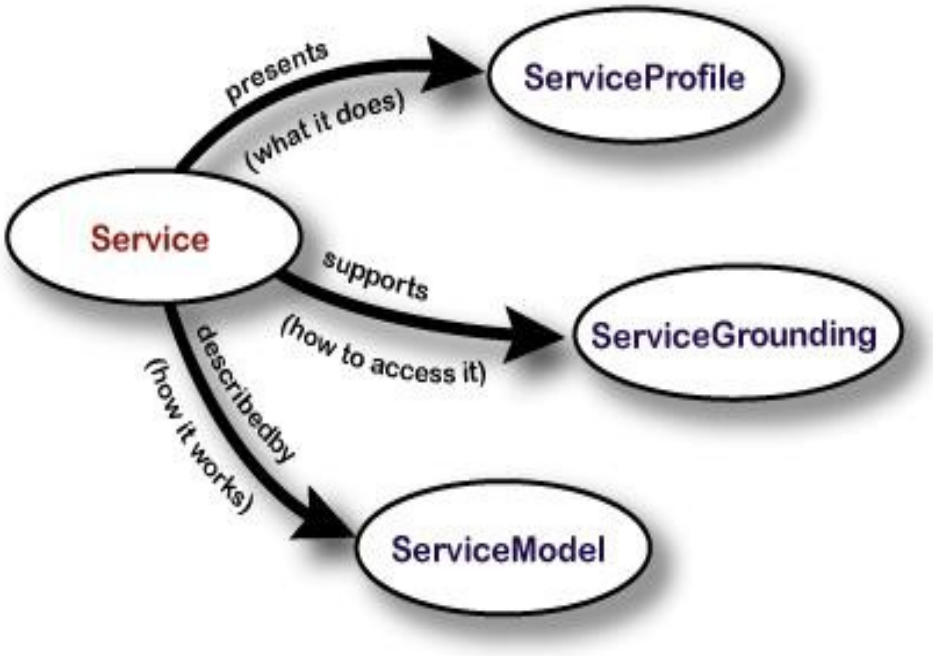


Figura 4: OWL-S, según [12]

Como se describe en la figura anterior, OWL-S está compuesta por tres partes fundamentales:

- *Service Profile*
- *Process Model*
- *Grounding*

2.5.1.1. Service Profile

Un service profile describe un servicio en función de tres tipos básicos de información:

- Que organización provee el servicio. Es información de contacto de la organización que provee el servicio (por ejemplo una dirección de correo de un operador que administra el servicio).
- Que función provee el servicio. Es el conjunto de transformaciones que el servicio es capaz de realizar. Específicamente describe las entradas requeridas por el servicio y las salidas generadas. También describe precondiciones requeridas por el servicio y los resultados esperables por la ejecución del mismo.
- Propiedades adicionales. Esta compuesto por distintos tipos de información, por ejemplo: *categoría de servicio*, que implica la pertenencia a un sistema de clasificación o categorización. *Calificación de calidad* así como existen servicios de excelente calidad existen de mala calidad incluso maliciosos, es así que un cliente puede querer requerir alguna calificación de calidad provista por un sistema especializado de calificación. Es responsabilidad del consumidor utilizar esta información y eventualmente verificar su exactitud. El tercer tipo de información provista aquí es una lista de parámetros de servicio, como por ejemplo máximo tiempo de respuesta, disponibilidad geográfica, etc.

2.5.1.2. Process Model

Según [12], para proveer una perspectiva detallada de como interactuar con un servicio es posible visualizarlo como un proceso[11]. Es importante destacar que un proceso no es un programa a ser ejecutado, sino una especificación de las maneras que un cliente puede interactuar con un servicio o conjunto de servicios relacionados.

Un proceso *atómico* es la especificación de un servicio que espera un mensaje y retorna un mensaje (mas allá de la complejidad de los mensajes). Un proceso *compuesto* es aquel que mantiene alguna clase de estado, cada mensaje que envía el cliente hace avanzar a través del proceso.

Un proceso puede tener un numero arbitrario de entradas (incluso ninguna), y un numero arbitrario de salidas. Las primeras representan la información que el proceso debe conocer para poder producir algún resultado, mientras que las últimas son la información que el proceso devuelve al consumidor o cliente. Puede haber además un conjunto de precondiciones, que deben ser alcanzadas para que el proceso sea satisfactoriamente invocado.

Finalmente el proceso puede tener cualquier numero de efectos. Las salidas y los efectos pueden depender de condiciones que deben ser verdaderas en el momento de la invocación.

---

11 OWL-S 1,1 define "*Process*" como subclase de "*ServiceModel"* [12]

Decimos que obtenemos un *resultado* cuando se satisface un numero de condiciones y se produce un conjunto de salidas y efectos[12].

2.5.1.3. Grounding

Con este termino se conoce a la especificación de los detalles de cómo acceder a un servicio, detalles que tienen que ver fundamentalmente con protocolos y formatos de mensajes. Se puede concebir como un *mapeo* de una especificación abstracta a una concreta de aquellos elementos descriptivos del servicio que son requeridos para interactuar con el, en particular, las entradas y las salidas de procesos atómicos.

Nótese que en OWL-S tanto *ServiceProfile* como *ProcessModel* son representaciones abstractas, únicamente el *ServiceGrounding* trata de un nivel concreto de especificación.

OWL-S no incluye una construcción abstracta para describir mensajes, mas bien se especifica el contenido concreto del mensaje, implícitamente mediante las propiedades *input* y *output* de algunos procesos atómicos. Asimismo, los procesos atómicos además de especificar las acciones básicas a partir de las cuales se componen procesos mas complejos, pueden ser vistos como primitivas de comunicación de una especificación abstracta de un proceso. Los mensajes concretos son especificados explícitamente en el *grounding*. La función central en el *grounding* es mostrar como las entradas (*inputs*) abstractas de un proceso atómico se

12 IOPE – Input, Output, Precondition, Effect.

implementan concretamente como mensajes, que comunican estos *inputs* y *outputs* en un formato pasible de ser transmitido.

Debido a la existencia de extensos trabajos en el área de especificación de mensajes, algunos adoptados ya por la industria se escogió WSDL[13] como modelo inicial para el mecanismo de *grounding* en OWL-S.

WSDL es "*un formato XML para describir servicios de red como un conjunto de puntos de acceso*[14] *operando en mensajes que contienen información tanto orientada a documentos como orientada a procesos"* [16]. Las operaciones y mensajes se describen de manera abstracta, vinculándose[15] luego a un protocolo de red en concreto y formato de mensaje definiendo así el punto de acceso.

Se puede observar que este concepto de vinculación (*binding*) de WSDL es consistente con el concepto de *grounding* en OWL-S.

13 WSDL – *Web Services Description Language*
14 *Endpoints* en el original.
15 *Binding*

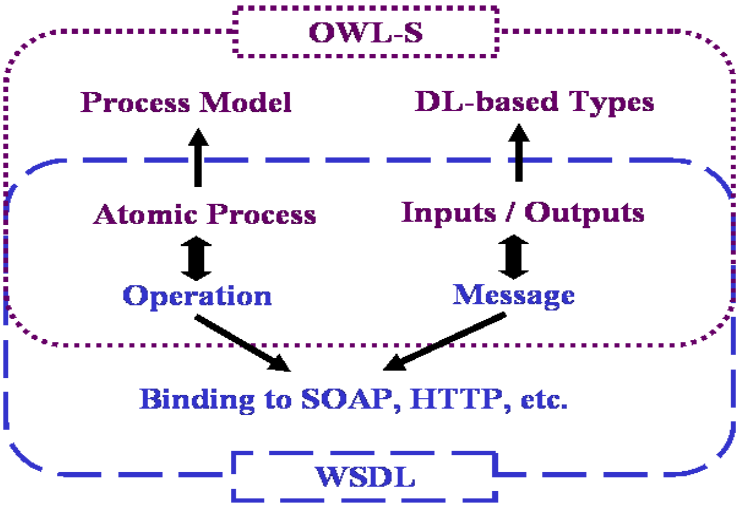


Figura 5: Relación entre OWL-S y WSDL según [12]

2.5.2. WSMO

De Acuerdo con [11] WSMO es una iniciativa para proporcionar semántica a los servicios Web, es un refinamiento y extensión del marco WSMF (*Web Service Modelling Framework*), mediante la generación de una meta-ontología para los Servicios Web Semánticos. WSMO es acompañada por un lenguaje formal, Web Services Modelling Language (WSML), que permite escribir descripciones de los servicios Web en concordancia con el modelo conceptual.

Los principios de diseño de WSMO están orientados de acuerdo a tres ejes fundamentales [17], [19]:

- Principios básicos de diseño de la web.
- Principios de diseño de la web semántica.
- Principios de diseño de computación distribuida orientada a servicios web.

Cumplimiento Web: WSMO se apoya en URI (Universal Resource Identifier) como medio de obtener la identificación de los recursos en la Web, el principio de espacio de nombres (NameSpace) para denotar el agrupamiento de información, soporta XML, descentralización de recursos y otras recomendaciones tecnológicas de W3C.

Basado en Ontologías: Todas las descripciones de recursos y datos intercambiados durante la utilización del servicio están basadas en ontologías que los definen y relacionan.

Estricto Desacoplamiento: Los recursos son especificados en forma independiente, cumpliendo con la naturaleza abierta y distribuida de la Web.

Centralismo de la Mediación: Se apunta al manejo de la heterogeneidad que aparece en todo ambiente abierto y que es necesario resolver para posibilitar la integración de servicios heterogéneos.

Separación Ontológica de Roles: Todos los requerimientos de los clientes son formulados en forma independiente de los servicios Web disponibles, se diferencia entre deseos del consumidor y servicios disponibles.

Semántica de Ejecución: La semántica de ejecución proporciona la realización práctica de WSMO, esto se logra a través de implementaciones como WSMX[18], implementación de referencia de WSMO.

Servicio versus Servicio Web: “Un servicio Web es una entidad computacional que (al ser invocado) tiene la capacidad de alcanzar un objetivo**.** Un servicio en cambio, es el valor real proporcionado por esta invocación. Esto implica que WSMO no especifica servicios sino servicios

Web, que son formas de buscar y comprar servicios.” [17]

2.5.2.1. Características de WSMO

Los elementos que caracterizan a WSMO se presentan en la siguiente figura [10], [11], [17], [19]:

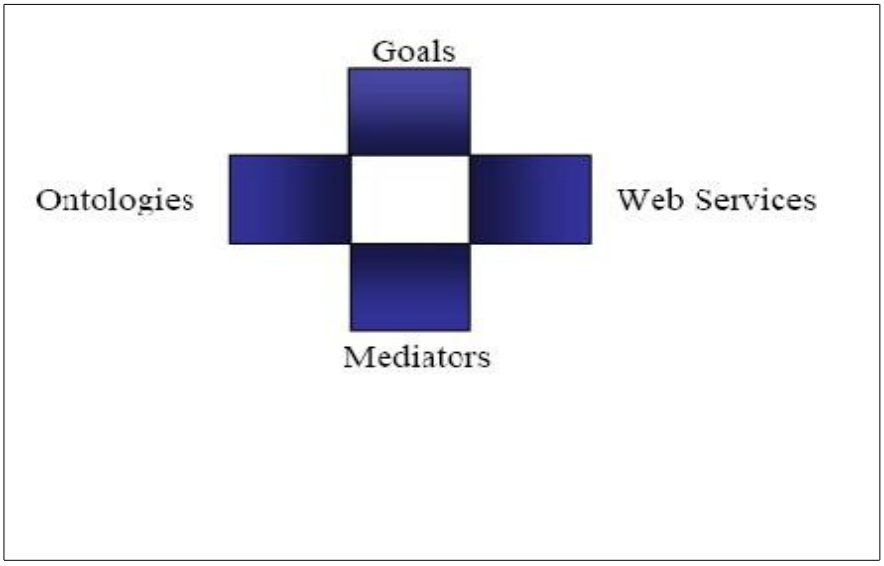


Figura 6: Características de WSMO según [10]

Ontologías: [19] Proporcionan terminologías (específicas de un dominio) para describir los otros elementos. Tienen dos propósitos principales: definir la semántica formal de la información y servir de nexo entre las terminologías humana y de máquina, así como las relaciones que existen entre las distintas terminologías (conceptos específicos del dominio de aplicación).

Servicios Web: [17] “Conectan computadoras y dispositivos usando los

protocolos estándar basados en la Web para intercambiar datos y combinarlos en nuevas formas. La distribución en la Web les da la característica de independencia con relación a la plataforma. Cada servicio Web representa una pieza atómica de funcionalidad que puede ser reutilizada para construir otras más complejas. WSMO describe los servicios Web desde tres perspectivas: propiedades no-funcionales; funcionalidad; comportamiento."

Objetivos: [17], [19], Son las intenciones que expresa un cliente cuando consulta un servicio Web, la funcionalidad que el servicio Web le proporcionará desde su punto de vista. Los objetivos y servicios Web se presentan como entidades independientes, asegurando el desacoplamiento entre requerimiento y servicio Web. Esta enfoque es conocido en base al objetivo (goal-driven approach), y se presenta como un elemento diferenciador de WSMO con relación a otros para abordar este problema.

Mediadores: [17], [19] Son los elementos introducidos a efectos de superar la heterogeneidad entre componentes descriptos en WSMO, [17] "permitiendo su encadenamiento para resolver incompatibilidades que aparecen a nivel de datos (mediando entre diferentes terminologías, resolviendo el problema de integración de diferentes ontologías), a nivel de procesamiento (mediando entre patrones de comunicación

heterogéneos), esto ocurre, por ejemplo, durante la comunicación entre servicios Web, o entre el consumidor y el servicio."

2.6. Clasificación de algoritmos de búsqueda de Web Services

Los algoritmos de búsqueda de servicios web no semánticos son anteriores a la aparición de la web semántica. Es por esto que los algoritmos de búsqueda de servicios web previos a la web semántica no fueron diseñados para trabajar con el modelo de datos semántico asociado a estos nuevos servicios web. Es así, que al introducir el concepto de web service semántico aparecen nuevos algoritmos que necesariamente toman en cuenta los aspectos semánticos de los servicios web.

No se trata de simplemente incorporar un nuevo modelo de datos para una representación nueva, sino de tomar en consideración *el significado en un contexto específico* de estos nuevos datos acerca de los servicios web.

Obsérvese que trabajar con los algoritmos tradicionales limitaría fuertemente la riqueza conceptual almacenada en la descripción semántica de los servicios web.

Una manera de conocer el comportamiento de un grupo de entidades (algoritmos en nuestro caso) es intentar clasificarlos de acuerdo a algún

patrón arbitrario y útil a nuestros efectos. La clasificación de algoritmos que se provee se basa en el paradigma o modelo de búsqueda subyacente utilizado para encontrar servicios web adecuados a un requerimiento predefinido (*Matchmaking*), por lo tanto, y de acuerdo a su comportamiento en el contexto de la información que buscan, se ofrece la siguiente clasificación de algoritmos de búsqueda en tres familias o clases fundamentales:

- Algoritmos Semánticos
- Algoritmos Sintácticos
- Algoritmos Híbridos

A continuación se enumeran los elementos presentes en cada algoritmo que justifican la categorización y se ejemplifican algoritmos que caen dentro de cada una de las categorías mencionadas.

Algoritmos Semánticos. Son aquellos algoritmos que son capaces de realizar únicamente búsquedas semánticas. Se basan en la clasificación ontológica de los elementos involucrados en la búsqueda, intentando encontrar algún *grado de concordancia* entre lo que busca el cliente y los elementos de los que se dispone en el repositorio de publicaciones. Tiene asociadas ontologías de dominio para su funcionamiento que involucra distintos grados de clasificación de las publicaciones. Un ejemplos de estos algoritmos es el algoritmo de Paolucci [13].

Algoritmos Sintácticos. Por oposición a la nomenclatura anterior, se denominan algoritmos sintácticos a aquellos algoritmos que no desarrollan búsquedas semánticas sino que se valen de otros mecanismos para encontrar concordancias entre lo solicitado por el cliente y lo que el repositorio de servicios web es capaz de proveer. Es usual encontrar aquí algoritmos que utilizan técnicas de *content based information retrieval.* El mas notable ejemplo de este tipo de algoritmos lo constituye UDDI [3], [4], [5].

Algoritmos Híbridos. Son aquellos algoritmos que combinan ambas técnicas de búsqueda. Realizan búsquedas semánticas basadas en ontologías subyacentes como los algoritmos semánticos y completan la búsqueda utilizando técnicas que utilizan los algoritmos sintácticos. Dos ejemplos notables son OWLS-MX [14] y Larks [15].

Las distintas familias de algoritmos presentadas introducen la necesidad de catalogar el desempeño de los diferentes algoritmos en determinados contextos y en un ambiente controlado. Es necesario saber cual es mejor en cada contexto, y resulta útil la medición de diferentes algoritmos de la misma familia en el mismo ambiente. Esta afirmación señala la necesidad de un ambiente de medición de algoritmos de acuerdo a determinadas métricas que puedan ser de interés y que sirvan como referencia de comparación.

Este ambiente de trabajo necesita estar dotado de determinadas

características que permitan la ejecución y medida de distintos algoritmos en condiciones de laboratorio reproducibles, permitiendo extraer las métricas que se seleccionen de manera automática.

3. Un ambiente de trabajo de medición de algoritmos de discovery – requerimientos funcionales.

Las necesidades básicas buscadas en un ambiente de trabajo de testeo y medición de algoritmos se agrupa en las siguientes áreas:

- Construcción de ambientes de ejecución de algoritmos.-
- Ejecución de algoritmo designado.
- Intercambiabilidad de algoritmos.
- Recolección automática de métricas.

3.1. Construcción de ambientes de ejecución de algoritmos

La ejecución de un algoritmo específico implica establecer todo el ambiente donde el mismo va a ejecutarse. Se trata de dotar al algoritmo de los elementos que debe buscar, y proveerle un ambiente controlado de ejecución a efectos de poder sacar conclusiones de su funcionamiento y de poder compararlo con otros algoritmos en idénticas condiciones de funcionamiento.

Se debe poder asociar un algoritmo a un juego de pruebas y conocer de los resultados arrojados sobre esos juegos de pruebas, de manera de que los mismos sean reproducibles y comparables.

En algoritmos híbridos y algoritmos semánticos se debe además disponer

de una ontología de referencia y servicios asociados de razonadores. Asimismo es necesario poder dotar al algoritmo de un espacio de búsqueda apropiado.

El espacio de trabajo generado así, que se compone de juegos de prueba, resultados esperados, ontologías, servicios de razonador debe ser múltiple, permitiendo cambiar fácilmente entre distintos espacios de trabajo aislados entre si para el mismo algoritmo. En la figura 7 se puede apreciar a modo esquemático las características del ambiente de trabajo planificado.

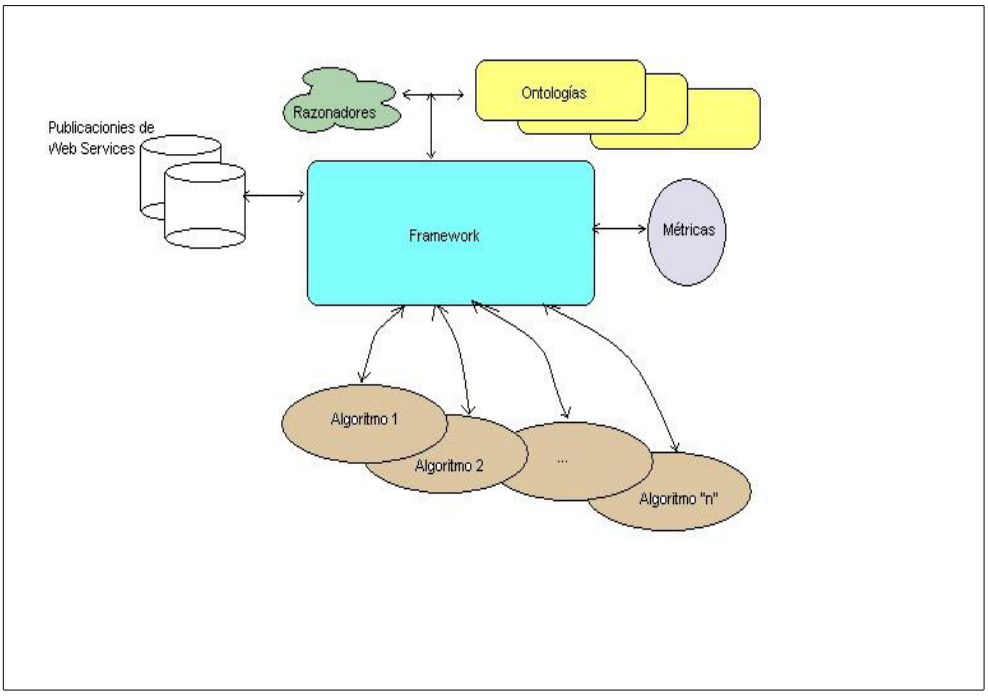


Figura 7: Esquema de características deseables del ambiente de trabajo

En la figura 7 se puede apreciar el ambiente de trabajo planificado como elemento central, disponiendo de servicios de razonadores y con un repositorio de ontologías de dominio. Se observa además un repositorio de publicaciones de servicios web. La interacción de los algoritmos con estos recursos se realiza a través del framework que provee mecanismos de recolección automática de métricas.

3.2. Ejecución de algoritmo designado

Tener la posibilidad de ejecutar un algoritmo arbitrario implica la disponibilidad de un conjunto de APIs que permitan invocar al algoritmo vinculándolo al ambiente de trabajo.

Este conjunto de apis deberá ser entregado al desarrollador a manera de contrato y como conjunto de operaciones disponibles para el algoritmo dentro del entorno de ejecución. El cumplimiento de este contrato dará como resultado un algoritmo medible por el entorno de trabajo propuesto.

No será posible sino a través de éste conjunto de operaciones acceder a ningún elemento del entorno de trabajo, como condición necesaria para que el entorno de trabajo pueda medir el desempeño del algoritmo en las condiciones prefijadas. Como resultado, este conjunto de operaciones implementadas en el ambiente serán el único punto de contacto entre un algoritmo en ejecución y el ambiente de trabajo.

Por otro lado, la posibilidad de ejecutar un algoritmo trae también como consecuencia la necesidad de la existencia de una interfaz de control desde donde sea posible realizar ciertas operaciones. Entre estas operaciones se destacan:

- Ordenar la ejecución de un algoritmo.
- Obtener los resultados de un algoritmo
- Asociar/disociar algoritmos a espacios de trabajo
- Consultar espacios de trabajo
- Asociar ontologías a espacios de trabajo
- Obtener métricas de la ejecución de un algoritmo.
- Administrar repositorios, tanto de ontologías como de publicaciones o repositorios de algoritmos.

3.3. Intercambiabilidad de algoritmos

Se deben proveer mecanismos para poder ejecutar distintos algoritmos sobre idéntico ambiente de trabajo, de manera de poder comparar su desempeño de acuerdo a métricas predefinidas.

Asimismo se requiere poder ejecutar un mismo algoritmo en distintos ambientes de trabajo para poder analizar su comportamiento bajo distintas condiciones.

Este mecanismo debe ser suficientemente ágil como para no sobrecargar la tarea esencial que es el diseño mismo del algoritmo.

En la figura 8, se puede apreciar el mecanismo de intercambiabilidad de algoritmos y ambientes de trabajo. Cada ambiente de trabajo está compuesto de un conjunto de servicios como espacio de búsqueda (publicaciones) y una ontología de referencia. Los algoritmos se asocian a cada espacio de trabajo de manera dinámica y se obtienen los resultados de medición de la ejecución en ese ambiente controlado. En la parte superior de la figura se esquematiza un algoritmo (Algoritmo “A”) trabajando en dos ambientes distintos (Ambientes de trabajo “1” y “2”), mientras que en la parte inferior se muestran dos algoritmos distintos (Algoritmo “A” y Algoritmo “B”) trabajando sobre el mismo ambiente de ejecución (“Ambiente de trabajo “1”).

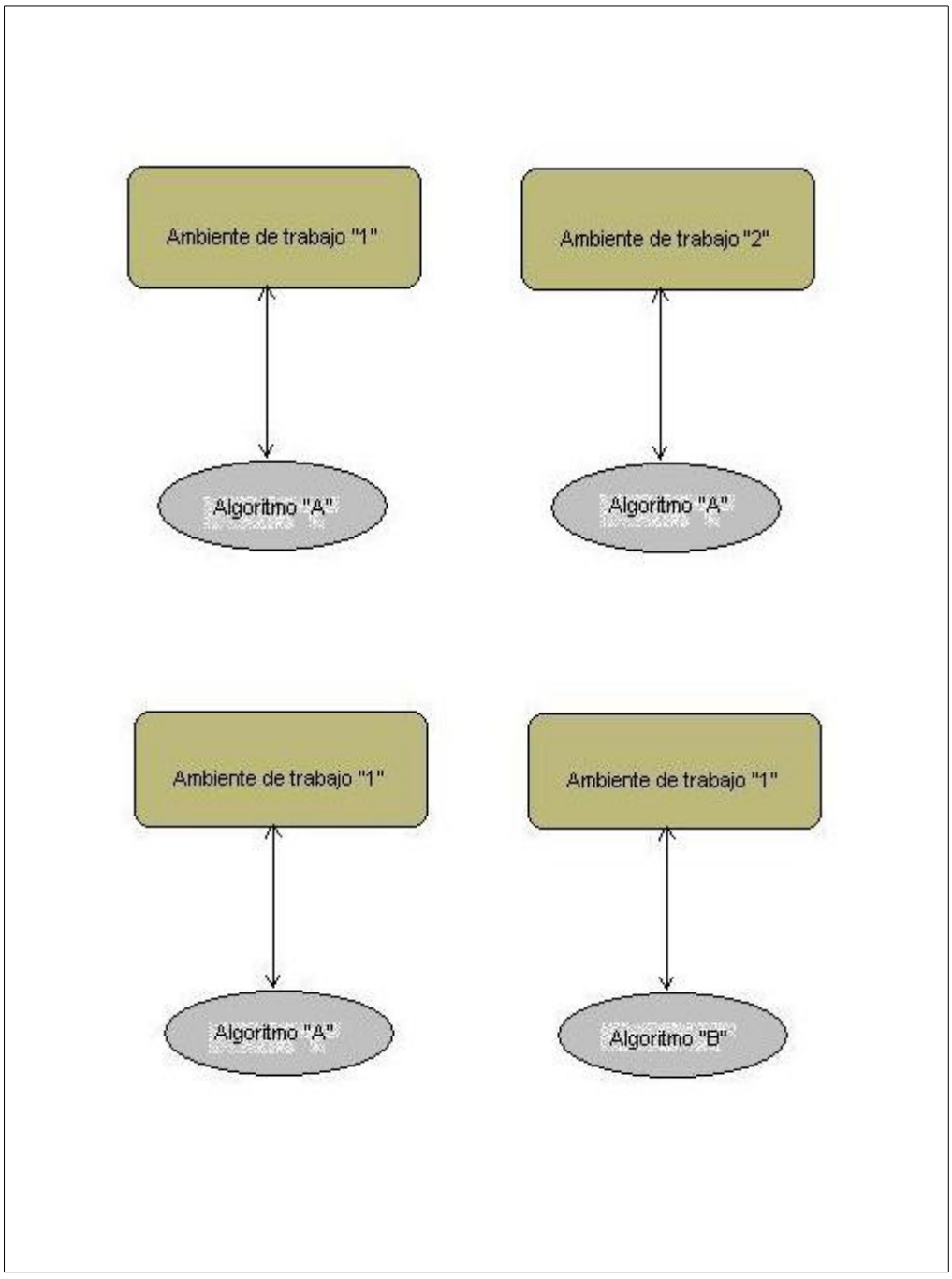


Figura 8: Intercambiabilidad de ambientes de trabajo y algoritmos.

3.4. ***Recolección automática de métricas***

Fuera de la definición formal de métrica, se establece que una métrica es una cuantificación de alguna propiedad de algo que queremos medir. En nuestro contexto se establece que una métrica es una cuantificación de una propiedad de un algoritmo que resulta interesante para el observador y que le permite compararlo con otros.

Existen diversos motivos para desear comparar diferentes algoritmos o bien distintas implementaciones de un mismo algoritmo. Algunos de estos motivos pueden ser:

- Investigar sus propiedades.
- Elegir entre distintas implementaciones de un mismo algoritmo para un proyecto específico.
- Elegir entre dos algoritmos distintos para la resolución de un problema.
- Mejorar una algoritmo existente o en desarrollo.

Los criterios de comparación entre algoritmos o implementaciones, cualesquiera que estos sean solo pueden ser valores medibles[16] y reproducibles en determinadas condiciones de trabajo.

Para poder entonces comparar distintos algoritmos desarrollados es

16 *"Lo que no se define no se puede medir. Lo que no se puede medir no se puede mejorar. Lo que no se mejora se degrada siempre*". William Thomson (Lord Kelvin)

necesario medir su comportamiento respecto a patrones específicos de medida. El marco de trabajo planificado deberá proveer estos patrones y permitirá medir los algoritmos respecto a estos patrones de manera automatizada. La recolección de éstas métricas debe ser provista por el mismo ambiente y se debe permitir que las métricas se tomen en idénticas condiciones de trabajo para todos los algoritmos que se desee comparar.

Para que el ambiente de trabajo presente utilidad a los desarrolladores de algoritmos nuevos es necesario poder automatizar la tarea de recolección de métricas a fin de permitir a los usuarios del entorno de trabajo centrarse en mejorar su desarrollo permitiendo ahorrar el tiempo que insumen tareas que se pueden automatizar.

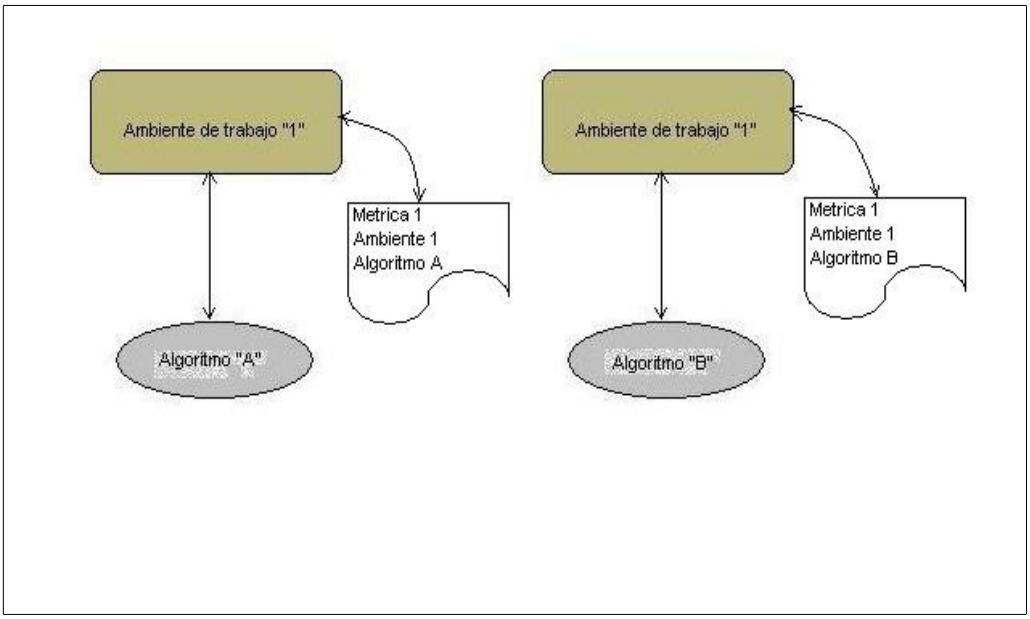


Figura 9: Recolección automática de métricas predefinidas.

En la figura 9 se ejemplifica el caso de comparación de dos algoritmos distintos en idénticas condiciones de trabajo. Los algoritmos “A” y “B” se ejecutan sobre el mismo ambiente de trabajo (denominado “Ambiente de trabajo “1” en la figura) obteniéndose valores de una métrica (“Métrica 1”) que eventualmente arrojará valores distintos para cada algoritmo, permitiendo la comparación repetible para ambos casos y en idénticas condiciones de trabajo.

No existen requerimientos específicos formulados relativos a qué métricas utilizar para la medición de algoritmos. Sin embargo es un efecto fuertemente deseado que la incorporación paulatina de distintas métricas se realice de una manera tal que no requiera demasiado esfuerzo, es decir, que realmente simplifique la tarea de medir.

Por tal motivo se implementó un conjunto básico de métricas que actúa como fundación de futuras métricas a incorporarse de manera de enriquecer la funcionalidad de este entorno de trabajo. El proceso detallado que ilustra paso a paso como incluir una nueva métrica en el entorno de trabajo se describe en el *apéndice B.*

Tratándose de algoritmos de búsqueda se ha optado por incorporar métricas generalmente utilizadas recuperación de información para medir el desempeño de algoritmos de búsqueda.

Las métricas seleccionadas basándose en el criterio expuesto son *Recall* y *Precision.* La primera es la proporción entre la cantidad de publicaciones relevantes recuperadas con respecto a la cantidad total de publicaciones relevantes en el espacio de búsqueda. La segunda es la proporción existente entre la cantidad de documentos relevantes recuperados y la cantidad de elementos recuperados.

Finalmente la recolección de métricas debe ser no solo automática sino que además transparente a la ejecución del algoritmo.

3.5. Resumen de funcionalidad

De las necesidades básicas a cubrir expuestas anteriormente, se desprenden algunos elementos entre los que resaltan:

1. La necesidad de la existencia de un repositorio general de ontologías, que se carguen a demanda al ambiente de trabajo. Es necesario disponer de un almacenamiento (repositorio) donde se puedan almacenar las ontologías de los distintos dominios de aplicación y donde puedan ser consultadas y accedidas a demanda. Deberá ser posible seleccionar u operar sobre una clase específica o individuo de una ontología específica residente en el repositorio. Es necesario que a partir de este *manejador* de ontologías se permitan las tareas básicas asociadas a la

clasificación semántica (determinar superclases de una clase, subclases, pertenencia de un individuo a una clase, clasificación de individuos, etc.). Es necesario además que a través de este repositorio se provean además las operaciones básicas de administración de ontologías, como por ejemplo, agregar ontologías al repositorio, enumerarlas, y fundamentalmente, asociar ontologías a ambientes de trabajo donde se ejecuten los algoritmos señalados.

2. La necesidad de un lenguaje de consulta. Es necesario poder realizar consultar arbitrarias sobre las ontologías residentes y sobre las representaciones de los web services. No solamente basadas en su estructura (taxonomía, jerarquía, etc.) sino por propiedades arbitrarias que puedan presentar los individuos de una ontología.

3. La necesidad de un repositorio de publicaciones de servicios web desde donde generar nuestro espacio de trabajo. Del mismo modo que con las ontologías, es requerido disponer de un almacenamiento de publicaciones de servicios web en alguna representación que permita accederlos y consultarlos a voluntad, ya sea por agentes humanos o por un algoritmo de búsqueda semántica de servicios web en tiempo de ejecución. Se requiere poder ejecutar funciones básicas sobre las publicaciones, como enumerarlas, permitir consultar sus descripciones, permitir

agruparlas en conjuntos específicos y asociarlas o disociarlas de espacios de trabajo, de manera de que un algoritmo pueda interactuar o no con ellas.

4. La necesidad de espacios de trabajo que vinculen las ontologías de dominio, las publicaciones de los servicios web y los servicios de razonamiento asociados. Denominaremos a estos como "*espacios de conocimiento y servicios*". En efecto, para poder dotar a un algoritmo de todos los elementos necesarios para su funcionamiento (ontologías de dominio, representaciones semánticas de servicios web) es necesario integrar estos elementos en un espacio de trabajo. Además, este espacio de trabajo debe estar aislado de los demás espacios de trabajo posibles. Es en ese cada uno de estos espacios específicos donde se deberá poder correr un algoritmo evitando que interfieran distintos espacios entre sí. Se requiere un entorno de trabajo dotado de múltiples espacios de trabajo que tengan sus propias características, donde existan distintas ontologías de dominio como referencia, y donde haya visibilidad de distintos conjuntos de servicios web para ser eventualmente encontrados por un algoritmo de matchmaking, y sobre los que se pueda operar de manera independiente a los demás (figura 10).

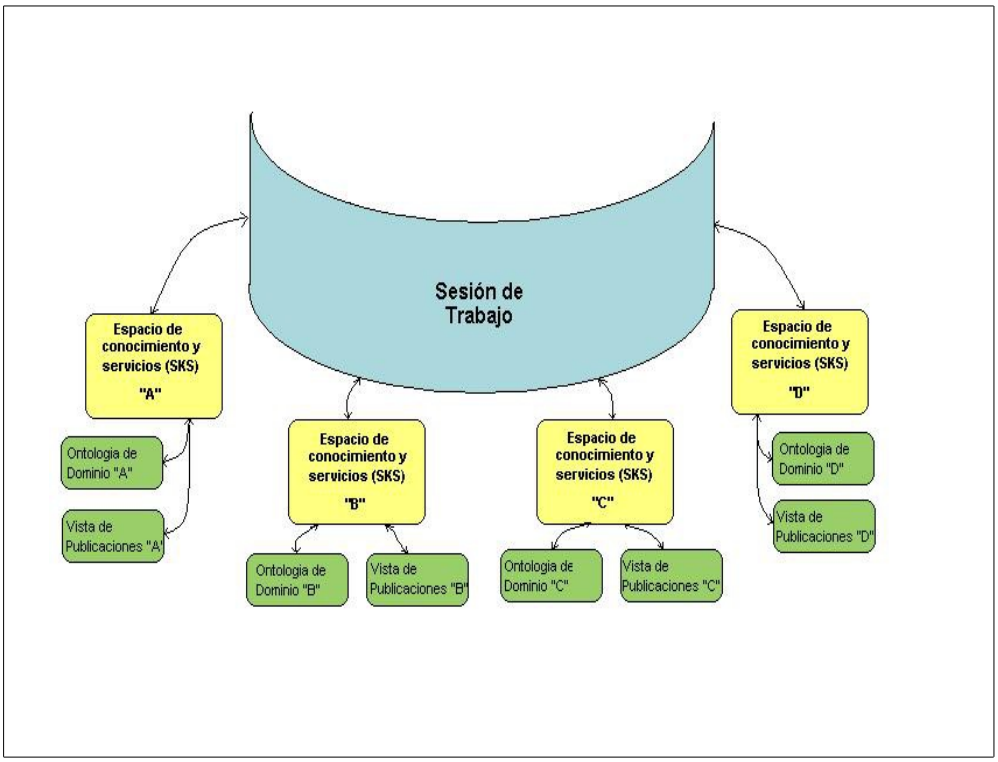


Figura 10: Espacios de Conocimiento y Servicios (SKS).

Es necesario poder asociar o disociar algoritmos a voluntad entre los distintos espacios de conocimientos y servicios. Se debe poder intercambiar arbitrariamente algoritmos entre distintos espacios de trabajo de manera de poder efectivamente comparar su comportamiento en idénticas condiciones de trabajo y de manera reproducible.

5. La necesidad de un repositorio de algoritmos desde donde elegir el algoritmo a probar. Es necesario disponer de las implementaciones

de los algoritmos para poder ejecutarlos. El entorno de trabajo deberá proveer un lugar desde donde elegir que algoritmo ejecutar en cada momento. De esta manera, el usuario del entorno de trabajo podrá tener disponible no solo su propio trabajo sino además las contribuciones previas de otros autores y propias de algoritmos con los cuales poder comparar su trabajo actual.

6. La necesidad de un cliente, para poder cargar un algoritmo al repositorio y poder controlar, consultar y construir el ambiente de trabajo deseado. Este cliente es concebido como una interfaz de comandos, a través de la cual el usuario interactúa con su propio ambiente. Este cliente será además la interfaz de un usuario con todo el sistema, desde donde estarán disponibles todas las funciones que el entorno de trabajo ofrece, administrar todos los repositorios, los espacios de trabajo, las ontologías, algoritmos, etc..

7. La necesidad de una máquina virtual[17] que aisla al algoritmo, provee una interfaz con el espacio de conocimiento y servicios y donde tiene lugar la recolección automática de métricas. Esta máquina virtual será provista al desarrollador en forma de API, con una interfaz bien definida y que permitan al algoritmo la utilización de los servicios programados en el entorno de trabajo.

17 DVM por *Discovery Virtual Machine.*

Para que un algoritmo pueda ser *medible* por el entorno de trabajo y para que pueda utilizar los espacios, las publicaciones y las ontologías disponibles a través de él, el mismo deberá ajustarse a una interfaz de comunicación con el entorno de trabajo así como a protocolos de llamada específicos y bien definidos. Esta interfaz toma la forma de API que es provista al desarrollador. La implementación de esa interfaz se encuentra en el servidor, que es donde el algoritmo se ejecuta finalmente. Toda otra tarea que el algoritmo ejecute fuera de esta API no será reconocida por el entorno de trabajo y por lo tanto no será medible. Por este motivo para que un algoritmo pueda utilizar el entorno de trabajo y todos sus servicios asociados necesariamente deberá ceñirse a esta interfaz.

Esta máquina virtual será la única interfaz entre el marco de trabajo y el algoritmo. Es en ella donde necesariamente tendrá lugar la recolección de las métricas.

3.6. Herramientas de Desarrollo

Seleccionar herramientas de desarrollo para la construcción de este ambiente de trabajo implicó investigar sobre herramientas disponibles de:

- Manejo de descripciones semánticas de servicios web.
- Repositorios de ontologías.

- Razonadores.
- Bibliotecas para manejo de ontologías y razonadores.
- Bases de Datos.
- Descripciones de servicios web como juegos de prueba.

3.6.1. Herramientas para acceso a descripciones semánticas de servicios web.

Definir que herramientas o bibliotecas utilizar para manejar descripciones semánticas de web services implica necesariamente tomar posición simultáneamente acerca que tipo de representación utilizar (OWL-S, WSMO, otras).

Las bibliotecas disponibles están en evolución por lo que se debe desarrollar de manera que el impacto de sustituir una biblioteca por otra sea mínimo. Se adopta la decisión de utilizar descripciones OWL-S (permitiendo soportar WSMO en el futuro) y dentro de OWL-S se optó por las API provistas por la universidad de Carnegie Mellon en http://www.daml.ri.cmu.edu/owlsapi/index.html

Existen alternativas a estas opciones, notoriamente las provistas por el proyecto de agentes web impulsado por la Universidad de Maryland, Institute for Advanced Computer Studies (Maryland Information and Network Dynamics Lab Semantic Web Agents Project) *Mindswap,* disponibles en http://www.mindswap.org

Se optó por las primeras por ser mas recientes, no existiendo otro criterio técnico a la fecha que apoye o descalifique esta decisión, sin desconocer que también existen desarrollos que interactúan con definiciones semánticas de web services que se basan en las proporcionadas por la universidad de Maryland.

3.6.2. Herramientas para la administración de Ontologías.

En cuanto a bibliotecas para manejo de ontologías, razonadores y consultas sobre descripciones semánticas, la decisión recayó sobre ***jena***, la biblioteca producto del proyecto de web semántica de HP labs (http://www.hpl.hp.com/semweb) disponible en internet a través de http://jena.sourceforge.net/downloads.html.

Esta elección resulta mas natural, dado que esta biblioteca es referida en toda la bibliografía consultada y es utilizada por las API seleccionadas para el manejo de descripciones semánticas de web services (CMU).

Jena provee:

- Una API de manipulación RDF
- Lectura y escritura de RDF en RDF/XML, N3, Triplas
- Una API OWL
- Persistencia
- Un motor SPARQL

- Interfaz DIG
- Profusa documentación y ejemplos.

Existen herramientas y entornos de trabajo que se basan en Jena y Dig. Entre los distintos entornos de trabajo evaluados se destaca ***hawk*** (Universidad de Lehigh). Hawk provee:

- Un repositorio que soporta OWL
- OWL-RDF writer
- OWL-RDF writer
- Modelo de almacenamiento, manipulación en memoria y Base de Datos
- Api de consulta
- Soporte para inferencia con razonadores con interfaz DIG

Lamentablemente no se encontró información suficiente sobre la evolución de ese proyecto para poder utilizar esta herramienta.

3.6.3. Razonadores

Entre los razonadores evaluados encontramos los siguientes:

- Fact ++(http://owl.man.ac.uk/factplusplus/)
- Kaon2 (http://kaon2.semanticweb.org/)
- Racer (http://www.racer-systems.com/)

- Pellet (http://pellet.owldl.com/)

De ellos fue seleccionado el último por motivos de licenciamiento y porque se conocen referencias de funcionamiento del mismo como servidor DIG. En la lista se presentan algunas opciones igualmente válidas por los mismos criterios, como Fact ++.

Una comparación de desempeño mas completa acerca de los razonadores mencionados se puede encontrar en [25]

3.6.4. Bases de Datos y juegos de prueba

Como Base de Datos se seleccionó Postgres, motivado por su licenciamiento y sus referencias técnicas, si bien las opciones disponibles eran muchas.

Finalmente, se dispuso de un conjunto de juegos de prueba para descripciones semánticas de servicios web, seleccionándose OWLS-TC, disponible desde (http://projects.semwebcentral.org/projects/owls-tc/).

3.6.5. Otros

En el transcurso de las distintas evaluaciones de herramientas, para analizar su funcionalidad e evaluar que podían aportar al proyecto se tomó contacto con ***WSMO Studio***, editor de ontologías, que puede

interactuar con un razonador y está disponible como plugin del entorno de desarrollo integrado ***Eclipse***. Otra herramienta evaluada fue el editor de ontologías OWL-S integrado de ***Protegé.*** Estas herramientas finalmente no fueron necesarias debido a la disponibilidad de juegos de prueba.

4. Arquitectura del ambiente de trabajo.

Existen algunos requerimientos no funcionales fundamentales que deben ser considerados:

- Independencia de las apis utilizadas. Esto significa proveer determinado grado de aislamiento de manera de que sea posible con un esfuerzo razonable sustituir las bibliotecas utilizadas por otras de otros fabricantes.-
- Posibilidad de separar distintas componentes del entorno de trabajo para poder ser reutilizados en otros proyectos.-
- Permitir trabajar de manera remota.-

Esto apunta a la determinación de una arquitectura esencialmente cliente servidor y con un acoplamiento bajo.

Por otro lado, de la descripción funcional y tomando en cuenta lo expuesto en el capítulo anterior, se pueden agrupar los componentes esenciales de acuerdo al esquema que se presenta en la figura 12.

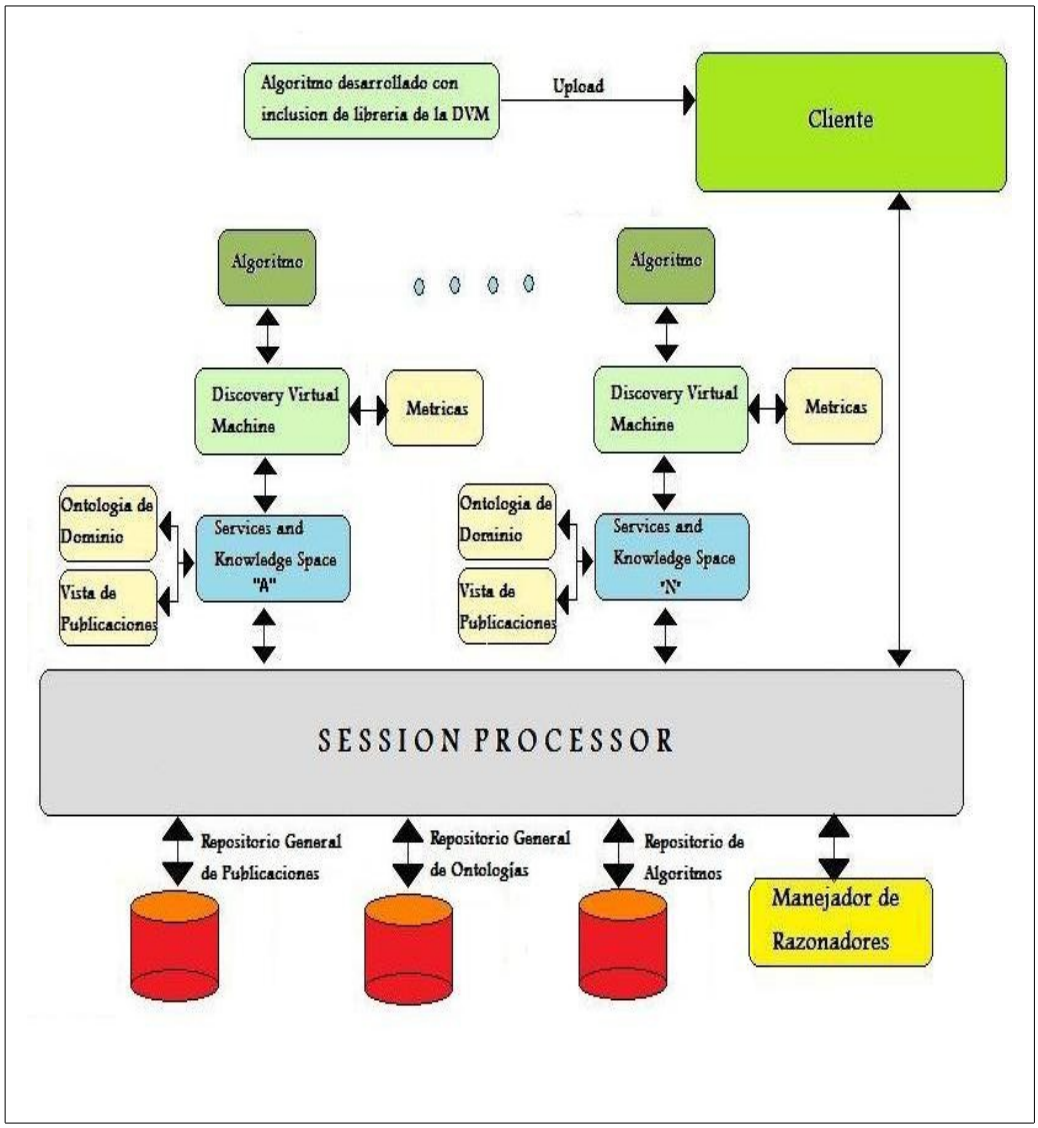


Figura 12: Descripción del ambiente de trabajo

4.1. Componentes esenciales

Los componentes básicos en los que se divide el entorno de trabajo, tomando como referencia la figura 12, son los siguientes:

- Manejador de repositorio de publicaciones.

- Manejador de ontologías.
- Manejador de razonadores (soporte de razonadores).
- Manejador de interrelaciones entre módulos.
- El cliente.

Estos componentes se describen a continuación.

4.1.1. Manejador de repositorio de publicaciones

El propósito de la existencia de una manejador de publicaciones y de un repositorio asociado, radica en la necesidad de operar con publicaciones, ya sea con todas ellas o con uno o varios subconjuntos específicos de publicaciones de interés. Este componente permite almacenar las publicaciones de servicios web expresadas en OWL-S o en WSMO (La versión inicial solo se soporta OWLS). Aquí se permite realizar las tareas básicas asociadas a la administración de publicaciones, listarlas, asignarlas a espacios de trabajo, construirlas a partir de una URL, levantarlas de un archivo, etc. Existe un repositorio general y además cada "e*spacio de conocimiento y servicios"* dispone de un repositorio propio o *vista de publicaciones* que es cargado a partir del primero a través de comandos enviados al servidor. Sobre estas publicaciones (que no son mas que descripciones semánticas de Web Services) se puede acceder individualmente, ya sea como una unidad o bien a cada una de sus partes (parámetros, etc.), tal como se han definido.

4.1.2. Manejador de ontologías

Al realizar búsquedas semánticas es necesario disponer de ontologías de referencia u ontologías de dominio. Del mismo modo que con las publicaciones, es necesario vincular estas ontologías de dominio a los espacios de trabajo que se consideren apropiados. También se les proporciona persistencia en una base de datos relacional.

4.1.3. Manejador de razonadores (soporte de razonadores)

Asociado al manejador de ontologías es necesario proveer mecanismos de razonamiento. Se logra adaptando un razonador DIG. Así se proveen las funciones básicas como la clasificación de individuos, la validación de una ontología, reportes de inconsistencias, análisis de la jerarquía de clases, etc. A través del soporte de razonamiento es posible también ejecutar consultas SPARQL ya sea sobre una ontología de dominio o sobre el conjuntos de descripciones semánticas de web services asociadas a un espacio de conocimiento y servicios (publicaciones).

4.1.4. Manejador de interrelaciones entre módulos

Con la finalidad de hacer funcionar de manera coordinada los demás elementos funcionales, es necesario disponer de un grupo de elementos que ene conjunto gobiernan la interrelación entre los módulos. Dentro de este grupo se encuentra un servidor que interpreta los comandos enviados desde un cliente, un manejador de sesiones que permite al

cliente operar en un ambiente aislado de otros clientes, el espacio de conocimiento y servicios que permite evaluar distintos algoritmos trabajando sobre distintos repositorios de publicaciones y utilizando distintas ontologías de referencia en una misma sesión, un repositorio de algoritmos a evaluar, y la discovery virtual machine necesaria para la recolección automática de métricas de desempeño.

4.1.5. El cliente

El cliente tiene como objetivo proporcionar una comunicación con el servidor, enviar los comandos correspondientes y desplegar las respuestas recibidas.

En las primeras versiones se utilizó un cliente *telnet* para enviar los comandos al servidor, pero pronto fue necesario realizar otras tareas para las cuales este cliente no era adecuado. En particular, la necesidad que motivó la creación del cliente fue la de enviar un algoritmo hasta el servidor para que sea almacenado y posteriormente ejecutado en el ambiente designado y con los parámetros que se deseara.

4.1.6. El espacio de conocimiento y servicios (SKS)

La necesidad de disponer distintos escenarios simultáneos para el mismo o distintos algoritmos motivó la posibilidad de creación dinámica de estos espacios.

Una vez que se tiene una sesión con el servidor, se pueden crear a través de comandos distintos espacios aislados donde existirán distintos repositorios de publicaciones de servicios web (tomadas del repositorio central) y distintas ontologías de referencia.

Se puede administrar, agregar o quitar de estos espacios las distintas representaciones semánticas de servicios web, así como distintas ontologías de referencia.

Se puede *navegar* a través de los distintos espacios simplemente cambiando de *espacio activo* que es el espacio por defecto en el que el usuario se encuentra en cada momento.

La denominación de *espacio de conocimiento y servicios* viene de la característica de estos espacios de disponer cada uno de su propia ontología de referencia arbitraria, con funciones de razonamiento integrado, y del repositorio de servicios propio de cada espacio.

Cada espacio permite además que se realicen consultas SPARQL ya sea sobre la ontología de referencia como sobre las descripciones semánticas de web services (publicaciones) que se depositan en su repositorio particular.

Esta misma funcionalidad de consultas SPARQL está disponible tanto para comandos ingresados desde un cliente como para consultas realizadas desde la API provista en la DVM.

4.1.7. La Discovery Virtual Machine y la recolección automática de métricas.

Medir el desempeño de un algoritmo requiere tener conocimiento de las tareas que éste internamente a medida que se va ejecutando. Para lograr éste objetivo se implementó una máquina virtual, que está compuesta por una API a través de la cual el algoritmo interactúa con el entorno de trabajo. En ésta API están definidas las funciones que el algoritmo de discovery puede realizar para realizar su cometido al mismo tiempo en que es medido su desempeño.

La DVM se libera como una biblioteca que debe ser incluida por el desarrollador.

La ventaja de éste enfoque radica en la posibilidad de extender la DVM, haciendo pública la nueva funcionalidad disponible.

Por otro lado, internamente la DVM tiene incorporadas un conjunto de funciones no públicas para el cálculo de métricas. De esta manera se consigue que la medición de aspectos relevantes del funcionamiento interno del algoritmo puedan ser medidos con transparencia a su desarrollo.

Este enfoque independiza al desarrollador de cualquier particularidad que pueda presentar una Discovery Virtual Machine. Mientras el desarrollador satisfaga la interfaz provista, el servidor se encargará de proveer la implementación. Esto permite además modificaciones en la DVM sin

impacto en la implementación del algoritmo.

Un caso que requiere la modificación de la DVM es precisamente la implementación de una nueva métrica.

Se resalta que todas las funciones relevantes que un algoritmo de discovery pueda ejecutar deberán ser aquellas provistas por la DVM, y esto incluye búsquedas, comparaciones, y todas aquellas funciones soportadas por una implementación de una DVM. De esto se desprende que para poder realizar mediciones sobre algoritmos cada vez mas sofisticados y complejos se debe hacer evolucionar la DVM para que pueda proveer mas operaciones disponibles cada vez mas complejas, para que puedan ser consumidas por los distintos algoritmos. Esto implicará además realizar nuevas liberaciones de las librerías al agregar nueva funcionalidad (operaciones disponibles para los algoritmos) en la Discovery Virtual Machine.

4.2. Restricciones en la construcción de un algoritmo medible por el entorno de trabajo.

La primera versión del entorno de trabajo solo admite algoritmos que cumplan con un protocolo de llamada bastante específico.

La especificación del protocolo de llamada de un algoritmo de discovery se encuentra en la librería de desarrollo que se libera para poder desarrollar un algoritmo medible por el entorno de trabajo.

En su versión inicial, el algoritmo de discovery deberá aceptar como parámetros:

- Conjunto descriptivo de los datos disponibles para el consumo del servicio web y que pueden ser aportados por el cliente. Estos datos serán parámetros de entrada del servicio web y deberán existir referencias a los mismos en la ontología de dominio.
- Conjunto descriptivo de los datos requeridos por el cliente y que el servicio web encontrado debe proveer. Del mismo modo que en el conjunto anterior, cada elemento debe estar referenciado en la ontología de dominio.

Como resultado el algoritmo devuelve una estructura de datos, también especificada en la librería que se libera, que provee una estructura ordenada por relevancia del conjunto de resultados. La relevancia, viene dado por el g*rado de concordancia* entre el servicio web encontrado y el servicio web buscado.

Esta estructura además provee un mapa (*WeightedUriMap*) entre los parámetros de entrada requeridos por cada web service provisto como solución y los parámetros de entrada ofrecidos por el cliente. Asimismo se provee un mapeo entre los parámetros de salida ofrecidos por el web

service y los requeridos por el cliente. Estos mapas entre los parámetros correspondientes están ponderados por grado de concordancia de acuerdo a su taxonomía dentro de la ontología de referencia.

Del grado de concordancia de ambos conjuntos surge un grado de concordancia global, que se traduce como la relevancia de la solución encontrada. La respuesta viene ordenada precisamente por relevancia.-

El algoritmo de ordenamiento provisto el trivial, pudiéndose sobrecargar generando una nueva implementación de la Discovery Virtual Machine. En la actualidad, generar una nueva implementación de la Discovery Virtual Machine implica realizar cambios menores en los espacios de conocimiento y servicios (SKS). Si se respeta la interfaz pactada (conocida como Discovery Virtual Machine) no será necesario liberar otra librería para desarrollo.

Respecto a la inclusión de otros parámetros de búsqueda, los parámetros candidatos deberán corresponder a aquellos atributos presentes en las publicaciones. Para que ello sea posible será necesario recurrir a diferentes representaciones semánticas aún no soportadas [20], [21].-

En este punto específico reviste interés soportar nuevas operaciones en la DVM, en particular para obtener nuevos criterios de similitud entre los

elementos residentes en el repositorio de publicaciones y los requerimientos que se puedan presentar. Específicamente se deberá soportar una función *distancia* [21] a efectos de poder obtener un grado de concordancia global y obtener nuevos criterios para seleccionar y ordenar servicios web candidatos para el consumo.

5. Una implementación de referencia – El algoritmo de Paolucci a través de la DVM.

5.1. Antecedentes, filosofía de desarrollo

Para presentar la implementación del algoritmo propuesto por Paolucci en [13] se ofrece a modo de introducción una descripción del mismo y su filosofía de desarrollo.

El trabajo plantea un punto de partida conocido, la necesidad de localización de web services a partir de una búsqueda semántica, por oposición a una búsqueda tradicional (palabras clave) o por la mera descripción de la interfaz que proveen.

En [13] se plantea que[18] un aspecto crucial en la búsqueda o coincidencia de funcionalidad (capability matching) es que únicamente puede ser realizado a nivel semántico.

Se parte de la concepción de una infraestructura para web services soportada por un conjunto de directorios en la Web. Este enfoque es un enfoque clásico, y de hecho ya existen repositorios basados en esa filosofía (UDDI, por ejemplo) aunque conceptualmente diferente, dado que se concentran en la realización de búsquedas por palabras clave exclusivamente.

Una publicación en esos registros[19] coincide con un requerimiento cuando

18 *"... in fact, one crucial fact in capability matching is that it can be done only at the semantic level..."*

19 *"advertisement"* según los autores.-

es ***suficientemente similar*** al requerimiento. Se apreciará que el algoritmo abarca tres niveles de similitud para publicaciones que concuerdan con requerimientos.

De acuerdo a este enfoque, el algoritmo está en algún grado expuesto a la explotación de publicaciones demasiado genéricas que intentan coincidir con todos los requerimientos (el problema se da igualmente cuando existen requerimientos demasiado genéricos). Los autores proponen asignar puntajes las publicaciones de acuerdo a grados de concordancia con el requerimiento[20]s.

En [13] se establece que un algoritmo debe:

- "Soportar concordancias semánticas flexibles entre publicaciones y requerimientos basados en las ontologías disponibles para los servicios y el algoritmo de concordancia". Esto quiere decir que el algoritmo debe analizar la/s ontología/s que describen o en las que se basan los web services (tanto de publicación como de requerimiento) para dar algún grado de similitud.

20 En nuestra opinión, este factor introduce la necesidad de disponer de mecanismos de clasificación previa (preferentemente automáticos) de manera de poder establecer una pre clasificación de las publicaciones que *pre penalice* las publicaciones genéricas. Esto se fundamenta en el hecho de que no es necesario conocer un requerimiento para saber que una publicación es *demasiado genérica* (siempre de acuerdo a criterios de concordancia predefinidos). Es posible adoptar "criterios de ingreso" a un repositorio que deben cumplir los web services para ser admitidos. Opinamos además que no es necesario tener en cuenta los casos en los que los *requerimientos* son demasiado genéricos, dado que pensamos que es deseable que se transfiera parte de la responsabilidad de la concordancia al cliente. Entendemos que este factor tenderá a estimular que los requerimientos no sean tan genéricos.

- “A pesar del grado de similitud de la concordancia, el algoritmo debe minimizar los falsos positivos y los falsos negativos. Inclusive el requerimiento debe poseer determinado control sobre la *cantidad de flexibilidad* que le permite al sistema.”
- “El algoritmo deberá estimular a las publicaciones y a los requerimientos a ser honestos con sus descripciones al costo de pagar un precio o no ser considerados como coincidencia o ser considerados inapropiadamente”.
- “El proceso de concordancia (matching) debe ser eficiente, no debe penalizar al cliente con retardos tales que lo hagan poco efectivo”.

Una premisa que sigue el algoritmo es que “una publicación concuerda con un requerimiento cuando el servicio provisto por quien publica puede ser de algún uso para el requerimiento. Específicamente una publicación concuerda con un requerimiento cuando todas las salidas del requerimiento concuerdan con las de la publicación, y todas las entradas de la publicación concuerdan con las entradas del requerimiento”.

5.2. El algoritmo

El cuerpo central del algoritmo es el siguiente [13]:

```
match (request) {
        recordMatch = empty list;
        for all adv in advertisements do {
                if match(request, adv) then
```

```
recordMatch.append(adv); }
        return sort (recordMatch);
}
```

El cuerpo principal admite un requerimiento (request) que es contra el cual se evaluarán las publicaciones. Para comenzar se parte de una lista vacía de resultados.

Posteriormente, para todos las publicaciones, si *concuerdan* con el requerimiento entonces se agregan a la lista de resultados. Finalmente se ordena el resultado por algún criterio y se devuelve.

Nótese que *concordar* implica la llamada a la subrutina *match* que se discute a continuación.

Como se citó anteriormente, “Una concordancia entre una publicación y un requerimiento consiste en la concordancia entre todas las salidas del requerimiento contra las salidas de la publicación; y todas las entradas de la publicación contra las entradas del requerimiento”. Esto requiere una concordancia para las entradas y otra para las salidas, por lo tanto dos rutinas de concordancia (matching).

Para la rutina de concordancia entre las salidas, “si una de las salidas del requerimiento no concuerda contra ninguna salida de la publicación, entonces la concordancia falla”. Esto indica que todas las salidas del requerimiento deben ser satisfechas, lo que garantiza que el

requerimiento se satisfaga. Pero el problema no termina allí, para que esa publicación efectivamente funcione se deben satisfacer todas sus entradas, por lo que el requerimiento las debe proveer. Vemos que la rutina de concordancia de las entradas y las salidas son análogas, se presenta a continuación la rutina de concordancia de las salidas.

```
outputMatch (outputsRequest, outputsAdvertisement) {
    globalDegreeMatch = Exact;
    for all out in outputsRequest do {
        find outA in outputsAdvertisement such that
            degreeMatch = maxDegreeMatch (outR,
                                          outA);
        if (degreeMatch = fail) {return fail;}
        if (degreeMatch < globalDegreeMatch)
            {globalDegreeMatch = degreeMatch;}
    }
    return globalDegreeMatch;}[21]
```

Esta rutina de comparación acepta como parámetros las salidas del requerimiento (es decir, lo que el cliente necesita) y las salidas de la publicación (es decir, lo que la publicación es capaz de proveer).

Se retorna un grado de similitud entre lo que espera el cliente y lo que puede proveer la publicación.

En primer lugar se inicializa el grado de concordancia "globalDegreeMatch" como "Exact" que es simplemente un valor inicial

21 en el paper figura "return sort(recorMatch)" creemos que es un error tipográfico pero no tenemos manera de comprobarlo.

escogido de un grupo que se analizará mas adelante.

Una vez inicializado, para todas y cada una de las salidas requeridas se busca entre las salidas que puede proveer la publicación aquella que su grado de concordancia sea máximo, es decir la mas parecida. Eso se hace mediante la invocación a la rutina maxDegreeMatch (outR, outA) que estudiaremos mas adelante. Este “máximo” puede ser el valor “fail”, es decir, directamente no hay concordancia entre lo buscado y lo ofrecido, entonces no se sigue buscando, la concordancia simplemente falló (recuérdese que se deben satisfacer *todas* las salidas esperadas del cliente). Si para esta instancia de búsqueda el grado de concordancia encontrado es menor al que se tiene actualmente (se comenzó con el valor “*Exact”*) entonces se actualiza el grado de concordancia a este valor. Como resultado, se tendrá el menor grado de concordancia hallado entre alguno de los parámetros de salida esperados y todos los ofrecidos.

Nótese que invirtiendo los parámetros se obtiene la rutina de comparación de las entradas, donde se deben satisfacer todas las entradas de la publicación.

Para calcular el *grado de concordancia* entre una salida requerida y una salida que puede proveer de una publicación, Paolucci propone una rutina que devuelve valores dentro de un conjunto discreto. Depende de la *relación entre conceptos* asociados con esas entradas y salidas, y aquí es donde se introduce la búsqueda semántica.

La rutina es la siguiente:

```
degreeOfMatch (outR, outA) {
        if (outA = outR) { return exact }
        if (outR.subclassOf (outA)) { return exact }
        if (outA.subsumes (outR))  { return plugIn }
        if (outR.subsumes (outA)) { return subsumes }
        return fail;
}
```

Se distinguen cuatro grados de concordancia:

- ***Exact*** – si los parámetros son equivalentes (outA = outR) se devuelve este valor. Del mismo modo, si lo que se requiere (outR) es subclase de lo que se provee (outA) se devuelve exact. Esto se hace asumiendo que al publicar outA el proveedor se compromete a entregar salidas consistentes con cada subtipo de OutA. De acuerdo a Paolucci esto equivale a decir que al publicar "VEHICULO" el proveedor se compromete a entregar "VAN", "SEDAN", "CAMIONETA". Si no fuera así y el proveedor solamente proporcionara "VAN" entonces una mejor estrategia sería restringir su publicación a "VAN". En mi modesta opinión, no estaría mal devolver un valor distinto a exact en este caso, de manera de reflejar mejor esta situación y no depender en esa medida de la voluntad del proveedor de realizar correctamente la publicación.
- ***Plug in***. En este caso outA es un conjunto que incluye outR, entonces, en otras palabras outA podría ser incorporado como

outR (plugged in según el autor). Paolucci cita a modo de ejemplo un servicio que provee cualquier tipo de “vehículos” podría ser utilizado por un cliente que espera “camionetas”. Este tipo de regla reconoce que existe una relación mas débil entre outR y outA que la anterior (exact).

- Subsumes. De manera análoga al caso anterior, en este caso outA está contenido en outR. El cliente puede utilizar la publicación para alcanzar sus objetivos, pero es probable que deba o modificar sus planes o hacer alguna tarea complementaria.
- ***Fail***. Es cuando no se identifica ningún tipo de relación entre publicación y requerimiento.

Finalmente presentamos la rutina de ordenamiento, que se utiliza para ordenar los resultados favorables obtenidos.

La rutina es la siguiente:

```
sortRule (match1, match2) {
        if (match1.output > match2.output)  {
                return (match1 > match2)}
        if ((match1.output = match2.output) &&
            (match1.input > match2.input)) {
                return (match1 > match2)}
        if ((match1.output = match2.output) &&
            (match1.input = match2.input)) {
                return (match1 = match2)}
        return (match1 < match2)[22]        }
```

22 Incompleto en el original.

De acuerdo a esta rutina, la concordancia de salidas tiene cierta prioridad, quedando la concordancia de entradas como criterio secundario. Esta rutina es en definitiva el criterio de ordenación de la rutina *sort* que se aprecia en el cuerpo principal del algoritmo. En síntesis, para dos resultados relevantes que se encuentran en el espacio de soluciones (denominado previamente *recordMatch*) el resultado de aplicar la subrutina sortRule devuelve cual de las dos soluciones es mas relevante que la otra. El código fuente del algoritmo se presenta en el apéndice C.

6. Ejemplo de interacción con el entorno de trabajo – La ejecución del algoritmo de Paolucci

A modo de ejemplo de interacción con el entorno de trabajo desarrollado se presentan un caso completo de ejecución del algoritmo de Paolucci, cuyo código fuente se presenta en el anexo correspondiente.

Durante esta interacción se muestra además otras tareas complementarias que pueden ser de ayuda en el momento de manipular el ambiente de trabajo.

La interacción con el ambiente es a través del comando digitados desde el cliente. Una lista completa de los comandos aceptados por el cliente se puede apreciar también en el anexo correspondiente. También existe ayuda en línea mediante el comando *help <nombre de comando opcional>.*

Todos los nombres de comando pueden ser modificados para reflejar mejor la terminología local sin necesidad de programación.

Existe una secuencia de tareas a cumplir para poder medir el algoritmo en este marco de trabajo:

- Construcción de una implementación del algoritmo.
- Enviar el algoritmo al servidor para su almacenamiento y posterior ejecución.

- Establecer el ambiente de trabajo de referencia.
- Ejecución específica del algoritmo.
- Recolección de métricas.

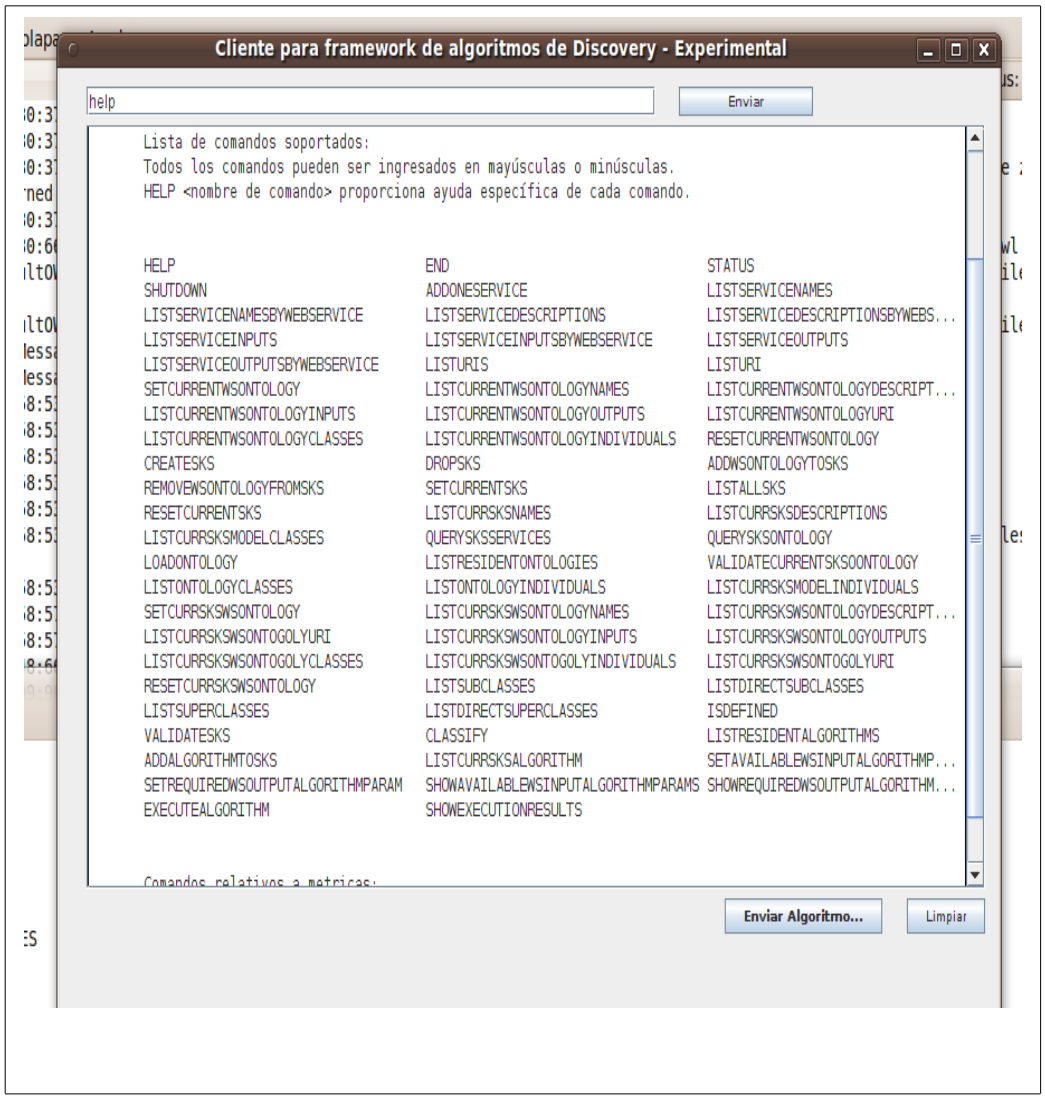


Figura 13: El cliente del ambiente de trabajo

La figura 13 muestra salida de la ejecución del comando "*help*", que

provee un listado de los comandos aceptados por el servidor.

6.1. Construcción de una implementación del algoritmo

Para que el algoritmo pueda ser ejecutado y medido en el marco de trabajo propuesto, existen restricciones en el momento de la implementación del algoritmo:

1. Debe ser implementado en Java. Para poder ejecutarlo de manera tal que no existan diferencias atribuibles al ambiente de ejecución es necesario establecer un ambiente de ejecución común. La elección de este lenguaje de programación obedece a varios factores, entre los que se destacan: a) su amplia difusión y b) sus condiciones de licenciamiento.
2. Debe utilizar la Discovery Virtual Machine, que será su interfaz con todos los servicios que provee el ambiente. Como quedó establecido en el punto anterior, la Discovery Virtual Machine tiene como funciones a) proveer un punto de acceso del algoritmo a las funciones y servicios que ofrece el ambiente, b) proveer una interfaz común para todos los algoritmos, unificando criterios para resolver servicios que pueda necesitar el algoritmo c) servir de punto de recolección de métricas.
3. Debe cumplir con cierta interfaz. Es necesario que el algoritmo

cumpla con cierto protocolo para su invocación y para su control. El cumplimiento de este requisito se logra exigiendo que el algoritmo herede de una clase base de la que puede sobrecargar solamente algunos métodos, mientras que otros se mantienen intactos y son de manejo interno para la Discovery Virtual Machine, que es el objeto que se utiliza para proveer la comunicación entre el algoritmo y su ambiente.

6.2. Envío del algoritmo al servidor para su almacenamiento y posterior ejecución.-

Una vez que se confecciona una implementación del algoritmo deseado, el algoritmo se hace conocer por el entorno de trabajo. Debido a las características elegidas para el entorno de trabajo en cuanto a su arquitectura, se debe utilizar un cliente para el envío del algoritmo (una clase java que contiene la implementación del algoritmo) al servidor. El servidor almacenará este algoritmo en un repositorio específico de algoritmos y lo hará disponible cuando se lo solicite.

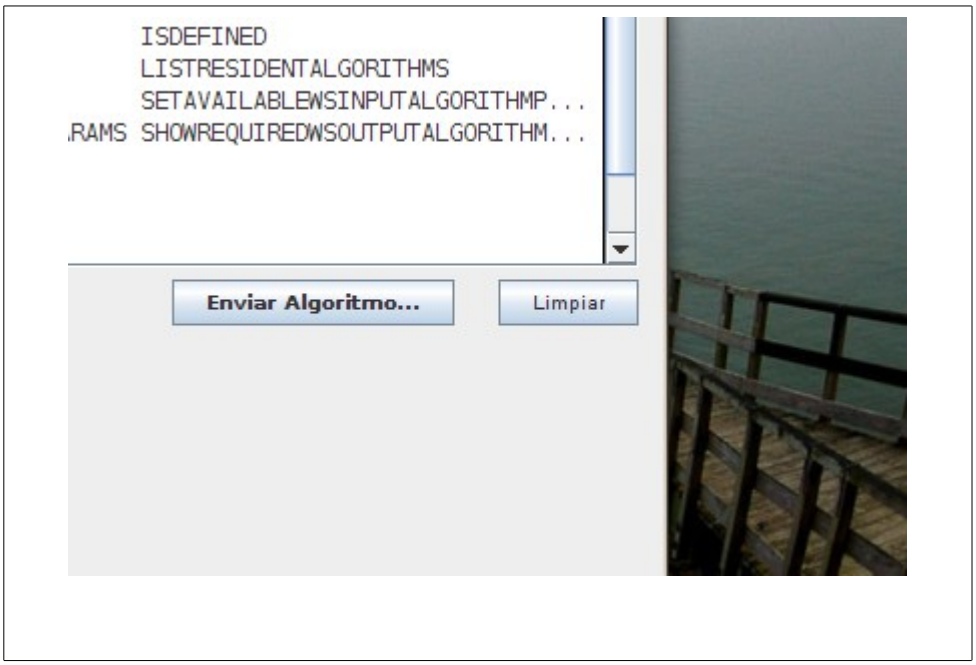


Figura 14: Detalle del cliente, función de envío de algoritmos al servidor.

Luego de seleccionar el algoritmo del sistema de archivos local y de la confirmación necesaria, el mismo es enviado al servidor donde quedará almacenado en el repositorio de algoritmos, quedando disponible para su uso posterior.

6.3. Establecer un ambiente de trabajo de referencia.-

Establecer un ambiente de trabajo de referencia es crear el espacio donde se ejecutará el algoritmo y donde buscará los servicios. Asimismo, si se desean recolectar métricas de búsqueda (como por ejemplo *recall* o *precision*) se deberá establecer cual es el conjunto de servicios que se

espera que el algoritmo encuentre en función de los parámetros de entrada que se le provean. Este espacio de trabajo, que a su vez provee funciones de razonamiento y búsqueda en una ontología de referencia (también llamada *ontología de dominio*) es el que recibe la denominación de *Services and Knowledge Space* (SKS). En Síntesis, establecer un ambiente de trabajo de referencia implica:

- Creación de un SKS.
- Proveer al SKS de una ontología de referencia.
- Poblar el SKS con descripciones de servicios web para realizar la búsqueda.
- Asociar un algoritmo seleccionado al SKS.
- Establecer los parámetros de funcionamiento del algoritmo.
- Opcionalmente definir la recuperación esperada.

6.3.1. Creación de un SKS.

Un SKS se crea enviando una orden al servidor para crearlo. Para crear un espacio de conocimiento y servicios, lo que se logra a través del comando *createsks <nombre_arbitrario>*.

En nuestro caso específico utilizaremos *createsks Test*[23].- Podemos crear hasta 10,000 espacios de trabajo por sesión e interactuar de manera alternada sobre ellos, no existiendo mayores restricciones para ello.

23 El servidor responde “000” si el comando se completa con éxito.-

El servidor le asigna un *handle* a cada sks creado, que es necesario para realizar operaciones sobre ellos (establecer el espacio de trabajo corriente, cargarle publicaciones, ontologías, etc.)

Para conocer el handle asociado a nuestro sks se procede con el comando *listallsks.* El comando devuelve el handle seguido del nombre de cada uno de los sks creados. El resultado de la ejecución del comando se presenta en la figura 15.

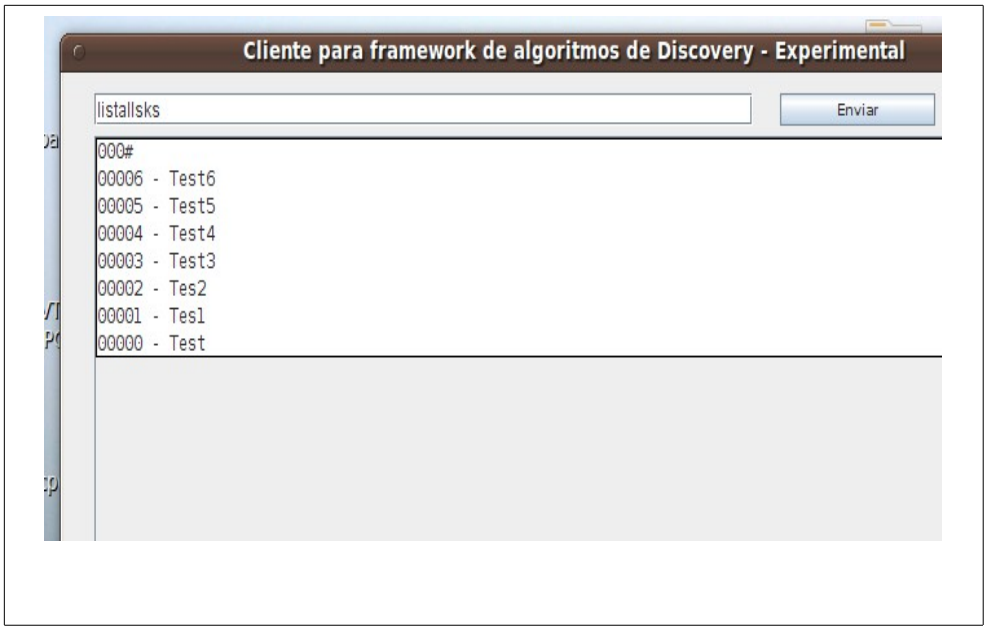


Figura 15: Listado de SKS

Finalmente se establece el espacio de trabajo actual, que será aquel sobre que se desarrolle la interacción. A partir de la ejecución de este comando, todos los subsiguientes se refieren al espacio actual. Esto se logra mediante la ejecución del comando *setcurrentsks <handle>* (en el

ejemplo que se presenta *setcurrentsks 0*

6.3.2 Proveer al SKS de una ontología de referencia.

El entorno de trabajo posee la capacidad de almacenar ontologías en la base de datos, lo único necesario es asociar, nuevamente mediante un comando, una ontología de referencia a un SKS. Es posible listar las ontologias almacenadas en la base de datos, lo que se consigue a través del comando *listresidentontologies*

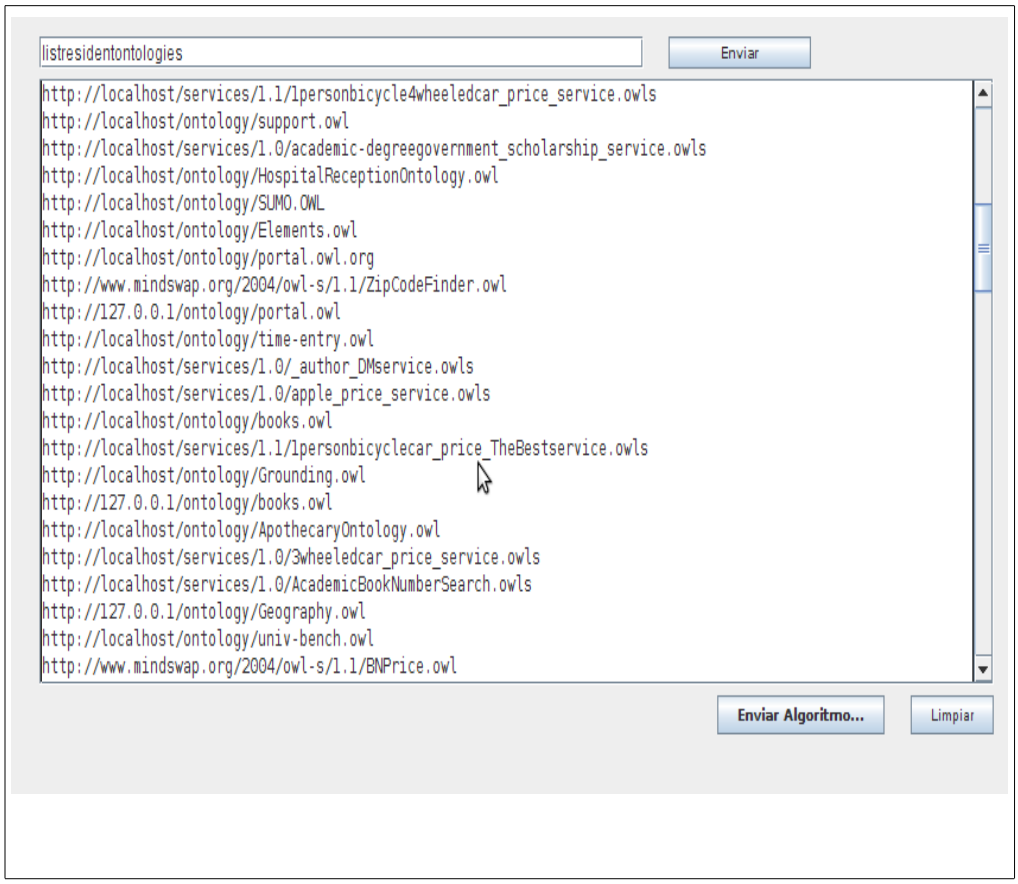


Figura 16: Listado de las ontologías residentes en la base de datos en nuestra prueba

Para asociar una ontología de dominio al sks actual se procede con el comando *loadontology <url>,* cuyo resultado se puede apreciar en la figura 16.

En nuestro ejemplo vamos a seleccionar http://127.0.0.1/ontology/simplified_sumo.owl

6.3.3 Poblar el SKS con descripciones de servicios web para realizar la búsqueda.

Del mismo modo que se almacenan otologías, existe un repositorio de publicaciones (descripciones de servicios web) que abarcan distintas áreas de conocimiento, del mismo modo que lo hacen las ontologías de referencia que se pueden seleccionar. Para dotar al SKS creado con servicios para su búsqueda, se invoca un conjunto de comandos que dejan asociados las descripciones de servicios al SKS corriente.

El primer paso es seleccionar las publicaciones sobre las que se desea trabajar, las que se pueden listar a través del comando *listservicenames*[24].

Este comando en particular devuelve los nombres de los servicios y sus *handles* correspondientes, como lo muestra la figura 17.

24 Existe un conjunto importante de comandos que permite administrar los repositorios tanto de descripciones de servicios web como de ontologías. Para mas información relativa los mismos referirse al apéndice correspondiente.

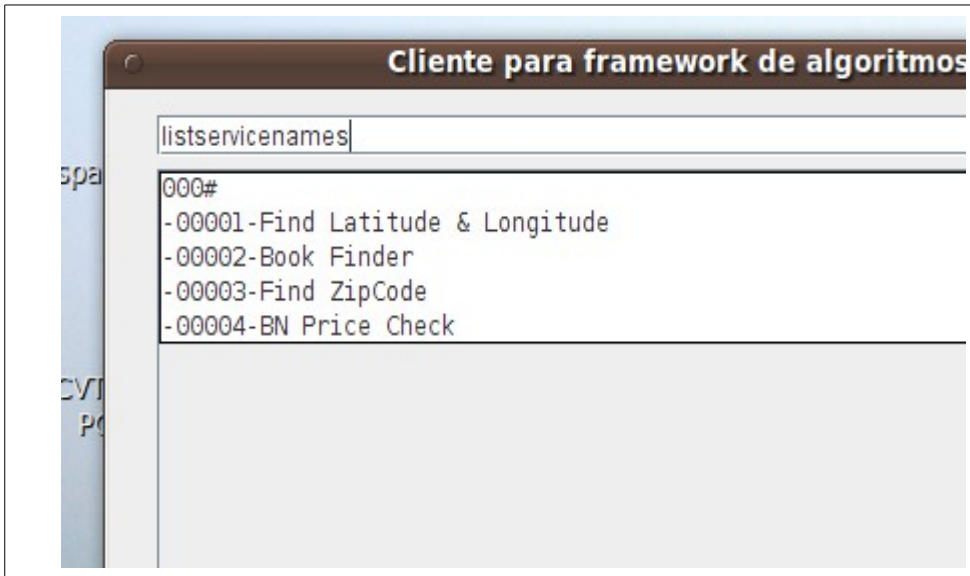


Figura 17: Listado de publicaciones disponibles en el ejemplo

Para el caso solo disponemos de 4 publicaciones de servicios web. Es posible cargar mas publicaciones dinámicamente.

Para asociar estas descripciones ontológicas de servicios web (publicaciones) al sks recientemente creado se digitará *addwsontologytosks <sks_handle> <web_service_ontology_handle>* Para nuestro caso específico será: addwsontologytosks 0 1 addwsontologytosks 0 2 addwsontologytosks 0 3 y addwsontologytosks 0 4

Existe un conjunto de tareas opcionales que se puede alcanzar con

distintos comandos a partir de este momento, como validar el espacio de trabajo, validar la ontología de dominio, ejecutar consultas sparql tanto sobre la ontología de dominio como sobre el conjunto de publicaciones, listar clases, individuos, proceder a la clasificación, listar clases ancestro directas o subclases directas, etc. Todos los comandos relacionados con estas tareas se encuentran descritos en el anexo correspondiente.

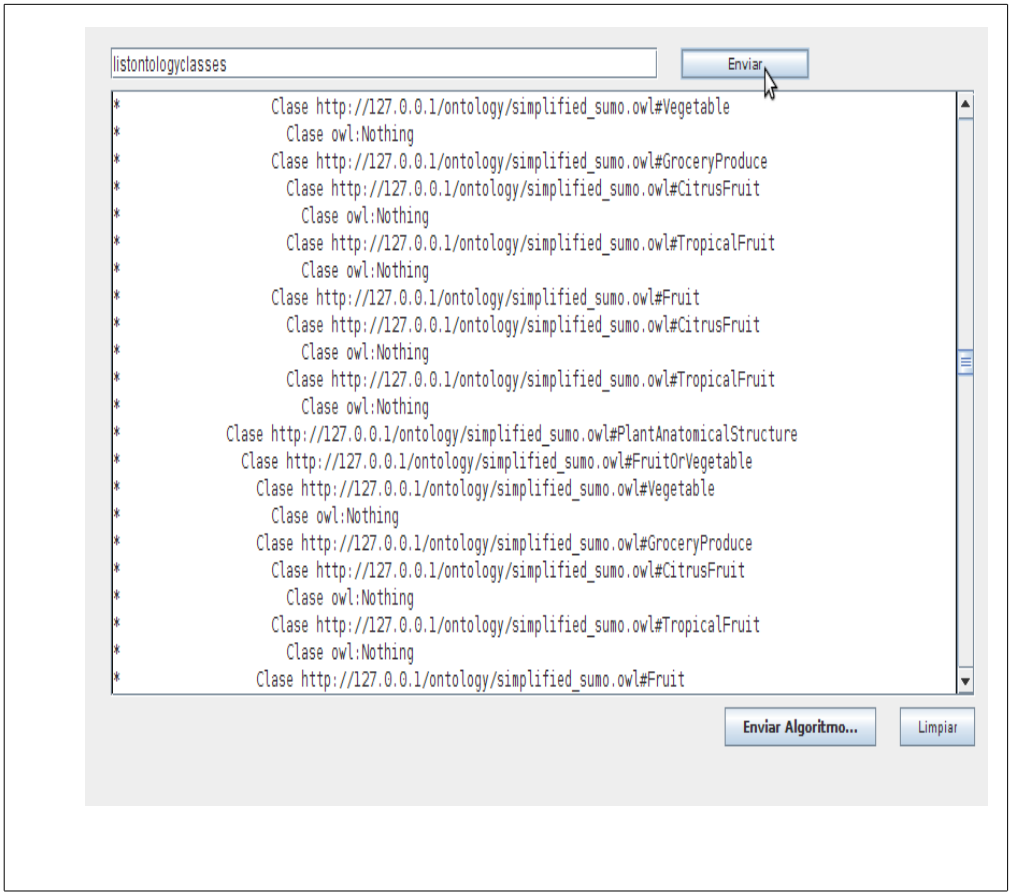


Figura 18: Listado de clases de la ontología de referencia (*listontologyclasses)*.

El listado de clases que se aprecia en la figura anterior y su indentación jerárquica es posible gracias a las funciones de razonamiento provistas por el entorno de trabajo.

Si no se dispusiera de un razonador, o este estuviera desconectado del ambiente se obtendría el mensaje de error de la figura 19:

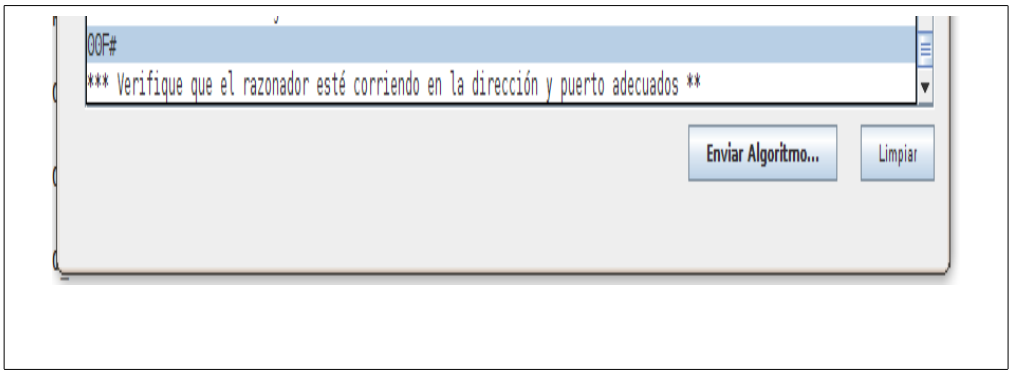


Figura 19: Mensaje de error que se produce al no disponer de funciones de razonador.

En este punto el espacio de trabajo resultante está dotado de un conjunto de publicaciones y de una ontología de referencia.

6.3.4 Establecer los parámetros de funcionamiento del algoritmo.

Un algoritmo desarrollado debe ser cargado al entorno de trabajo y posteriormente asociado al espacio de sevicios (sks) corriente.

El repositorio de algoritmos previamente cargados se puede consultar a través del comando *listresidentalgorithms,* tal como se muestra en la figura 20.

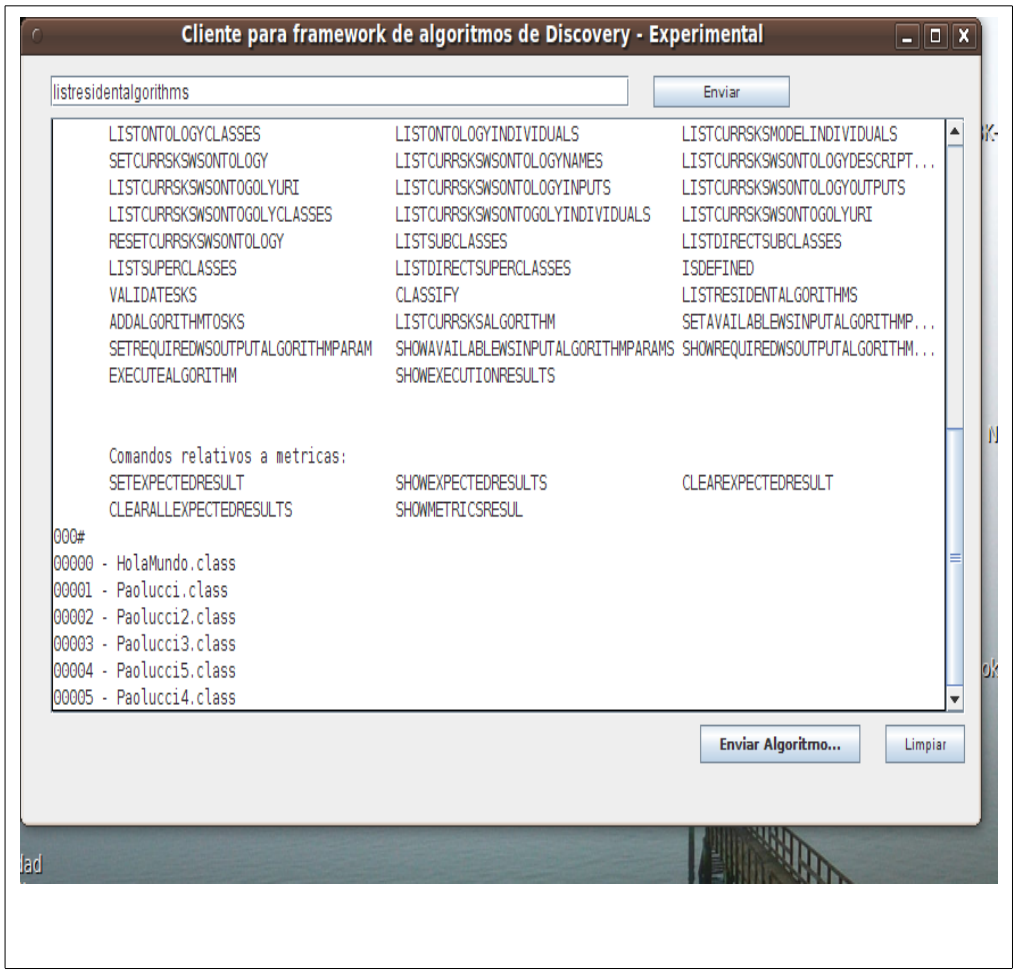


Figura 20: Listado de algoritmos del repositorio.

Nuevamente el valor que se presenta precediendo el nombre del archivo de cada algoritmo es el *handle* del algoritmo correspondiente.

Para asociar el algoritmo con el sks actual se procede con el comando *addalgorithmtosks <handle_sks> <handle_algoritmo>.* En nuestro caso vamos a seleccionar el algoritmo “Paolucci2.class”. Es mismo es una variante a la presentada en el anexo, donde se incluyen numerosas invocaciones a la api *say* de la *DiscoveryVirtualMachine* con el único motivo de poder mostrar en pantalla el funcionamiento interno del algoritmo.

De acuerdo a la figura anterior, para la asociación se ejecuta el comando *addalgorithmtosks 0 2.* En este punto se dispone de un espacio de servicios que contiene publicaciones a ser buscadas, un algoritmo de búsqueda y una ontología de referencia.

Establecer los parámetros de funcionamiento del algoritmo es proveer información de los parámetros de I/O requeridos para que una publicación de servicio web satisfaga la búsqueda. El grado de concordancia semántica obtenido en la búsqueda de cada publicación declarada en un SKS permitirá establecer un criterio de relevancia para el ordenamiento final que el algoritmo retornará. La decisión del orden de las publicaciones retornadas es decisión del algoritmo, pero la Discovery Virtual Machine provee mecanismos para identificar el grado de concordancia (*degree of match*) de una publicación respecto a un requerimiento.

En el caso particular del ejemplo planteado se asume que el algoritmo va a buscar un servicio web que dado una cadena de caracteres el servicio

le devuelve un libro.

Si se listan los servicios asociados al espacio de búsqueda actual (comando *lisrtcurrsksnames*), se observa que el servicio cuyo nombre es “Book Finder” puede ser apropiado, tal como se observa en la figura 21.

Figura 21: Servicios disponibles en el SKS actual.

Se puede analizar la composición de este servicio tal como está publicado, para ello se procede a listarlo (figura anterior), establecerlo como el servicio actual dentro del sks y finalmente listar sus parámetros de entrada y salida, lo que se logra con los comandos , *setcurrskswsontology <handle>, listcurrskswsontologyinputs y listcurrskswsontologyoutputs* (figura 22).

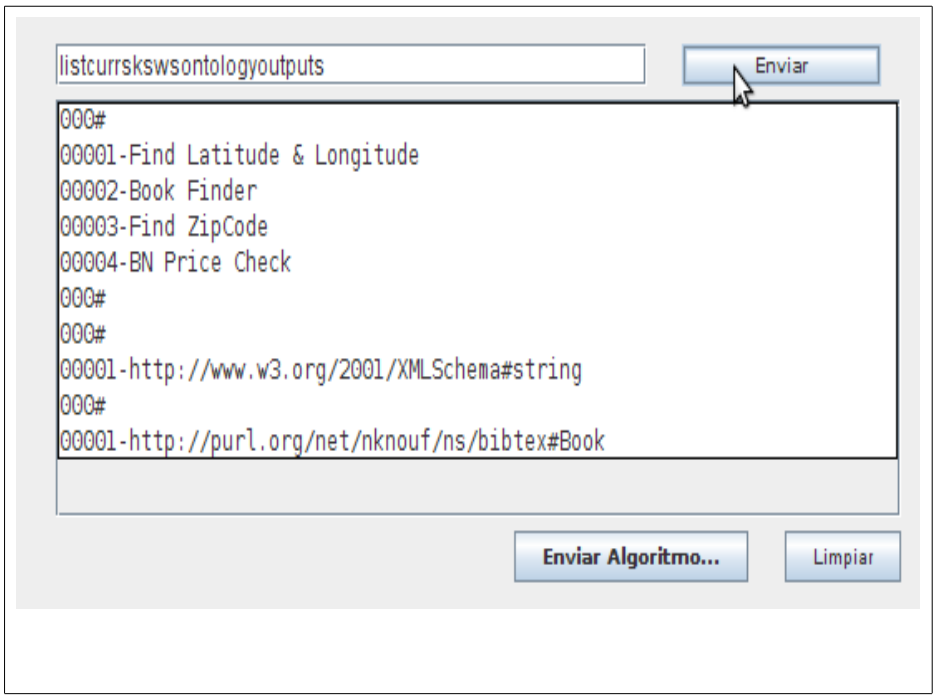


Figura 22: Listado de entradas y salidas de "Book Finder"

Para que el algoritmo encuentre algún grado de concordancia entre el requerimiento del usuario y la publicación devuelta, se le deben comunicar al algoritmo los parámetros disponibles (que serán parámetros de entrada de un servicio web) y los parámetros requeridos al servicio web (que es lo que el usuario necesita).

Para simplificar se proporciona al algoritmo a ejecutar esos parámetros, lo que se logra mediante los comandos *setavailablewsinputalgorithmparam <uri>* y *setrequiredwsoutputalgorithmparam <uri>*

En el caso particular del ejemplo estos comandos toman la siguiente

forma:

setavailablewsinputalgorithmparam

http://www.w3.org/2001/XMLSchema#string

y

setrequiredwsoutputalgorithmparam

http://purl.org/net/nknouf/ns/bibtex#Book

No existen restricciones al agregado o eliminación de parámetros de entrada o salida. En el ejemplo se establece un parámetro disponible como entrada para el servicio web y un parámetro requerido como salida.

Con el único objetivo de observar la recolección automática de *Recall* y *Precision* se establece que el servicio web denominado “Book Finder” es nuestro *resultado esperado* de la ejecución del algoritmo. Esto se logra mediante el comando *setexpectedresult* Book Finder.

Nuevamente aquí no existen restricciones a agregar, quitar o examinar resultados esperados, incluso es posible trabajar sin resultados esperados (lo que elimina la posibilidad en este caso del cálculo de *Recall* y *Presicion*).

6.4. Ejecución específica del algoritmo.

La ejecución del algoritmo se logra a través del comando *executealgorithm*

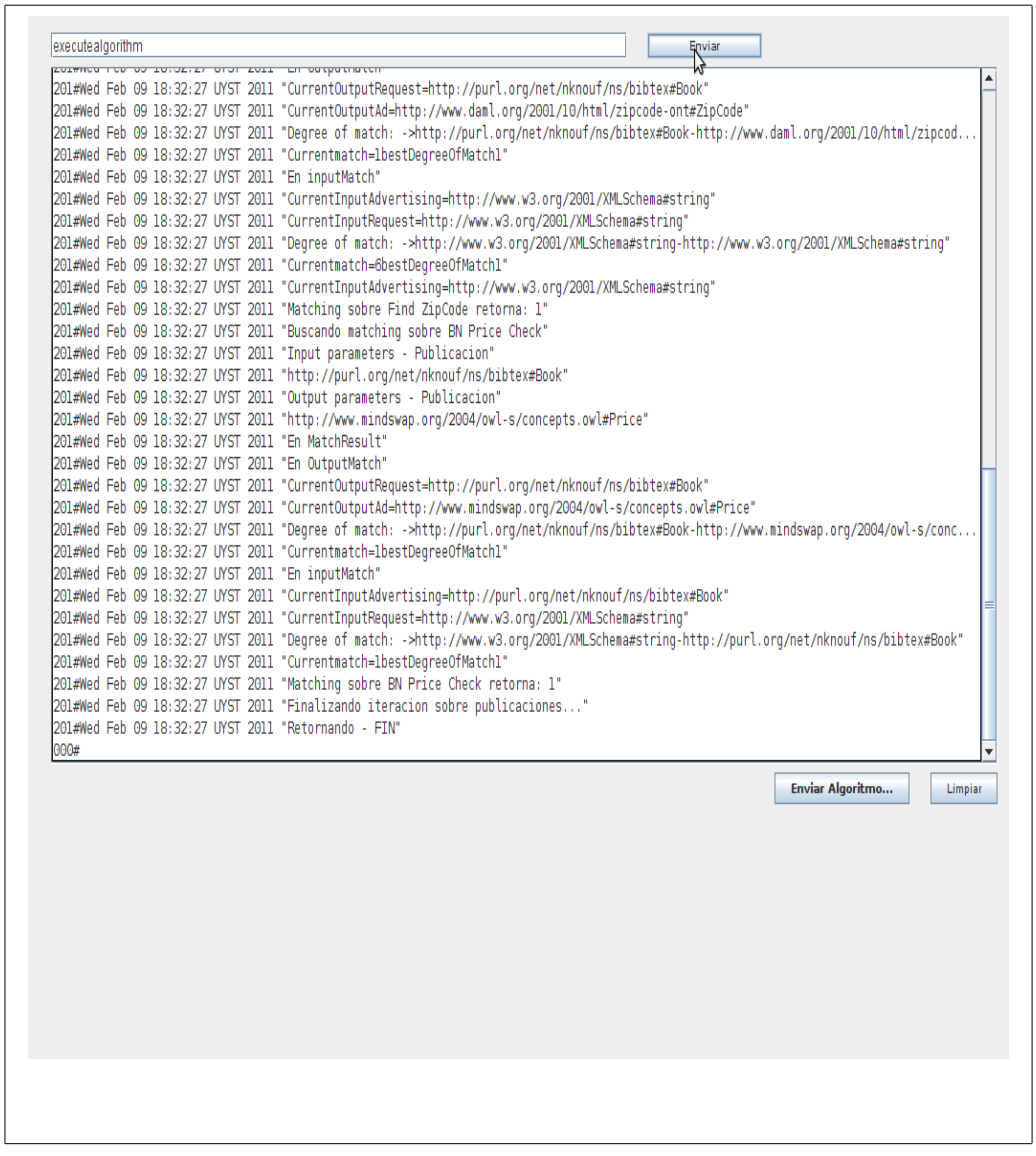


Figura 23: Trazas de ejecución del algoritmo de Paolucci

En la figura 23 se pueden observar las trazas que va dejando el algoritmo a lo largo de su ejecución. Si no se producen fallas o excepciones en la ejecución del algoritmo el servidor lo indica, del mismo modo que en la ejecución de cualquier comando, comunicando al cliente el código de respuesta “000”.

Los resultados que devuelve el algoritmo se pueden solicitar al servidor mediante el comando *showexecutionresults* que se muestra en la figura 24.

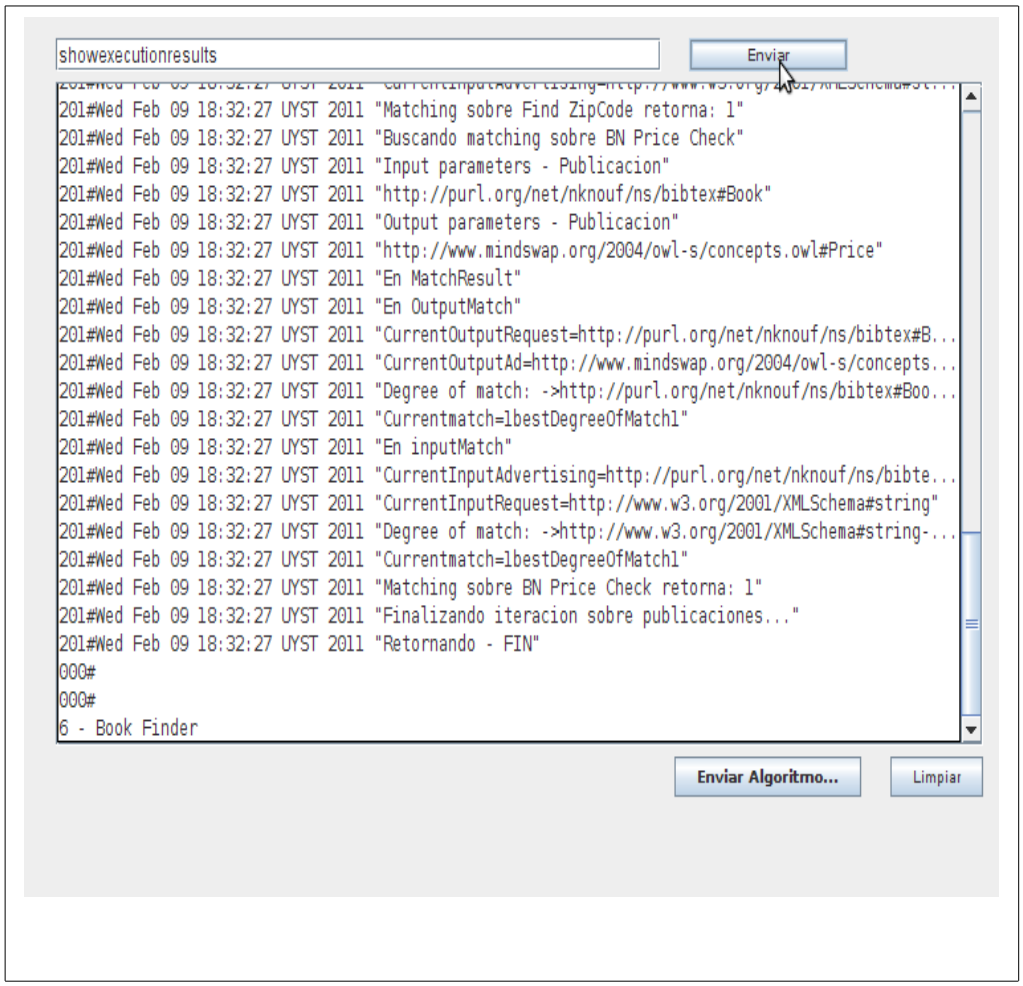


Figura 24: Resultado de la ejecución.

En la figura se puede apreciar que el algoritmo fue efectivamente capaz de encontrar un servicio de nombre “Book Finder” que coincide con nuestro resultado esperado.

6.5 Recolección de métricas.

Las métricas se recolectan de manera automática, la recolección

comienza y finaliza con el inicio y finalización del algoritmo. Una vez que el algoritmo finaliza solo se debe solicitar al servidor que muestre las métricas recolectadas.-

En tiempo de ejecución se puede ejemplificar la interacción de los componentes mediante la figura 25:

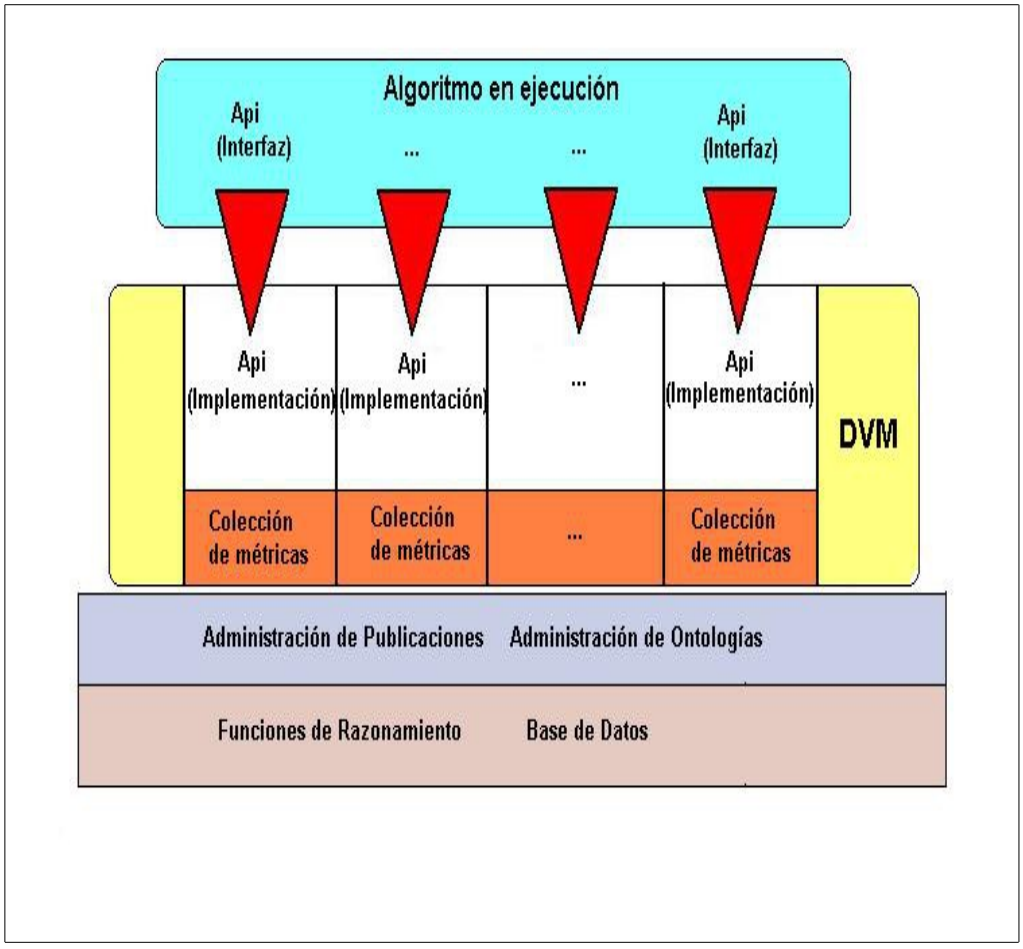


Figura 25: La recolección de métricas en tiempo de ejecución.

El algoritmo en ejecución interactúa con el entorno de trabajo a través de la interfase provista. Esa interacción se esquematiza en rojo en la figura 25. La implementación de ésta interfaz se encuentra embebida en la *Discovery Virtual Machine.*

A su vez, cada implementación tiene acceso a los servicios de mas bajo nivel (administración de Ontologías, administración de publicaciones, realización de consultas en SPARQL u otros) Estos servicios se hacen disponibles a la DVM en cada sesión por el administrador de sesiones. Es precisamente a nivel de sesión que se restringe la ejecución de un algoritmo al espacio de trabajo especificado (SKS).

Las implementaciones de las API esconden la invocación correspondiente a la recolección de métricas. Esto se hace mediante la invocación en la entrada y en la salida de cada rutina de los métodos relacionados con las métricas.

Para el caso del ejemplo, posteriormente a la ejecución del algoritmo se solicitan las métricas recolectadas mediante el comando *showmetricsresults*

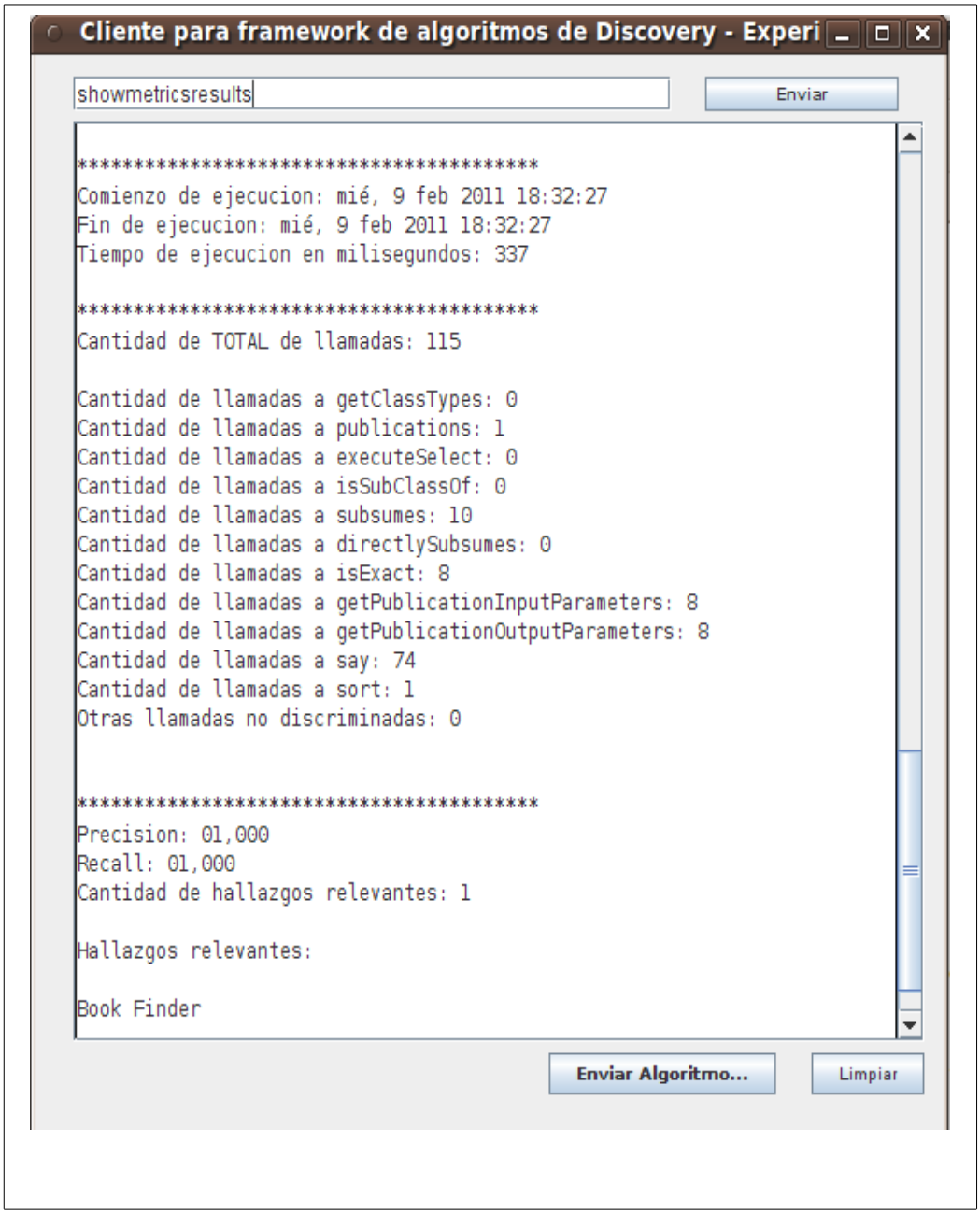


Figura 26: La recolección de métricas en el ejemplo planteado.

El caso presentado en la figura 26 se muestran valores peculiares para

recall y *precision.*

Es posible analizar que pasaría con estos valores si se esperaran mas resultados válidos. Para ello se puede agregar al conjunto de resultados esperados otro servicio, como por ejemplo “BN Price Check” (*setexpectedresult* BN Price Check).

Al ejecutar nuevamente el algoritmo (*executealgorithm*) y solicitar las métricas el resultado obtenido sería el que se muestra en la figura 27:

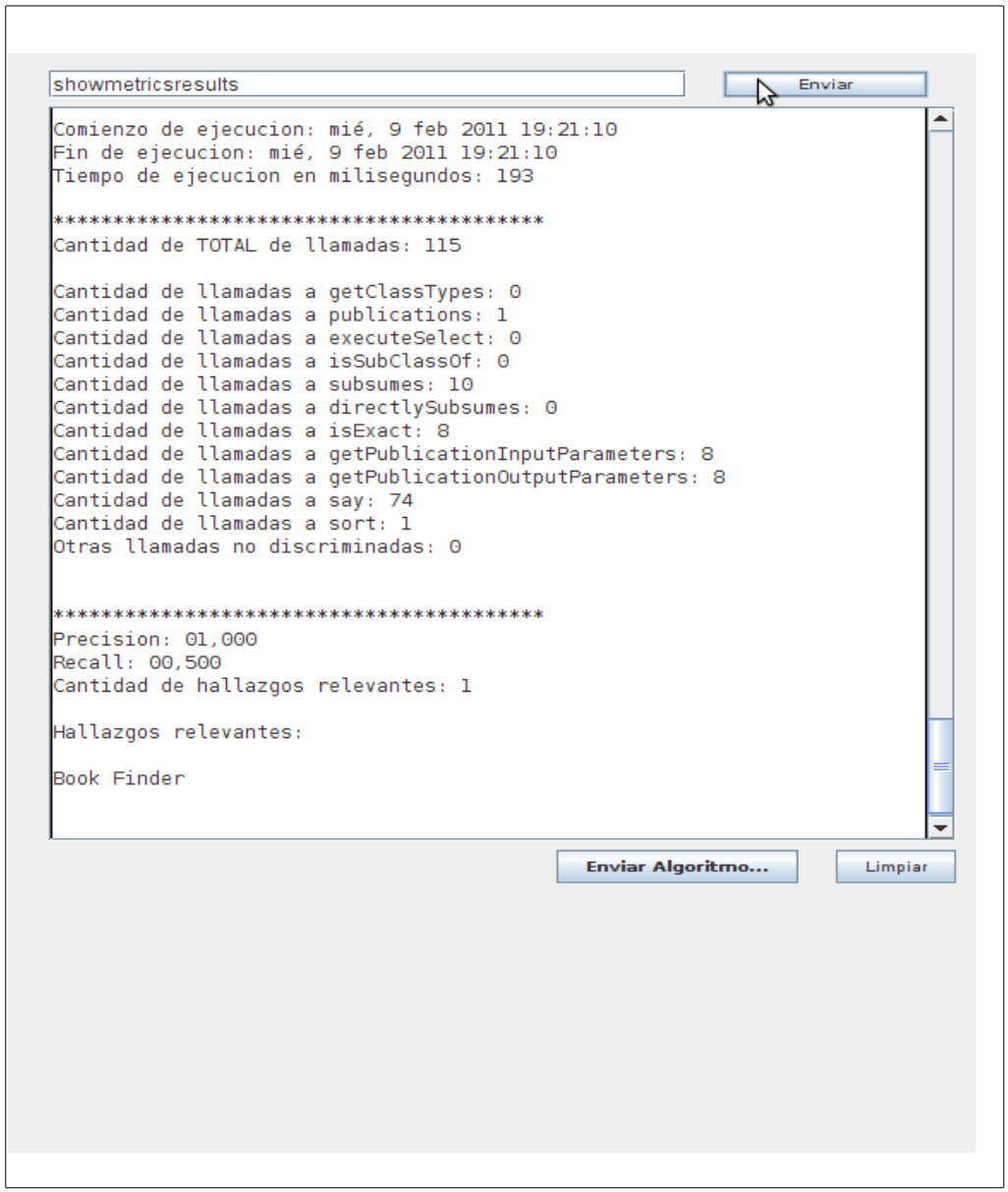


Figura 27: Variación de las métricas de acuerdo al nuevo escenario

En el ejemplo planteado se agregó un nuevo servicio como resultado esperado. Como era de esperar el algoritmo vuelve a encontrar el servicio

que había encontrado en la corrida anterior, por lo tanto se detecta una variación en el valor de la métrica de *Recall* que se produce al hacer variar el conjunto de servicios relevantes esperados de la ejecución. El valor encontrado concuerda con el que era esperable encontrar, dado que:

Recall: Cantidad de servicios relevantes recuperados / Cantidad de servicios relevantes

En el caso planteado, Recall = ½ = 0,5 , que es el valor que arroja el entorno de trabajo.

7. Conclusiones

Se ha desarrollado un entorno de trabajo que potencia las alternativas disponibles para:

1. Permitir construir algoritmos con metas cuantificables y predefinidas.

   Es posible a partir del entorno de trabajo provisto desarrollar un algoritmo con una meta específica preestablecida y medible. La utilización del entorno de trabajo permite identificar cuando se han alcanzado objetivos concretos expresados en términos de métricas. Es posible conocer si un algoritmo cumple con determinadas especificaciones mediante una medición objetiva, simplificando las tareas de testing al proveer mecanismos automáticos para la verificación de especificaciones no funcionales.

   Permite establecer con precisión en las especificaciones de un algoritmo respecto a métricas que se deben cumplir, y provee un mecanismo objetivo de certificación.

2. Comparar el trabajo realizado con patrones objetivos.

   Es factible desarrollar y refinar continuamente un algoritmo, encarando un proceso sistemático y continuo de mejora, siendo posible medirlo contra versiones anteriores utilizando parámetros específicos.-

Dados algunos criterios específicos de aceptación de un algoritmo expresados en modo de métricas, a partir de la utilización del entorno de trabajo es posible definir el momento preciso en que el algoritmo supera las métricas preestablecidas.

De este modo es entonces posible determinar dentro de un conjunto de algoritmos de matchmaking cual de ellos se comporta de manera mas adecuada a nuestros fines basados en criterios objetivos y cuantificables de funcionamiento y que a su vez son reproducibles.

3. Automatización de la tarea de recolección de métricas.

   La métrica es un indicador, con una interpretación y un contexto asociados, que compara un aspecto específico de algo, en nuestro caso un algoritmo. La automatización y utilización de herramientas para la medición y recolección de métricas elimina del proceso todos los factures humanos que puedan incidir en la medida introduciendo errores.

   En la medida en que la cantidad de métricas que deseamos conocer crece, el proceso de extracción de las mismas se hace cada vez mas complejo. Proveer un ambiente donde la recolección de métricas sea automática favorece y promueve la medición y por lo tanto el conocimiento mas profundo de los algoritmos utilizados.

4. Disminuir el esfuerzo necesario para la reproducción de contextos.

   Conjuntamente con la recolección de métricas, es necesario poder registrar el contexto en que se extraen estas métricas. Esto es necesario para poder reproducir los resultados obtenidos.

   Los resultados y mediciones que se extraigan de la ejecución de los algoritmos no tienen valor si no se pueden reproducir y comparar entre si.

   Estos resultados pueden depender del contexto en el que se obtengan, es decir, pueden depender de la ontología de dominio, de la cantidad de publicaciones donde el algoritmo deba buscar, o incluso del razonador con el que se trabaje.

5. Administración de juegos de prueba y ambientes de trabajo.

   El entorno en el que se ejecutan estos algoritmos es un entorno complejo debido a varios factores, entre los que resaltan los recursos que se utilizan (razonadores, repositorios de publicaciones, ontologías de referencia).

8. Trabajo Futuro

- Enriquecimiento/extensión de la DVM

  La DVM es el punto de contacto entre el entorno de trabajo y los distintos algoritmos de búsqueda semántica que se deseen medir. Por ese motivo el enriquecimiento de la Discovery Virtual Machine permitirá a los algoritmos que se desarrollen realizar tareas que hasta el momento no puedan realizar.

  En este punto específico existen muchas mejoras posibles, como por ejemplo:

- Flexibilizar el protocolo de llamada que deben seguir los algoritmos que se ejecutan en el entorno de trabajo. Hasta el momento el protocolo de llamada está predefinido y todos los algoritmos que son ejecutados son invocados mediante este protocolo. Es posible mejorar este aspecto permitiendo flexibilizar el desarrollo de algoritmos.

- Enriquecer las funciones específicas de la DVM. Si la Discovery Virtual Machine puede realizar mas funciones, estas funciones se harán disponibles a los nuevos algoritmos que se deseen desarrollar (a través de una biblioteca que se libera, como interfaz a la DVM).

- Incorporación de nuevas métricas. Incorporar nuevas métricas en

la DVM permitirá descubrir nuevos aspectos de los algoritmos estudiados. Esto permitirá agregar nuevas dimensiones de conocimiento y comparación entre los distintos algoritmos que se desarrollen.

- Potenciar el manejador de sesiones.

  El manejador de sesiones es la pieza del entorno de trabajo que interactúa directamente con el cliente del mismo. Incrementar la funcionalidad de manejador de sesiones significa aumentar las posibilidades de lo que puede realizar un cliente.

  En este punto es posible mejorar la clasificación de juegos de pruebas y resultados como primer paso inmediato.

- Mejoras en el cliente.

  El cliente es la “interfaz con el humano” del entorno de trabajo. Mejorar el cliente tiene impacto directo en la practicidad y facilidad de utilización del mismo. Entre las mejoras posibles al cliente destacamos:

- Introducción de un lenguaje de srcipting o ejecución de macros. A través de un lenguaje de scripting es posible automatizar completamente una sesión de trabajo. Dado que la comunicación entre el cliente y el servidor se desarrolla a través de comandos y parámetros documentados, enviados desde el cliente al servidor y

sus correspondientes respuestas, es posible introducir un parser que interprete un lenguaje sencillo y traduzca cada uno de los mensajes y que los mismos puedan ser leídos desde un archivo. Un archivo así confeccionado puede ser interpretado por ese parser de manera de agilizar el trabajo de establecer los ambientes de prueba y la ejecución de cada algoritmo.

- Mejoras visuales. Las mejoras visuales que se puedan introducir harán mas amigable y deberán guiar al usuario de manera de una manera intuitiva.
- Grabación de sesiones. La grabación de sesiones deberá tener una doble finalidad. La primera como registro de lo ocurrido en una sesión de manera de permitir comunicar y almacenar los resultados obtenidos en las distintas pruebas y mediciones realizadas. El segundo cometido es permitir generar un archivo que pueda ser el punto de partida del lenguaje comentado en el primer párrafo.

Bibliografía

[1] ***Web Services Glossary***, W3C, Editores Hugo Haas W3C, Allen Brown Microsoft Corporation, disponible en Internet 15/12/2006 en http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[2] ***Understanding Web Services XML, WSDL, SOAP and UDDI***, Eric Newcomer, Independent Technology Guides, Pub. Addison-Wesley Professional, 2002, ISBN 978-0-201-75081-2

[3] ***UDDI v3: The Registry Standard for SOA,*** OASIS UDDI Specification Technical Committee, disponible en Internet el 15/11/2006 en http://www.oasis-open.org/presentations/uddi_v3_webcast_20050222.pdf

[4] ***UDDI Version 3 Features List Richard Harrah, Hewlett-Packard Company,*** Sam Lee, Oracle Corporation, Joel Munter, Intel Corporation Claus Von Riegen, SAP AG, editado por Karsten Januszewski, Microsoft Corporation, Ed Mooney, Sun Microsystems, Inc., 2004, disponible en Internet el 16/11/2006 en http://www.uddi.org/pubs/uddi_v3_features.htm

[5] ***UDDI Spec Technical Committee Draft***, Tom Bellwood, Massimo Paolucci, Katia Sycara *et al,* editado por Luc Clement, Systinet, Andrew Hately, IBM, *et al* 19/10/2004, Disponible en Internet el 16/11/2006 en http://uddi.org/pubs/uddi_v3.htm

[6] ***World Wide Web Consortiun***, Ossi Nykänen W3C, disponible en Internet el 15/12/2006 en http://www.w3c.tut.fi/talks/2003/0331umedia-on/slide6-0.html

[7] ***OWL Web Ontology Language Overview***, W3C Recommendation 10 Feb 2004, Deborah L. McGuinness (Knowledge Systems Laboratory, Stanford University), Frank van Harmelen (Vrije Universiteit, Amsterdam), W3C 2004, disponible en Internet el 20 de enero de 2007 en http://www.w3.org/TR/2004/REC-owl-features-20040210/

[8] ***Choosing an Ontology Languaje***, Anna V. Zhdanova, Uwe Keller, Treansactions on Engineering, Computing and Technology, V4 Feb 2005, pub: World Enformatika Societym, ISSN 1305-5313, Disponible en internet el 20 de enero de 2007 en http://www.waset.org/pwaset/v4/v4-13.pdf

[9] ***SWRL: A Semantic Web Rule Language Combining OWL and RuleML***, W3C Member Submission 21 May 2004, Ian Horrocks, Network Inference, Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies, Harold Boley, National Research Council of Canada, Said Tabet, Macgregor, Inc. Benjamin Grosof, Sloan School of Management, MIT, Mike Dean, BBN Technologies, W3C 2004, disponible en Internet el 25 de enero de 2007 en http://www.w3.org/Submission/2004/SUBM-

SWRL-20040521/

[10] ***Web Services Modelling Ontology***, Roman D, Keller U et al, *Applied Ontology Journal 1 – 2005 (p 77 – 106)*, ed: IOS Press, disponible en Internet el 20/07/2006 http://www.debruijn.net/publications/wsmo.pdf

[11] ***D2v1.2. Web Service Modeling Ontology (WSMO), WSMO Final Draft 13 April 2005***, Roman D, Keller U et al, ed: Roman D, Keller U et al, disponible en Internet el 21/07/2006 en http://www.wsmo.org/TR/d2/v1.2/20050413/

[12] ***OWL-S Semantic Markup for Web Services***, Martin D, Burstein M et al, ed: Martin D, disponible en internet el 24/7/2006 en http://www.daml.org/services/owl-s/1.1/overview/

[13] ***Semantic Matching of Web Services Capabilities,*** Proceedings of the First International Semantic Web Conference, LNCS 2342, Massimo Paolucci, Takahiro Kawamura, Terry R. Payne and Katia Sycara, Editor: Springer-Verlag, 2002, disponible en Internet el 22/11/2006 en http://eprints.ecs.soton.ac.uk/7606/01/ISWC2002-Matchmaker.pdf

[14] ***Automated Semantic Web Service Discovery with OWLS-MX, AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems,*** Matthias Klusch, German Research Center for Artificial Intelligence, Multiagent Systems Group Saarbruecken, Germany, Benedikt Fries, University of the Saarland Computer Science Department Saarbruecken, Germany, Katia Sycara, Carnegie Mellon University Robotics Institute, Pittsburgh PA, USA, Editor: ACM Press, disponible en Internet el 27/11/2006 http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/owlsmx-06-final.pdf

[15] ***Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems***, K. Sycara, The Robotics Institute - Carnegie Mellon University, Pittsburg, M. Klusch, German Research Center for Artificial Intelligence, Multiagent Systems Group Saarbruecken, Jianguo Lu, Computer Science Department, Univeristy Of Toronto, Kluwer Academic Publishers, 2002, disponible en interner el 5/12/2006 en www.cs.cmu.edu/~softagents/papers/LARKS.pdf.

[16] ***Web Services Description Language (WSDL) 1.1***, Erik Christensen, Microsoft Corporation, Francisco Curbera, IBM Research, Greg Meredith, Microsoft Corporation, Sanjiva Weerawarana, IBM Research, W3C Note 15 March 2001, disponible en internet el 15/12/2006 en http://www.w3.org/TR/wsdl

[17] ***Web Services Modelling Ontology (WSMO)*** Sena, O., Trabajo no

publicado preparativo de tesis magistral, Facultad de Ingeniería, Universidad de la República, 2008, citado con autorización del autor.

[18] ***WSMX Web Services Execution Environment***, implementación de referencia de WSMO, disponible en internet el 23/07/2007 en http://www.wsmx.org

[19] ***Towards Intelligent Web Services: The Web Service Modeling Ontology (WSMO)***, Cristina Feier, Digital Enterprise Research Institute, Instituto de informatica, Universidad de Innsbruck Dumitru Roman1, Digital Enterprise Research Institute, Instituto de informatica, Universidad de Innsbruck, Axel Polleres, Digital Enterprise Research Institute, Instituto de informatica, Universidad de Innsbruck, John Domingue, Knowledge Media Institute, The Open University, Milton Keynes, UK, Michael Stollberg, Digital Enterprise Research Institute, Instituto de informatica, Universidad de Innsbruck, Dieter Fensel, Digital Enterprise Research Institute, Instituto de informatica, Universidad de Innsbruck., disponible en Internet el 2/1/2007 en http://members.deri.at/~michaels/publications/wsmo-icic.pdf

[20] ***Hacia un modelo genérico para la calidad de los servicios web***, O. Sena, R. Motz, Tesis de maestría no publicada, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay., pub CEDI 2007, Zaragoza, disponible en internet 29/07/2009 en https://www.fing.edu.uy/inco/grupos/csi/wiki/Camaleon/images/b/b0/QoS1.pdf

[21] ***Multidimensional SemanticWeb Services Matching***, Guzmán Llambías, Regina Motz, Alvaro Rettich, Marco Scalone, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay 2008, LA-WEB 2008, disponible en Internet el 29/07/2009
https://www.fing.edu.uy/inco/grupos/csi/wiki/Camaleon/images/2/2d/MDMOnto.pdf

[22] ***Wordnet,*** resultado de búsqueda, disponible en Internet el 8/08/2009 en http://wordnetweb.princeton.edu/perl/webwn?s=ontology

[23] ***A Translation Approach to Portable Ontology Specifications,*** Thomas R. Gruber, Knowledge Systems Laboratory, Departamento de Ciencias de la Computación de la Universidad de Stanford Publicado en *Knowledge Acquisition*, 5(2):199-220, 1993, disponible en Internet el 8/08/2009 en http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[24] ***Encyclopedia of Database Systems***, definición de Thomas R. Gruber, Publicado por Ling Liu and M. Tamer Özsu, Springer-Verlag, 2009, disponible en Internet el 8/08/2009 en http://tomgruber.org/writing/ontology-definition-2007.htm

[25] ***Benchmarking OWL Reasoners,*** J¨urgen Bock1, Raphael Volz1, FZI Research Center for Information Technologies, Karlsruhe, Germany Peter Haase2, Qiu Ji2, Institute AIFB, University of Karlsruhe, Germany, pub Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), disponible en Internet disponible en Internet el 12/07/2008 en http://*www.aifb.uni-karlsruhe.de/WBS/pha/publications/owlbenchmark.pdf*

[26] ***Introducing knwledge graph*** - resultado de búsqueda, disponible en Internet el 15/05/2012 en *http://www.google.com/insidesearch/features/search/knowledge.html*

[27] ***Introducing the Knowledge Graph: things, not strings*** - resultado de búsqueda, disponible en Internet el 16/05/2012 en http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html#!/2012/05/introducing-knowledge-graph-things-not.html

Apéndice A – Diagramas de clase.-

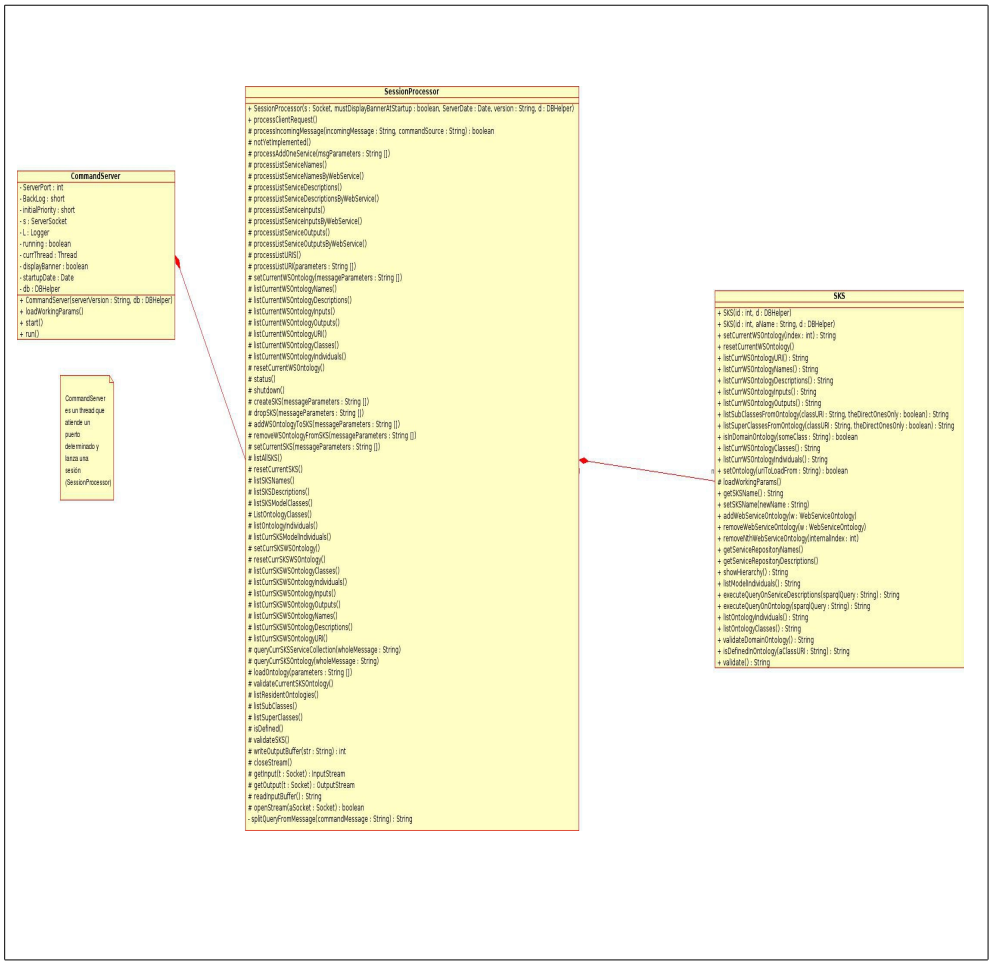
En éste capítulo se presentan las clases mas importantes que componen el entorno de trabajo, excluyendo por tanto las clases secundarias o dependientes de implementación.-

El servidor y el procesador de sesiones

El servidor, es un servidor clásico multithreaded que tiene asociado un procesador de sesiones. El procesador de sesiones es el responsable de interpretar y resolver los comandos que envía el cliente en esta sesión.-

Por otro lado, a cada sesión le corresonde un número no acotado de SKS, de manera de permitir que en una misma sesión se puedan medir distintos algoritmos en distintos ambientes de trabajo asociado.-

Estas relaciones se explican en la figura siguiente:

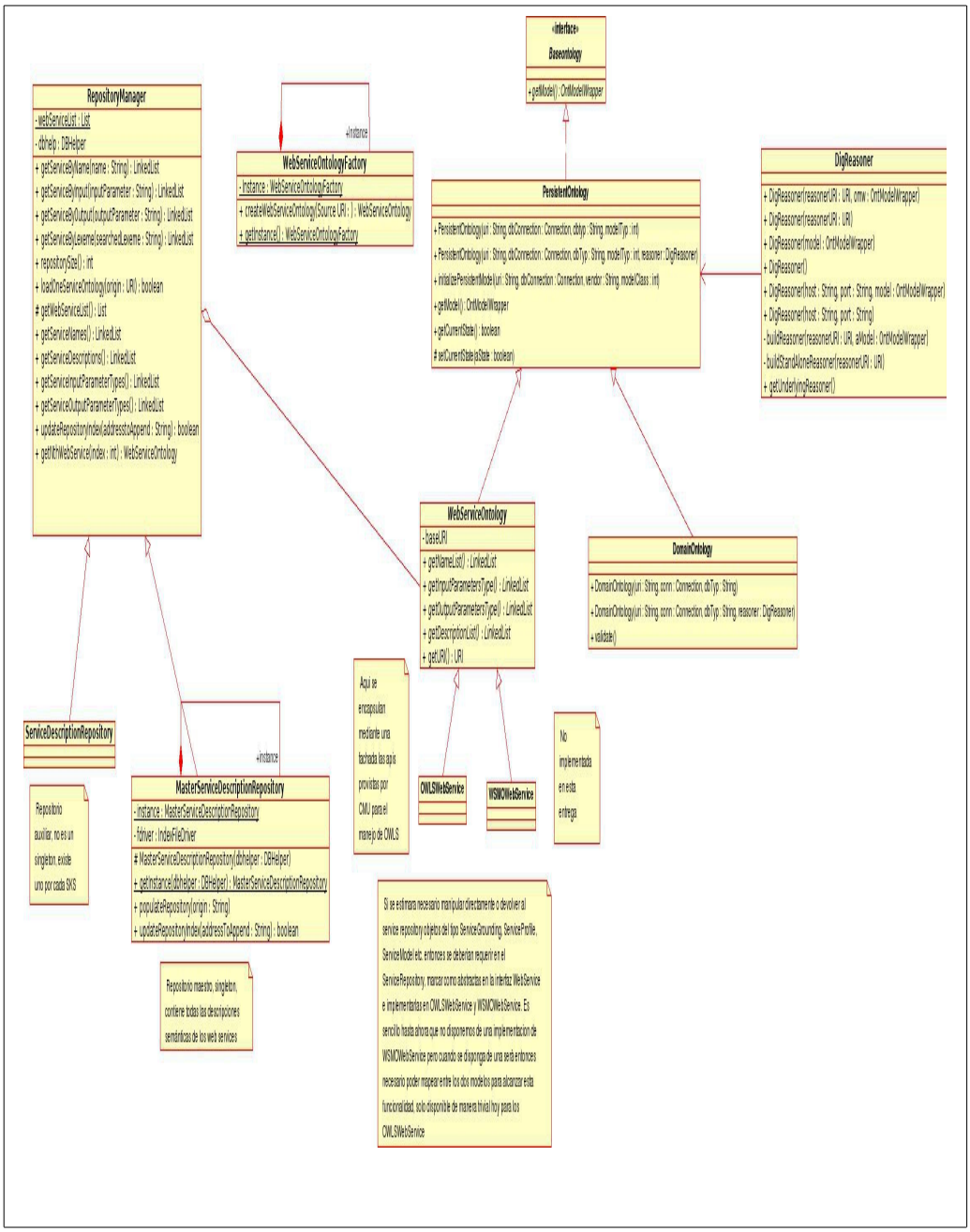

CommandServer
- ServerPort : int
- BackLog : short
- initialPriority : short
- s : ServerSocket
- L : Logger
- running : boolean
- currThread : Thread
- displayBanner : boolean
- startupDate : Date
- db : DBHelper
+ CommandServer(serverVersion : String, db : DBHelper)
+ loadWorkingParams()
+ start()
+ run()
CommandServer es un thread que atiende un puerto determinado y lanza una sesión (SessionProcessor)
SessionProcessor
+ SessionProcessor(s : Socket, mustDisplayBannerAtStartup : boolean, ServerDate : Date, version : String, d : DBHelper)
+ processClientRequest()
# processIncomingMessage(incomingMessage : String, commandSource : String) : boolean
# notYetImplemented()
# processAddOneService(msgParameters : String [])
# processListServiceNames()
# processListServiceNamesByWebService()
# processListServiceDescriptions()
# processListServiceDescriptionsByWebService()
# processListServiceInputs()
# processListServiceInputsByWebService()
# processListServiceOutputs()
# processListServiceOutputsByWebService()
# processListURIS()
# processListURI(parameters : String [])
# setCurrentWSOntology(messageParameters : String [])
# listCurrentWSOntologyNames()
# listCurrentWSOntologyDescriptions()
# listCurrentWSOntologyInputs()
# listCurrentWSOntologyOutputs()
# listCurrentWSOntologyURI()
# listCurrentWSOntologyClasses()
# listCurrentWSOntologyIndividuals()
# resetCurrentWSOntology()
# status()
# shutdown()
# createSKS(messageParameters : String [])
# dropSKS(messageParameters : String [])
# addWSOntologyToSKS(messageParameters : String [])
# removeWSOntologyFromSKS(messageParameters : String [])
# setCurrentSKS(messageParameters : String [])
# listAllSKS()
# resetCurrentSKS()
# listSKSNames()
# listSKSDescriptions()
# listSKSModelClasses()
# ListOntologyClasses()
# listOntologyIndividuals()
# listCurrSKSModelIndividuals()
# setCurrSKSWSOntology()
# resetCurrSKSWSOntology()
# listCurrSKSWSOntologyClasses()
# listCurrSKSWSOntologyIndividuals()
# listCurrSKSWSOntologyInputs()
# listCurrSKSWSOntologyOutputs()
# listCurrSKSWSOntologyNames()
# listCurrSKSWSOntologyDescriptions()
# listCurrSKSWSOntologyURI()
# queryCurrSKSServiceCollection(wholeMessage : String)
# queryCurrSKSOntology(wholeMessage : String)
# loadOntology(parameters : String [])
# validateCurrentSKSOntology()
# listResidentOntologies()
# listSubClasses()
# listSuperClasses()
# isDefined()
# validateSKS()
# writeOutputBuffer(str : String) : int
# closeStream()
# getInput(it : Socket) : InputStream
# getOutput(it : Socket) : OutputStream
# readInputBuffer() : String
# openStream(aSocket : Socket) : boolean
- splitQueryFromMessage(commandMessage : String) : String
SKS
+ SKS(id : int, d : DBHelper)
+ SKS(id : int, aName : String, d : DBHelper)
+ setCurrentWSOntology(index : int) : String
+ resetCurrentWSOntology()
+ listCurrWSOntologyURI() : String
+ listCurrWSOntologyNames() : String
+ listCurrWSOntologyDescriptions() : String
+ listCurrWSOntologyInputs() : String
+ listCurrWSOntologyOutputs() : String
+ listSubClassesFromOntology(classURI : String, theDirectOnesOnly : boolean) : String
+ listSuperClassesFromOntology(classURI : String, theDirectOnesOnly : boolean) : String
+ isInDomainOntology(someClass : String) : boolean
+ listCurrWSOntologyClasses() : String
+ listCurrWSOntologyIndividuals() : String
+ setOntology(urlToLoadFrom : String) : boolean
# loadWorkingParams()
+ getSKSName() : String
+ setSKSName(newName : String)
+ addWebServiceOntology(w : WebServiceOntology)
+ removeWebServiceOntology(w : WebServiceOntology)
+ removeWithWebServiceOntology(internalIndex : int)
+ getServiceRepositoryNames()
+ getServiceRepositoryDescriptions()
+ showHierarchy() : String
+ listModelIndividuals() : String
+ executeQueryOnServiceDescriptions(sparqlQuery : String) : String
+ executeQueryOnOntology(sparqlQuery : String) : String
+ listOntologyIndividuals() : String
+ listOntologyClasses() : String
+ validateDomainOntology() : String
+ isDefinedInOntology(aClassURI : String) : String
+ validate() : String

El manejador del repositorio de publicaciones.

De acuerdo a lo establecido por los requerimientos, el administrador o manejador del repositorio de publicaciones debe:

- Determinar sobre qué conjunto(s) de descripciones semánticas se va a trabajar
- Seleccionar un conjunto de descripciones semánticas del conjunto, o sea, dar el acceso a un subconjunto del universo de descripciones semánticas con que se va a trabajar.
- Acceder a cada una de la descripciones seleccionadas de forma de poder pasarlas como parámetro y acceder a sus partes.

De acuerdo a estos requerimientos se modeló el siguiente conjunto de clases:



Los componentes fundamentales que se pueden apreciar en la figura son los siguientes:

- ***ServiceRepository***.- Esta es la clase que provee los servicios esenciales al algoritmo. El acceso a los web services se realiza através de ella. Cualquier otro servicio que se desee agregar y hacer disponible a un algoritmo se debe agregar aqui. Estructuralmente es un singleton, existe un único objeto en memoria en todo momento de ejecución del entorno de trabajo. Dado que los detalles de implementación hicieron que se trate de una colección en memoria, pero dotada de persistencia. Internamente debe ser vista como una coleccion de WebServices (genéricos) que se describen mas abajo. Tiene vinculación con una fàbrica (factory) de WebServices, que es la encargada de la construcción de los mismos.

- ***WebServiceOntologyFactory***. Es una clase (singleton nuevamente) que es la encargada de crear objetos que implementan la interfaz **WebServiceOntology**. Sirve para aislar al ServiceRepository de los detalles de la creación de WebServices.

- ***WebServiceOntology***. Es la interfaz que deben implementar las clases que se agrupen bajo esta característica. Como se aprecia, **OWLSWebService** y **WSMOWebServic**e la implementan.

- ***WSMOWebService***. Se incluye para facilitar desarrollos futuros, aunque al momento no se soportan representaciones WSMO. Lo que se implementa la posibilidad de incluir el soporte WSMO en el futuro, es decir, parte de la jerarquía que el desarrollador deberá seguir para evitar incompatibilidades soportando WSMO u OWL-S indistintamente.

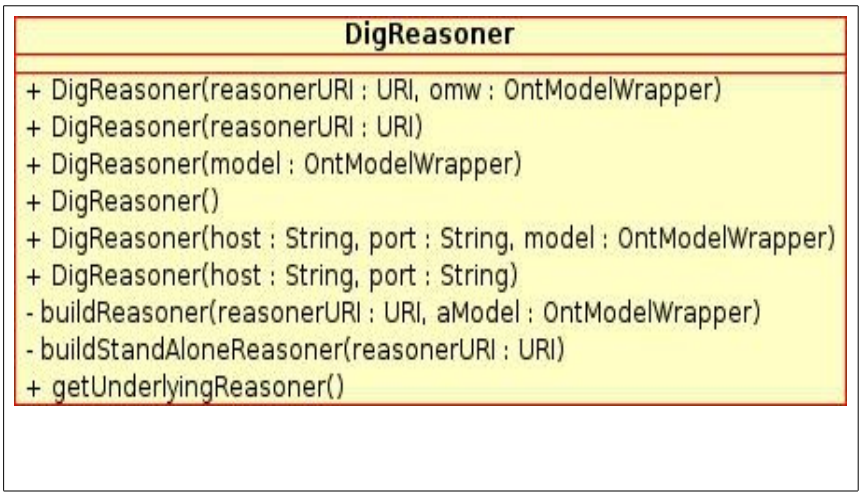
- ***OWLSWebService***. Aquì se desarrolla la funcionalidad de acceso a los elementos definidos para OWL-S encapsulando las API provistas por CMU.

Algunas consideraciones: Se definó no implementar el acceso e elementos fundamentales como service profile o model o grounding, dado que no son elementos que estén presentes en WSMO e implicarìan un mapeo entre ambos modelos (si bien esto es factible). Eso implicaría definir metodos de acceso o manipulacion a nivel de la interfaz WebService (lo que se hace sin problemas), implementar a nivel de OWLSWebService (lo que es trivial dadas las API utilizadas) e implementar a nivel de WSMOWebService (lo que no es trivial cuando se soporte WSMO, si bien ahora es simplemente implementar un método vacío). Cualquier servicio extra que se desee hacer disponible (en cuanto a repositorio) para un algoritmo se deberá ofrecer a través de ServiceRepository. Se preseume que se necesaitarán mas servicios,

entendemos que este diseño favorece la adición de nuevos servicios sin demasiados problemas, delegando la implementación específica de cada servicio a partir de la interfaz WebService.

Interfaz con Razonador Dig

Para la interfaz con el razonador Dig que corresponda se modela una clase que permite cumplir con los servicios básicos. La misma se presenta a continuación



Apéndice B – Como se programa una nueva métrica

Conceptualmente, desde el punto de vista del servidor una métrica es un objeto correspondiente a una clase que implementa la interfaz “Metric”. Las métricas se agrupan en una colección de métricas que simplifica su comunicación con la DVM.

Los distintos métodos implementados en la DVM están obligados a notificar a esta colección en el momento de su invocación y en su salida.

La colección de métricas a su vez notifica a cada métrica del evento sucedido.

A continuación se ejemplifica la implementación de una métrica arbitraria.

Para el caso mostraremos como implementar una métrica que mida el conteo de las llamadas a las funciones de la Discovery Virtual Machine por parte de un algoritmo.

Primer paso, implementar la interfaz “Metric”

La interfaz Metric actualmente desarrollada se muestra a continuación:

```
/*************************************************************************
*                                Metric.java
*
*************************************************************************/
/**
* Descripcion:  Es la interfaz que debe cumplir una Metrica.
*                    Cualquier métrica que sea incorporada al framework debe cumplir
*                    con esta interfaz y luego ser cargada por la DVM.
*
```

```
* Modificaciones:
*   Author: Gustavo Núñez
*   nro:
*   Fecha: Noviembre/2008
*   Notes:
**/

package FING.Camaleon.Discovery.VirtualMachine.Metrics;

import FING.Camaleon.Discovery.Framework.Algorithm.MatchResult;
import java.util.Vector;


public interface Metric {


// Setup comunica una coleccion de URL con el
// resultado esperado en la corrida del algoritmo.
// Permite ademas hacer inicializaciones.
public void setup (Vector <String> expectedSet);

// Es invocada al principio y al final de la invocacion de
// cada metodo de la DVM. El parametro que se pasa es una
// constante (ver Constants.java) que indica que metodo
// acaba de comenzar o finalizar.
public void update(int track);

// Devuelve un reporte con el resultado del calculo de la metrica.
public String getResults ();

// Es invocada justo antes de la invocacion del algoritmo de discovery.
// Tambien es un lugar idoneo para la inicializacion.
public void start ();

// Es invocada justo despues de la invocacion del algoritmo de discovery.
public void stop ();

// Es util para reinicializar, se invoca a pedido del cliente.
public void clear();

// Aqui es donde se computan los calculos finales, es invocada inmediatamente
// despues de "stop". Recibe como resultado los resultados del algoritmo de
// discovery
public void doFinal (Vector <MatchResult> results);

}
```

Para facilitar su comprensión se resaltan en color los distintos métodos que se deben implementar.

Como punto de partida para implementar la métrica del ejemplo debemos considerar dos cosas:

1. Las llamadas posibles a los métodos de la DVM.

2. Las constantes que nos permiten realizar los conteos requeridos.

Para el primer paso, una inspección de la implementación en el servidor de la DVM (DiscoveryVirtualMachineImp.java) nos permite apreciar los siguientes métodos disponibles:

```
public Iterator <String> getClassTypes (boolean directTypes, String IndividualURI)
public Iterator <String> publications ()
public SparqlResultSet executeSelect ( int queryDestination, String query)
public boolean isSubClassOf (String aClassUri, String anotherClassUri)
public boolean isDirectSubClassOf (String aClassUri, String anotherClassUri)
public boolean subsumes (String aClassUri, String anotherClassUri)
public boolean directlySubsumes (String aClassUri, String anotherClassUri)
public boolean isExact (String aClassUri, String anotherClassUri)
public Iterator <String> getPublicationInputParameters (String publicationURI)
public Iterator <String> getPublicationOutputParameters (String publicationURI)
public void say (String msg)
public Vector <MatchResult> sort (Vector <MatchResult> v)
```

La discusión de la implementación de cada uno de estos métodos escapa al alcance de este capítulo.

Una inspección mas profunda de para cada uno de los métodos permite apreciar existe una invocación, como primera y última instrucción de un método de actualización de toda la colección de métricas.

Como ejemplo tomemos el método **say (String msg)**:

```
public void say (String msg) {
        mc.update (Constants.startsay);
        L.Log(5, msg);
        String msgToReturn = null;
        myNetworkConnection = S.getClientConnection ();
        SimpleDateFormat sdf = new SimpleDateFormat ("yyyy/MM/dd - HH:mm:ss.SSS");
        Date currDateTime = new Date ();
        String formattedDate = sdf.format(currDateTime);
        msgToReturn = new String (Constants.SAY_ANSWER + "#"+ currDateTime +
```

```
                                        " \""+ msg + "\"\n");
myNetworkConnection.writeOutputBuffer (msgToReturn);
mc.update (Constants.endisExact);
// Devolver a cliente el mensaje ...
}
```

El objeto **mc** es del tipo MetricCollection (clase implementada en MetricCollection.java), y tiene como objetivo agrupar a todas las métricas. El método *update* notifica a todas las métricas que deberán realizar acciones necesarias para lograr su cometido. El parámetro que se pasa es una constante entera que indica en que situación se encuentra la DVM. Estas constantes se definen en Constants.Java y son las que se detallan a continuación:

```
// Constantes relacionadas a metricas.-
// Por cada función invocable de la DVM se debe generar dos constantes
// en esta seccion. Cada funcion invocada actualizará a la entrada y a la salida
// al conjunto de metricas con estas constantes.
// De este modo cada metrica conocerá que funciones se están invocando para
// saber así como proceder en cada caso.-

public static final int startgetClassTypes = 0;
public static final int endgetClassTypes = 1;
public static final int startpublications =2;
public static final int endpublications=3;
public static final int startexecuteSelect=4;
public static final int endexecuteSelect=5;
public static final int startisSubClassOf=6;
public static final int endisSubClassOf=7;
public static final int startisDirectSubClassOf=8;
public static final int endisDirectSubClassOf=9;
public static final int startsubsumes=10;
public static final int endsubsumes=11;
public static final int startdirectlySubsumes=12;
public static final int enddirectlySubsumes=13;
public static final int startisExact=14;
public static final int endisExact=15;
public static final int startgetPublicationInputParameters=16;
public static final int endgetPublicationInputParameters=17;
public static final int startgetPublicationOutputParameters=18;
public static final int endgetPublicationOutputParameters=19;
public static final int startsay=20;
public static final int endsay=21;
public static final int startsort=22;
public static final int endsort=23;
```

Las constantes utilizadas en el ejemplo son las que se resaltan. Como se detalla en la documentación de interfaz Constants, se definen dos constantes por cada método que se implementa en la DVM, uno para notificar a cada métrica el comienzo del método, y otro para notificar la finalización del mismo. La implementación de cada método debe garantizar la notificación. como única manera de poder detectar el ingreso y la salida del método.

La colección de métricas (MetricCollection.java) se describen a continuación:

```
/*************************************************************************
*                                MetricCollection.java
*
*************************************************************************/
/**
* Descripcion:  Colección de métricas, clase de conveniencia para poder
*                    manejar todas las metricas desde un solo objeto dentro de
*                    la DVM.
*
* Modificaciones:
*   Author: Gustavo Nunez
*   nro:
*   Fecha: Noviembre/2008
*   Notes:
**/

package FING.Camaleon.Discovery.VirtualMachine.Metrics;

import FING.Camaleon.Discovery.Framework.Algorithm.MatchResult;
import java.util.Vector;
import java.util.Iterator;

public class MetricCollection {

protected Vector <Metric> internalCollection; // soporte de la coleccion de metricas.

public MetricCollection () {
        internalCollection = new Vector <Metric> ();}

public void addMetric (Metric m) {
        internalCollection.add (m);}
```

```
public void clearAllMetrics () { internalCollection.removeAllElements();}

public void setup (Vector <String> expectedResultSet ) {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.setup (expectedResultSet);
        }
}

public void update (int track) {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.update (track);
        }
}

public String getMeasure () {
        Iterator <Metric> i = internalCollection.iterator ();
        String result = "\n****************************************\n";
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                result = result  + currentMetric.getResults() ;
                result = result + "\n****************************************\n";
        }
        return result;
}
public void startAllMetrics() {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.start();
        }
}

public void doFinal (Vector <MatchResult> t) {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.doFinal (t);
        }
}

public void stopAllMetrics() {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.stop();}
        }
public void clear () {
        Iterator <Metric> i = internalCollection.iterator ();
        while (i.hasNext()) {
                Metric currentMetric = i.next ();
                currentMetric.clear();}
        }
}
```

Como se advierte en el listado anterior, la implementación del la colección de métricas está basada en un vector de métricas. Esta clase encapsula la colección y toda la comunicación entre la DVM y cada métrica se realiza a través de esta colección.

El método *update* recorre la colección invocando el método update de cada métrica con el parámetro (constante numérica predefinida) con el que es invocado.

Finalmente, con los elementos anteriores es posible implementar una métrica que cuente las invocaciones a los métodos de la DVM por parte de un algoritmo de discovery.

Una implementación se presenta a continuación:

```
/*********************************************************************
*                                NumberOffCallsMetric.java
*
*********************************************************************/
/**
* Descripcion:  Es una metrica de ejemplo que cuenta la cantidad de llamadas
*                    a funciones de la DVM de un algoritmo.
*
*
* Modificaciones:
*   Author: Gustavo Nunez
*   nro:
*   Fecha: Noviembre/2008
*   Notes:
**/
package FING.Camaleon.Discovery.VirtualMachine.Metrics;

import FING.Camaleon.Discovery.Framework.Core.Constants;
import FING.Camaleon.Discovery.Framework.Algorithm.MatchResult;
import java.util.Vector;

public class NumberOffCallsMetric implements Metric, Constants {

private int globalcounter;
```

```
private int countgetClassTypes;
private int countpublications;
private int countexecuteSelect;
private int countisSubClassOf;
private int countisDirectSubClassOf;
private int countsubsumes;
private int countdirectlySubsumes;
private int countisExact;
private int countgetPublicationInputParameters;
private int countgetPublicationOutputParameters;
private int countsay;
private int countsort;
private int indscriminated;

public NumberOffCallsMetric () {
        globalcounter = 0;
        countgetClassTypes = 0;
        countpublications = 0;
        countexecuteSelect = 0;
        countisSubClassOf = 0;
        countisDirectSubClassOf = 0;
        countsubsumes = 0;
        countdirectlySubsumes = 0;
        countisExact = 0;
        countgetPublicationInputParameters = 0;
        countgetPublicationOutputParameters = 0;
        countsay = 0;
        countsort = 0;
        indscriminated = 0;
        }

public void setup (Vector <String> expect) {
        // inicialización.-
        // En este caso particular no se hace uso de parámetros.
        globalcounter = 0;
        countgetClassTypes = 0;
        countpublications = 0;
        countexecuteSelect = 0;
        countisSubClassOf = 0;
        countisDirectSubClassOf = 0;
        countsubsumes = 0;
        countdirectlySubsumes = 0;
        countisExact = 0;
        countgetPublicationInputParameters = 0;
        countgetPublicationOutputParameters = 0;
        countsay = 0;
        countsort = 0;
        indscriminated = 0;

        }

public void update(int a) {
        // no se hace uso de parámetros aquí.
        switch (a) {
        case Constants.startgetClassTypes: countgetClassTypes ++; break;
        case Constants.endgetClassTypes: break;
        case Constants.startpublications:countpublications ++; break;
        case Constants.endpublications: break;
        case Constants.startexecuteSelect: countexecuteSelect ++; break;
```

```
		case Constants.endexecuteSelect: break;
		case Constants.startisSubClassOf: countisSubClassOf ++; break;
		case Constants.endisSubClassOf: break;
		case Constants.startisDirectSubClassOf: countisDirectSubClassOf ++; break;
		case Constants.endisDirectSubClassOf:break;
		case Constants.startsubsumes: countsubsumes++; break;
		case Constants.endsubsumes:break;
		case Constants.startdirectlySubsumes:countdirectlySubsumes++; break;
		case Constants.enddirectlySubsumes:break;
		case Constants.startisExact: countisExact++; break;
		case Constants.endisExact:break;
		case  Constants.startgetPublicationInputParameters:countgetPublicationInputParameters+
+; break;
		case Constants.endgetPublicationInputParameters:break;
		case
Constants.startgetPublicationOutputParameters:countgetPublicationOutputParameters++; break;
		case Constants.endgetPublicationOutputParameters:break;
		case Constants.startsay: countsay++; break;
		case Constants.endsay:break;
		case Constants.startsort: countsort++; break;
		case Constants.endsort:break;
		default: indscriminated ++; break;
			}
		};
public String getResults () {
		String report = "";

		report = "Cantidad de TOTAL de llamadas: " + globalcounter + "\n\n" +
			"Cantidad de llamadas a getClassTypes: "+ countgetClassTypes+ "\n" +
			"Cantidad de llamadas a publications: "+ countpublications+ "\n" +
			"Cantidad de llamadas a executeSelect: "+ countexecuteSelect+ "\n" +
			"Cantidad de llamadas a isSubClassOf: "+ countisSubClassOf+ "\n" +
			"Cantidad de llamadas a subsumes: "+ countsubsumes+ "\n" +
			"Cantidad de llamadas a directlySubsumes: "+ countdirectlySubsumes+ "\n" +
			"Cantidad de llamadas a isExact: "+ countisExact+ "\n" +
			    "Cantidad de llamadas a getPublicationInputParameters: "+
countgetPublicationInputParameters+ "\n" +
			   "Cantidad de llamadas a getPublicationOutputParameters: "+
countgetPublicationOutputParameters+ "\n" +
			"Cantidad de llamadas a say: "+ countsay+ "\n" +
			"Cantidad de llamadas a sort: "+ countsort+ "\n" +
			"Otras llamadas no discriminadas: "+indscriminated+"\n\n";
		return report;
		}
public void start () {
		// nada que hacer aqui...
		}
public void stop () {
		// nada que hacer aqui...
		}

public void clear () {		globalcounter = 0;
		countgetClassTypes = 0;
		countpublications = 0;
		countexecuteSelect = 0;
		countisSubClassOf = 0;
		countisDirectSubClassOf = 0;
		countsubsumes = 0;
		countdirectlySubsumes = 0;
```

```
        countisExact = 0;
        countgetPublicationInputParameters = 0;
        countgetPublicationOutputParameters = 0;
        countsay = 0;
        countsort = 0;
        indscriminated = 0;}

public void doFinal (Vector <MatchResult> results) {
        // no es necesario utilizar este parámetro aqui.
        globalcounter = countgetClassTypes +
                        countpublications +
                        countexecuteSelect +
                        countisSubClassOf +
                        countisDirectSubClassOf +
                        countsubsumes +
                        countdirectlySubsumes +
                        countisExact +
                        countgetPublicationInputParameters +
                        countgetPublicationOutputParameters +
                        countsay +
                        countsort +
                        indscriminated;

 }
}
```

En nuestra implementación hemos decidido llevar un contador global de llamadas y un contador particular para cada llamada a cada método. Se definen como privados en el cuerpo de la clase.

En la constructora se inicializan a cero estos contadores.

En el método *Setup* se vuelven a inicializar a cero estos contadores, si bien esto no era estrictamente necesario. Obsérvese que el método mencionado recibe un vector que almacena *resultados esperados* del algoritmo. La invocación de este método es opcional, y se hace a partir de un comando ingresado por el usuario previo a la corrida del algoritmo. Es claro que este método no es útil para esta métrica en particular, pero si lo es para otras (particularmente *recall* y *precision*) y tienen una implementación impuesta por la interfaz *Metric.*

El método *update (int)* recibe como parámetro una de las constantes ta mencionadas y es invocado dentro del método homónimo de la *MetricCollection.* En este caso en particular se incrementa el contador correspondiente del método invocado, acumulándose así la cantidad de invocaciones a cada método. Para este caso, las notificaciones originadas por las salidas a cada método son ignoradas por no ser de ninguna utilidad.

El método *getResults* devuelve una cadena de caracteres arbitraria, y sirve para comunicar los resultados de la métrica para consumo humano. En esta implementación se devuelve una cadena de caracteres con texto que describe a los contadores y muestra su resultado final. Su invocación es posterior a la ejecución al algoritmo.

Los métodos *Start* y *Stop* no son útiles en este contexto, pero si lo serían en una métrica que midiera, por ejemplo, el tiempo de ejecución de un algoritmo, dado que se invocan inmediatamente antes de la invocación del algoritmo de discovery e inmediatamente a continuación de su finalización, respectivamente.

El método *clear* tiene como objetivo volver a inicialirar las variables utilizadas deshechando los resultados obtenidos. Es útil cuando se intenta medir sucesivamente distintos algoritmos, de manera de no arrastrar resultados de una medida a otra.

El método *doFinal* contiene los cómputos finales necesarios para obtener los valores buscados. En el caso particular que se expone, se ha dispuesto a modo de ejemplo de un contador global que es la suma de todos los anteriores. En esta función precisamente es donde tiene lugar esa suma. En otros contextos (recall y precision notablemente) se obtienen los valores finales de esas métricas dado que éste método acepta como parámetros los resultados que devuelve el algoritmo de discovery luego de su ejecución.

Luego de la implementación de la métrica expuesta, solo queda vincularla de manera apropiada a la implementación de la DVM.

Para ello se debe:

1. Declarar y definir la métrica en la DVM.
2. Asociarla a la colección de métricas.

Para lograrlo se procede de la siguiente manera:

```
/*************************************************************************
*                              DiscoveryVirtualMachineImp.java
*
*************************************************************************/
/**
* Descripcion:  Es la implementación de una Discovery Virtual Machine.
*                    Es funcionalmente mas rica que las API que serán entregadas a
*                    los desarrolladores. Los objetos de esta clase además de cumplir
*                    con la interfaz DVM, estarán en contacto directo con un SKS
*                    en el que se basarán para implementar los métodos necesarios
*                    sobre las ontologías de turno y los repositorios correspondientes.
*
* Modificaciones:
*   Author: Gustavo Núñez
*   nro:
*   Fecha: Noviembre/2007
```

```
*   Notes:
**/

package FING.Camaleon.Discovery.VirtualMachine.Implementations;

import FING.Camaleon.Discovery.VirtualMachine.DiscoveryVirtualMachine;
import FING.Camaleon.Discovery.VirtualMachine.Sparql.SparqlResultSet;
import FING.Camaleon.Discovery.Framework.Ontology.SKS;
import FING.Camaleon.Discovery.Framework.Utils.Comm.TCPIPConnection;
import FING.Camaleon.Discovery.Framework.Core.Constants;
import FING.Camaleon.Discovery.Framework.Algorithm.MatchResult;
import FING.Camaleon.Discovery.Framework.Algorithm.DiscoveryAlgorithm;
import FING.Camaleon.Discovery.VirtualMachine.Metrics.*;

import java.util.Iterator;
import java.util.LinkedList;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Vector;

public class DiscoveryVirtualMachineImp implements DiscoveryVirtualMachine, Constants {
private SKS S;

private final int inTheDomainOntology = 0;    // Constantes que definen donde
private final int inTheServicesRepository = 1;              // ejecutar un query.
protected TCPIPConnection myNetworkConnection;
protected FING.Camaleon.Discovery.Framework.Utils.Logger L;

protected DiscoveryAlgorithm DA;
protected Vector <MatchResult> currentResults;

// Metricas de aqui en adelante.-
protected MetricCollection mc;                                  // Colección de metricas.
public Vector <String> expectedResultCollection; // resultados esperados de la ejecucion.
// las nuevas metricas deben ser declaradas aqui.
protected ExecutionTimeMetric etm;
protected NumberOffCallsMetric noc;
protected PrecisionAndRecallMetric par;

public DiscoveryVirtualMachineImp (SKS mySKS) {
                        S   = mySKS;
                        L   = FING.Camaleon.Discovery.Framework.Utils.Logger.getInstance();
                        mc  = new MetricCollection();
                        // las nuevas metricas deben ser creadas aqui
                        etm = new ExecutionTimeMetric();
                        noc = new NumberOffCallsMetric();
                        par = new PrecisionAndRecallMetric();
                        // las nuevas metricas deben ser agregadas a la
                        // coleccion de metricas.
                        mc.addMetric (etm);
                        mc.addMetric (noc);
                        mc.addMetric (par);
                        expectedResultCollection = S.getExpectedResultsCollection();}
```

En primer lugar se declara el package donde reside la métrica, en nuestro caso es FING.Camaleon.Discovery.VirtualMachine.Metrics

Luego se define el objeto **noc** del tipo que nos interesa *NumberOfCallsMetric*

Se inicializa en la constructora de la DVM (operador *new*) y para finalizar se agrega este objeto a la colección de métricas (invocación coloreada mc.addMetric (noc).

Apéndice C - Una implementación del algoritmo de Paolucci a través de la DVM

A continuación se presenta el código fuente de la implementación de referencia presentada en el capítulo correspondiente.

```
/*****************************************************
*                                Paolucci.java                                *
*****************************************************/
// Una implementacion del algoritmo de Paolucci de Matchmaking.
// Autor Gustavo Nunez -
// La presente es una implementacion del algoritmo de Paolucci publicado
// en el trabajo Semantic Matching of Web Service Capabilities,
// (Massimo Paolucci et al) ISWC, pages 333-347. Springer Verlag, 2002,
// disponible en Internet http://eprints.ecs.soton.ac.uk/7606/1/ISWC2002-Matchmaker.pdf

// Comentarios:
// Se trata de una implementacion adaptada al Framework de construccion de
// algoritmos semanticos, y que por lo tanto utiliza la Discovery Virtual Machine
// provista por este. A su vez cumple con la interfaz de llamada impuesta por
// el framework, por lo que devuelve una coleccion de resultados ranqueada y con
// un mapa de entradas/salidas para cada parametro de cada web service devuelto
// Es por ese motivo que esta implementacion consta de estructuras auxiliares
// (MatchResult, WeightedURIMap) que intervienen en el pasaje de parametros
// entre las rutinas y que no forman parte del trabajo originalmente expuesto
// por Paolucci, pero que no hacen a lo esencial de su algoritmo de busqueda.

package REPOSITORY.ALGORITHMS;

import FING.Camaleon.Discovery.Framework.Algorithm.DiscoveryAlgorithm;
import FING.Camaleon.Discovery.Framework.Algorithm.MatchResult;
import FING.Camaleon.Discovery.Framework.Algorithm.WeightedURIMap;

import java.util.Vector;
import java.util.Iterator;

public class Paolucci extends DiscoveryAlgorithm {

public Vector <MatchResult> discover (Vector <String> availableInput, Vector
<String> requiredOutput) {
        DVM.say ("Comenzando implementacion de algoritmo de Paolucci...");
        // inicializamos la coleccion que vamos a devolver ...
        Vector <MatchResult> results = null;
        // y uno de trabajo.
        Vector <MatchResult> unsortedResults = new Vector <MatchResult> ();
```

```
        // levantar las publicaciones del repositorio ...
        Iterator <String> publicaciones = DVM.publications();

        // iterar sobre las publicaciones ...
        DVM.say ("Comenzando iteracion sobre publicacionies...");
        while (publicaciones.hasNext()) {
                // Iteradores sobre los parametros de entrada ...
                Iterator <String> availableInputIterator = (availableInput.subList(0,

        availableInput.size())).iterator();
                Iterator <String> requiredOutputIterator = (requiredOutput.subList(0,

        requiredOutput.size())).iterator();

                String publicacion = publicaciones.next();
                DVM.say ("Buscando matching sobre " + publicacion);
                MatchResult currMatchResult = matches (publicacion,
                                                availableInputIterator,
                                                requiredOutputIterator);
                int resultValue = currMatchResult.getWeight();
                DVM.say ("Matching sobre " + publicacion + " retorna: " +
                                                resultValue);
                if (resultValue != FAIL) {unsortedResults.add (currMatchResult);}

        }
        DVM.say ("Finalizando iteracion sobre publicaciones...");
        results = DVM.sort (unsortedResults);
        DVM.say ("Retornando - FIN");
        return results;

} // fin del algoritmo.


public MatchResult matches ( String publication,
                                Iterator <String> ReqAvailableInputIterator,
                                Iterator <String> ReqNeededOutputIterator ) {
// No publicado en el paper de Paolucci  - GN
        int outputDegreeOfMatch;
        int inputDegreeOfMatch;
        int globalMatch=FAIL;
        Iterator <String> inputPublicationParameters =
                                DVM.getPublicationInputParameters(publication);
        Iterator <String> outputPublicationParameters =
                                DVM.getPublicationOutputParameters(publication);

        // para obtener los mapeos de parametros.
        Vector <WeightedURIMap> inputMapCollection = new Vector
                                                        <WeightedURIMap> ();
        Vector <WeightedURIMap> outputMapCollection = new Vector
```

```
                                                    <WeightedURIMap> ();

        outputDegreeOfMatch = outputMatch (ReqNeededOutputIterator,
                                           outputPublicationParameters,
                                           outputMapCollection);
        inputDegreeOfMatch = inputMatch (ReqAvailableInputIterator,
                                         inputPublicationParameters,
                                         inputMapCollection);
        globalMatch = minimum (outputDegreeOfMatch, inputDegreeOfMatch);
        // Si hay un FAIL, globalMatch vale FAIL y se trata en la
        // rutina llamadora.
        MatchResult aMatch = new MatchResult (publication, globalMatch,
                                           inputMapCollection,
                                           outputMapCollection);
        return aMatch;
}

public int outputMatch ( Iterator <String> outputsRequest,
                         Iterator <String> outputsAdvertising,
                         Vector <WeightedURIMap> outputMappingCollection) {
        // Aqui hay que satisfacer la salida de requerimiento (cada una)
        // con lo que puede proveer la publicacion.
        int globalDegreeOfMatch = EXACT;
        int bestDegreeOfMatch= FAIL;
        while (outputsRequest.hasNext()) {
                String currentOutputRequest = outputsRequest.next ();
                int currentMatch=FAIL;
                String mapCandidate = null;
                String currentOutputAdvertising = null;
                while (outputsAdvertising.hasNext()) {
                        currentOutputAdvertising = outputsAdvertising.next();
                        currentMatch = degreeOfMatch (currentOutputRequest,
                                                      currentOutputAdvertising);
                        if (greater (currentMatch, bestDegreeOfMatch)) {
                                bestDegreeOfMatch = currentMatch;
                                mapCandidate = currentOutputAdvertising;
                        }
                }
                // si lo mejor que encontramos para este "outputrequest" es
                // un fail, quiere decir que no podemos satisfacer la salida
                // de requerimiento, por lo tanto se retorna FAIL y no se continua
                if (bestDegreeOfMatch == FAIL) {
                                outputMappingCollection.removeAllElements();
                                return FAIL;}
                globalDegreeOfMatch = minimum (globalDegreeOfMatch,
                                               bestDegreeOfMatch);
                // para mapear parametros.
                WeightedURIMap currentMap = new WeightedURIMap
                                                    (mapCandidate,
                                                    globalDegreeOfMatch,
```

```
                                                        currentOutputRequest);
                outputMappingCollection.add (currentMap);
        }
        return globalDegreeOfMatch;
}

public int inputMatch ( Iterator <String> inputsRequest,
                        Iterator <String> inputsAdvertising,
                        Vector <WeightedURIMap> inputMappingCollection ) {
        // aqui hay que satsfacer a la entrada de la publicacion con los
        // parametros que hay disponibles en este requerimiento.
        // caso distinto al "outputmatch"

        int globalDegreeOfMatch = EXACT;
        int bestDegreeOfMatch= FAIL;
        while (inputsAdvertising.hasNext()) {
                String currentInputAdvertising = inputsAdvertising.next ();
                int currentMatch=FAIL;
                String mapCandidate = null;
                String currentIntputRequest=null;
                while (inputsRequest.hasNext()) {
                currentIntputRequest = inputsRequest.next();
                currentMatch = degreeOfMatch (currentIntputRequest,
                                              currentInputAdvertising);
                        if (greater (currentMatch, bestDegreeOfMatch)) {
                                bestDegreeOfMatch = currentMatch;
                                mapCandidate = currentIntputRequest;
                        }
                }
                // si lo mejor que encontramos para este "inputsAdevertising" es
                // un fail, quiere decir que no podemos satisfacer la entrada.
                // DE LA PUBLICACION por lo tanto se retorna FAIL y no se continua
                if (bestDegreeOfMatch == FAIL) {
                                inputMappingCollection.removeAllElements();
                                return FAIL;}
                globalDegreeOfMatch = minimum (globalDegreeOfMatch,
                                               bestDegreeOfMatch);
                // para mapear parametros.
                WeightedURIMap currentMap = new WeightedURIMap
(currentInputAdvertising,
                                                        globalDegreeOfMatch,
                                                        mapCandidate);

                inputMappingCollection.add (currentMap);
        }
        return globalDegreeOfMatch;
}

public boolean greater (int aDegree, int anotherDegree) {
// Esta rtutina se hace para comparar dos grados de concordancia.Set/2008
```

```
// A la fecha, las constantes estan programadas de manera que un grado
// de concordancia alto tiene un valor numerico alto, por lo que se puede
// confiar en el valor numerico de las constantes. Si esto cambiara se puede
// de todos modos reformular esta funcion comparadora de manera de adecuarla a
// las necesidades del caso.
        return (aDegree > anotherDegree);
}

public int minimum (int aDegree, int anotherDegree) {
// Esta rtutina se hace para comparar dos grados de concordancia.Set/2008
// A la fecha, las constantes estan programados de manera que un grado
// de concordancia alto tiene un valor numerico alto, por lo que se puede
// confiar en el valor numerico de las constantes. Si esto cambiara se puede
// de todos modos reformular esta funcion comparadora de manera de adecuarla a
// las necesidades del caso.
        if (aDegree <= anotherDegree) { return aDegree;}
        return anotherDegree;
        }

public int maximum (int aDegree, int anotherDegree) {
// Esta rtutina se hace para comparar dos grados de concordancia.Set/2008
// A la fecha, las constantes estan programados de manera que un grado
// de concordancia alto tiene un valor numerico alto, por lo que se puede
// confiar en el valor numerico de las constantes. Si esto cambiara se puede
// de todos modos reformular esta funcion comparadora de manera de adecuarla a
// las necesidades del caso.
        if (aDegree >= anotherDegree) { return aDegree;}
        return anotherDegree;
        }

public int degreeOfMatch (String reqURI, String advURI){
        if (DVM.isExact(reqURI,advURI)) {return EXACT;}
        if (DVM.isDirectSubClassOf (advURI, reqURI)) {return EXACT;}
        if (DVM.subsumes (advURI, reqURI)) {return PLUGIN;}
        if (DVM.subsumes (reqURI, advURI)) {return SUBSUMES;}
        return FAIL;
}

}
```

Apéndice D – Historia de Liberaciones

Release inicial.- Se integraron las bibliotecas seleccionadas y se confeccionó un primer programa que las contuviera. En el interin se incorporaron funciones de *logging* e internacionalización. Se incluye además un archivo de configuración que le da mayor flexibilidad al ambiente de trabajo permitiendo personalizar algunas propiedades del ambiente de trabajo. Esta versión se liberó el 3 de Junio de 2007.-

Release 0.1.- Se profundizó en el manejo de repositorio, sin lograr una funcionalidad específica relativa al repositorio de descripciones semánticas pero si logrando algunos avances significativos en el manejo de las bibliotecas designadas. La fecha de liberación fue el 13 de junio de 2007.-

Release 0.2.- Se consolidó la utilización de algunas de las bibliotecas utilizadas hasta el momento con algunos avances importantes en cuanto a diseño. Se realizaron pruebas cruzadas de descripciones semánticas de servicios con las bibliotecas disponibles, obteniendo resultados satisfactorios. Se eliminó la necesidad de disponer de copias locales de estas descripciones semánticas (cambios en la arquitectura interna del repositorio) y se avanzó en la incorporación de un razonador (junto con la

eliminación de algunos errores menores de programación detectados en el transcurso de todas las pruebas). La fecha de liberación corresponde al 25 de Junio de 2007.-

Release 0.3.- Se incluyen cambios en la arquitectura, a partir de esta versión el ambiente de trabajo pasó a ser un servidor, incorporando a partir de aquí toda la funcionalidad como un servicio (al que provisoriamente se accede a través de *telnet* sobre un puerto conocido). Surge la necesidad de un pequeño lenguaje (un pequeño protocolo de mensajería) entre el cliente y el servidor. La fecha de liberación fue el 4 de Julio de 2007.-

Release 0.4.- Con la arquitectura orientada definitivamente y las bibliotecas a utilizar completamente integradas, se proporcionan al servidor comandos para interactuar con el cliente. Esta versión incorpora una gran cantidad de comandos (mas de 15 comandos distintos, de acuerdo con las notas a la liberación). Adquiere preponderancia el concepto de sesión, como entidad que aísla y aglutina la interacción entre el cliente y el servidor, y como tanto como un lugar idóneo para desarrollar la funcionalidad esperada de *interrelaciones entre los* módulos, que tiene su mayor exponente en la recolección de métricas.

Esta versión se liberó el 25 de Julio de 2007.-

Release 0.5 (Night Build).- Esta versión incorporó el concepto de SKS (*Services and Knowledge Space*) que en esencia es un espacio donde se aíslan un conjunto de descripciones semánticas de servicios y una ontología base, sobre el que el algoritmo interactuará de manera aislada (un algoritmo en una ejecución interactuará con un SKS). Dado que el soporte de ontología no es completo (no existe soporte de razonamiento ni clasificación en esta versión) es que se denominó a esta versión como "pre alfa" o "night build". Sin embargo, la funcionalidad incorporada permite interactuar con un dominio especifico de conocimiento (eso es conceptualmente el SKS). Existe, por sesión, soporte para múltiples SKS. Esta versión se liberó el 15 de Agosto de 2007.-

Release 0.6.- Incorporación de consultas SPARQL, tanto sobre la colección de web services pertenecientes a cada SKS como a las ontologías de dominio subyacentes. Esta versión se liberó el 5 de Setiembre de 2007.-

Release 0.6A.- Esta versión solo corregía bugs detectados de la versión 0.6, fue liberada el 19/9/2007.-

Release 0.7A.- Soporte de persistencia. En esta versión se incluye la posibilidad de almacenar modelos en una base de datos relacional y realizar consultas SPARQL sobre ellos. Se dejan afuera de esta liberación, por cuestiones de agenda: a) Soporte de persistencia para las descripciones semánticas de los web services, b) Soporte de persistencia para el modelo común (unión de los modelos de descripciones semánticas) dentro de los SKS y c) Software adicional para la carga inicial de las ontologías (se cargan únicamente dentro del contexto de un SKS). Esta versión se liberó el 10 de octubre de 2007.**-**

Release 0.7B.- Ampliación de funcionalidad de soporte de persistencia. En esta versión se incluye un conjunto de mecanismos que permite cargar la base de datos con ontologías de dominios cuyas URI se listan en un archivo de URIs que se encuentra referenciado en el archivo de configuración. Se incluyen mecanismos idóneos para cargar y manipular ontologías de dominio. No se incluyen en esta versión: a) Soporte de persistencia para las descripciones semánticas de los web services, b) Soporte de persistencia para el modelo común (unión de los modelos de descripciones semánticas) dentro de los SKS. Esta versión se liberó el 15 de Octubre de 2007.-

Release 0.7C.- Ampliación de funcionalidad de soporte de persistencia, etapa final. En esta versión se incluye un conjunto de mecanismos que permite cargar la base de datos con descripciones semánticas de web services además de las ontologías de dominio incluidas en la versión anterior. Como se descubrieron problemas en la carga multithreaded donde aparentemente el driver utilizado de la base de datos no soporta paralelismo, se serializaron las tareas de carga solucionando los inconvenientes detectados. Se desarrolla además la persistencia de los modelos unión de las descripciones semánticas en cada SKS. Se incorpora definitivamente, el juego de pruebas con las descripciones (ontologías) provistas por OWL-S TC 2.1. Para su instalación se requiere a) apache y b) OWLS-TC 2.1. Esta versión se liberó el 6 de Noviembre de 2007.-

Release 0.8.- Inclusión de razonadores. El avance mas importante que proporciona esta versión es la inclusión de los razonadores como un elemento activo.- Fue una liberación mayor, que incluyó un conjunto de 15 nuevos comandos que comienzan a modelar la funcionalidad pedida para los razonadores. Quedaron fuera de esta versión todos los aspectos relativos a clasificación de individuos y clasificación de clases dentro de una jerarquía. Estas funciones se incluyen en las siguientes entregas.

Esta versión se liberó el 13 de Noviembre de 2007.-

Release 0.81.- Finaliza el desarrollo de la funcionalidad requerida a los razonadores. Se implementa un conjunto de comandos que hacen uso de las funciones del razonador clasificando clases, listando individuos y validando SKS. Esta versión se liberó entre el 15 y el 19 de Noviembre de 2007, fecha en que se terminaron de redactar las notas a la liberación.-

Release 0.82.- Versión menor, corrige el criterio de validación de SKS. A partir de esta versión para validar un SKS se procede de esta manera: a) Se valida la ontología de dominio. Si la ontología no es consistente se falla y se genera una lista de inconsistencias. Se continúa b) generando una lista con todos los parámetros (de entrada y de salida) de los web services de ese SKS, para luego c) buscar todas estas clases en la ontología de dominio, si alguna no se encuentra en la ontología entonces se falla y se genera una lista de las clases que no están definidas en la ontología de dominio. Esta versión se liberó el 20 de Noviembre de 2007.-

Release 0.83.- Repositorio de algoritmos. Comienzo del desarrollo de la Discovery Virtual Machine y clases específicas para ejecutar un algoritmo. Se aislaron las clases de comunicaciones del *SessionProcessor* y se

confeccionó un *cliente* de manera de permitir la "subida" al servidor de una archivo ".class" que implemente un algoritmo de matchmaking. A su vez, este cliente permite realizar todas las tareas que se podían realizar con *telnet* (cliente anterior). A partir de esta versión el ambiente de trabajo solo será utilizable a través de este cliente específico.

Se agrega la clasificación de individuos dentro de una ontología (motivo por el cual esta liberación aún integra la serie "8,x". Esta versión se liberó el 11 de Diciembre de 2007.-

Release 0.90.- Tratamiento de algoritmos.

Esta versión incluye los mecanismos necesarios para la ejecución de un algoritmo, establecer sus parámetros y recoger los resultados.

Se provee el entorno de ejecución (parámetros de entrada y salida que se requerirán a un algoritmo de discovery), se vincula un SKS a un algoritmo a ejecutar.

Se comienza a dotar de funciones a la Discovery Virtual Machine para que pueda ofrecer al algoritmo las primitivas necesarias para alcanzar sus cometidos. Esta versión se liberó el 11 de Junio de 2008.-

Release 0.91.- Enriquecimiento de Discovery Virtual Machine. A partir de

esta versión la DVM soporta querys sparql. Esto permite que un algoritmo ejecute consultas sparql directamente.

Esta versión se liberó el 21 de Julio de 2008.-

Release 0.92.- Enriquecimiento de funcionalidad de Virtual Machine, segunda etapa. Esta versión incluye una Discovery Virtual Machine que posee funciones de clasificación y funciones auxiliares. Es por eso, que a partir de esta versión un algoritmo puede realizar tareas de clasificación a través de la DVM vinculada a un SKS. Concomitantemente con esta liberación se confeccionó el primer algoritmo que se ejecutó sobre el ambiente de trabajo (Hola Mundo), haciendo uso de la DVM liberada. Esto implicó el desarrollo de una librería que debe ser incorporada por el desarrollador para construir un algoritmo semántico medible. Se cumplió el ciclo completo de creación de SKS, subida del algoritmo desde el cliente al servidor, y ordenar la ejecución. Esta versión (conjuntamente con la librería asociada) se liberó el 12 de Agosto de 2008.-

Release 0.93.- Enriquecimiento de funcionalidad de Virtual Machine, tercera etapa. En esta versión se introduce el concepto de relevancia de búsqueda y ordenación por ese criterio. La motivación viene dada en que los algoritmos de búsqueda devuelven un conjunto de datos *ordenados*

por un criterio de relevancia asociado al grado de concordancia (*degree of match*). Por lo tanto se incluyeron cambios en la interfaz del algoritmo (protocolo de llamada, tipos de datos retornados). A partir de esta versión fue posible alcanzar una primera implementación del algoritmo de Paolucci. Esta versión se liberó el 10 de Octubre de 2008.-

Release 1.00.- Incorporación de métricas. En esta fase se incorpora a la DVM la posibiliad de recolectar métricas automáticamente de la ejecución del algoritmo que se testea.- Esta versión se liberó el 25 de noviembre de 2008.-

Apéndice E – Compendio de comandos aceptados por el servidor.

Todos los comandos enviados del cliente al servidor pueden ser digitados tanto en mayúsculas como en minúsculas. El resultado de la ejecución de cada comando viene siempre acompañado de un código de retorno que se despliega en el cliente.

HELP – Despliega un texto mostrando todos los comandos disponibles. Acepta un parámetro <Nombre_de_cmando> opcional que despliega la información necesaria específica de ese comando.

END - Finaliza la sesión, termina la ejecución del cliente.

STATUS - Invoca al recolector de memoria y proporciona informacion de estado del servidor.

SHUTDOWN - Finaliza el servidor (debe ser invocado dos veces).

AddOneService <serviceURI> - Agrega una descripción de servicio.

ListServiceNames - Lista todos los servicios disponibles.

ListServiceNamesByWebService - Lista servicios disponibles por Web Service.

ListServiceDescriptions - Lista las descripciones de todos los servicios disponibles.

ListServiceDescriptionsByWebService - Lista las descripciones por

Web Service.

ListServiceInputs - Lista las entradas requeridas por todos los servicios disponibles.

ListServiceInputsByWebService - Lista las entradas requeridas por Web Service.

ListServiceOutputs - Lista las salidas requeridas por todos los servicios disponibles.

ListServiceOutputsByWebService - Lista las salidas requeridas por Web Service.

ListURIS - Lista las URI de base de los web services.

ListURI <indice> - Lista la URI del WS almacenado en el lugar <indice>.

SetCurrentWSOntology <indice> - Establece al WS del lugar <indice> del repositorio omo ws por defecto.

ListCurrentWSOntologyNames - Lista el/los nombres en la ontología por defecto.

ListCurrentWSOntologyDescriptions - Lista la(s) descripciones en la ontología por efecto.

ListCurrentWSOntologyInputs - Lista los parámetros de entrada regustrados en la ntología por defecto.

ListCurrentWSOntologyOutputs - Lista los parámetros de salida en la

ontología por efecto.

ListCurrentWSOntologyURI - Lista la URI base de la ontología por defecto.

resetCurrentWSOntology - Descarta el valor por defecto de una ontología de servicios web.

CREATESKS <nombreopcional> - Crea un "Services and Knowledge Space" con un nombre opcional.

DROPSKS <handle> - Elimina un Services and Knowledge Space con definido por el índice que se pasa como parámetro.

ADDWSONTOLOGYTOSKS <sksHandle> <WSOntologyIndex> - Agrega al Services and Knowledge Space referenciado por sksHandle la WebServiceOntology definida por WSOntologyIndex

REMOVEWSONTOLOGYFROMSKS <sksHandle> <WSOntologyIndex> - Elimina del Services and Knowledge Space referenciado por sksHandle la WebServiceOntology definida por WSOntologyIndex

SETCURRENTSKS <sksHandle> - Establece el Services and Knowledge Space referenciado por "sksHandle" como SKS por defecto.

LISTALLSKS - Despliega la lista completa de Service And Knowledge Spaces.

RESETCURRENTSKS Descarta el valor por defecto para el Service and Knowledge Space.

LISTCURRSKSNAMES - Despliega los nombres de los servicios en el Service and Knowledge Space actual.

LISTCURRSKSDESCRIPTIONS - Despliega las descriptiones de los servicios en el Service and Knowledge Space actual.

LISTCURRSKSMODELCLASSES - Despliega las características del modelo compuesto por los web services en el SKS actual.

QUERYSKSSERVICES <SPARQLQuery> - Ejecuta una consulta SPARQL sobre el modelo generado por todos los servicios del SKS actual.

QUERYSKSONTOLOGY <SPARQLQuery> - Ejecuta una consulta SPARQL sobre la ontología de base correspondiente al SKS actual.

LOADONTOLOGY <uritoloadfrom> - Carga una ontología en la base de datos o bien la levanta de la base de datos, haciéndola disponible. Se debe estar en el contexto de un SKS para activar una ontología de dominio.

LISTRESIDENTONTOLOGIES – Lista las ontologías de dominio residentes en la Base de Datos.

LOADONTOLOGY <*Ontology_URI*> – Carga una ontología en el

contexto de un sks, con soporte de persistencia.

QUERYSKSSERVICES - Ejecuta una consulta SPARQL sobre el modelo generado por todos los servicios del SKS actual.

QUERYSKSONTOLOGY - Ejecuta una consulta SPARQL sobre la ontología de base correspondiente al SKS actual.

LISTSUBCLASSES <*ClassURI*> - Lista las subclases de la clase de la uri, en la ontología de dominio del SKS actual.-

LISTDIRECTSUBCLASSES <*ClassURI*> - Lista la subclases directas de la clase de la uri, en la ontología de dominio del SKS actual.-

LISTSUPERCLASSES <*ClassURI*> - Lista las superclases de la clase de la uri, en la ontología de dominio del SKS actual.-

LISTDIRECTSUPERCLASSES <*ClassURI*> - Lista las superclases directas de la clase de la uri, en la ontología de dominio del SKS actual.-

ISDEFINED <ClassURI> - Contesta si la clase de la uri está definida en la ontología de dominio del actual SKS.

VALIDATESKS - Valida un SKS mediante la validación de la ontología de dominio, y verificando que cada clase del modelo unión de las descripciones semánticas de los web services pertenezca además a dicha ontología de dominio. Valido para el SKS actual.

CLASSIFY <IndividualURI> - Devuelve las clases a las que pertenece un

individuo en la ontología de dominio considerada. Valido para el SKS actual.

VALIDATECURRENTSKSONTOLOGY - Valida la ontología de dominio asociada al SKS actual, Produce un reporte de inconsitencias asociado a un código de retorno.

LISTONTOLOGYCLASSES - Lista las clases pertenecientes a la ontología de dominio en el contexto del SKS actual. Se presentan con una indentación que representa jerarquía de clases.

LISTONTOLOGYINDIVIDUALS - Lista los individuos pertenecientes a la ontología de dominio en el contexto del SKS actual.

LISTCURRSKSMODELINDIVIDUALS - Lista los individuos pertenecientes al modelo común (de todos los servicios) de un SKS (el actual).-

LISTCURRENTWSONTOLOGYCLASSES - Lista las clases de la descripción semántica corriente de web services (descripción de web service corriente fuera del contexto de un sks).

LISTCURRENTWSONTOLOGYINDIVIDUALS - Lista los individuos de la descripción semántica corriente de web services (descripción de web service corriente fuera del contexto de un sks).

SETCURRSKSWSONTOLOGY <index> - Establece una descripción de web service (es decir una WSOntology) como la descripción por defecto

en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYNAMES - Enumera el o los nombres asociados al Web Service por defecto en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYDESCRIPTIONS - Lista las descripciones asociadas al Web Service por defecto en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYINPUTS - Lista las entradas requeridas por el web service por defecto en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYOUTPUTS - Lista las salidas provistas por el web service por defecto en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYCLASSES - Lista las clases que presenta el web service por defecto en el contexto de un SKS (el sks corriente).

LISTCURRSKSWSONTOLOGYINDIVIDUALS - Lista los individuos que presenta la descripción semántica del Web Service por defecto en el contexto de un SKS (el sks corriente).

ListCurrSKSWSOntogolyURI - Lista la URI asociada a la descripción semántica del web service por defecto en el contexto de un SKS (el sks corriente).

RESETCURRSKSWSONTOLOGY - Descarta el valor por defecto de una

ontología de servicios web en el contexto del SKS corriente (el sks corriente).-

SETAVAILABLEWSINPUTALGORITHMPARAM <URI> Agrega un parámetro de entrada del algoritmo de discovery. Corresponde un parámetro de un elemento disponible para el llamado de un web service semántico.

SETREQUIREDWSOUTPUTALGORITHMPARAM <URI> - Agrega un parámetro de entrada del algoritmo de discovery. Corresponde a un parámetro de un elemento que es necesario que un servicio web semántico devuelva.

SHOWAVAILABLEWSINPUTALGORITHMPARAMS - Muestra parámetros de entrada del algoritmo de discovery. Muestra aquellos parámetros que están disponibles para ser consumidos por distintos servicios web que serán devueltos por el algoritmo.

SHOWREQUIREDWSOUTPUTALGORITHMPARAMS - Muestra parámetros de entrada del algoritmo de discovery. Muestra aquellos parámetros que son requeridos como salida en aquellos servicios que devolverá el algoritmo.

EXECUTEALGORITHM - Ejecuta el algoritmo de discovery asociado al SKS actual.

SHOWEXECUTIONRESULTS - Devuelve el resultado (lista de URLS) de

la ejecución del algoritmo.

SETEXPECTEDRESULT <nombre> **-** Almacena el nombre que se indica como un resultado esperado de la ejecución. Existen métricas (notablemente recall y precision) que requieren conocer el conjunto de publicaciones que debe devolver el algoritmo para realizar sus cómputos. Con este comando se van agregando URLs de publicaciones que se espera que el algoritmo devuelva. Para funcionar correctamente, este comando se envía al servidor previo a la ejecución del algoritmo, dado que el cálculo se produce automáticamente al final de la ejecución.-

SHOWEXPECTEDRESULTS - Devuelve la lista de resultados esperados de la ejecución de un algoritmo. Es la lista ingresada a través de los comandos *SetExpectedResult*.

CLEAREXPECTEDRESULTS <nombre> - Elmina la nombre que se indica de la lista de resultados esperados de una ejecución. Tiene por efecto que no se considere la URL que se pasa como parámetro como un resultado relevante entre los que devuelve el algoritmo.

CLEARALLEXPECTEDRESULTS - Elmina la lista de resultados esperados de una ejecución.

SHOWMETRICSRESULTS - Muestra todas las métricas recolectadas de la última ejecución. Este comando se debe enviar posteriormente a la ejecución del algoritmo.