Descubrimiento semántico de Servicios Web RESTful en entornos móviles

Tesis para optar al título de Ingeniero en Electrónica y Telecomunicaciones

Segundo Gonzalo Acte Coral
Carlos Arturo Cabrera Camacho

Director: Ing. Fulvio Yesid Vivas

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación de Servicios Avanzados en Telecomunicaciones
Popayán, Mayo de 2012

Resumen

Estamos siendo testigos del aumento de los Servicios Web publicados tanto SOAP (*Service Object Access Protocol*) como REST (*Representation State Transfer*), gracias a que las tecnologías de desarrollo permiten crear servicios de forma más rápida, en cualquier lenguaje y con mejores prestaciones para el usuario. Con el crecimiento que ha tenido la Web 2.0, los servicios Web REST han ganado importancia gracias a factores como su simplicidad y escalabilidad, lo que ha producido un incremento importante en el número de servicios disponibles en la Web. El descubrimiento de dichos servicios se ha convertido en un verdadero desafío ya que no se cuenta con un estándar para la descripción y publicación (como WSDL y UDDI en los servicios Web SOAP) y todo se ha resumido a un emparejamiento sintáctico entre las peticiones de un usuario y las palabras claves que describen el servicio para poder encontrarlos.

Es deseable que los servicios que habitualmente son consumidos desde un computador de escritorio también lo sean desde un dispositivo móvil, ya que este mercado está creciendo rápidamente y ofrece valor agregado a los usuarios como la posibilidad de acceder a los servicios en cualquier lugar y tiempo. Pero el entorno impone barreras adicionales como el tamaño de los dispositivos, las bajas velocidades de sus procesadores, memoria y batería limitada, entre otros aspectos. En el ambiente móvil se prefieren los Servicios Web RESTful ya que son más simples y requieren menor procesamiento de los dispositivos para ser consumidos.

En este sentido, este trabajo de grado propone un mecanismo de descubrimiento de Servicios Web RESTful que incluye el enriquecimiento semántico de éstos para salvar la barrera sintáctica, y así tener mejores resultados en el proceso de descubrimiento, con un algoritmo que tiene en cuenta el contexto de entrega ya que los servicios serán consumidos desde un dispositivo móvil.

Tabla de contenido

Lista de Figuras

Lista de Tablas

Lista de Ejemplos

Capítulo 1

Introducción

1.1. Definición del problema

El descubrimiento de Servicios Web se define como la capacidad de localizar servicios a partir de parámetros y descripciones públicas de su funcionamiento, los cuales son aportados por los consumidores. El objetivo del proceso es localizar un Servicio Web apropiado que cumpla con los criterios funcionales de búsqueda que proporciona el agente o usuario final (Booth, Haas et al. 2004). Un mecanismo de descubrimiento eficiente permite una construcción eficaz de sistemas de software a partir de elementos individuales que son intercambiados dinámicamente con el fin de proveer un servicio (Fensel, Keller et al. 2005).

El proceso de descubrimiento se hace complejo en el contexto de los Servicios Web RESTful debido a que no cuentan con un documento formal legible por máquinas para especificar el formato de las entradas y salidas, ni el tipo de petición HTTP (*Hypertext Transfer Protocol*) que debe usarse con el fin de invocar el servicio (Lathem 2007). En este campo se han desarrollado algunas iniciativas como hRESTS (*HTML for RESTful Web Services*), WADL (*Web Application Description Language)* y WSDL 2.0 (*Web Services Description Language v2.0*) que facilitarían el descubrimiento sintáctico, sin embargo, debido a que no existe un estándar para la publicación de esta clase de Servicios Web, se presentan ambigüedades en el nombre, lo que hace compleja su búsqueda. Por ejemplo, si se tienen dos servicios descritos como Sumadora y Adicionadora, son distintos sintácticamente pero equivalentes lógicamente, y en el caso de una eventual búsqueda se podría descartar uno de ellos (Hermida, Corrales et al. 2009). Por lo tanto, es preciso buscar la manera de agregar anotaciones semánticas que permitan soportar algún mecanismo de recuperación más sofisticado (Maleshkova, Pedrinaci et al. 2009).

Por otro lado, la creciente evolución de las tecnologías móviles sugiere la posibilidad de descubrir servicios existentes en la Web de manera ubicua, a través de diversos dispositivos, no obstante esto supone dificultades adicionales como la capacidad de procesamiento limitada y la heterogeneidad de terminales (Bianchini, De Antonellis et al. 2006). Consecuentemente, para permitir la participación activa de un usuario en la Web 2.0 es necesario capturar el contenido que genera en cualquier momento y lugar, lo cual es posible gracias a las herramientas idóneas que le proporciona el dispositivo móvil (García Hervás 2008).

Teniendo en cuenta lo anterior, se hace necesario tener un método de descubrimiento de Servicios Web RESTful, que permita la búsqueda de estos de forma semántica y que se adapte al entorno móvil, es decir, que tenga en cuenta el contexto de entrega y el dispositivo desde el cual se hace la petición (Gutiérrez 2009; Kalin 2009). Con el fin de proporcionar una solución al problema planteado, surge la siguiente pregunta de investigación: ¿Cómo realizar el descubrimiento semántico de Servicios Web RESTful, en entornos móviles?

1.2. Escenarios de Motivación

Tener un motor de búsqueda de Servicios Web RESTful que opere de manera semántica en un entorno móvil podría ser útil en los siguientes escenarios:

Composición de servicios: La composición de servicios se define como el proceso mediante el cual dos o más servicios que ofrecen una funcionalidad básica se combinan para ofrecer una más compleja (Hermida 2010). Los Servicios Web de funcionalidades básicas o atómicas son muy comunes ya que son implementados de esa forma pensando en la reutilización, siguiendo el principio de "no reinventar la rueda", pero los usuarios por lo general tienen necesidades complejas que requieren que varios Servicios simples trabajen en conjunto. La composición de servicios no se aborda en esta tesis pero este proceso tiene como pilar fundamental el descubrimiento de servicios, así que es necesario un descubrimiento de calidad. Suponemos el caso en que un usuario disfruta de sus vacaciones y necesita de un servicio completo que le facilite trabajos como reservar vuelos y habitaciones, encontrar restaurantes y eventos para su distracción; este servicio debe ser creado combinando otros que cumplan funcionalidades específicas (como reservar una habitación en un hotel) y es en este caso que el descubrimiento de dichos servicios se vuelve importante.

Servicios dependientes del contexto: El entorno móvil se caracteriza por ser heterogéneo, las peticiones pueden provenir de un sinnúmero de dispositivos con diferentes características. Continuando con el ejemplo de las vacaciones del usuario, si desea reservar una habitación son indeseables los servicios que no pueden ser consumidos desde el dispositivo por limitaciones en el hardware.

Flexibilidad al buscar servicios: La más grande ventaja que traerá contar con un motor de búsqueda que se base en semántica es la flexibilidad que se le ofrece al usuario a la hora de buscar servicios, siguiendo con el ejemplo si el usuario desea saber el estado del clima para programar sus salidas a sitios turísticos, y cuenta con un motor de búsqueda basado en sintáctica, si su búsqueda se realiza como "clima", se podrían descartar servicios que cumplen con los requerimientos del usuario pero que podrían estar descritos como meteorología o tiempo, un motor de búsqueda semántico no tendría este problema.

Servicios Web REST: Los servicios Web REST, como se menciona en diversas secciones de este trabajo, se han incrementado en número y su adopción por parte de empresas y desarrolladores está más consolidada con el pasar de los días. De esta manera, para el desarrollo de trabajos futuros en el área de los servicios Web REST, es de suma importancia contar con un motor de descubrimiento que opere sobre éstos.

Investigación en REST: motivar la investigación y desarrollo de trabajos similares dentro de la comunidad universitaria (especialmente la Universidad del Cauca), mostrando como atractivo el hecho de usar tecnologías que están creciendo (como los servicios web RESTful) y que posiblemente dominaran el mercado.

1.3. Objetivos

Objetivo General

- Proponer un mecanismo de descubrimiento de Servicios Web RESTful basado en semántica.

Objetivos Específicos

- Adaptar un algoritmo para el descubrimiento semántico de Servicios Web RESTful.

- Implementar el algoritmo definido para la realización del descubrimiento semántico de Servicios Web RESTful.

- Evaluar el algoritmo definido e implementado para establecer su eficiencia y precisión.

1.4. Estructura del documento

A continuación se presenta una descripción resumida del contenido de cada capítulo:

En el capítulo 2 se presentan los conceptos generales asociados a este trabajo de grado: servicios Web RESTful, sus principales características, su importancia, las ventajas y desventajas que tiene seguir la arquitectura REST. Conceptos como el descubrimiento de servicios y el enriquecimiento semántico también son presentados en este capitulo, esto con el fin de entender la necesidad de estos procesos, las ventajas y desventajas, los diferentes métodos mediante los cuales se podría realizar, y finalmente, se presenta una breve descripción de como se encuentra posicionado el consumo de servicios desde dispositivos móviles.

En el capítulo 3 se presenta conceptos de tecnologías usadas en el desarrollo de la solución. Algunas de los estándares y tecnologías usados son:

- hRESTS para la descripción de los servicios web RESTful.
- RDF como modelo estándar para el intercambio de datos en la web y para la publicación de servicios en el repositorio.
- XSLT para la transformación de los archivos de descripción a RDF.
- WURFL como repositorio de dispositivos desde el cual se puede obtener información básica o detallada de los dispositivos móviles para usar con cualquier propósito, en nuestro caso, obtener las características desde el cual se solicitan los servicios.
- WordNet como una base de datos léxica del idioma ingles que puede ser usada como ayuda para ampliar la consulta y obtener los servicios.

En el capitulo 4 se presenta todo el proceso de desarrollo de la solución, la adaptación de algoritmos, el levantamiento de requerimientos, el modelado del sistema (casos de uso, diagrama de clases y paquetes), arquitectura de referencia y el diagrama de despliegue que dan soporte al descubrimiento de servicios.

En el capitulo 5 se presentan los resultados obtenidos con el sistema solución, la metodología empleada para obtener los resultados, se consideran aspectos como la

eficiencia de la búsqueda e indicadores de desempeño. El análisis de los resultados esta enfocado al entorno móvil.

En el capitulo 6 se presenta las conclusiones del trabajo y las posibles alternativas de trabajos futuros relacionados con el descubrimiento semántico de servicios Web RESTful o con la mejora del sistema construido en este trabajo.

Capítulo 2

Estado Actual del Conocimiento

A continuación se presentan las bases teóricas en las cuales se encuentra enmarcado este proyecto y algunos de los trabajos más relevantes usados para la consecución de los objetivos. Una de las metas que se desea conseguir es la construcción de una base de conocimiento que quedará como aporte de este trabajo de grado cuya utilidad se vea reflejada en trabajos futuros relacionados con este u otros temas afines al descubrimiento de servicios Web basados en REST. Esta base de conocimiento será constituida tomando como parámetros los lineamientos propuestos en (Serrano Cartaño 2008), en donde se definen unas fases de referencia para ejecutar una investigación documental y que se aprecian en la Figura 1.

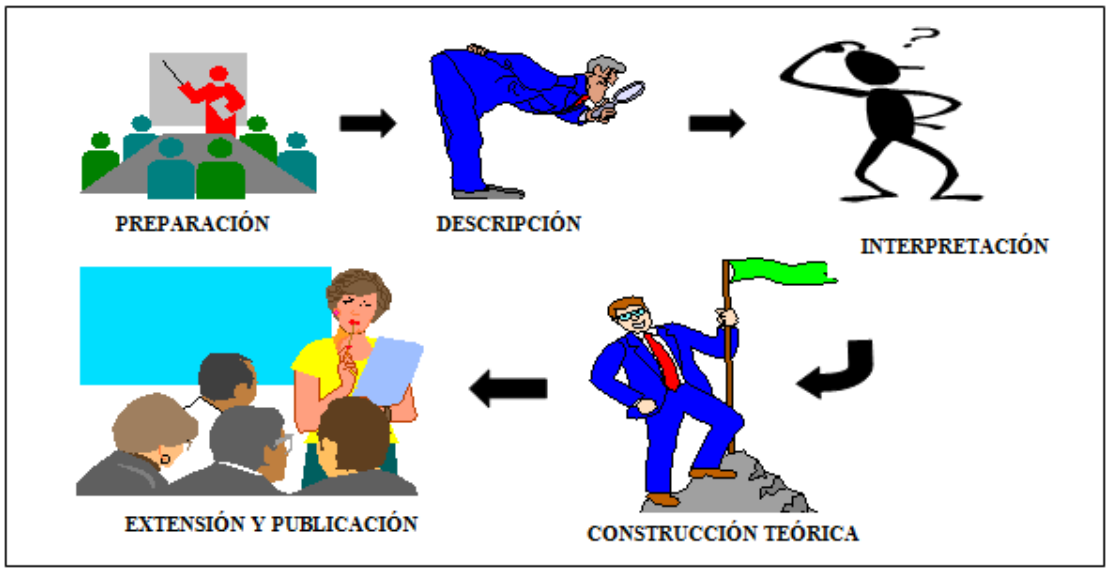


Figura 1. Fases de referencia para la construcción del estado del arte. Tomado de (Serrano Cartaño 2008)

En forma resumida, las actividades que se especifican en cada una de las fases de referencia en (Serrano Cartaño 2008) son:

- ***Fase de Preparación:*** en esta fase se efectúan todos los detalles que tienen que ver con la orientación de la investigación. Se define un lenguaje común, y además se definen el tema central y los núcleos temáticos[1] del estudio.
- ***Fase de Descripción:*** en esta fase se ejecuta una descripción detallada de las unidades de análisis[2] que comprenden cada uno de los núcleos temáticos a través de unos formatos definidos denominados fichas. En estos formatos se

[1] Un núcleo temático se define como un subtema del tema central de investigación que delimita un campo de conocimiento.
[2] La frase "unidad de análisis" se refiere al documento que trata un núcleo temático o subtema perteneciente al área temática seleccionada con el fin de construir una visión global de conocimiento.

especifica toda la información relevante que aporta cada unidad de análisis, la temática que aborda, su metodología, entre otros aspectos.

- ***Fase de Interpretación:*** en esta fase se amplía el estudio realizado en la fase anterior, tomando como base el conjunto de fichas definidas anteriormente para cada unidad de análisis de cada núcleo temático. Esta fase va más allá de lo meramente descriptivo.
- ***Fase de construcción teórica:*** en esta fase se realiza un balance del resultado del estudio por cada núcleo temático de la fase anterior con dos propósitos principales:
    - presentar el estado actual del conocimiento en la temática central estudiada.
    - identificar focos de investigación que permitan orientar el proceso investigativo que se desea llevar a cabo.
- ***Fase de Extensión y Publicación:*** esta fase consiste en la divulgación de los resultados obtenidos en todo el proceso.

Para el desarrollo de la base de conocimiento de este trabajo de grado se abordaron las cinco fases presentadas anteriormente. La divulgación de los resultados (fase de referencia cinco) se hace de forma escrita en esta monografía. Igualmente, se tienen en cuenta las actividades de referencia planteadas en (Serrano Cartaño 2008) *"para la elaboración de la base inicial de conocimiento de un proyecto de desarrollo tecnológico"*, que constituyen, según se menciona en el documento, una personalización del "Modelo para la Investigación Documental" descrito en (Serrano Cartaño 2008). Así, a continuación se describe el alcance de la base de conocimiento establecida en el marco de este trabajo de grado.

Declaración del alcance de la Base de Conocimiento

Para cumplir con el objetivo principal de este trabajo de grado (proponer un mecanismo de descubrimiento de Servicios Web RESTful basado en semántica) es necesario realizar un análisis del estado del conocimiento relacionado con la temática que sirva como soporte para la construcción de la solución planteada. Conjuntamente, es necesario realizar la base de conocimiento con el fin de identificar vacíos, limitaciones, dificultades, tendencias y logros obtenidos en la realización de mecanismos de descubrimiento de servicios Web REST para diferenciar claramente la contribución aportada en esta temática. Explícitamente, se espera que el soporte teórico aclare los avances realizados en la temática central abordada y dé respuesta a los siguientes interrogantes:

- ¿Qué tecnologías son utilizadas para la descripción de servicios Web SOAP y REST?
- ¿Qué tecnologías son utilizadas para realizar el proceso de descubrimiento de servicios Web REST?
- ¿Qué tecnologías son utilizadas para adicionar semántica al proceso de descubrimiento?
- ¿Qué algoritmos de descubrimiento son utilizados para soportar el descubrimiento de servicios REST?
- ¿Qué se debe tener en cuenta para que el descubrimiento de servicios REST sea posible desde dispositivos móviles?

Esta declaración del alcance servirá para la posterior revisión y validación de la base de conocimiento, la cuál es una actividad de referencia reseñada en (Serrano Cartaño 2008).

De esta manera, la presentación del estado del arte se divide en dos partes: en la primera se presentan conceptos generales necesarios para entender el contexto del proyecto, y en la segunda se presentan los trabajos relacionados más relevantes relacionados con la temática central del proyecto que a su vez son divididos en cuatro núcleos temáticos principales: algoritmos de descubrimiento de servicios Web, descripción de servicios Web RESTful, semántica y entorno móvil.

2.1. Conceptos generales

2.1.1. Descubrimiento de Servicios y descubrimiento semántico

Gracias al paradigma de SOA, miles de servicios Web disponibles en Internet se convierten en puntos de entrada para aplicaciones de mayor complejidad, debido al proceso de Composición de servicios. Pero, para hacer esto posible, antes esos servicios deben ser descubiertos (Bachlechner, Siorpaes et al. 2006).

El descubrimiento de servicios es un proceso que tiene como finalidad encontrar un Servicio Web apropiado que cumpla con los requerimientos que proporciona el solicitante, que puede ser un agente software o un usuario final. Se fundamenta en el emparejamiento de descripciones abstractas del servicio que se desea consumir con anotaciones semánticas de los Servicios Web disponibles (Fensel, Keller et al. 2005). El proceso de descubrimiento debe ser capaz de reconocer y calcular la similitud entre los requerimientos proporcionados por un Servicio Web y los requeridos por el solicitante (Sycara, Paolucci et al. 2003). En (Burstein, Bussler et al. 2005) se define el descubrimiento de servicios como un proceso mediante el cual un cliente identifica servicios candidatos que puedan satisfacer sus objetivos y se definen tres actores principales:

- **Proveedores de servicios:** ofrecen servicios Web a los solicitantes.
- **Solicitantes de servicios:** buscan servicios que cumplan con sus objetivos.
- ***Matchmakers*:** reciben descripciones de servicios Web disponibles ofrecidos por los proveedores y los emparejan contra los requerimientos de los solicitantes.

Las tareas que cada actor debe realizar con el fin de permitir el proceso de descubrimiento son:

- Los proveedores deben describir los servicios Web que ofrecen en términos de sus capacidades y restricciones.
- Los solicitantes deben crear caracterizaciones abstractas o requerimientos formales de los servicios que buscan con el fin de facilitar el emparejamiento con las capacidades publicadas de los servicios.
- Los solicitantes deben interactuar con *Matchmakers* que puedan responder a las consultas de descripciones de los servicios ofrecidos.
- Los *Matchmakers* deben comparar las descripciones de las consultas y las capacidades de los servicios.

- Los solicitantes deben decidir si los servicios retornados pueden satisfacer las precondiciones especificadas en la consulta, con el fin de consumir el servicio.

En el contexto de REST, al no existir una forma estandarizada y ampliamente aceptada para la publicación de servicios Web, se produce una generalizada ambigüedad en el nombre de los servicios, lo que hace necesario que el mecanismo de descubrimiento sea semántico. Para ello, la comunidad de la Web semántica establece que los servicios Web deben estar descritos en un lenguaje expresivo con una semántica bien definida, con el fin de superar las limitaciones del descubrimiento sintáctico (Joines, Willenborg et al. 2003).

Existen diferentes aproximaciones para soportar el descubrimiento semántico de servicios, que se mencionan a continuación (Keller, Lara et al. 2005):

2.1.1.1. Descubrimiento basado en palabras claves

Es una de las maneras posibles de realizar el descubrimiento de servicios Web. Puede ser usado para filtrar y clasificar, de una manera rápida, una gran cantidad de descripciones de servicios. En su forma básica, este tipo de descubrimiento es realizado emparejando las palabras claves proporcionadas por el solicitante contra las palabras claves extraídas de la descripción de los servicios, teniendo en cuenta la estructura de dicha descripción (propiedades funcionales, no funcionales, objetivos, precondiciones y efectos, etc.). Este tipo de descubrimiento puede ser utilizado también para encontrar relaciones entre palabras clave proporcionadas por el usuario y conceptos de una ontología utilizada para formalizar servicios y objetivos (Keller, Lara et al. 2005).

2.1.1.2. Descubrimiento basado en descripciones semánticas simples de servicios

Antes de definir formalmente esta modalidad de descubrimiento, es necesario definir el concepto de ontología ya que es clave en el proceso de descubrimiento basado en descripciones semánticas.

2.1.1.2.1. Ontologías

En el contexto de intercambio de información, una ontología es una descripción formal de conceptos jerarquizados y sus relaciones acerca de un dominio específico de conocimiento. Son utilizadas generalmente para definir una estructura formal con el fin de expresar un vocabulario acerca de un tópico específico, de manera que pueda ser compartido por máquinas o agentes software. Las ontologías proporcionan una terminología compartida y explícita, en donde se definen las relaciones entre los conceptos que la conforman, lo que las hace ideales para la descripción de servicios Web y de objetivos proporcionados por el solicitante. Adicionalmente, las ontologías pueden expresarse en lenguajes avanzados de lógica que tienen características bastante completas, lo que permite que a través de herramientas software estandarizadas se permitan utilizar servicios de inferencia sobre el dominio específico acerca del cual trata la ontología (Keller, Lara et al. 2004; McGuinness and Van Harmelen 2004). Para modelar ontologías existen diferentes lenguajes; entre los más destacados están OWL, RDF, WSML, entre otros. Las ontologías permiten el descubrimiento semántico, cuya conceptualización se presenta en los siguientes párrafos.

Las ontologías (cuya definición se presenta en la anterior sección) surgen como un mecanismo que proveen un vocabulario común para referirse a algún concepto en algún dominio específico. Este tipo de descubrimiento adiciona información semántica legible por máquinas a las descripciones de los servicios y a las consultas realizadas por el solicitante (mediante anotaciones semánticas que apuntan a conceptos definidos en una ontología) a través de un lenguaje formal, como WSML (*Web Service Modeling Language*). Asimismo, se permite que el mecanismo de recuperación realice inferencia sobre la ontología y de esta manera obtenga el máximo provecho sobre el dominio de conocimiento específico, lo que se ve reflejado en los altos valores de *precision y recall* que se logran (Keller, Lara et al. 2004).

En este tipo de descubrimiento se describen las capacidades de los servicios como un conjunto de objetos que el servicio está en capacidad de ofrecer. Así, se abstraen ejecuciones concretas del servicio Web y la relación entre el estado previo de la ejecución del servicio y el estado posterior del mismo. Concretamente, según este modelo, un servicio Web es visto como un objeto computacional que puede ser invocado por un cliente, el cuál en tiempo de ejecución proporciona toda la información necesaria para que el mecanismo de descubrimiento identifique y entregue el servicio concreto que ha sido solicitado. En este tipo de descubrimiento no se tiene en cuenta las entradas del servicio y su relación con las salidas y efectos de la ejecución del servicio Web. Para esta clase de descubrimiento se considera WSML DL como el lenguaje candidato y se recomiendan razonadores lógicos como RACER[3], FACT++[4] y Pellet[5] (Keller, Lara et al. 2004; Keller, Lara et al. 2005).

Sin embargo, a pesar de la precisión que se logra con este mecanismo y a la inclusión de información semántica legible por máquinas, las clases de información semántica que pueden ser expresadas en esta alternativa de descubrimiento son limitadas. Para ciertas aplicaciones y usuarios en las que se desee modelar algunos matices del comportamiento o funcionalidad de los servicios Web, buscando lograr un nivel más alto de automatización en un sistema software dinámico basado en servicios, es necesario contar con un mecanismo de descubrimiento más moderno (Keller, Lara et al. 2004).

2.1.1.3. Descubrimiento basado en descripciones semánticas enriquecidas.

En este tipo de descubrimiento se extiende la propuesta de descripción de los servicios Web en términos de un conjunto de objetos que el servicio está en capacidad de ofrecer, incrementando el nivel de detalle del modelo de servicio utilizado. Se

[3] RACER (*Renamed Abox and Concept Expression Reasoner):* Es un razonador semántico que provee servicios de inferencia (estándar y no estándar) altamente optimizados para manejar aplicaciones con ontologías robustas. Ofrece soporte para OWL, persistencia de datos y también tiene la posibilidad de realizar consultas a través un lenguaje propietario llamado *nRQL* (new *Racer Query Language).* Más información en: http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[4] Es un razonador semántico diseñado para la variante OWL-DL. FACT++ es implementado en C++ con el fin de maximizar la portabilidad y mejorar la eficiencia de la herramienta. Más información en: http://owl.man.ac.uk/factplusplus/

[5] Es un razonador semántico OWL 2 para el lenguaje Java. Es ideal para aplicaciones que necesitan realizar procesos avanzados de inferencia sobre información utilizando OWL. Además, Pellet tiene soporte para perfiles OWL 2 incluyendo OWL 2 DL. Más información en: http://clarkparsia.com/pellet/

considera la variante WSML Full para el modelamiento de los servicios Web. Conjuntamente, se tiene en cuenta la relación entre las entradas y las salidas y efectos de las ejecuciones del servicio Web, hecho no considerado en la modalidad de descubrimiento anterior. De esta manera, una ejecución de un servicio Web es descrita por un conjunto de salidas y efectos (que dependen de los valores específicos de los parámetros de entrada) y un servicio Web es visto como una colección de posibles ejecuciones y modelado como una recopilación de conjuntos de entradas y efectos. En la **Figura 2** se puede apreciar el modelo de servicio extendido para soportar esta modalidad de descubrimiento (Keller, Lara et al. 2004; Keller, Lara et al. 2005).

Como se observa en la **Figura 2. Modelo de servicio extendido basado en conjuntos**, cuando se invoca un servicio Web *ws* con valores de entrada $i_1 \ldots i_n$ con un estado previo del servicio *pre-state*, la ejecución del servicio resulta en un estado posterior diferente *post-state* donde el servicio entrega un conjunto de elementos como su salida $ws^{out}(i_1 \ldots i_n)$ y un conjunto de efectos $ws^{eff}(i_1 \ldots i_n)$. En el framework WSMO el cliente especifica su deseo como un objetivo (lo que espera conseguir del servicio Web). Así, la descripción del objetivo se compone de un conjunto de información deseada $goal^{out}$ así como un conjunto de efectos deseados $goal^{eff}$. De esta manera se determina el grado de correspondencia entre el servicio Web y los objetivos especificados por el cliente (Keller, Lara et al. 2004).

2.1.2. Servicios Web

Un servicio Web es un sistema software diseñado para soportar la interacción interoperable entre máquinas sobre una red que generalmente es Internet (Booth, Haas et al. 2004), esto quiere decir que pueden comunicarse entre sí, sin importar el lenguaje en el que están programados o la plataforma en la cual están alojados, gracias al uso de tecnologías Web estándar abiertas como HTTP y XML (*eXtensible Markup Language*) (Cavanaugh 2006). Esta integración de aplicaciones se realiza a alto nivel, centrándose más en servicios y menos en protocolos de red (Kreger 2001; Cavanaugh 2006). Además, los servicios Web buscan definir un mecanismo estandarizado para describir, localizar y comunicarse con aplicaciones en línea, ofreciendo una solución a la incompatibilidad presente en servicios estilo Web 2.0, convirtiendo cada aplicación que hace parte del servicio completo (antes incompatible con las otras) en un componente de servicio Web, el cual es descrito utilizando tecnologías estándar (Curbera, Duftler et al. 2002).

Cuando las aplicaciones migraron a una red se hizo necesario tener un mecanismo para que éstas pudiesen comunicarse entre sí, inicialmente surgieron una serie de protocolos propietarios entre los que se destacan COM (*Componet Object Model*) de Microsoft, y CORBA (*Common Object Request Broker Architecture*) de OMG (*Object Management Group*), cuyo gran problema era la interoperabilidad, ya que entre estos protocolos no existía una forma de enviarse mensajes. Es así como aparece SOAP para solucionar este inconveniente, adoptado como un trabajo conjunto entre varias de las empresas que ya tenían su protocolo propietario.

2.1.2.1. Servicios Web SOAP

SOAP es un protocolo basado en XML para el intercambio de mensajes y llamadas a procedimiento remoto RPC (*Remote Procedure Call*). Fue inicialmente creado por

Microsoft y posteriormente desarrollado en conjunto con otras compañías como Developmentor, IBM, Lotus, y UserLand. Los mensajes SOAP tienen una estructura muy simple: un elemento principal con dos secundarios (ambos en formato XML), uno de los cuales contiene la cabecera o *header* y el otro contiene el cuerpo del mensaje o *body.* SOAP no define un protocolo de transporte sino que utiliza los existentes como HTTP o SMTP; además, adicionalmente a la estructura básica del mensaje, la especificación define un modelo que detalla cómo los destinatarios deben procesar los mensajes SOAP y precisa también actores, que muestran quién debería procesar el mensaje (Curbera, Duftler et al. 2002).

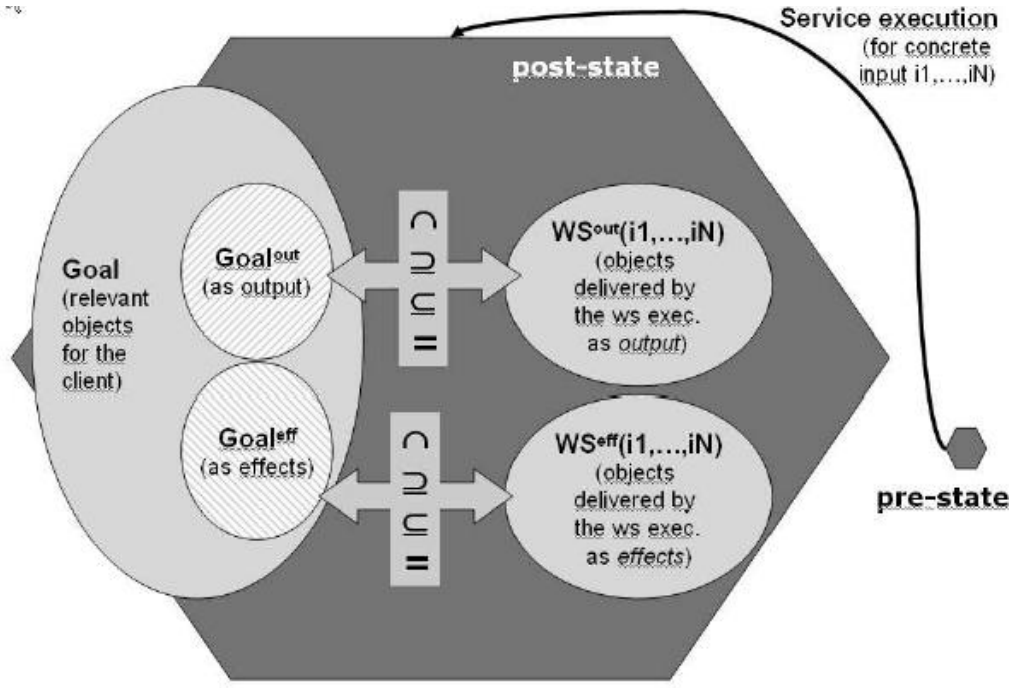


Figura 2. Modelo de servicio extendido basado en conjuntos

Por otro lado, SOAP sólo ofrece una comunicación básica, simple, y no describe en detalle los mensajes que deben ser intercambiados para interactuar exitosamente con el servicio. Para solucionar este inconveniente aparece WSDL, un formato XML desarrollado por Microsoft que describe una interfaz del servicio Web en donde especifica cómo interactuar con el servicio, proporcionando al usuario un punto de contacto. La descripción WSDL de un servicio provee dos tipos de información: una descripción a nivel de aplicación (es decir, la interfaz del servicio como tal, basada en XML), y los detalles específicos dependientes del protocolo que los usuarios deben seguir para accederlo (en otras palabras, los detalles concretos con respecto a *cómo y dónde* la interfaz del servicio puede ser accedida) (Curbera, Duftler et al. 2002; Toma, Steinmetz et al. 2008).

Pero no todo en SOAP es perfecto. Sus mensajes necesitan de un nivel de detalle alto tanto para la solicitud como para la respuesta, lo que genera mayor consumo de recursos como el ancho de banda, lo cual los hace inadecuados para ser consumidos desde dispositivos con capacidades limitadas de procesamiento como los terminales móviles (Hamad, Saad et al. 2009; Aijaz, Chaudhary et al. 2010), lo cual es una prioridad en este trabajo de grado. Además, pueden existir problemas de

interoperabilidad cuando en la interfaz del servicio están presentes tipos de datos nativos o lenguajes de construcción de la implementación del servicio. Igualmente, la escalabilidad se ve implicada ya que los métodos que se definen en SOAP no son estándar, y así las implementaciones prematuras de los servicios tienen enormes inconvenientes de interoperabilidad y de mantenimiento por tecrceros (Pautasso, Zimmermann et al. 2008).

2.1.3. REST

REST es un estilo arquitectónico descrito por Roy Fielding en su Trabajo de Doctorado en Información y Ciencias de la Computación (Fielding 2000). Fielding sostiene que a diferencia de los sistemas transaccionales en los cuales los servidores necesitan mantener complejas sesiones con los clientes, la Web reduce la obligación del servidor debido a preguntas auto-contenidas con respuestas simples. En muchos casos, la respuesta involucra el envío de un conjunto de datos pre computados (el estado representacional) a través de la red. El cliente puede esencialmente navegar a través de un amplio rango de recursos existentes pre computados siguiendo links de un recurso a otro. En REST, hay muchos objetos, cada uno identificado por una URI y muy pocos métodos (GET, POST, PUT, DELETE). Cada petición enviada a un objeto da como resultado la transferencia de una representación de este objeto (típicamente en forma de un documento XML, ó JSON). Este documento permite al cliente tener la oportunidad de cambiar el estado del objeto mediante la navegación a una URI diferente, la cual es referenciada en el documento (zur Muehlen, Nickerson et al. 2005).

REST tiene la intención de evocar una imagen de cómo se comporta una aplicación Web bien diseñada: una red de páginas Web, donde el usuario interactúa a través de una aplicación seleccionando links (transiciones de estado), lo que da como resultado que la siguiente página, que representa el siguiente estado de la aplicación, sea transferida al usuario y entregada para su uso (Fielding 2000).

Uno de los más importantes principios de REST es la primacía de recursos que son exclusivamente identificados por un conjunto de URI opacas (*opaque URIs*), con el objetivo de evitar acoplamiento entre clientes y servidores, de tal manera que no se realicen suposiciones acerca de la estructura de la URI. REST requiere una interfaz uniforme, es decir, un conjunto de operaciones o métodos con semántica conocida que cambie el estado de los recursos (Alarcon and Wilde 2010).

La interfaz depende del esquema de URI. Por ejemplo, para HTTP, los métodos estándar son GET, POST, PUT, DELETE y OPTIONS. Los métodos son externos a los recursos, y son invocados enviando mensajes estándar al servidor Web indicando la URI del recurso requerido, el método, el payload del mensaje y los metadatos. Un recurso puede tener múltiples representaciones que obedecen a un formato estandarizado o un tipo de media (*media type*) (como por ejemplo text/html, application/xml, etc) y puede ser negociado con el servidor Web. Las representaciones transmiten el estado de las interacciones del cliente dentro de la aplicación y contienen hipervínculos (*hyperlinks*) que permiten a los clientes descubrir otros recursos o cambiar el estado del recurso representado.

Las principales características de REST son:

Arquitectura Cliente/Servidor: las funciones que corresponden a un servicio completo se dividen entre dos entidades, el cliente y el servidor (que no necesariamente deben ser maquinas distintas), el cliente realiza peticiones y el servidor las procesa y generalmente regresa una respuesta al cliente, todo esto comunicándose con métodos estándar como los protocolos TCP/IP. La interacción con el usuario final fue delegada a los clientes, mientras que el procesamiento pesado, manejo de datos, seguridad e integridad son labores de los servidores. Dividir estas funciones hace que un sistema sea más modular y por lo tanto el mantenimiento y desarrollo ágil son una de sus mayores ventajas (Boss 2002). Otra ventaja de usar una arquitectura cliente/servidor es la centralización del control sobre todos los recursos.

La arquitectura permite que múltiples clientes se puedan conectar a un servidor y realizar peticiones en simultaneo, por esta razón los cuellos de botella son el problema clásico del modelo, la congestión que generan muchas peticiones simultaneas hacen que los tiempos de respuesta sean más lentos además de que se requiere un mayor ancho de banda en caso de que el cliente y el servidor no estén alojados en el mismo equipo. Otro problema que generalmente se presenta es la disponibilidad del servicio, si un servidor no está funcionando y no se tienen réplicas de este, los usuarios no podrán acceder a los recursos que requieren, esto no sucedería en otro tipo de arquitecturas como por ejemplo la P2P (*Peer to Peer*) (Márquez Avendaño and Zulaica Rugarcía 2004).

Sin estados: los servicios sin estado son mucho más simples de diseñar, escribir y distribuir. El cliente debe enviar en la petición hacia el servidor los datos necesarios para que esta pueda ser atendida con éxito. Además, éste es el encargado de guardar el estado de los recursos, lo cual permite una liberación de carga de procesamiento del lado del servidor.

Generalidad de interfaces: a diferencia de SOAP, en donde el desarrollador puede definir una serie de métodos nuevos para cada aplicación que desarrolle, el estilo de arquitectura de REST requiere que se utilicen los métodos definidos por HTTP (POST, GET, PUT y DELETE) para hacer uso de los recursos, los cuales son entidades localizables mediante una URL (*Uniform Resource Locator*).

En los Servicios Web existe algo conocido como “nombre”, estos describen conceptos o archivos almacenados en la red, es decir que cada nombre está representado por una URI (*Uniform Resource Identifier*). Las acciones que se pueden realizar sobre cada nombre recibe la denominación de “verbo”, lo que busca REST es que un grupo reducido de verbos sea universal, es decir que apliquen para todos los nombres, de no ser así, conocer todas las acciones que se pueden aplicar sobre los recursos que existen en la web sería una tarea imposible, además de que se tendrían problemas de redundancia (Cowan 2005).

Hipermedios: es una tecnología de presentación y acceso a la información, la cual describe recursos como texto, imágenes, video, sonido, etc., y la manera en que estos están conectados de modo jerárquico o asociativo. El ejemplo más claro de hipermedia es la Web, en donde todos los recursos están representados por una URI y muchos de ellos están relacionados entre sí y son accedidos desde otros recursos (Tramullas 1997).

2.1.3.1. Servicios Web RESTful

Los Servicios Web que siguen los patrones de diseño de REST son llamados RESTful. Este tipo de servicios Web, que siguen fielmente los principios de la Web, actualmente están teniendo un gran crecimiento. Empresas como Amazon, eBay, Yahoo, Twitter, entre muchas otras, ofrecen interfaces REST de sus servicios, tanto para consumidores como para programadores. Este tipo de servicios Web pueden ser vistos como una serie de recursos identificados a través de una URL, de tal manera que los clientes que deseen usar este tipo de recursos acceden a sus representaciones a través de una serie de métodos estandarizados que describen la acción a ser realizada sobre el recurso (Toma, Steinmetz et al. 2008).

2.1.3.1.1. Ventajas y desventajas de los servicios Web RESTful

Según (Pautasso, Zimmermann et al. 2008) los servicios Web RESTful presentan las siguientes fortalezas:

- Son concebidos como simples debido a que aprovechan los conocidos estándares de la W3C y la IETF como HTTP, XML, URI, MIME y su infraestructura generalizada.
- Los clientes y servidores HTTP están disponibles para la mayoría de lenguajes de programación y plataformas hardware y software.
- Gracias a los hipervínculos y las URI, en REST es posible descubrir recursos Web sin la necesidad de la inscripción obligatoria a un repositorio centralizado.
- REST permite libertad de acción para optimizar el rendimiento de un servicio Web, ya que ofrece la posibilidad de escoger formatos de mensajes ligeros para las respuestas del servidor, como JSON, o incluso texto plano para tipos de datos muy simples. Además soporta funcionalidades como caché, *clustering,* y balanceo de carga.

En (Pautasso, Zimmermann et al. 2008) también se listan las siguientes debilidades:

- No existe claridad con respecto a las mejores prácticas para construir servicios Web RESTful.
- Para peticiones que tengan grandes cantidades de datos de entrada, no es posible codificar esos datos en la URI del recurso, lo cual podría causar un código de error al hacer una petición de ese recurso al servidor.

2.1.3.2. REST vs SOAP

Los principales servicios Web en internet utilizan REST para ofrecer sus funcionalidades. Pero esto no quiere decir que SOAP esté acabado. SOAP es ampliamente usado al interior de organizaciones por aplicaciones empresariales con el fin de integrar sistemas heredados. Sin embargo, en términos generales, REST presenta significativas ventajas con respecto a SOAP, las cuales se resumen en la Tabla 1(singh 2009).

Criterio	REST	SOAP
Flexibilidad y simplicidad	El uso de URI para la representación de cada recurso hace que los servicios Web RESTful sean más ligeros. De tal	Los servicios Web SOAP se consideran un poco más complejos ya que el cliente necesita conocer en detalle las operaciones

	manera que un desarrollador, para acceder a un recurso, simplemente tiene que crear o modificar una URL	y su semántica, debido a que éstas no son estándar
Uso de ancho de banda	Las peticiones y respuestas pueden ser cortas, haciendo un uso más eficiente de los recursos	Requiere un *wrapper* XML para cada petición y respuesta, lo que se traduce en un mayor uso de ancho de banda
Seguridad	Las peticiones REST van sobre HTTP o HTTPS. Así, del lado del servidor se puede filtrar cada mensaje analizando el comando HTTP usado en la petición. De esta forma, una petición GET siempre se considera segura ya que por definición ella no puede modificar ningún dato	Las peticiones SOAP habitualmente usan el método POST para comunicarse con un servicio Web específico, y sin examinar a fondo el *wrapper de SOAP* en el cual vienen las peticiones, es imposible determinar si la petición es o no segura. Hay que decir que la tarea de examinar a fondo las peticiones SOAP es algo difícil y consume recursos adicionales
Manejo de tipos	Del lado del cliente es necesario procesar la respuesta, que puede venir en cualquier formato estandarizado como XML, JSON, o incluso texto plano	Debido a que se definen métodos personalizados, del lado del cliente se recibe un tipo de dato específico (por ejemplo *String* o *Integer*) o un objeto
Complejidad del lado del cliente	Simplemente se hacen peticiones HTTP, lo cual es relativamente sencillo	Se requiere una librería para consumir el servicio, ya que las peticiones no son HTTP puras
Complejidad del lado del servidor	Desplegar métodos puede ser relativamente difícil ya que se necesita convertirlos a una representación formal como XML	Normalmente las librerías SOAP del lado del servidor se encargan del procesamiento de las solicitudes y respuestas, lo que podría representar una complejidad relativamente baja

Tabla 1. Comparación de REST vs SOAP

Hay que decir también que es factible que SOAP y REST convivan juntos. No son necesariamente excluyentes, ya que SOAP es ampliamente usado en escenarios de integración de aplicaciones empresariales profesionales con una vida útil más larga y requerimientos de QoS (*Quality Of Service*) avanzados, mientras que los servicios Web RESTful son usados para integraciones sobre la Web (Pautasso, Zimmermann et al. 2008). Aunque REST no es perfecto (como se observa en la Tabla 1, puede representar más complejidad del lado del servidor cuando se construye un servicio Web) si posee muchas ventajas con respecto a SOAP, razón por la cual este trabajo

de grado se centra en definir un mecanismo de Descubrimiento Semántico de servicios Web RESTful.

2.1.3.3. Servicios Web RESTful en entornos móviles

(Hamad, Saad et al. 2009) realizó una investigación sobre el consumo de Servicios Web desde dispositivos móviles, tanto SOAP como RESTful, se centró principalmente en el tiempo de respuesta a las peticiones de los usuarios y en el tamaño de los mensajes que se envían desde el servidor hasta el cliente (dispositivo móvil), esta investigación tomo en cuenta aspectos limitantes del entorno móvil, como la duración de la batería, memoria y baja velocidad de procesamiento. Con todo esto, se llegó a la conclusión de que el consumo de Servicios Web desde dispositivos móviles se puede hacer de manera más eficiente si se usa la arquitectura REST para crear los servicios. Los resultados de la investigación se muestran en la Figura 3 y Figura 4.

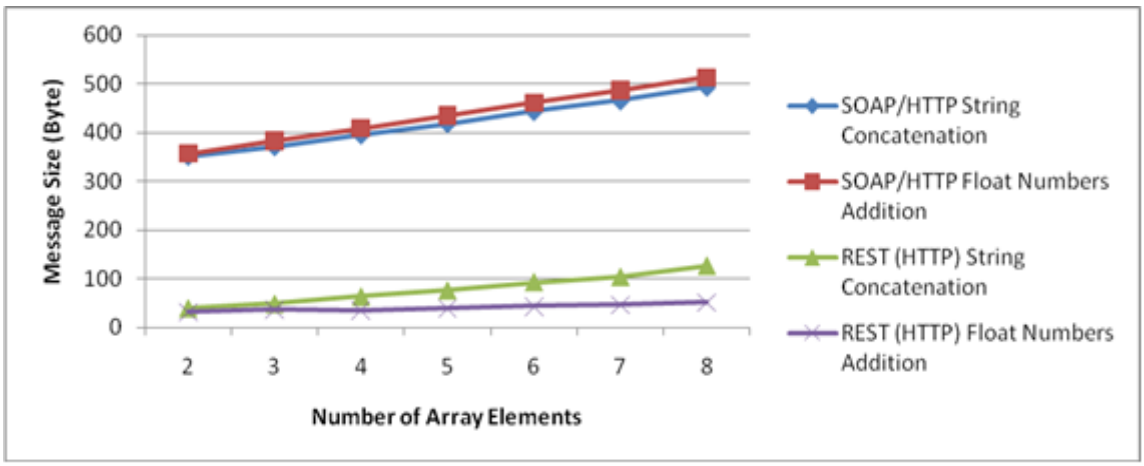


Figura 3. RESTful vs SOAP tamaño de los mensajes

Se usaron dos servicios, uno que recibe como entrada dos palabras y regresa al cliente las palabras concatenadas y otro que suma dos números. Como puede apreciarse en la Figura 3 los servicios Web SOAP emplean mensajes que superan a los de REST hasta en 4 veces, y la Figura 4 muestra que los tiempos de respuesta de los servicios RESTful son de hasta la mitad comparado con los tiempos de los servicios SOAP. Este comportamiento es deseable y mucho más si se trata de un entorno móvil.

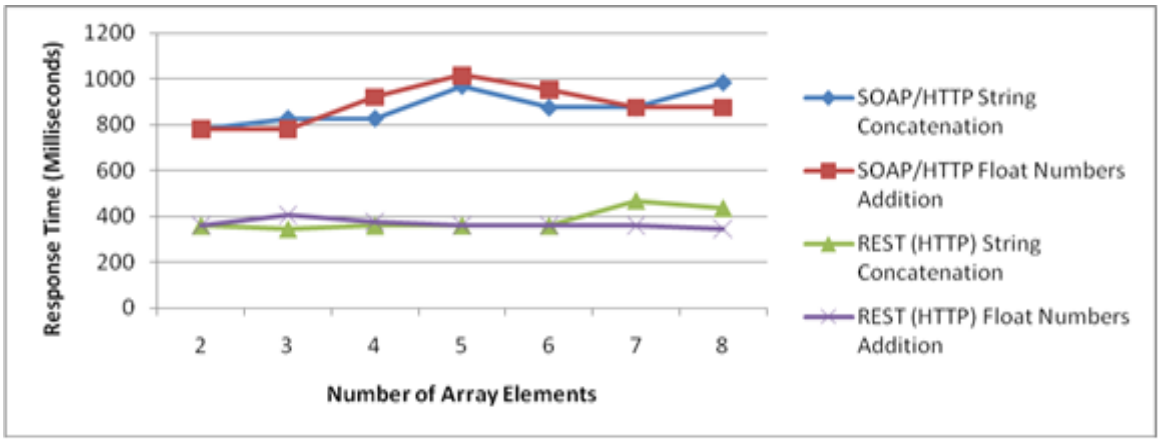


Figura 4. RESTful vs SOAP tiempos de respuesta

2.1.4. Entorno móvil

El reciente crecimiento de la capacidad que han adquirido los dispositivos móviles para conectarse a redes como Internet a velocidades cada vez mayores y con mejores prestaciones, ha sido un aliciente para que este proyecto enmarque su investigación en el entorno móvil. Según (Herrera de la Cruz 2005) las principales causas a las cuales se les atribuye que la conexión de dispositivos inalámbricos a Internet se de con mayor frecuencia son:

- Nuevas tecnologías de conexión como UMTS en donde se alcanzan velocidades de transferencia de hasta 2Mbps y que evidencian el gran avance en este tema comparándose con tecnologías anteriores como GPRS en donde se alcanzaban 115Kpbs. La Figura 5 muestra la evolución de las redes.
- La conexión para dispositivos móviles es ofrecida por los operadores de telecomunicaciones a precios cada día menores, incentivando su uso.
- Expansión de las redes inalámbricas WiFi de acceso gratuito, principalmente en sitios comerciales públicos, así como redes domesticas.

En la actualidad, y por un proceso rápido de desarrollo de nuevas tecnologías para dispositivos móviles, estos cuentan con características que les permiten consumir servicios, ya sea en forma de aplicaciones nativas o en forma de servicios web. A pesar de sus favorables características, aun es importante considerar, a la hora de desarrollar servicios, que estos tienen limitaciones tanto en su hardware como en su software; restricción en el tamaño de sus pantallas, procesadores de baja velocidad, memoria limitada, consumo de batería y bajas velocidades de conexión (comparándose con las conexiones para computadores de escritorio) son algunas de las limitaciones que deben ser tenidas en cuenta (Hamad, Saad et al. 2009).

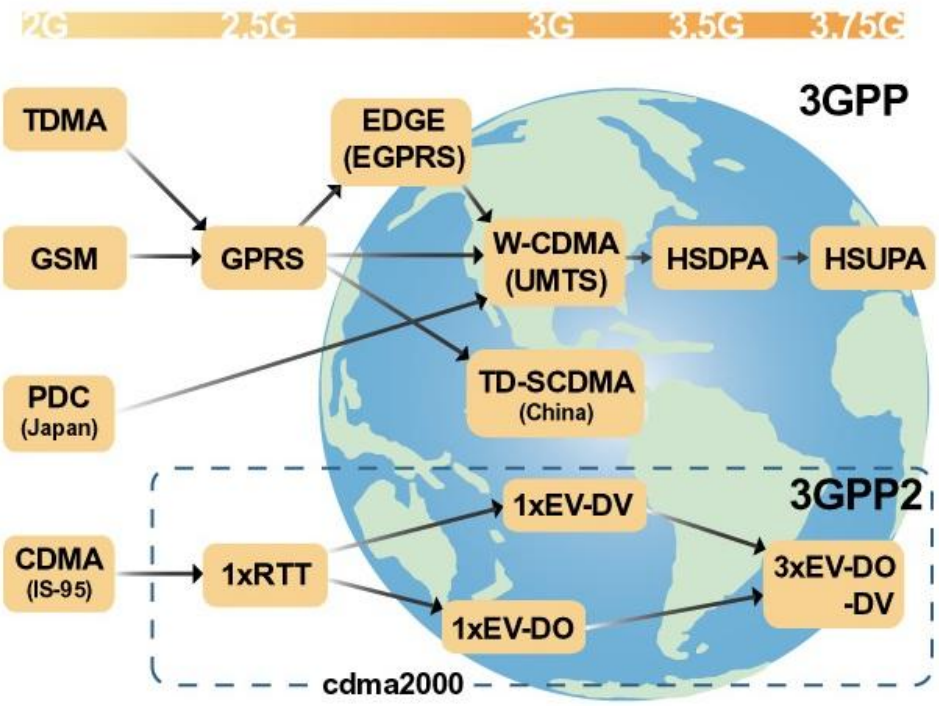


Figura 5. Evolución de las redes celulares

Debido al notable crecimiento de terminales móviles en el mundo, las aplicaciones y servicios disponibles para las diferentes plataformas han crecido exponencialmente, permitiendo en la actualidad conseguir un servicio para necesidades muy específicas. Pero anteriormente no se tenia esta gama de servicios disponibles para dispositivos, los servicios web inicialmente eran usados desde móviles tal cual se hace desde computadores de escritorio, lo cual era complejo, ya que, imágenes grandes o menús extensos, que son de fácil exploración desde un computador, difícilmente se podían visualizar desde un dispositivo móvil, y no sólo era cuestión de presentación, sino también de tecnologías usadas en la elaboración de los servicios y paginas web, ya que, si una página hacía uso de frameworks para mejorar sus funcionalidades, posiblemente se presentaría errores si se accedía desde dispositivos móviles, basta recordar la dificultad que existía antes para abrir paginas con flash o el uso de Javascript para evidenciar las incompatibilidades entre el entorno móvil y el entorno de escritorio (Herrera de la Cruz 2005).

Pero no todo en los dispositivos móviles son limitaciones, el entorno ofrece una característica única que posiblemente no es tan eficiente desde un dispositivo de escritorio y es la información de contexto. Métodos no tan eficientes como Cell-ID o más precisos como el uso de un GPS son usados para determinar de forma aproximada la ubicación del dispositivo móvil con diversos objetivos, entre los cuales esta enriquecer los servicios con información de localización (servicios llamados LSB). Un claro ejemplo de servicio enriquecido con información de localización es *Google Latitude* en donde a través de la localización del dispositivo se puede ver en donde se encuentra una persona en cualquier momento en un mapa(ilico.net 2011).

2.2. Trabajos relacionados

En esta sección se presentan los trabajos relacionados con el área temática que se aborda en este trabajo de grado. Los trabajos, producto de la investigación documental realizada, son clasificados en cuatro núcleos temáticos que son: algoritmos de descubrimiento de servicios Web, descripción de servicios Web RESTful, semántica y entorno móvil. A continuación se presenta un análisis de todos los trabajos relevantes por cada núcleo temático, en donde se menciona el aporte y las brechas de cada uno para dar cumplimiento a los objetivos planteados en este trabajo de grado.

2.2.1. Algoritmos de descubrimiento de servicios Web

En este núcleo temático se especifican las diferentes unidades de análisis relacionadas con algoritmos que permitan el descubrimiento de servicios Web.

2.2.1.1. Adaptive Matchmaking for RESTful Services based on hRESTS and MicroWSMO

(Lampe, Schulte et al. 2010) presenta una propuesta para el emparejamiento de servicios Web RESTful que adapta distintas metodologías de algoritmos utilizados en el emparejamiento de servicios Web SOAP tradicionales. Los autores se centran en la descripción de hRESTS, formato utilizado para la descripción sintáctica de los servicios Web REST y MicroWSMO, como extensión de hRESTS, para la inclusión de información semántica. El algoritmo de emparejamiento propuesto se centra en la premisa de que las operaciones proveen la funcionalidad esencial que un agente busca. De esta manera, los valores de similitud de todos los niveles de abstracción definidos (servicio, operación, entradas y salidas) son agregados al nivel de operación.

Además, una vez determinado el grado de correspondencia entre dos operaciones, se propone el cálculo de un equivalente numérico correspondiente utilizando un estimador OLS (*Ordinary Least Square*).

El algoritmo propuesto tiene en cuenta algunas características específicas de los servicios Web RESTful, como la especificación explícita de un valor de similitud numérico entre los verbos HTTP (GET, POST, PUT, DELETE) de la petición y la oferta de servicios, permitiendo así determinar inicialmente el grado de correspondencia. En el trabajo se presenta un prototipo que implementa el algoritmo propuesto, el cual es evaluado utilizando una colección de prueba de servicios Web RESTful. Para la evaluación se utilizan algunas métricas del dominio de recuperación de información, con el fin de evaluar la calidad del algoritmo. El conjunto de servicios (originalmente en formato SAWSDL) es transformado en servicios con formato hRESTS y anotaciones semánticas tipo MicroWSMO utilizando un documento XSLT.

El principal aporte de (Lampe, Schulte et al. 2010) en este trabajo de grado es la descripción del proceso de emparejamiento que se realiza entre dos servicios Web con el fin de determinar su grado de correspondencia y además, se consideran algunos aspectos importantes de los servicios Web REST para tener en cuenta en el sistema de descubrimiento. Conjuntamente, se consideran aspectos semánticos al utilizar ontologías de dominio para la descripción semántica de los servicios Web REST a través de anotaciones tipo MicroWSMO sobre hRESTS.

Por otro lado, existen algunas brechas en el trabajo se enumeran a continuación:

- No se consideran algunos aspectos característicos de los servicios Web RESTful, como la posibilidad de poder filtrar por el tipo de contenido las respuestas a las consultas realizadas en el sistema. Esto se debe a que hRESTS no considera este aspecto en la descripción que hacen de los servicios.
- No se tiene en cuenta el contexto de entrega. En el trabajo no se consideran aspectos de movilidad, o la conexión; aspectos que pueden ser utilizados para filtrar las ofertas de servicio con el fin de acercarse más a los requerimientos del usuario.

2.2.1.2. Using DNS for REST Web Service Discovery

(Algermissen 2010) Destaca la popularidad que han ganado los Servicios Web RESTful y matiza la importancia y la necesidad de contar con un mecanismo de descubrimiento. En este contexto, propone como solución el uso de la tecnología DNS (*Domain Name System*), manifestada en la especificación DNS-SD (*DNS-Service Discovery*). Presenta las ventajas de adoptar DNS-SD para el descubrimiento de servicios, haciendo énfasis en el amplio soporte con que cuenta y en las características de ubicuidad. Además, reitera la importancia que tiene el descubrimiento de servicios en SOA (*Service Oriented Architecture*) y la facilidad con que DNS-SD permite realizar este proceso en los Servicios Web RESTful.

El principal aporte de (Algermissen 2010) para el desarrollo de esta propuesta de trabajo de grado es el ejemplo práctico de implementación con el que se demuestra que el descubrimiento de servicios puede ser efectuado en sistemas RESTful a través

de la especificación DNS-SD. Igualmente, el aporte conceptual del documento es de gran relevancia.

Por otro lado, las brechas identificadas de este trabajo son:

- El descubrimiento de Servicios Web se realiza de forma sintáctica y no semántica.
- No presenta un algoritmo para realizar el descubrimiento de servicios de forma semántica.

2.2.1.3. Algorithm for Web Service Discovery Based on Information Retrieval Using WordNet and Linear Discriminant Functions

(Sotolongo, Kobashikawa et al. 2007) Presenta un algoritmo de búsqueda para Servicios Web SOAP que hace uso de la base de datos léxica WordNet para agregar semántica, obteniendo un conjunto de sinónimos de las palabras de la petición original y buscando servicios a partir de éstas. Emplea una función discriminante lineal para determinar la importancia de las búsquedas, tanto con la petición original como con las peticiones generadas a partir del uso de WordNet.

Con el uso del algoritmo propuesto se realizan las medidas de fiabilidad (*precision*), porcentaje de los servicios relevantes recuperados (*recall*) y calidad del emparejamiento (*overall*), los cuales al ser comparados con las medidas obtenidas con cuatro algoritmos basados en sintáctica, presentan un mejor comportamiento en un escenario de prueba compuesto por 48 servicios reales.

El principal aporte de (Sotolongo, Kobashikawa et al. 2007) para el desarrollo de esta propuesta de grado es la ilustración del algoritmo de búsqueda para Servicios Web SOAP, ya que ésta se realiza de forma semántica mediante el uso de WordNet. El algoritmo podría ser usado en Servicios Web RESTful mediante una adaptación.

La brecha identificada en este trabajo es:

- No propone un algoritmo de búsqueda para Servicios Web RESTful, ya que el planteado trabaja con Servicios Web SOAP.

2.2.1.4. Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching

(Bellur and Kulkarni 2007) Presenta un algoritmo de emparejamiento para servicios Web semánticos, cuyo modelamiento e implementación se basa en grafos bipartitos. En el trabajo, los autores enfatizan la importancia del descubrimiento de servicios dentro de SOA, y presentan un análisis detallado del algoritmo presentado por Massimo Paloucci[6] y otros en (Paolucci, Kawamura et al. 2002), utilizado para el proceso de emparejamiento de servicios Web. Los autores identifican las fortalezas y debilidades del algoritmo presentado en (Paolucci, Kawamura et al. 2002), y plantean unas modificaciones al algoritmo original; principalmente proponen la inclusión de un modelo de las peticiones y las ofertas de servicios Web, y el proceso de emparejamiento como tal, a través de grafos bipartitos. Asimismo, comparan la complejidad del algoritmo propuesto por ellos con respecto al original y adicionalmente

[6] Más información disponible en: http://www.discovery.dist.unige.it/

se presentan comparaciones en términos de rendimiento de ambos algoritmos, encontrando resultados favorables.

El principal aporte de (Bellur and Kulkarni 2007) para el desarrollo de este trabajo de grado es el análisis y las modificaciones que se realizan al algoritmo de emparejamiento, lo que proporciona un punto de vista bastante claro para evaluar la posibilidad de utilizar este algoritmo como base para el desarrollo del mecanismo de descubrimiento de servicios Web REST. Además, se muestra de manera clara la forma en que los grafos bipartitos se utilizan para el proceso de emparejamiento.

Sin embargo, existen algunas brechas en el trabajo que se detallan a continuación:

- El algoritmo de emparejamiento propuesto no considera aspectos relacionados con los servicios Web REST. Es decir, no se considera ningún aspecto relacionado con las información presente en la descripción de los servicios para REST, ya que la colección de servicios utilizada para la evaluación experimental del algoritmo está basada en OWL-S.
- El algoritmo propuesto no considera aspectos relacionados con el contexto de entrega de los servicios (entorno móvil).

2.2.1.5. Análisis

Los algoritmos de descubrimiento estudiados en las unidades de análisis de este núcleo temático abordan el tema desde diversos puntos de vista. Las aproximaciones más sofisticadas para la ejecución del descubrimiento se basan en servicios Web SOAP, aprovechando el lenguaje utilizado para su descripción (WSDL). Teniendo en cuenta las necesidades originadas de esta investigación, reflejadas en la pregunta de investigación *(¿Cómo realizar el descubrimiento semántico de Servicios Web RESTful, en entornos móviles?)*, algunos autores solucionan con éxito en sus trabajos algunos componentes de estas necesidades, pero de forma aislada. Valga como ejemplo el trabajo realizado en (Sotolongo, Kobashikawa et al. 2007) en donde se implementa un mecanismo de recuperación semántico pero NO para servicios Web REST; o también el trabajo en (Lampe, Schulte et al. 2010) en donde no se permite filtrar los servicios por el tipo de respuesta e igualmente no se consideran aspectos de movilidad. Así, no se encontró un algoritmo que diera solución a todas las necesidades planteadas en los objetivos de este trabajo de grado.

2.2.2. Descripción de servicios Web RESTful

En este núcleo temático se puntualizan las distintas unidades de análisis relacionadas con el proceso de descripción de servicios Web REST.

2.2.2.1. Supporting the Creation of Semantic RESTful service Descriptions

(Maleshkova, Pedrinaci et al. 2009) Presenta la gran importancia que están tomando los Servicios Web RESTful en la actualidad y su rápido crecimiento gracias a redes sociales como Facebook, Twitter o empresas como Google, entre muchas otras. El trabajo que realizan los autores se basa en solucionar dos problemas identificados en los Servicios Web RESTful, que son: la falta de anotaciones que sirvan a las máquinas para poder comunicarse entre ellas y la carencia de semántica al momento de publicar los servicios. De esta forma, propone hacer uso de hRESTS y de microWSMO para hacer una descripción formal de los servicios de forma semántica.

hRESTS introduce una serie de etiquetas adicionales al documento HTML de un servicio, las cuales, son procesables por los dispositivos que las leen, ligeras, de fácil uso y suficientes para describir las propiedades claves del servicio. Una vez que se tienen estas etiquetas se puede utilizar microWSMO para agregar anotaciones, lo cual contribuye a que procesos como la composición y clasificación de servicios sea más eficiente.

Como aporte más significativo se presenta a SWEET (Semantic Web sErvices Editing Tool), una aplicación Web que hace uso de las funcionalidades de hRESTS y microWSMO para ayudar a la creación de descripciones semánticas para Servicios Web RESTful que presenta al usuario una serie de herramientas diseñadas para tal fin.

El principal aporte de (Maleshkova, Pedrinaci et al. 2009) para el desarrollo de esta propuesta de trabajo de grado es la posibilidad de contar con una herramienta como SWEET que facilita el enriquecimiento de los Servicios Web de una manera rápida y eficiente, permitiendo que los esfuerzos se centren en la adaptación e implementación del mecanismo de descubrimiento. El uso de microWSMO puede facilitar el trabajo en entornos móviles, donde los recursos son limitados ya que es un formato ligero.

Adicionalmente, las brechas identificadas en este trabajo son:

- Debido a que no hay una estandarización en la forma de describir semánticamente servicios REST, la aplicabilidad de SWEET se limita a un entorno de prueba.
- No se muestran los resultados que demuestren que mediante el uso de SWEET se mejora el proceso de descubrimiento de servicios.

2.2.2.2. EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies

(Alowisheq, Millard et al. 2009) Identifica como problema principal que una descripción sintáctica de los Servicios Web RESTful no es suficiente para generar un mecanismo de descubrimiento que sea automático o semiautomático, y que las entradas y salidas no son garantía para intuir qué tipo de servicio se está consumiendo. Los autores presentan un enfoque denominado EXPRESS (*EXPressing REstful Semantic Services Using Domain Ontologies*) que propone explotar los recursos ofrecidos por la Web semántica, y así tener un mínimo de diseño y desarrollo al momento de agregar notaciones semánticas a los Servicios Web RESTful.

Propone el uso de formatos y estándares como RDF (*Resource Description Framework*) y OWL (*Ontology Web Language*) para añadir semántica a las páginas HTML de un Servicio Web RESTful. Finalmente, hace una comparación de EXPRESS con SA-REST y WSMO lo cual permite verificar que el primero es comparable en eficiencia con sus contendores.

El principal aporte de (Alowisheq, Millard et al. 2009) para el desarrollo de esta propuesta de trabajo de grado es la ilustración mediante un ejemplo del uso de la metodología que se recomienda en EXPRESS, la cual podría ser adoptada para enriquecer Servicios Web en el desarrollo del proyecto.

Y por otro lado, la principal brecha identificada en este trabajo es:

- No presenta un mecanismo para descubrir Servicios Web RESTful con EXPRESS, solamente se propone un mecanismo para enriquecer semánticamente las descripciones de los servicios Web REST. La realización de descubrimiento y composición de servicios se declara como un trabajo futuro de la investigación.

2.2.2.3. Linking Data from RESTful Services

(Alarcon and Wilde 2010) Presenta una conceptualización bastante completa de la esencia y objetivos de la Web semántica, resaltando principalmente la importancia que tiene, según la tendencia actual, el extender los recursos legibles por humanos de la Web con información semántica codificada en una forma procesable por máquinas, de manera que pueda ser extraída con el fin de poder incrementar el grado de satisfacción del usuario de la Web.

Por otro lado, también se presenta una completa conceptualización de REST, enumerando todas sus características, objetivos y restricciones. Basados en estas consideraciones, los autores proponen un lenguaje para la descripción de servicios Web RESTful llamado ReLL, que registra en un archivo con formato XML los recursos, sus representaciones (que pueden ser más de una para cada recurso) y los links que pueden llevar a otro recurso relacionado.

Adicionalmente, se exhibe la manera en que se transforman las descripciones ReLL a RDF con el propósito de guardar las descripciones en un repositorio. Conjuntamente, se muestra la implementación de un "rastreador" o *crawler* para descripciones tipo ReLL (aplicadas en tres ejemplos específicos) que analiza sintácticamente la información de los archivos y produce unas tripletas RDF de los recursos y sus relaciones entre sí.

El principal aporte de (Alarcon and Wilde 2010) para el desarrollo de este trabajo de grado es la conceptualización que se hace de REST, en donde se enumeran los principios y conceptos trascendentales con el fin de generar un lenguaje de descripción adecuado para servicios Web RESTful. Este último ítem también es de suma importancia, ya que se propone un novedoso mecanismo para la descripción sintáctica de servicios Web REST.

Las brechas identificadas en el trabajo son:

- No presenta ningún algoritmo de descubrimiento. El trabajo de los autores se centra en identificar los principales conceptos de REST con el fin de definir un mecanismo apropiado de descripción para servicios Web RESTful.
- No define un mecanismo para adicionar semántica a los servicios descritos con el lenguaje de descripción propuesto, ya que este lenguaje es definitivamente sintáctico.

2.2.2.4. Análisis

Hasta este punto, a través del estudio de la literatura relacionada con el tema central de esta investigación, es claro que con el fin de soportar un mecanismo adecuado de recuperación de servicios REST, es necesario definir para éstos un componente de

descripción. En este sentido, existen diferentes aproximaciones que intentan dar solución al inconveniente de la definición de una descripción para servicios Web REST. Quizás el lenguaje que más se adapta a las restricciones definidas en REST es el presentado en (Alarcon and Wilde 2010), en donde se define un lenguaje para la descripción sintáctica de los recursos de los servicios; sin embargo, es un lenguaje relativamente nuevo, razón por la cual no existen servicios descritos con él para realizar pruebas. Este ítem es de gran relevancia para este trabajo de grado, ya que es necesario realizar una etapa de evaluación del algoritmo implementado y para ello se necesita contar un número apto de servicios Web descritos. Por consiguiente, es necesario considerar otras opciones. En el capítulo 3 se presentan las diferentes alternativas para la descripción de los servicios y en el capítulo 4 se presentan unos criterios definidos con el fin de seleccionar el más apropiado.

Por otro lado, todas las alternativas analizadas en este núcleo temático dan solución al problema de la descripción de los servicios Web REST de una manera sintáctica. No obstante, es posible utilizar un mecanismo adicional para complementar semánticamente el lenguaje de descripción seleccionado.

2.2.3. Semántica

En este núcleo temático se puntualizan las diversas unidades de análisis relacionadas con los mecanismos que buscan adicionar semántica al proceso de Descubrimiento de servicios.

2.2.3.1. Development of Retrieval Methods for RESTful Web Services using Semantic Technologies

(Seung-Jun Cha 2010) Resalta la importancia de la Web 2.0 y de los servicios Web RESTful como parte activa de la misma, ya que se han difundido rápidamente (lo que se ve reflejado en el incremento del número de servicios disponibles en la Web). Igualmente, destaca el problema de encontrar servicios específicos que cumplan con los requerimientos del usuario manejando simplemente recuperación de servicios basado en palabras claves. Así, propone el uso de tecnologías semánticas basadas en servicios Web RESTful.

Específicamente los autores proponen un método de recuperación de servicios Web utilizando dos tecnologías: SAWSDL para embeber información semántica en las páginas de descripción de cada servicio y RDFa para transformar las descripciones en formato HTML de cada servicio de “legibles por humanos” a “legibles por máquinas” mediante la adición de un conjunto de extensiones, a nivel de atributos, a las páginas XHTML. De esta manera se extrae la información en RDF de todas las descripciones de los servicios y se guardan en un repositorio; el usuario del sistema realiza la consulta mediante palabras claves que son extendidas a través del uso de una ontología de dominio de prueba o a través de WordNet para aquellas que no tienen un concepto semántico asociado en la ontología.

El principal aporte de (Seung-Jun Cha 2010) para este trabajo de grado es el ejemplo de aplicación de ontologías de dominio y de propósito general para la implementación del sistema de recuperación de servicios Web RESTful.

Las brechas identificadas en este trabajo son:

- Los servicios Web descritos son guardados en una base de datos relacional definida arbitrariamente por los autores, lo que hace difícil tener acceso a ella y además, hace al sistema poco escalable.
- La ontología de dominio utilizada es una ontología de prueba escrita por los autores, cuyo vocabulario es bastante reducido. Así, muchos servicios Web serían descartados si se utiliza ésta.

2.2.3.2. SA-REST: Adding semantics to REST-based Web Services

(Lathem 2007) Presenta a los Mashups como la esencia de las aplicaciones Web 2.0, ya que surgen como un nuevo servicio creado a partir de 2 o más preexistentes. Los Mashups basan su éxito y popularidad en la innovación de los Servicios Web RESTful y en la evolución de los lenguajes de programación, principalmente los que siguen las técnicas AJAX.

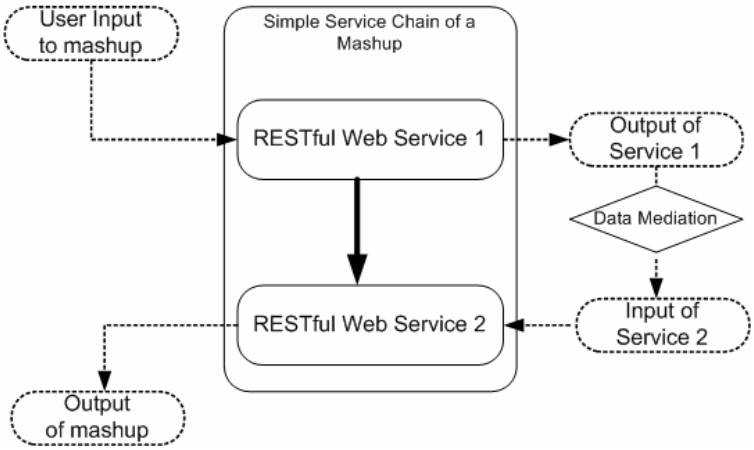


Figura 6. Arquitectura típica de un Mashup que usa dos servicios

Señala que el lugar apropiado para agregar anotaciones semánticas es el archivo de descripción WSDL de un servicio, pero ante la ausencia de estos archivos en el contexto RESTful, se puntualiza la importancia que tienen los microformatos como OWL-S (*Semantic OWL*), WSMO y WSDL-S (*Semantic WSDL*) ya que permiten agregar semántica a las páginas HTML de los servicios. Finalmente, recomienda el uso de RDF ya que está estandarizado por la W3C (*World Wide Web Consortium*) y muestra un ejemplo de un documento enriquecido de esta forma.

Por otro lado, presenta un estudio de las herramientas de creación de Mashups que existen actualmente e ilustra la manera en que estos interactúan (ver Ilustración 1). Se destaca que el componente de mediación de datos es el automatizado. También, introduce el concepto de SMashup (*Semantic Mashup*) que usa Servicios Web RESTful enriquecidos con anotaciones SA-REST y muestra dos ejemplos que se implementaron utilizando herramientas como Craig'sList y Google Map Mashup.

El principal aporte de (Lathem 2007) para el desarrollo de esta propuesta de trabajo de grado es el nivel de detalle con el que se muestra el enriquecimiento de servicios con SA-REST para posteriormente facilitar el descubrimiento y composición de Servicios RESTful.

Las brechas identificadas en este trabajo son:

- No presenta la forma en que se realizaría el descubrimiento para los Servicios Web RESTful ya que el documento se centra en la creación de Mashups a partir de los Servicios Web RESTful enriquecidos con SA-REST.
- No especifica ningún algoritmo de descubrimiento de servicios.

2.2.3.3. Semantic Web Service Automation with Lightweight Annotations

(Vitvar and Fensel 2009) Presenta un enfoque ligero para la descripción semántica de servicios Web SOAP y RESTful, el cual se fundamenta en una ontología simple que modela los Servicios Web y su semántica (WSMO-Lite). Muestra dos mecanismos de anotación que implementan esta ontología, uno para descripciones de servicios RESTful (MicroWSMO) y otro para descripciones tradicionales WSDL (SAWSDL).

Define cuatro tipos de semántica de servicios que pueden ser utilizados para describir Servicios Web:

- ***Information model*:** define la semántica de las entradas, salidas y los mensajes de falla del servicio.
- ***Functional semantics*:** define la funcionalidad del servicio.
- ***Nonfunctional semantics*:** define los detalles específicos al entorno de ejecución o implementación del servicio.
- ***Behavioral semantics*:** especifica el protocolo que el cliente necesita seguir para interactuar con el servicio.

Los autores presentan también ejemplos de documentos de descripción tipo WSDL con anotaciones WSMO-Lite para servicios Web SOAP, y tipo hRESTS, el formato utilizado por MicroWSMO para agregar semántica a servicios Web RESTful. Conjuntamente, para lograr el objetivo de automatización de servicios, se definen de forma genérica algoritmos para cada tarea que debe llevarse a cabo (descubrimiento, oferta, filtrado, clasificación, selección e invocación de servicios), en lo que se denomina Entorno de Ejecución Semántico (*Semantic Execution Environment SEE*).

El principal aporte de (Vitvar and Fensel 2009) para el desarrollo de esta propuesta de trabajo de grado es la claridad con la que definen conceptos claves relacionados con el descubrimiento de servicios y las anotaciones semánticas de los mismos. Igualmente, los algoritmos genéricos que se especifican y que facilitan el proceso de descubrimiento, proporcionan una referencia de gran relevancia para alcanzar el objetivo general de este proyecto.

Por otro lado, la principal brecha identificada en este trabajo es:

- Se presenta sólo la definición genérica del algoritmo de descubrimiento más no su implementación.

2.2.3.4. Análisis

Los trabajos relacionados con semántica analizados en este núcleo temático se abordan desde diferentes puntos de vista. Para adicionar semántica a las descripciones de servicios Web se utilizan ontologías de dominio para extender las consultas realizadas por los clientes, concretamente en el trabajo de (Seung-Jun Cha

2010) y también como un vocabulario común para relacionar partes de la descripción de los servicios con conceptos en una ontología. Asimismo, la semántica también se aplica a través del uso de herramientas como bases de datos léxicas, por ejemplo WordNet, lo que permite generar una consulta enriquecida que mejora los resultados obtenidos a través de mecanismos de recuperación basados simplemente en palabras claves.

Adicionalmente, se identificaron mecanismos para la adición de anotaciones semánticas a descripciones de servicios REST ya existentes, descritas en trabajos como (Vitvar and Fensel 2009; Gomadam, Ranabahu et al. 2010), que permiten apuntar a conceptos semánticos definidos en documentos especializados.

En síntesis, existen diversas alternativas para adicionar semántica a descripciones sintácticas de servicios REST. En los capítulos posteriores se determinarán las alternativas escogidas con su respectiva justificación.

2.2.4. Entorno móvil

En este núcleo temático se especifican las unidades de análisis más relevantes relacionadas con los servicios Web en entornos móviles.

2.2.4.1. Desarrollo Web Orientado a dispositivos móviles

(Herrera de la Cruz 2005) Presenta la razón por la cual, actualmente, los servicios desarrollados para dispositivos móviles está creciendo rápidamente y los usuarios se están conectando cada vez más a Internet desde sus dispositivos, son presentadas razones técnicas y el impulso que le ha dado los precios bajos ofrecidos por los operadores.

Se presenta la evolución de las velocidades de conexión de los dispositivos móviles, ya sea con la empresa de telecomunicaciones que le presta el servicio o a través de WiFi.

También se aborda de forma resumida la manera en cómo se ha venido consumiendo los servicios Web desde un dispositivo móvil, para hacer ver la necesidad del desarrollo independiente de servicios especialmente diseñados para un contexto móvil. Pero para esto es importante determinar con exactitud el tipo de dispositivo que está accediendo a una página o que está solicitando un servicio, así que se presentan dos de las alternativas más usadas para esto: UAprof y WURFL.

UAPorf es una especificación definida por la OpenMobile Allianze que recoge las características de los dispositivos inalámbricos, aunque esta especificación aún es vigente y puede ser usada, se presenta la razón por la cual ya no lo es de forma masiva, debido principalmente al mal uso que le han dado los fabricantes de dispositivos, ya que no se ha respetado las convenciones sobre el uso de términos, atributos o funcionalidades, lo que eventualmente produciría mas trabajo para los desarrolladores añadiendo excepciones a sus códigos para trabajar con todos los dispositivos.

Finalmente se presenta a WURFL como un repositorio de dispositivos que nació a partir de los problemas que se presentó con UAProf, y que presenta una serie de ventajas respecto a su “competidor” entre las cuales se destaca que es un fichero XML

que es alimentado por los fabricantes y por los desarrolladores, razón por la cual se puede conseguir información actualizada. Otra de las ventajas es que la comunidad open-source a desarrollado API en diferentes lenguajes, o por lo menos, los más usados como Java, PHP, .net y Ruby, entre otros. Además muestra el uso de la API java y de un framework basado en WURFL, esto no fue considerado ya que se uso la guía oficial de la API para su implementación y el framework no es de interés para este proyecto.

El aporte de (Herrera de la Cruz 2005) para este trabajo de grado es la ilustración del uso de la API de WURLF en el lenguaje de programación Java, que es el seleccionado para la implementación del prototipo de este proyecto.

La principal brecha identificada en este trabajo es:

- No se presenta una comparación técnica del funcionamiento de WURFL con respecto a UAProf.
- El ejemplo mostrado no utiliza la última versión de WURFL.

2.2.4.2. Performance Evaluation of RESTful Web Services for Mobile devices

La investigación del trabajo que resume este artículo (Hamad, Saad et al. 2009) se fundamenta en el hecho de que los servicios están siendo consumidos desde dispositivos móviles inalámbricos con mayor frecuencia, y considerando las limitaciones del contexto móvil (restricciones en las dimensiones de la pantalla, bajas velocidades de procesamiento, bajas capacidades de almacenamiento, duración de las baterías entre otros aspectos) los autores ven importante evaluar el desempeño que tiene tanto el consumo de servicios SOAP como servicios RESTful, para determinar cuál es más aconsejable consumir considerando las limitaciones ya mencionadas.

Para llevar a cabo sus pruebas realizaron una implementación que consistió en un servicio RESTful y su contraparte en SOAP, y los alojaron en servidores *Glassfish*. Los terminales móviles fueron emulados con *Sun Mobile Emulator* configurándoles una velocidad de procesamiento de 512 bytecodes/millisecond y una velocidad de conexión de 9600 bps. Los servicios desarrollados en Java consisten en un programa que concatena cadenas de texto y otro en el cual se suma números. Las pruebas realizadas consisten en enviar desde dos hasta ocho cadenas de texto y números, recuperando el tamaño de los mensajes transmitidos para hacer estas operaciones y el tiempo que se demoran en procesarlas (ver Figura 3 y Figura 4), y a partir de éstas se determina que el uso de servicios Web RESTful en entornos móviles otorga ventajas frente a los SOAP, principalmente en los tiempos de procesamiento y tamaño de los mensajes. Esta conclusión se puede emplear como un motivo más para justificar el uso de servicios Web RESTful en este proyecto.

El aporte de (Hamad, Saad et al. 2009) para el desarrollo de este trabajo de grado es la fundamentación para el uso de servicios RESTful sobre SOAP en entornos móviles si sólo consideramos las restricciones del dispositivo.

La brecha de este trabajo es:

- No se realiza un descubrimiento de los servicios, ya que la ubicación de estos es plenamente conocida, por lo que los tiempos de respuesta no consideran este proceso.

2.2.4.3. Evolution and Usability of Mobile Phone Interaction Styles

(Kiljander 2004) Es un documento muy completo que tiene por objetivo determinar los pasos que deben tomar los fabricantes de dispositivos pensando en los nuevos retos tecnológicos y en la usabilidad de sus productos. El documento inicia con un estudio sobre los hábitos de consumo de las personas en cuanto a dispositivos móviles y como este es afectado en un buen porcentaje por interfaces agradables y un dispositivo usable, además de las principales razones por las que un usuario compra un dispositivo móvil (estadística que es tomada de un estudio de Nokia del año 2003 y que se muestra en la Figura 7) y una figura que muestra el número de dispositivos vendidos según sus características y tecnología (Figura 8).

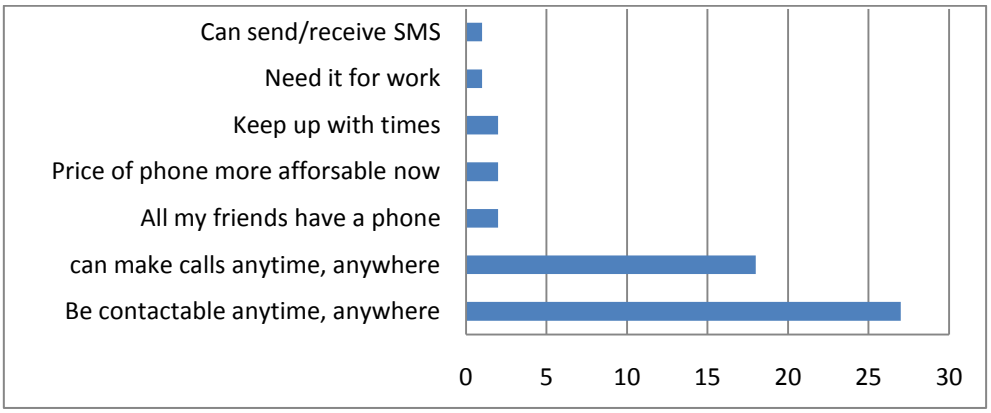


Figura 7. Razones principales por las que un usuario adquiere un celular por primera vez

También resalta la importancia del contexto y lo posiciona como una de las diferencias fundamentales con un entorno de escritorio y por la cual los servicios para estos dispositivos pueden ser más atractivos (aunque hace especial énfasis en los servicios ofrecidos por los operadores).

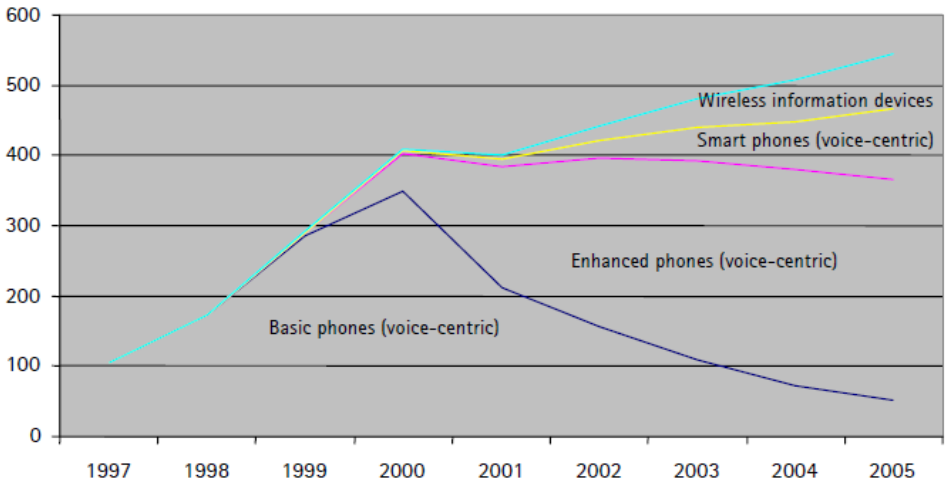


Figura 8. Terminales móviles vendidos por año

Del documento se destaca que a medida que la tecnología avanza se pueden hacer servicios más completos, siempre y cuando no se aumente demasiado la complejidad al momento de usarlos.

Finalmente se hace una comparación entre los dispositivos de los principales fabricantes, lo cual no es de mucho interés para esta investigación.

El principal aporte de (Kiljander 2004) para este trabajo de grado son los estudios realizados por el autor en la industria móvil en los que se resalta la masificación de la industria celular en los últimos años, lo que da origen a una oportunidad para brindarle al usuario servicios que le solucionen algunas necesidades.

La principal brecha identificada en este trabajo es:

- No se habla de los servicios Web aplicados al entorno móvil, lo cuál es un aspecto fundamental en esta investigación.

2.2.4.4. A System Architecture for Context-Aware Service Discovery

(Doulkeridis, Loutas et al. 2006) Considerando que debido a los avances tecnológicos cada día se ofrecen más servicios para los dispositivos móviles, y que este entorno presenta restricciones para que puedan ser descubiertos o consumidos eficientemente, los autores proponen una arquitectura que tenga en cuenta estas restricciones para que solo los servicios relevantes sean tenidos en cuenta a la hora de ser presentados como el resultado de una búsqueda, entendiéndose como relevantes los servicios que cumplen con los requerimientos de un usuario o agente y que además no representen una carga exagerada para el dispositivo que solicita el servicio. Esta arquitectura representa una mejora en la calidad de la búsqueda pero un mayor procesamiento; por lo tanto es evaluada la relación costo / carga.

El documento provee una definición de contexto como la información implícita relacionada tanto con el usuario solicitante (agente o usuario final) como con el proveedor, que pueden afectar la utilidad de los servicios recuperados. Se evidencia que es importante tener un mecanismo para recuperar información sobre el dispositivo que solicita los servicios (*user-context*) y que además se requiere que los servicios ofrezcan información sobre cómo pueden ser consumidos más eficientemente, sobre que características mínimas debe cumplir un dispositivo móvil para que este pueda consumir el servicio sin restricciones o complicaciones (*service-context*).

La arquitectura que se propone se muestra en la Figura 9 y se pueden sacar las siguientes conclusiones sobre esta:

- Es importante tener un repositorio de perfiles de dispositivos, aunque en el documento no se hace recomendación alguna sobre cual usar.
- La arquitectura puede usar una gran variedad de repositorio de servicios, por ejemplo UDDI o ebXML.
- Aunque la arquitectura evidencia que se necesita un repositorio de perfiles de usuario, que se alimenta con las preferencias de los usuarios, en nuestro caso este modulo es despreciado ya que no es de interés la personalización de servicios.

Los autores del documento realizan una serie de pruebas que les sirve para concluir que su arquitectura reduce significativamente servicios que no podrían ser consumidos fácilmente por un dispositivo específico. Rescatamos de este documento esta conclusión para argumentar que el descubrimiento tenga en cuenta el contexto.

La principal brecha es que no se presenta una comparación que indique en cuánto se incrementan los tiempos de procesamiento para recuperar los servicios, haciendo uso de un mecanismo que recupere la información de contexto.

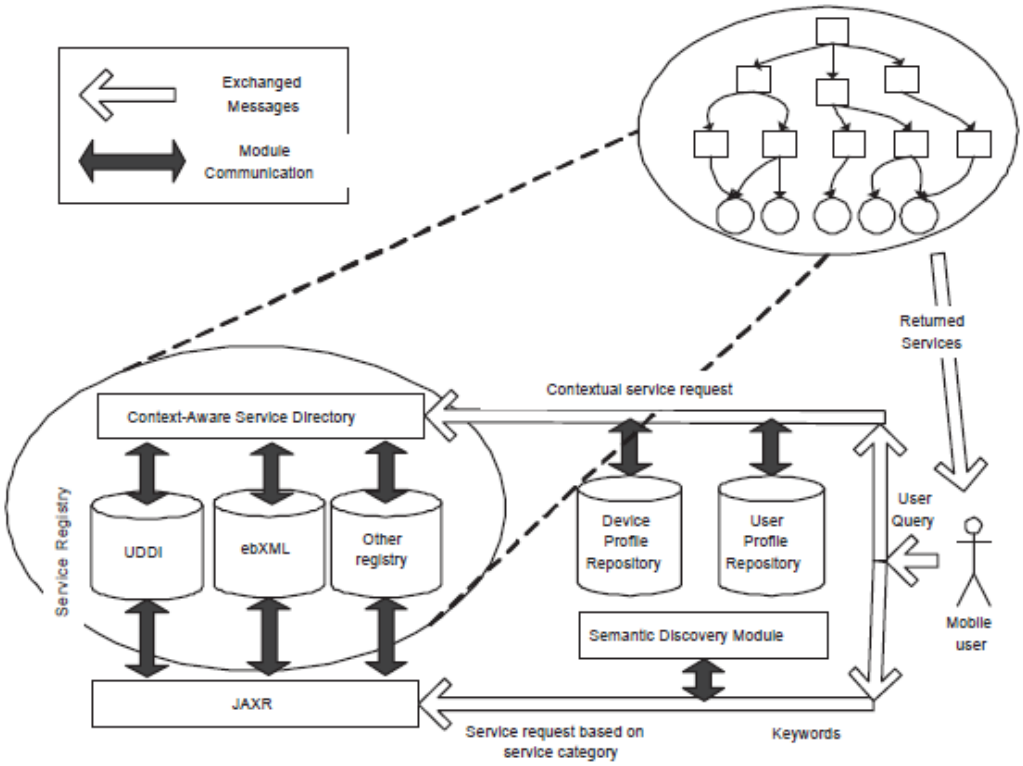


Figura 9. Arquitectura para el descubrimiento de servicios que tiene en cuenta el contexto

2.2.4.5. Análisis

Los trabajos relacionados con este núcleo temático definido en esta investigación arrojan para nosotros un resultado bastante diciente: los servicios Web REST son más apropiados que los servicios Web SOAP para ser consumidos desde dispositivos móviles. Esta afirmación se aborda en el trabajo de (Hamad, Saad et al. 2009).

Adicionalmente, se estudian alternativas para permitir identificar los servicios Web adecuados para ser consumidos desde el dispositivo móvil según sus características específicas; esto permite filtrar la oferta de servicios Web según los requerimientos del usuario.

Además, se demuestra que la industria móvil está en crecimiento y que constituye una oportunidad para poder prestar servicios Web que puedan ser consumidos desde dispositivos móviles y que puedan solucionar algunas necesidades comunes de los usuarios.

2.3. Resumen

En este capítulo se presentó el estudio de la literatura relacionada con el tema central abordado en este trabajo de grado. La temática se dividió en dos etapas: en la primera se presentaron los conceptos generales relacionados con el descubrimiento semántico de servicios Web REST. Concretamente, se identificaron los enfoques para realizar descubrimiento de servicios, se definió el concepto de Servicio Web, se definieron las características de REST (identificando sus ventajas y desventajas) y además se identificaron aspectos relevantes al entorno móvil, primordialmente las ventajas y particularidades que ofrece.

En una segunda etapa, se detallaron los principales trabajos relacionados con el tema central de investigación abordado y que conforman la base de conocimiento construida en este trabajo de grado. Estos trabajos o unidades de análisis se agruparon en cuatro núcleos temáticos y se realizó un análisis sintético por cada uno de ellos.

En síntesis, gracias a la base de conocimiento construida, se lograron identificar las tecnologías utilizadas para la descripción de servicios Web REST, para ejecutar el proceso de descubrimiento en REST, asimismo se identificaron diferentes enfoques que permiten adicionar semántica a las descripciones sintácticas REST y adicionalmente se identificaron las características que ofrece el entorno móvil, permitiendo establecer que gracias (entre otras cosas) a su simplicidad, los servicios REST son más idóneos para ser consumidos desde dispositivos móviles que los servicios Web SOAP. Una vez especificadas las tecnologías relacionadas con los distintos componentes identificados en el planteamiento del problema del marco de este proyecto, se tienen las herramientas necesarias para seleccionar el método idóneo para la implementación del mecanismo de descubrimiento de servicios.

Capítulo 3

Especificaciones y Tecnologías Asociadas

En este capítulo se presentan las diferentes tecnologías asociadas a la solución planteada para permitir el descubrimiento de servicios REST. Todas las tecnologías aquí mencionadas hacen parte del proceso llevado a cabo para lograr los objetivos propuestos en este trabajo de grado.

3.1. Descripción sintáctica de Servicios Web RESTful

Los Servicios Web RESTful deben su éxito en la industria gracias a las bondades que ofrecen sus propiedades, entre las cuales se destaca su alta escalabilidad, como resultado de un diseño de acoplamiento flexible. Este acoplamiento flexible o "*loose coupling"* tiene una evocación positiva ya que significa que el Servicio Web asume sólo un pequeño conjunto de supuestos y de esta manera se minimiza el impacto de cambio, permitiendo que el Servicio evolucione de manera independiente y haciendo que los sistemas orientados a servicios se desarrollen de manera fácil y económica (Pautasso and Wilde 2009). Adicionalmente, los Servicios Web RESTful se caracterizan por su facilidad para el despliegue, ya que se construyen sobre la infraestructura Web y estándares asociados (Alarcón and Wilde 2010).

El número de servicios Web RESTful que están disponibles en la Web ha aumentado considerablemente en gran parte gracias al fenómeno de la Web 2.0, en donde tanto proveedores de servicios y compañías especializadas como usuarios finales contribuyen de igual manera creando contenidos y aplicaciones en línea. Sin embargo, no existe un lenguaje de descripción estándar para este tipo de servicios, ni tampoco poseen un archivo de descripción pública de su funcionalidad que permita a los clientes o consumidores tener una manera automatizada de consumirlos. De esta manera, la mayoría de los servicios Web RESTful cuentan generalmente con un archivo de texto en formato HTML en donde se *describe en lenguaje natural y no entendible por máquinas* todos los detalles del mismo. En este sentido, existen algunas iniciativas que intentan dar solución a este inconveniente, las cuales se mencionarán en los siguientes párrafos.

3.1.1. WSDL 2.0

Es una especificación para describir interfaces de aplicaciones Web basadas en SOAP, pero que también tiene soporte para aplicaciones que no utilicen este protocolo (Toma, Steinmetz et al. 2008). WSDL 2.0 provee un modelo y un formato en XML para describir servicios Web. Además, permite separar la descripción de la funcionalidad abstracta ofrecida por un servicio. De esta manera, describe un servicio Web en dos etapas: una abstracta y otra concreta. En un nivel abstracto, WSDL 2.0 describe un servicio Web en términos de los mensajes que éste envía y recibe; estos mensajes son descritos independientemente del formato específico utilizando un sistema de tipos, comúnmente un esquema XML (Chinnici, Gudgin et al. 2003). Por otro lado, en un nivel concreto, se especifican detalles de transporte y otros relacionados con una o más interfaces (Lin 1998).

Las descripciones WSDL 2.0 tienen dos partes: la descripción del servicio y la descripción de interfaces. Una descripción de servicio WSDL 2.0 especifica cómo potenciales clientes podrían interactuar con el servicio descrito y representa la afirmación de que el servicio implementa y se ajusta completamente a lo que el documento WSDL 2.0 describe. Por otro lado, una interfaz WSDL 2.0 describe las potenciales interacciones con un servicio Web. Las interacciones descritas en la interfaz no son obligatorias; es decir, la descripción de una operación dentro de una interfaz WSDL 2.0 no implica que la operación deba ocurrir, sino que, si de alguna manera la interacción es iniciada, entonces la operación declarada describe cómo esa interacción debe ocurrir (Lin 1998).

3.1.2. WADL

Es un lenguaje XML diseñado con el fin de proveer un formato de descripción de protocolo procesable por máquinas para aplicaciones Web basadas en HTTP, especialmente las que usan XML para su comunicación (Toma, Steinmetz et al. 2008). WADL define una serie de elementos que describen todos los detalles del servicio, de manera análoga a los servicios Web en el contexto SOAP. A través de estos elementos se especifican los recursos, identificados por una URI, las entradas y salidas de los métodos HTTP asociados a cada recurso, las representaciones de los estados de los recursos, entre otros detalles (Toma, Steinmetz et al. 2008). WADL surge junto a WSDL 2.0 como un medio legible por máquina para describir formalmente una interfaz para servicios o aplicaciones Web que sigan el estilo REST (Toshiro Takase 2008). En (Toma, Steinmetz et al. 2008) se enumeran los elementos que están presentes en un documento WADL, los cuales son:

- ***application*:** es un elemento de nivel superior que contiene la descripción total del servicio. Puede contener elementos como: *grammars*, *resources*, *method*, *representation* y *fault*.
- ***grammars*:** actúa como un contenedor para las diferentes estructuras XML intercambiadas durante la ejecución del protocolo descrito por el documento WADL. Las estructuras se incluyen utilizando la etiqueta o elemento *include*.
- ***resources*:** actúa como contenedor de los recursos de la aplicación. Cada recurso se describe a través de la etiqueta *resource*, que a su vez puede contener subelementos como *method* o *resource*.
- ***method*:** describe la entrada y salida de un método HTTP que puede ser aplicado a un recurso. Puede tener dos subelementos:
  - *request*: describe la entrada a ser incluida cuando se aplica un método HTTP a un recurso.
  - *response*: describe la salida que resulta tras aplicar un método HTTP a un recurso.
- ***representation:*** describe la representación de un recurso y puede ser declarado a nivel global como un subelemento de *application* o a nivel local como subelemento de *request* o *response*, o también puede ser referenciado externamente.
- ***fault***: similar a un elemento de tipo *representation*, con la diferencia de que *fault* referencia una condición de error.

3.1.3. ReLL (Resource Linking Language)

Es un lenguaje sintáctico para la descripción de servicios Web RESTful introducido por los doctores Erik Wilde[7] y Rosa Alarcon[8] y otros, en trabajos como (Alarcon and Wilde 2010; Alarcón and Wilde 2010) . En (Alarcon and Wilde 2010) se definen las principales características que debería tener un lenguaje para la descripción de servicios Web RESTful, entre ellas encontramos:

- Se debe evitar el acoplamiento entre el servidor y el cliente
- Los recursos deben soportar múltiples representaciones
- Se debe considerar las relaciones entre recursos como una propiedad fundamental
- La descripción de los recursos debe ser parcial de manera que pueda ser completada o modificada, con el fin de conseguir escalabilidad
- Debe ser lo más simple posible para que pueda ser adoptado ampliamente por los desarrolladores de servicios Web REST

ReLL modela los servicios como un conjunto de recursos y la relación existente entre ellos. Adicionalmente considera todas las anteriores características, incluidas las principales restricciones para el diseño de servicios RESTful (identificación de recursos, relación entre los mismos, e interfaz uniforme para acceder a los recursos relacionados). En la Figura 10. Esquema de descripción de ReLL se muestra un esquema del modelo para la descripción de servicios de ReLL.

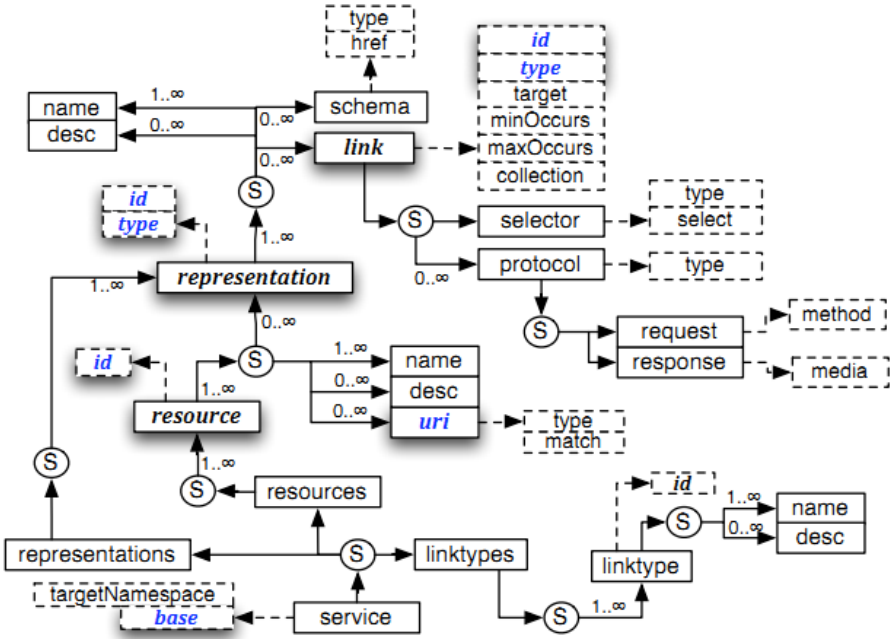


Figura 10. Esquema de descripción de ReLL

En la Figura 10 los rectángulos representan elementos y los rectángulos con línea punteada representan los atributos de estos elementos. Los círculos con la letra "S" representan secuencias. Como se muestra en la Figura 10, un servicio (*service*)

[7] Más información disponible en: http://dret.net/netdret/
[8] Más información disponible en: http://dcc.puc.cl/gente/usuarios/ralarcon

expone un conjunto de recursos (*resource*) que tienen un identificador único, un nombre, una descripción y un patrón URI (opcional) para describir los posibles parámetros utilizados para invocar al recurso. Asimismo, un recurso puede tener una o varias representaciones (*representations*) que pueden ser asociadas a esquemas con el fin de ofrecer la posibilidad de validación. Cada representación puede contener cualquier número de *links*. Un *link* es recuperado utilizando selectores (*selectors*) que dependen del formato de la representación, por lo tanto se deben interpretar de manera diferente y en diferentes lenguajes según el lenguaje de representación que retorne el recurso. De esta manera, para representaciones en formato XML, el ejemplo más claro de un selector es el lenguaje XPath (*XML Path Language*). Por último, un *link* define una posible asociación desde la representación del recurso que contiene el link hacia otro recurso a través del atributo *target*, que contiene el *id* único del otro recurso al que se desea referenciar y además pueden contener *protocol* que por cada link especifica las reglas para la interacción con el recurso (Alarcon and Wilde 2010).

3.1.4. hRESTS (HTML Microformat for Describing RESTful Web Services)

Es un microformato[9] cuyo propósito es proporcionar representaciones *"legibles por máquina"*[10] para las descripciones comunes de los servicios Web REST, usualmente disponibles en lenguaje natural en formato HTML. Los microformatos aprovechan las facilidades de las páginas HTML o XHTML como los atributos *class* y *rel* para marcar los fragmentos de interés en las páginas Web. Igualmente, traducen la jerarquía de elementos HTML a una jerarquía de objetos y sus propiedades. Surgió debido a la necesidad de contar con descripciones de servicios que sean procesables por máquinas. Los autores notaron que los servicios Web RESTful usualmente eran descritos en páginas Web HTML en donde especificaban todos los detalles del servicio Web: la URI a la cual se debía hacer la petición, el método HTTP que debía utilizarse, los parámetros de entrada, los tipos de respuesta, etc.; pero estas descripciones carecían de información legible por máquinas. hRESTS está soportado en un modelo que representa los servicios Web como una serie de operaciones con entradas y salidas, cuyo orden de ejecución es determinado por la hipermedia presente en cada respuesta del servidor al cliente (Kopecký, Gomadam et al. 2008).

En la Figura 11 se aprecia el modelo de servicio de hRESTS, en donde se definen todas las clases que se pueden referenciar (como anotaciones legibles por máquina) para la descripción de servicios Web REST. hRESTS establece que un servicio tiene una o varias operaciones, y éstas tienen entradas y salidas; además se definen propiedades de las operaciones como la dirección y el método utilizado para invocar la

[9] Los microformatos son una serie de formatos simples y abiertos, construidos sobre estándares existentes y ampliamente adoptados, cuyo objetivo es resolver problemas simples, adaptándose a los comportamientos y patrones de uso actuales. Otra definición especifica que son pequeños patrones de HTML para representar cosas comúnmente publicadas en las páginas Web como personas, eventos, entradas de blogs, comentarios y etiquetas. Algunos ejemplos de microformatos son: RSS, GeoURL (para localización), hCalendar (para eventos de calendario), entre muchos otros. Toda la información al respecto está disponible en: http://microformats.org/

[10] Las descripciones de la mayoría de los servicios Web RESTful usualmente están escritas en lenguaje natural y en formato HTML o XHTML, lo que hace que tareas como la composición y el descubrimiento de estos servicios sean totalmente manuales. Para proporcionar cierto nivel de automatización a estos procesos es necesario agregar ciertas anotaciones que son transparentes al usuario final y que permiten que esa descripción escrita en lenguaje natural pueda también ser utilizada por las máquinas para realizar tareas específicas

operación, la cuál es una característica de los servicios REST (utilizar una interfaz uniforme para la interacción de los servicios Web a través de los métodos HTTP).

```
@prefix hr:   <http://www.wsmo.org/ns/hrests#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix wsl:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

# classes and properties of the WSMO-Lite minimal service model
wsl:Service a rdfs:Class .
wsl:hasOperation a rdf:Property ;
   rdfs:domain wsl:Service ;
   rdfs:range wsl:Operation .
wsl:Operation a rdfs:Class .
wsl:hasInputMessage a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:hasOutputMessage a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:Message a rdfs:Class .

# hRESTS properties added to the above model
hr:hasAddress a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range hr:URITemplate .
hr:hasMethod a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range xsd:string .

# a datatype for URI templates
hr:URITemplate a rdfs:Datatype .
```

Figura 11. Modelo de servicio de hRESTS

Los elementos principales de hRESTS que se pueden apreciar en la **Figura 12** se especifican a continuación:

- ***service:*** indica que el elemento, en este caso la etiqueta *div* es parte del microformato hRESTS. Puede contener una o más operaciones y tener un título (*label*).
- ***operation:*** indica que el elemento, en este caso otra etiqueta *div*, contiene una descripción de una operación del servicio Web. Especifica la dirección y el método usado por la operación, y puede contener una descripción de las entradas y salidas de la misma. Puede ser usado también en hyperlinks (etiqueta <*a href*>) y en formularios (etiqueta <*form*>).
- ***address:*** especifica la URI de la operación. Puede ser usado en etiquetas *span* o en un link (<*a href*>).
- ***method:*** especifica el método HTTP usado por la operación.
- ***input:*** especifica la descripción de la entrada de la operación.
- ***output:*** especifica la descripción de la salida de la operación.

- ***label:*** especifica una etiqueta en lenguaje natural para un servicio, operación o mensaje.
- ***defaulting:*** se refiere al caso en el cual una página contiene algunas descripciones de operaciones pero no contiene elementos con la clase *service.* En ese caso, el analizador que procesa el documento debería asumir que la página describe un servicio simple sin etiqueta *label*.

```
<div class="service" id="svc">
<p>Description of the
  <span class="label">ACME Hotels</span> service:</p>
<div class="operation" id="op1"><p>
  The operation <code class="label">getHotelDetails</code> is
  invoked using the method <span class="method">GET</span>
  at <code class="address">http://example.com/h/{id}</code>,
  with <span class="input">the ID of the particular hotel replacing
     the parameter <code>id</code></span>.
  It returns <span class="output">the hotel details in an
     <code>ex:hotelInformation</code> document.</span>
</p></div></div>
```

Figura 12. Ejemplo de una descripción de servicio hRESTS. Tomado de (Kopecký, Gomadam et al. 2008)

3.2. Enriquecimiento Semántico de Servicios Web RESTful

Debido al incremento en el número de servicios Web RESTful disponibles en la red, la búsqueda y recuperación de éstos basados en palabras claves seguramente es insuficiente, ya que los usuarios solamente recuperarían los servicios cuyo nombre coincida exactamente con el que busca (Seung-Jun Cha 2010). Por otro lado, la plataforma Web es ampliamente usada por infinidad de aplicaciones de diferentes tipos (*e-commerce, t-learning,* redes sociales, etc.) desarrolladas bajo estándares en un intento primario por compartir toda clase de documentos a través de una red global, distribuida y heterogénea de computadoras. Este nuevo paradigma establece una tendencia clara que va desde una Web centrada en las personas hacia una centrada en aplicaciones software, lo que le permite abrirse a un nuevo mundo de posibilidades. Para ello, es necesario adicionar semántica a la información disponible en la Web (Web Semántica[11]) y desarrollar componentes software escalables y que soporten interoperabilidad universal (Nandigam, Gudivada et al. 2005). Teniendo en cuenta lo anterior, con el fin de proveer un mecanismo de Descubrimiento de Servicios Web RESTful, es necesario agregar cierto grado de semántica a la descripción de los mismos, con el fin de mejorar los resultados de recuperación de un servicio, ya que pueden existir dos o más que realicen la misma función pero que tengan un nombre distinto (serían diferentes a nivel sintáctico pero equivalentes a nivel semántico). A

[11] La Web semántica es definida como "la Web de datos". La visión de una Web semántica fue concebida por ***Tim Berners-Lee***, el inventor de WWW (World Wide Web). Tim Berners-Lee la definió como *"una Web de datos que pueden ser procesados directa e indirectamente por máquinas"*

continuación se presentan las iniciativas que intentan adicionar anotaciones semánticas a las descripciones de servicios Web existentes.

3.2.1. SAWSDL (Semantic Annotations for WSDL and XML Schema)

SAWSDL propone un mecanismo para enriquecer las descripciones funcionales de un servicio Web adicionando semántica a sus descripciones WSDL. Es una recomendación de la W3C desde Agosto de 2007. Concretamente, SAWSDL presenta un conjunto de extensiones de los atributos para WSDL y XML Schema que permite la descripción de los aspectos semánticos de los servicios. Así, SAWSDL parte de la suposición de que existe ya un modelo semántico del servicio Web, y describe un mecanismo para enlazar o relacionar ese modelo semántico con las descripciones funcionales sintácticas capturadas por WSDL a través de un conjunto de anotaciones que describen las entradas, salidas y las operaciones de un servicio Web (Farrell and Lausen 2007; Toma, Steinmetz et al. 2008) .

En (Toma, Steinmetz et al. 2008), se define para SAWSDL la siguiente terminología:

- ***Modelo Semántico****:* se refiere a un conjunto de representaciones legibles por máquina utilizadas para modelar un área de conocimiento o alguna parte del mundo, incluido el software. Un ejemplo de un modelo semántico son las ontologías.
- ***Concepto****:* es un elemento de un modelo semántico, el cual debe ser identificable a través de una URI. Un ejemplo de un concepto puede ser un clasificador en algún lenguaje, el valor de la propiedad de una instancia de una ontología o un axioma.
- ***Anotación semántica****:* es una información adicional en un documento que identifica o define un concepto en un modelo semántico con el fin de describir parte de dicho documento. Las anotaciones semánticas son de dos tipos: identificadores explícitos de conceptos, o identificadores de mapeos desde WSDL a conceptos o viceversa.
- ***Semántica****:* se refiere a un conjunto o conjuntos de conceptos identificados por anotaciones.

SAWSDL propone anotaciones semánticas básicas con el fin de ser usadas en la descripción de interfaces, operaciones, elementos y atributos de archivos WSDL y XML-Schema (Toma, Steinmetz et al. 2008; Kopecký, Vitvar et al. 2009), las cuales se detallan a continuación:

- ***modelReference:*** utilizado en cualquier componente en el modelo del servicio para apuntar o mapear a conceptos semánticos apropiados identificados por diferentes URI.
- ***liftingSchemaMapping:*** utilizado para mapear o asociar conceptos en el nivel del servicio (usualmente una descripción en formato XML) a un nivel semántico (generalmente RDF), a través de transformaciones apropiadas. Usualmente es usado para tareas de post-*descubrimiento.*
- ***loweringSchemaMapping:*** utilizado para mapear o asociar conceptos en el nivel semántico a un nivel de servicio. Usualmente es usado para tareas de post-*descubrimiento.*

Utilizando los elementos anteriores es posible crear anotaciones, como se aprecia en el Ejemplo 1 (nótese que las líneas que interesan en este contexto están resaltadas). En él se presenta un ejemplo de anotaciones SAWSDL para una interfaz WSDL de órdenes de compra. Las anotaciones aparecen tanto en elementos WSDL como en atributos XSD. ***modelReference,*** como ya se explicó anteriormente, identifica un concepto en un modelo semántico que describe el elemento al cuál es enlazado. En este ejemplo, el elemento *OrderRequest* es descrito por el concepto "OrderRequest" en la ontología cuya URI es *"http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder".* Asimismo, *loweringSchemaMapping* también es asociada al elemento *OrderRequest* para apuntar a un tipo de mapeo, en este caso un documento XML, el cual muestra cómo los elementos dentro de *OrderRequest* pueden ser mapeados desde datos semánticos en el modelo.

```
<wsdl:description
  targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl">

  <wsdl:types>
    <xs:schema
targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
      elementFormDefault="qualified">
      <xs:element name="OrderRequest"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/
purchaseorder#OrderRequest"

sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/ma
pping/RDFOnt2Request.xml">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="customerNo" type="xs:integer" />
            <xs:element name="orderItem" type="item" minOccurs="1"
maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="item">
        <xs:all>
          <xs:element name="UPC" type="xs:string" />
        </xs:all>
        <xs:attribute name="quantity" type="xs:integer" />
      </xs:complexType>
      <xs:element name="OrderResponse" type="confirmation" />
      <xs:simpleType name="confirmation"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/
purchaseorder#OrderConfirmation">
        <xs:restriction base="xs:string">
          <xs:enumeration value="Confirmed" />
          <xs:enumeration value="Pending" />
          <xs:enumeration value="Rejected" />
        </xs:restriction>
      </xs:simpleType>
    </xs:schema>
  </wsdl:types>

  <wsdl:interface name="Order"
```

```
sawsdl:modelReference="http://example.org/categorization/products/elec
tronics">
    <wsdl:operation name="order" pattern="http://www.w3.org/ns/wsdl/in-out"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/
purchaseorder#RequestPurchaseOrder">
      <wsdl:input element="OrderRequest" />
      <wsdl:output element="OrderResponse" />
    </wsdl:operation>
  </wsdl:interface>
</wsdl:description>
```

Ejemplo 1. Documento de descripción de servicio WSDL con anotaciones semánticas de tipo SAWSDL. Tomado de (Farrell and Lausen 2007)

3.2.2. SA-REST

SA-REST es un *poshformat*[12] que permite agregar metadatos adicionales (típicamente metadatos ontológicos) a las descripciones en HTML o XHTML de los servicios REST. Así, metadatos de varias fuentes como ontologías o taxonomías pueden ser embebidas en las descripciones, facilitando tareas como mediación de datos e integración y búsqueda de servicios (Gomadam, Ranabahu et al. 2010).

Las propiedades básicas de SA-REST, que son especificadas utilizando los atributos *class* y *title* definidos por la especificación HTML, se detallan a continuación:

- ***domain-rel****:* permite una descripción de información de dominio para un recurso. Es usada cuando un recurso tiene contenido que abarca varios dominios.
- **sem-*rel****:* captura la semántica de un link, permitiendo la adición de anotaciones a documentos de terceros. Esta propiedad puede sólo ser usada con el elemento <*a*>.
- ***sem-class****:* puede ser usado para marcar una entidad simple dentro de un recurso. La entidad puede ser un fragmento de texto u objetos embebidos como video, o audio.

```
<p>
A <b><span class="sem-class" title="http://tap.stanford.edu/#computer"> computer </span></b>
is a <a href="/wiki/Machine" title="Machine">machine</a> that manipulates
<a href="/wiki/Data_(computing)" title="Data (computing)">data</a> according
to a set of <a href="/wiki/Source_code" title="Source code">instructions</a>.
</p>
<p>
<span class="domain-rel" title="http://www.owl-ontologies.com/ComputingOntology.owl#History_of_Computing" >
Although mechanical examples of computers have existed through much of recorded human
history, the first electronic computers were developed in the mid-20th century (1940–1945).</span> </p>
```

Figura 13. Ejemplo de una descripción SA-REST. Tomado de (Gomadam, Ranabahu et al. 2010)

[12] *Poshformats* son formatos de datos construidos a partir del uso de nombres de clases semánticas. Se diferencian de los microformatos, los cuales son el conjunto apropiado de "poshformatos" desarrollados través de los principios y procesos de los microformatos. Ver: http://microformats.org/wiki/poshformats

La Figura 13 presenta un ejemplo de una descripción enriquecida con SA-REST. SA-REST utiliza algunas anotaciones propietarias para describir servicios Web REST basadas en una versión extendida del modelo de servicios definido para hRESTS. Adicionalmente, los documentos SA-REST pueden ser procesados utilizando XSLT para extraer la información relevante.

3.2.3. WSMO (Web Service Modeling Ontology)

Es la mayor iniciativa en el área de los Servicios Web Semánticos iniciada en Europa (Toma, Steinmetz et al. 2008). WSMO provee especificaciones ontológicas para los elementos principales de los servicios Web semánticos. Otros autores la definen como un modelo conceptual que define la sintaxis y la semántica que tienen los elementos que describen un servicio Web semántico (Toma, Steinmetz et al. 2008; Carrión Jiménez 2010).

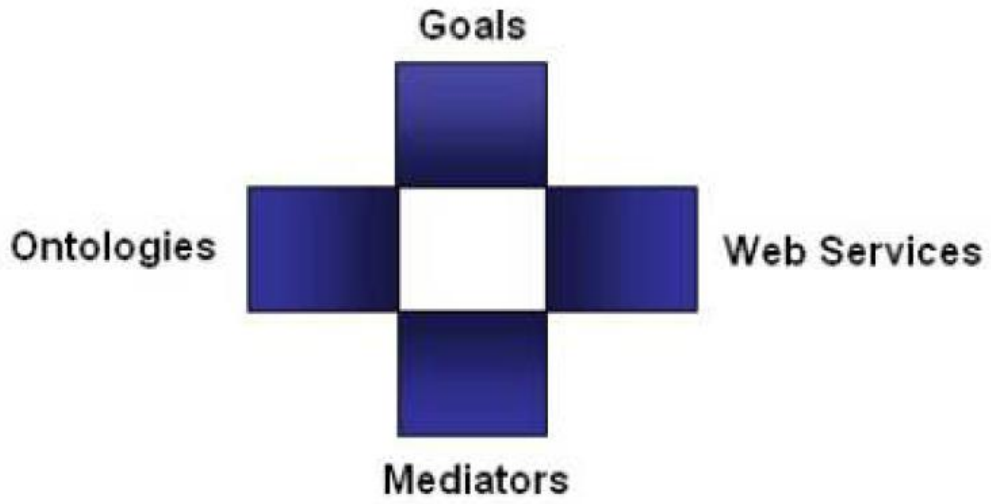


Figura 14. Modelo Conceptual de WSMO

En la Figura 14 se muestra el modelo conceptual de MicroWSMO. Se compone de cuatro elementos principales que son: Ontologías, Objetivos, Servicios Web y Mediadores. Cada uno de estos elementos se especifica a continuación:

- **Ontologías:** proporciona la semántica formal para la terminología utilizada dentro de todos los componentes de MicroWSMO. Las ontologías no sólo se centran exclusivamente en aspectos sintácticos, sino que adicionalmente proveen definiciones formales que son procesables por máquina y que permiten a componentes software y aplicaciones de terceros tener en cuenta su significado (Roman, Keller et al. 2005).
- **Servicios Web:** representan entidades computacionales capaces de proveer acceso a los servicios, que a su vez proporcionan valor en un dominio. Así, las descripciones de servicios Web se componen de capacidades (*capabilities*), interfaces y trabajo interno del servicio, descritas a través de la terminología definida por las ontologías (Roman, Keller et al. 2005).
- **Objetivos:** describen aspectos relacionados con los deseos de los usuarios con respecto a la funcionalidad requerida. Los aspectos relevantes de los objetivos son descritos a través de ontologías (Roman, Keller et al. 2005).

- **Mediadores:** describen los elementos encargados de gestionar los problemas de interoperabilidad entre los elementos anteriormente referidos, o también se interconectan para ofrecer servicios complejos (que se componen de dos o más servicios). Los mediadores resuelven problemas de interoperabilidad a nivel de datos, a nivel de protocolo y a nivel de procesos (Roman, Keller et al. 2005; Carrión Jiménez 2010).

3.2.3.1. MicroWSMO

Es una extensión de hRESTS que adiciona anotaciones semánticas a la descripción de servicio propuesta en esta especificación. Para expresar esta semántica, concretamente MicroWSMO adopta la ontología de servicio de WSMO-Lite (Fensel, Fischer et al. 2010). MicroWSMO cumple una función análoga a la de SAWSDL, ya que se ubica en la capa de anotaciones semánticas independiente de la tecnología de servicios Web utilizada (WSDL/SOAP o REST/HTTP). La Figura 15 ilustra esta situación.

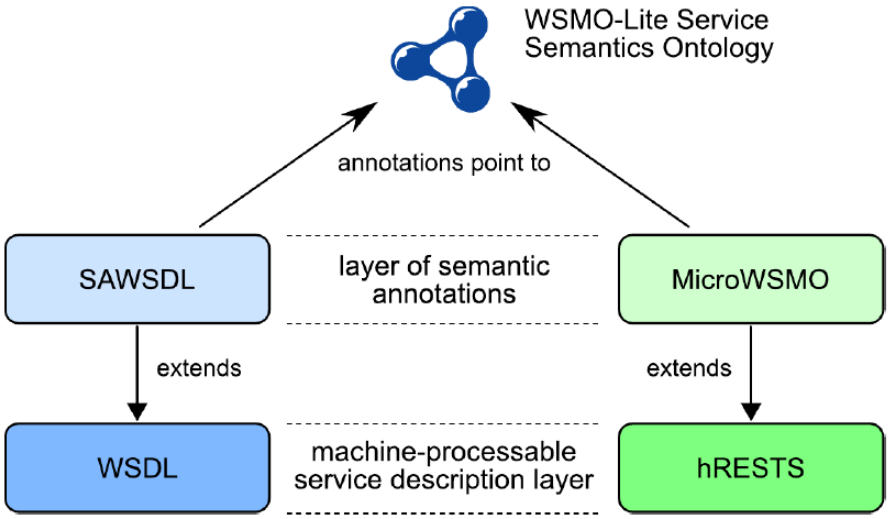


Figura 15. Posicionamiento de MicroWSMO y WSMO-Lite. Tomado de (Fensel, Fischer et al. 2010)

3.2.3.2. WSMO-Lite

Está inspirado en la ontología WSMO, pero sólo se centra en un subconjunto de ésta. Ha sido creado debido a la necesidad de una ontología de servicio liviana que esté construida sobre los estándares más recientes de la W3C (Fensel, Fischer et al. 2010). Es considerada como "la próxima evolución revolucionaria después de SAWSDL, completando las anotaciones SAWSDL con descripciones de servicio semánticas concretas" (Fensel, Fischer et al. 2010).

Como ya se ha mencionado anteriormente, SAWSDL no define un modelo semántico como tal sino que sólo define las anotaciones que se agregan a los archivos de descripción para enlazar o relacionar dicho documento con una descripción de la semántica del servicio (modelo semántico). De esta forma, ese modelo semántico es implementado utilizando las anotaciones definidas por WSMO-Lite. Sin embargo, WSMO-Lite no está ligado obligatoriamente a SAWSDL, ya que sólo lo utiliza como un mecanismo de anotación para archivos de descripción; WSMO-Lite puede ser utilizado

en cualquier descripción de servicio legible por máquinas combinado con mecanismo apropiado de anotación (Fensel, Fischer et al. 2010).

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix owl: <http://www.w3.org/2002/07/owl#> .
4  @prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
5
6  wsl:Ontology a rdfs:Class;
7    rdfs:subClassOf owl:Ontology.
8  wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class.
9  wsl:NonFunctionalParameter a rdfs:Class.
10 wsl:Condition a rdfs:Class.
11 wsl:Effect a rdfs:Class.
```

Figura 16. Ontología de servicio de WSMO-Lite. Tomado de (Fensel, Fischer et al. 2010)

En la Figura 16 se puede observar la ontología de servicio de WSMO-Lite. La semántica de los elementos de WSMO-Lite se explica a continuación:

- *wsl:Ontology:* define un contenedor para una colección de aseveraciones o afirmaciones acerca del modelo de información de un servicio. Es una subclase de *owl:Ontology* limitada a ontologías que pueden servir como modelos de información.
- *wsl:FunctionalClassificationRoot:* marca una clase que es la raíz de una clasificación la cual también incluye todas las subclases RDFS de la clase raíz. Una clasificación (taxonomía) de las funcionalidades de un servicio puede ser usada para la descripción funcional del mismo.
- *wsl:NonFunctionalParameter:* declara un marcador de posición para una propiedad no funcional específica de dominio.
- *wsl:Condition y wsl:Effect:* ambas conforman una capacidad (*capability*) en una descripción de servicio funcional. Se espera que ambas usen un lenguaje lógico concreto para describir las expresiones lógicas para condiciones y efectos.

Anotaciones WSMO-Lite para WSDL

En (Fensel, Fischer et al. 2010) se definen unos tipos particulares de anotaciones soportadas por WSMO-Lite que se clasifican en:

- *Anotaciones de referencia:* son anotaciones que apuntan a un concepto semántico de WSMO-Lite **desde** un componente WSDL (como *interface, operation, service, etc.*). En SAWSDL se representa este tipo de anotación mediante el atributo *modelReference. (Ver (Farrell and Lausen 2007)).*
- *Anotaciones de transformación:* son anotaciones que especifican una transformación de datos llamada *lifting* cuando se trata desde un componente de un esquema XML a un concepto de una ontología, y *lowering* cuando es de un concepto de una ontología a XML. En SAWSDL se representa este tipo de anotación mediante los atributos *liftingSchemaMapping* y *loweringSchemaMapping. (*Ver (Farrell and Lausen 2007)).

En el Ejemplo 2 se puede apreciar un prototipo de una ontología de dominio donde se utilizan los elementos de WSMO-Lite.

```
# namespaces and prefixes
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wsl:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix ex:   <http://example.org/onto#> .
@prefix xs:   <http://www.w3.org/2001/XMLSchema#> .

# domain ontology
ex:Customer  a  rdfs:Class .
ex:hasService  a  rdf:Property ;
    rdfs:domain ex:Customer ; rdfs:range ex:Service .
ex:Service  a  rdfs:Class .
ex:hasConnection  a  rdf:Property ;
    rdfs:domain ex:Customer ; rdfs:range ex:NetworkConnection .
ex:NetworkConnection  a  rdfs:Class .
ex:providesBandwidth  a  rdf:Property ;
    rdfs:domain ex:NetworkConnection ; rdfs:range xs:integer .
ex:VideoOnDemandService rdfs:subClassOf ex:Service .

# capability description example
ex:VideoOnDemandSubscriptionPrecondition  a  wsl:Condition ;
    rdf:value
        "Prefix (ex <http://example.org/onto#>)
         Prefix (pred <http://www.w3.org/2007/rif-builtin-predicate#>)
         And (?Customer#ex:Customer
              ?Customer[ex:hasConnection->?Connection]
              ?Connection#ex:NetworkConnection
              ?Connection[ex:providesBandwith->?Y]
              External(pred:numeric-greater-than(?Y 1000)))" .
ex:VideoOnDemandSubscriptionEffect  a  wsl:Effect ;
    rdf:value
        "Prefix (ex <http://example.org/onto#>)
         Prefix (pred <http://www.w3.org/2007/rif-builtin-predicate#>)
         And (?Customer#ex:Customer
              ?Customer[ex:hasService->?Service]
              ?Service#ex:VideoOnDemandSubscription)" .
# nonfunctional property example
ex:PriceSpecification rdfs:subClassOf wsl:NonfunctionalParameter .
ex:VideoOnDemandPrice  a  ex:PriceSpecification ;
    ex:pricePerChange "30"^^ex:euroAmount ;
    ex:installationPrice "49"^^ex:euroAmount .

# classification example
ex:SubscriptionService  a  wsl:FunctionalClassificationRoot .
ex:VideoSubscriptionService rdfs:subClassOf ex:SubscriptionService .
ex:NewsSubscriptionService rdfs:subClassOf ex:SubscriptionService .
```

Ejemplo 2. Prototipo de una ontología de dominio que muestra los diferentes tipos de anotaciones WSMO-Lite. Tomado de (Fensel, Fischer et al. 2010)

En el Ejemplo 2 se define un prototipo de una ontología de dominio para un servicio de telecomunicaciones, y además se definen también una condición y efecto para un servicio de subscripción de video bajo demanda, un ejemplo de una propiedad no funcional (que describe el precio del servicio) y un ejemplo de una clasificación de servicios simple con tres niveles o categorías. El lenguaje utilizado en el ejemplo para

expresar las capacidades (condición y efecto) del servicio es RFI (*Rule Interchange Format*[13])

Posicionamiento de las anotaciones WSDL

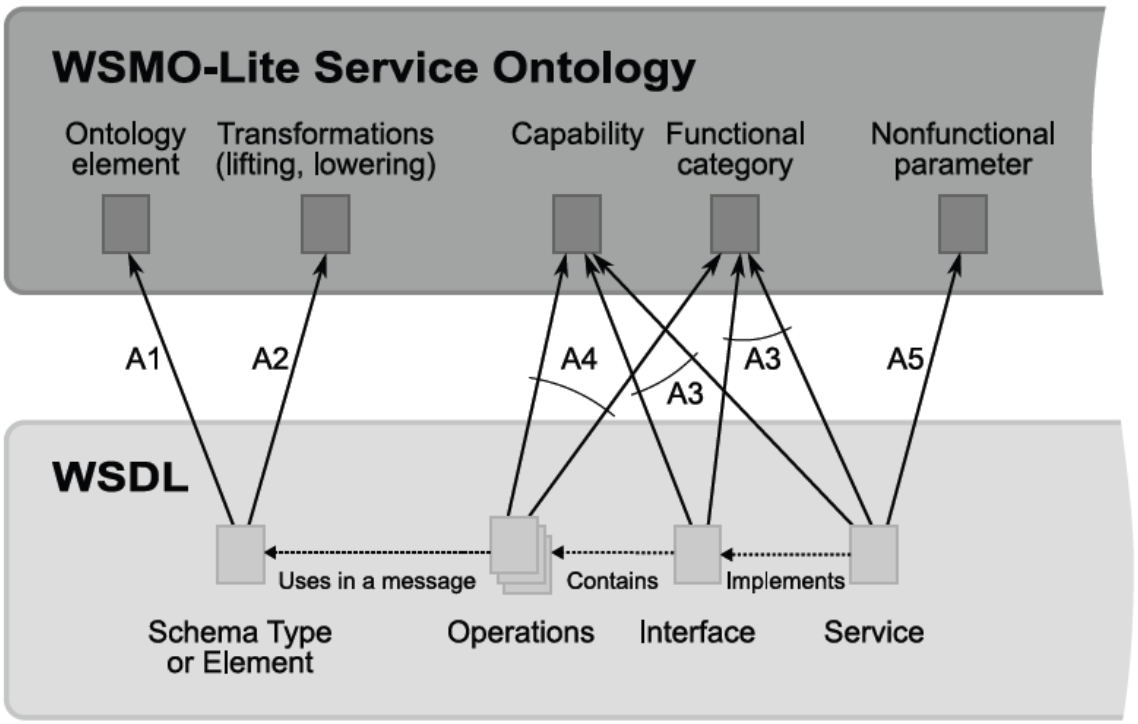


Figura 17. Anotaciones del tipo WSMO-Lite en WSDL. Tomado de (Fensel, Fischer et al. 2010)

Las anotaciones WSMO-Lite para WSDL se aprecian en la Figura 17 y se definen como:

- **A1. Anotaciones ontológicas del esquema XML:** el esquema es usado en WSDL para describir mensajes (definiciones de tipos y declaraciones de elementos) que pueden relacionar clases de una ontología del modelo de información de un servicio con los mensajes.
- **A2. Anotaciones de transformación del esquema XML:** para comunicarse con un servicio, es necesario transformar los datos entre su modelo semántico y las estructuras de mensajes XML.
- **A3. Anotaciones funcionales de Interfaces y Servicio:** este tipo de anotaciones aplican tanto para servicios Web concretos como para interfaces. Una anotación de referencia apunta a una descripción funcional apropiada desde un servicio o interfaz. Un ejemplo de estas anotaciones se ve en la Figura 18:

[13] Para más información ver: http://www.w3.org/standards/techs/rif#w3c_all

```
<wsdl:interface name="NetworkSubscription"
      sawsdl:modelReference="http://example.org/onto#VideoSubscriptionService
                             http://example.org/onto#VideoOnDemandSubscriptionPrecondition
                             http://example.org/onto#VideoOnDemandSubscriptionEffect" >
   <wsdl:operation name="CheckNetworkConnection" ... />
</wsdl:interface>
```

Figura 18. Ejemplo de anotaciones funcionales en un fragmento de una descripción WSDL. Tomado de (Fensel, Fischer et al. 2010)

- **A4. Anotaciones funcionales de operaciones de interfaces WSDL:** este tipo de anotaciones aplican para operaciones de interfaces, con el fin de indicar sus funcionalidades particulares. Una anotación de referencia apunta de una operación a su descripción funcional apropiada.
- **A5. Anotaciones no-funcionales de un servicio WSDL*:*** este tipo de anotaciones aplican a una instancia concreta de un servicio Web (Servicio WSDL). Una anotación de referencia puede apuntar desde un componente de servicio a una propiedad no-funcional. Un ejemplo de estas anotaciones se ve en la Figura 19:

```
<wsdl:service name="ExampleCommLtd"
      interface="NetworkSubscription"
      sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
   <wsdl:endpoint ... />
</wsdl:service>
```

Figura 19. Ejemplo de anotaciones no-funcionales. Tomado de (Fensel, Fischer et al. 2010)

3.3. XML (Extensible Markup Language)

Es un formato de texto simple y flexible, derivado de SGML[14], originalmente diseñado para afrontar el reto de las publicaciones electrónicas a gran escala. Fue desarrollado por el XML Working Group[15] formado por la W3C en 1996. Describe una serie de objetos de datos llamados documentos XML, y describe en parte el comportamiento de programas de computadores que procesan estos documentos. XML juega hoy en día un importantísimo papel en el intercambio de datos entre aplicaciones y servicios Web (Bray, Paoli et al. 1997; Quin 2003). Los objetivos de diseño para XML (Bray, Paoli et al. 1997) son:

- Debe ser directamente utilizable a través de Internet
- Debe soportar una amplia variedad de aplicaciones
- Debe ser compatible con SGML
- Debe ser fácil escribir programar que procesen documentos XML
- El número de características opcionales en XML debe mantenerse al mínimo, idealmente cero

[14] Standard Generalized Markup Language: es una tecnología estándar ISO para la definición de lenguajes de marcado generalizado para documentos. HTML es un ejemplo de un lenguaje de marcado. Información detallada disponible en: http://www.w3.org/TR/html4/intro/sgmltut.html
[15] http://www.w3.org/2000/xp/Group/

- Los documentos XML deberían ser razonablemente claros y legibles por humanos
- El diseño XML debe ser preparado rápidamente y debe ser formal y conciso
- Los documentos XML deben ser fáciles de crear
- La brevedad en el marcado de XML es de importancia mínima

3.4. XML Schema

XML Schema un lenguaje basado en XML, que se convirtió en una recomendación de la W3C desde el 2 de Mayo del 2001, y cuyo propósito es describir la estructura de un documento basado en XML; por lo tanto, XML Schema es considerado como una alternativa para DTD (Document Type Definition), incluso algunos autores lo consideran no sólo como una alternativa sino como un sucesor. El lenguaje XML Schema es también conocido como XML Schema Definition (XSD), éste proporciona un medio para la definición de la estructura, contenido y semántica de los documentos XML. XML es un lenguaje de marcado muy flexible, que permite al usuario definir etiquetas personalizables, lo que puede causar muchos problemas de interoperabilidad. Así, XML Schema constituye una alternativa valiosa ya que permite definir el orden de las etiquetas, sus atributos, los tipos de datos permitidos en el documento, validación de los datos, restricciones, y mucho más. La Figura 20 muestra un pequeño ejemplo de un documento XML y su respectivo documento XML Schema (Figura 21).

```
<?xml version="1.0"?>
<book id ="1">
    <author>Dan Brown</author>
    <title>The Da Vinci Code</title>
    <country>United States</country>
    <genre>Mystery</genre>
    <pages>454</pages>
</book>
```

Figura 20. Documento de ejemplo XML

En el esquema XML de la Figura 21 se aprecia que el elemento *book* es de tipo complejo (debido a que contiene atributos y elementos subyacentes). El atributo *id* requerido (obligatorio) y es de tipo entero. De esta manera, todos los documentos XML asociados a este esquema deben definir un atributo *id* para la etiqueta *book*. Además, también se define que los elementos subyacentes (*author, title, country, genre y pages)* deben estar en secuencia. Finalmente, para el elemento *pages* se define una restricción: el valor de la etiqueta *pages* debe estar en el rango de 0 a 900.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="book">
  <xs:complexType>
    <xs:attribute name="id" type="xs:integer" use="required" />
    <xs:sequence>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
      <xs:element name="genre" type="xs:string"/>
      <xs:element name="pages">
        <xs:simpleType>
            <xs:restriction base="xs:integer">
                <xs:minInclusive value="0" />
                <xs:maxInclusive value="900" />
            </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Figura 21. Documento de ejemplo XML Schema

3.5. XSLT

Es un lenguaje definido para la transformación de documentos XML en otros documentos XML. XSLT hace parte de la familia de recomendaciones que definen la transformación y presentación de documentos XML llamada XSL (*Extensible Stylesheet Language*)[16], de ahí provienen sus siglas (*XSL Transformations*). Una transformación en el lenguaje XSLT se expresa como un documento XML bien formado de acuerdo a las recomendaciones normativas de XML y al uso de los espacios de nombres. El documento está formado por dos tipos de elementos: elementos definidos por XSLT (que se caracterizan por la respectiva referencia a su espacion de nombres que es: *http://www.w3.org/1999/XSL/Transform*) y elementos que no están definidos por XSLT. Una transformación expresada en XSLT describe unas reglas para transformar un árbol de etiquetas XML de origen en otro árbol de etiquetas de resultado, a través de una asociación de patrones con plantillas. Estas reglas tienen dos partes: un patrón que es comparado con los nodos en el árbol del documento de origen y una plantilla que es instanciada para formar el árbol del documento resultante. Así, el árbol de etiquetas resultante puede ser diferente al árbol

[16] La familia de recomendaciones XSL se compone de tres partes:
- XSLT: lenguaje para transformar documentos XML.
- XPath (*XML Path Language*): lenguaje para acceder a ciertas partes de un documento XML, el cual es utilizado por muchos otros lenguajes (entre ellos XSLT).
- XSL-FO (*XSL Formatting Objects*): es un vocabulario XML para declarar la semántica del formato.

de etiquetas de origen y además, el documento de transformación XSLT puede ser aplicado una amplia clase de documentos (Clark 1999).

```
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      lang="en">
    <head>
        <title>Sales Results By Division</title>
    </head>
    <body>
        <table border="1">
            <tr>
                <th>Division</th>
                <th>Revenue</th>
                <th>Growth</th>
                <th>Bonus</th>
            </tr>
            <xsl:for-each select="sales/division">
                <!-- order the result by revenue -->
                <xsl:sort select="revenue"
                          data-type="number"
                          order="descending"/>
                <tr>
                    <td>
                        <em><xsl:value-of select="@id"/></em>
                    </td>
                    <td>
                        <xsl:value-of select="revenue"/>
                    </td>
                    <td>
                        <!-- highlight negative growth in red -->
                        <xsl:if test="growth &lt; 0">
                             <xsl:attribute name="style">
                                 <xsl:text>color:red</xsl:text>
                             </xsl:attribute>
                        </xsl:if>
                        <xsl:value-of select="growth"/>
                    </td>
                    <td>
                        <xsl:value-of select="bonus"/>
                    </td>
                </tr>
            </xsl:for-each>
        </table>
    </body>
</html>
```

Ejemplo 3. Ejemplo de un documento de transformación XSLT. Tomado de (Clark 1999)

El Ejemplo 3 muestra un documento XSLT para transformar datos de entrada en HTML.

3.6. RDF

Resource Description Framework es un lenguaje para la representación de recursos en la Web. También es definido como un framework cuyo propósito es representar la información alojada en la Web. Se convirtió en una recomendación W3C desde Febrero del 2004 y hace parte de la *W3C's Semantic Web Activity*. RDF ha sido concebido con el fin de representar meta información (título, autor, fecha de modificación, copyright, etc.) relacionada con recursos Web (páginas Web, documentos Web, etc.). RDF fue ideado para situaciones en las que la información presente en la Web necesita ser procesada por aplicaciones y no sólo ser presentada

al usuario; estas aplicaciones pueden utilizar muchas herramientas de procesamiento para RDF que existen hoy en día, ya que RDF provee un framework común para el intercambio de información (Manola and Miller 2004).

El lenguaje se basa en la idea de que cada recurso en la Web está representado por identificadores Web o URI (*Universal Resource Identifiers*) y está descrito por propiedades y valores de esas propiedades.

En la Figura 22 se ve un conjunto de declaraciones RDF representadas de forma gráfica.

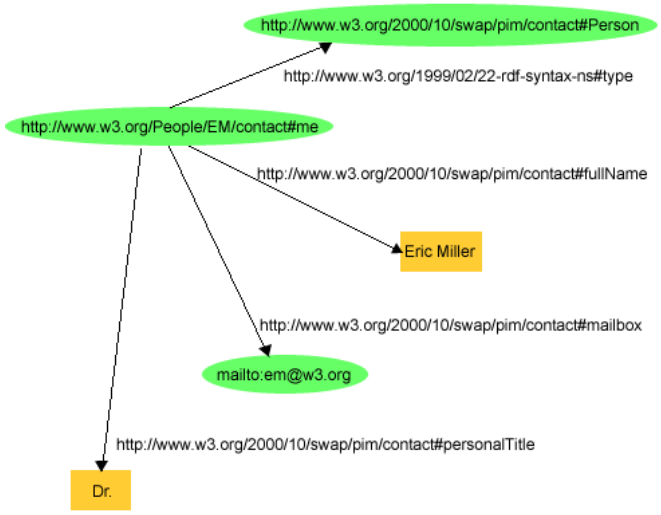


Figura 22. Ejemplo de un grupo de declaraciones RDF, vistas de forma gráfica. Tomado de (Manola and Miller 2004)

Las declaraciones RDF representadas en el ejemplo son: "hay una Persona identificada por http://www.w3.org/People/EM/contact#me cuyo nombre es **Eric Miller,** cuya dirección **email** es **em@w3.org**, y cuyo título es **Dr.**"

De una manera formal, para intercambiar o guardar modelos gráficos de declaraciones RDF se utiliza una sintaxis basada en XML (denominada *RDF/XML*). En la Figura 23 se aprecia un fragmento del ejemplo de la Figura 22 en la notación *RDF/XML.*

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Figura 23. Ejemplo de declaraciones RDF en notación RDF/XML. Tomado de (Manola and Miller 2004)

A continuación se exponen de forma breve algunos conceptos referentes a RDF; para ello, se considera la siguiente declaración:

http://www.example.org/index-html tiene un **creador** cuyo valor es **Jack Sparrow**

Ejemplo 4. Declaración RDF en lenguaje natural

RDF está basado en la idea de que las cosas que se describen tienen propiedades que a su vez tienen valores, y que los recursos pueden ser descritos a través de declaraciones o sentencias que definen esas propiedades y valores. Así, las tres partes que conforman una declaración RDF (también llamadas tripletas RDF) son:

- **Sujeto:** es la parte que identifica el recurso al cual la declaración se refiere. En el anterior ejemplo, el sujeto es la URL http://www.example.org/index.html.
- **Predicado:** es la parte que identifica la propiedad del sujeto. En el anterior ejemplo, el predicado es **"creador".**
- **Objeto:** es la parte que identifica el valor de la propiedad. En el anterior ejemplo, el objeto es **"Jack Sparrow".**

Formalmente, con el fin de ser identificadas únicamente sin lugar a confusiones para que puedan ser intercambiados entre máquinas, cada una de las partes que conforman tripletas RDF debe ser identificada con una URI (Manola and Miller 2004). Como se puede apreciar en la Figura 23, en la notación formal *RDF/XML* es común el uso de espacios de nombres para declarar las tripletas.

3.7. OWL

Es un lenguaje de marcado usado para publicar y compartir datos a través de redes haciendo uso de ontologías. Es aprobado y recomendado por la W3C, razón por la cual se podría considerar como un estándar, ya que ha sido ampliamente usado en muchos campos, especialmente el académico. OWL se diferencia de sus predecesores en que esta perfectamente acoplado con la arquitectura de la WWW y la Web semántica específicamente.

OWL se complementa con RDF para obtener las siguientes capacidades:

- Distribución a través de diferentes sistemas.
- Escalable
- Compatible con estándares web para la accesibilidad e internacionalización
- Extensible

OWL tiene tres variantes, que se diferencian entre sí por el grado de expresividad que manejan, representado en etiquetas disponibles para armar la ontología, que son: OWL Lite, OWL DL y OWL Full (ordenados de menor a mayor expresividad). OWL se usa para representar el significado de los recursos y la relación entre ellos, lo que formalmente se conoce como ontología.

OWL DL (*Description Logics*) como lenguaje web de ontología está diseñado para una expresividad media ya que se usan todas las construcciones del lenguaje OWL pero con ciertas restricciones. A continuación se presentan las construcciones más usadas:

- ***Class:*** define un grupo de recursos que pertenecen a un mismo grupo por compartir algunas de sus propiedades.

- ***disjointWith***: se utiliza para establecer que una clase es disjunta con otra, lo que significa que si un recurso ya pertenece a una clase, entonces no puede pertenecer a su clase disjunta. Por ejemplo, si A es una instancia de hombre, entonces A ya no puede ser una instancia de mujer.
- ***rdfs:subClassOf*:** sirve para crear una jerarquía entre clases, en el Ejemplo 5 se puede concluir que HealthInsuranceNumber es una subclase de Number
- ***FunctionalProperty*:** se usa para definir propiedades que tendrán un valor único, así que su cardinalidad será de cero o uno.
- ***rdfs:domain*:** este reduce los recursos a los cuales pueden aplicarse una propiedad determinada.
- ***rdfs:range*:** reduce los recursos que una propiedad puede tener como su valor.
- ***rdf:type:*** indica a qué tipo pertenece el sujeto de la declaración.

A continuación se presenta una ontología descrita con OWL DL en donde se emplean los constructores mencionados anteriormente:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://127.0.0.1/ontology/ApothecaryOntology.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://127.0.0.1/ontology/ApothecaryOntology.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Costs"/>
  <owl:Class rdf:ID="HealthInsuranceNumber">
    <owl:disjointWith>
      <owl:Class rdf:ID="TelephoneNumber"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Number"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Hospital">
    <owl:disjointWith>
      <owl:Class rdf:ID="EmergencyStation"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="MedicalOrganisation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:FunctionalProperty rdf:ID="Drug_hasCosts">
    <rdfs:range rdf:resource="#Costs"/>
    <rdfs:domain rdf:resource="#Drug"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Ejemplo 5. Ontología descrita con OWL DL

3.8. WURFL (Wireless Universal Resource File)

A diferencia de una arquitectura de un servicio que presta funcionalidades a dispositivos de grandes capacidades, en donde detectar el ambiente de ejecución se hace del lado del cliente para no cargar esta responsabilidad al servidor, en un ambiente móvil en donde no se dispone de recursos o mecanismos (como *Javascript*)

para realizar esta labor, el servidor es el encargado de detectar a quien se entrega el servicio mediante las cabeceras de los mensajes (parámetro *User-Agent* de HTTP), es en este caso en donde el uso de repositorio de dispositivos como WURFL se hace tan importante.

WURFL es un archivo que funciona como un repositorio de descripción de dispositivos o DDR (*Device Description Repository*) en donde se encuentran almacenadas de forma detallada las particularidades de la mayoría de dispositivos móviles que existen en el mercado.

Es ampliamente usado por desarrolladores en sus programas para detectar, mediante el uso de API, las características del dispositivo desde el cual están accediendo al programa y así adaptar o negar funcionalidades o la presentación de los servicios, por ejemplo se puede prever la capacidad de procesamiento de un dispositivo y notificar al usuario que la aplicación no funcionará de forma óptima debido a limitaciones en el hardware, esto si no cumple con los requisitos mínimos para un procesamiento eficaz de la aplicación.

Técnicamente la información es almacenada en un archivo XML en el cual, mediante el uso de etiquetas, se define el nombre de un dispositivo y se informa sobre las características de éste.

3.8.1. Ventajas de WURFL

Las principales ventajas de usar WURFL son las siguientes:

- Contiene información de la mayoría de dispositivos móviles que existen en el mercado, con un nivel de detalle sobre sus características suficientes para satisfacer las necesidades de los desarrolladores de aplicaciones móviles.
- Es el repositorio más usado, por encima de UAProf, se puede decir que usar WURFL es trabajar con el estándar, lo usan compañías como Facebook o Google. Esto garantiza que el desarrollo de un modulo con WURFL pueda ser fácilmente reutilizado en otros proyectos software.
- WURFL es de uso libre tanto para proyectos open-source como privados.
- La base de información tiene un gran crecimiento gracias a los aportes de la comunidad, esto asegura que WURFL es un repositorio muy completo y garantiza que la información que se encuentra en el archivo XML sea siempre la más actualizada.
- Tiene API para los lenguajes más usados de programación (PHP, Perl, Ruby, Python, .Net, C+, Java). Estas interfaces de programación son desarrolladas por la compañía encargada del mantenimiento de WURFL (*ScientialMobile*) y conforman la API estándar o desarrolladas por terceros (API no estándar). En nuestro caso nos interesa la API estándar desarrollada para JAVA.
- WURFL es personalizable gracias al uso de un “patch”. El repositorio puede ser modificado a nuestra conveniencia, se pueden agregar dispositivos que por algún motivo especial no se encuentren en el repositorio, o hacer la aplicación más ligera y rápida al eliminar registros que por las características únicas de una aplicación no vaya a utilizar todos los dispositivos del mercado.

3.8.2. WURLF vs UAProf

UAProf (*User Agent Profile*) es otro repositorio de dispositivos en donde se almacena las características de estos, y además permite guardar las preferencias de los usuarios, el perfil de un usuario, para alcanzar el mismo objetivo que WURFL, mejorar las aplicaciones para dispositivos móviles teniendo en cuanta el contexto.

A pesar de que UAProf nació con el propósito de unificar la información de las características de todos los dispositivos y preferencias de uso de estos por parte de los usuarios, UAProf actualmente no tiene mucho éxito (aunque es usado) principalmente por el mal uso que le dan los fabricantes, aunque la mayoría de creadores de dispositivos móviles se acogieron a UAProf, estos no publican todas las características de sus dispositivos, además no hay un estándar sobre como publicar, existen ambigüedades en los nombres. OMA y ninguna otra organización se ha preocupado por estandarizar los archivos descriptores.

WURFL es un proyecto open-source que inicialmente se alimentó de las especificaciones publicadas en UAProf y de los aportes de la comunidad, la ventaja principal radica en que los nombres con los cuales se especifican las características están estandarizados y agrupados en 4 tipos:

- ***Product_info*:** contiene la información genérica del dispositivo como el nombre y la marca.

Capacidad	Tipo	Descripción
brand_name	string	Marca (ej: Nokia)
model_name	string	Modelo (ej: N95)
marketing_name	string	Además de la marca y modelo, algunos dispositivos incluyen el nombre comercial (ej: BlackBerry 8100 Pearl, Nokia 8800 Scirocco, Samsung M800 Instinct).
is_wireless_device	boolean	Retorna información acerca si el dispositivo es inalámbrico o no. Los teléfonos móviles y las PDA son consideradas dispositivos inalámbricos, mientras que los PC de escritorio o portátiles no
uaprof,uaprof2,uaprof3	String (URL)	Las URL UAProf
device_os	String	Información acerca del Sistema Operativo.
device_os_version	String	Versión del Sistema Operativo
mobile_browser	String	Información acerca del navegador del dispositivo (Openwave, Nokia, Opera, Access, Teleca,...)
mobile_browser_version	String	Versión del navegador

Tabla 2. WURFL grupo product_info

- ***Markup*:** almacena información sobre los lenguajes soportados por el dispositivo.

Capacidad	Tipo	Descripción
xhtml_support_level	[-1 |..| 4]	Suponiendo que el dispositivo es compatible con alguna forma de XHTML, esta capacidad determina el nivel de soporte de estas etiquetas: Nivel "-1": No soporta XHTML, en estos casos el dispositivo normalmente soporta WML. Nivel "0": Soporte básico XHTML. Pantalla mínima de 100 pixel. No soporta CSS. Soporte básico o nulo de tablas. Soporte de formularios básicos. Nivel "1": XHtml con cierto soporte de CSS. Anchura mínima de pantalla 120 pixel. Soporte básico de tablas. Nivel "2": Mismas características que en el nivel 1, pero en un futuro podrían variar. Nivel "3": Excelente soporte CSS. Bordes y márgenes correctamente aplicados. Anchura mínima de pantalla 164 pixel. Mayor soporte a tablas complejas. Soporta imágenes de fondo, incluso aplicadas desde estilos. Nivel 4: Igual que el nivel 3 pero con soporte AJAX.
preeferred_markup	string	Indica cual es el mejor lenguaje de marcas para el dispositivo.
html_web_3_2	true/false	Soporta HTML versión 3.2.
html_web_4_0	true/false	Soporta HTML versión 4
multipart_support	true/false	Soporta formularios multipart

Tabla 3. WURFL grupo markup

- ***Display*:** contiene información sobre lo que ofrece el dispositivo para el despliegue de información como la resolución.

Capacidad	Tipo	Descripción
resolution_width	int	Anchura de la pantalla en pixels
resolution_height	int	Altura de la pantalla en pixels
max_image_width	int	Anchura máxima de las imágenes
max_image_height	int	Altura máxima de las imágenes

Tabla 4. WURFL grupo display

- ***Streaming*:** información sobre el tipo de multimedia que soporta el dispositivo en s*treaming*.

Capacidad	Tipo	Descripción
streaming_video_width	true/false	True, si el dispositivo puede visualizar videos por streaming
streaming_real_media	"none",8,9,10	El dispositivo acepta streaming en RealMedia
streaming_3gpp	true/false	El dispositivo acepta streaming en 3GPP
streaming_mp4	true/false	El dispositivo acepta streaming en MP4
streaming_wmv	"none",7,8,9	El dispositivo acepta streaming en WMV
streaming_mov	true/false	El dispositivo acepta streaming en MOV
streaming_flv	true/false	El dispositivo acepta streaming en FLV
streaming_3g2	true/false	El dispositivo acepta streaming en 3GPP 2

Tabla 5. WURFL grupo streaming

Por último, detectar el ambiente de ejecución de un servicio supone una carga de procesamiento extra que es necesaria.

3.8.3. API

Como ya se mencionó anteriormente se tiene un conjunto de API desarrolladas tanto por *ScientiaMobile* como por terceros para muchos lenguajes de programación, en nuestro caso usamos la API de Java desarrollada por *ScientiaMobile*.

La función principal de la interfaz de programación es detectar el dispositivo desde el cual se hace una petición y obtener las características de éste. *WURFLManager* es el encargado de pasar una petición HTTP a *Device* (hablando de objetos) y así tener todas las propiedades del dispositivo desde donde se generó el HTTP como si fuesen atributos de la clase *Device*, aunque los atributos se consiguen mediante un método que recibe como parámetro el nombre de la característica que se desea obtener. Esto puede ser útil en cuanto a procesamiento si sólo deseamos conocer una característica en especial.

El uso y configuración de la API es fácil y se explica muy bien en la página de *ScientiaMobile*, además siempre se puede contar con el *JavaDoc* para explorar las posibilidades de la API oficial.

3.9. Resumen

En este capítulo se presentaron todas las tecnologías asociadas a la solución, que de alguna manera fueron utilizadas con el fin de soportar el mecanismo de descubrimiento de servicios Web REST. Adicionalmente se describieron todas las iniciativas (identificadas a partir de la base de conocimiento construida) para definir un mecanismo de descripción para servicios Web REST y para extender semánticamente estas descripciones. En el siguiente capítulo se expone el mecanismo de descripción y de adición semántica seleccionado.

Capítulo 4

Solución propuesta

En este capítulo se muestra el proceso de adaptación seguido en este trabajo con el fin de proponer un mecanismo de descubrimiento de Servicios Web RESTful basado en semántica. Este proceso de adaptación incluye un procedimiento previo de descripción de dichos servicios con el fin de proporcionarles la capacidad de soporte para la implementación del algoritmo de descubrimiento. Posteriormente se describe la adaptación del algoritmo de descubrimiento y los aportes realizados en este aspecto. Asimismo, se presenta la implementación de dicho algoritmo a través del desarrollo del prototipo de prueba que soporta el mecanismo de descubrimiento.

4.1. Selección del formato de descripción de los servicios Web REST

Figura 24. Actividades para la selección del formato de descripción para los servicios Web RESTful

En el capítulo 3, y gracias a la base de conocimiento construida en el marco de este trabajo de grado, se especificaron todas las alternativas para describir sintácticamente los servicios Web REST con el objetivo de soportar algún mecanismo de descubrimiento. Adicionalmente también se detallaron las alternativas para adicionar semántica a las descripciones sintácticas de los servicios. Así, inicialmente se realiza una selección de alguna de las alternativas disponibles para ello. Este proceso incluye una serie de actividades que se listan en la Figura 24 y que se presentan a continuación.

4.1.1. Análisis de alternativas de descripción para servicios Web RESTful

Con la finalidad de seleccionar el formato de descripción más adecuado según las necesidades dadas en este trabajo de grado, se definieron una serie de criterios de selección que se muestran en la Tabla 6.

Criterios	*WSDL 2.0*	*WADL*	*ReLL*	*hRESTS*
Colección de servicios de prueba	✗	✗	✗	✓
Soporte para múltiples representaciones	✗	✗	✓	✗
Baja Complejidad	✗	✗	✗	✓
Ajuste a otras características REST	✗	✓	✓	✓
Amplia adopción en la comunidad	✗	✗	✗	✓
Herramientas de soporte para los servicios	✓	✗	✗	✓

Tabla 6. Comparación de las diversas alternativas para descripción de servicios REST

Aunque no está enmarcado en los objetivos del proyecto, la definición del formato de descripción para servicios REST fue uno de los aspectos que demandó más tiempo en el desarrollo del proyecto, ya que fue necesario analizar a profundidad la literatura para encontrar algunas alternativas que dieran solución a este inconveniente. En la anterior tabla se definieron algunos criterios de comparación para seleccionar el formato más adecuado, con el fin de dar soporte al mecanismo de descubrimiento. A continuación se dará una breve explicación de cada uno de los criterios definidos:

- ***Colección de servicios de prueba:*** este ítem es quizás el más importante de todos. Hace referencia a que, con el fin de validar el algoritmo de descubrimiento implementado, se necesitan un mínimo de servicios descritos en el lenguaje seleccionado para permitir realizar una serie de pruebas de rendimiento y precisión.
- ***Soporte para múltiples representaciones:*** una de las características de los servicios Web REST es que permiten que los recursos pertenecientes tengan múltiples representaciones en distintos formatos. Así, sería ideal que el formato de descripción permitiera definir el tipo de respuesta que dará el servicio al cliente y el contenido enviado en la respuesta, para, idealmente, ofrecer al usuario la posibilidad de seleccionar el formato y contenido que más le convenga.
- ***Baja complejidad:*** este aspecto hace referencia a que el lenguaje de descripción debe ser relativamente simple, para que pueda ser ampliamente adoptado por los proveedores de servicios, ya que éstos buscan que el mecanismo para describir sus servicios y publicar su funcionalidad sea lo más simple posible.

- ***Ajuste a otras características REST:*** hace referencia a que el lenguaje adopte otras características REST, como la de soportar una interfaz uniforme para la interacción con los servicios REST a través de los métodos HTTP, la identificación de los recursos a través de URI, entre otras.
- ***Amplia adopción en la comunidad:*** es ideal que el lenguaje sea adoptado ampliamente por la comunidad de desarrollo, con el fin de tener bastantes herramientas para la implementación del mecanismo de descubrimiento.
- ***Herramientas de soporte para los servicios:*** también es ideal que existan herramientas que permitan la generación automática de código, o que permita la descripción de los servicios de la manera más simple posible.

Inicialmente se consideraron alternativas como WSDL 2.0 o WADL como formato de descripción; sin embargo, estas alternativas fueron descartadas debido a que su formato de descripción produce un alto nivel de acoplamiento entre el servidor y el cliente, dada la complejidad del archivo de descripción definido (Alarcon and Wilde 2010). Por otro lado, probablemente la opción que más se adhiere a las restricciones de REST es el lenguaje ReLL. Sin embargo, este lenguaje fue descartado y finalmente se optó por la adopción de ***hRESTS*** como lenguaje de descripción por dos razones principales:

- ReLL está más orientado a describir los servicios como recursos y las relaciones existentes entre éstos. Esto lo hace algo complejo y definitivamente no adecuado para algunos servicios Web REST ya existentes debido a que el lenguaje es relativamente nuevo y no existen herramientas que faciliten la descripción. Por el contrario, hRESTS es un formato más “maduro” que lleva mucho más tiempo en la comunidad. De esta manera, existen herramientas como la presentada en (Maleshkova, Pedrinaci et al. 2010) que permiten describir servicios REST de una manera relativamente simple, y en alto nivel (sin tener en cuenta detalles profundos de implementación).
- ReLL no cuenta con un conjunto de servicios Web descritos con el fin de realizar pruebas, lo cual es un inconveniente insalvable. En contraposición, hRESTS cuenta con un conjunto de servicios descritos con el fin de realizar pruebas. Esto se explicará detalladamente en los siguientes capítulos.

Adicionalmente, y como se especifica en (Lampe, Schulte et al. 2010), la gran mayoría de servicios Web RESTful son descritos en lenguaje natural en una página Web tipo HTML. hRESTS aprovecha este tipo de páginas, añadiéndoles información “legibles por máquinas” a través de una serie de etiquetas para marcar los fragmentos de interés de la descripción con el fin de hacer que los servicios sean descubiertos; esto se hace de manera transparente al usuario final, es decir, que la página de descripción se seguirá visualizando de la misma manera. Por lo tanto, hRESTS constituye la alternativa más práctica (de mayor adopción) en la actualidad.

4.1.2. Extensión del formato de descripción

En hRESTS, como se mencionó en apartes anteriores de este documento, los servicios Web se describen a partir de operaciones pertenecientes a un servicio, que tienen entradas y salidas (análogo a WSDL en los servicios Web tradicionales). Sin embargo, en hRESTS se definen propiedades que son características de los servicios Web RESTful, como la URI de cada recurso y el método HTTP con el cuál se invoca

éste. De esta manera, un algoritmo de descubrimiento no puede abordar una descripción hRESTS del mismo modo que una descripción WSDL; es por esta razón que es necesaria una adaptación de dicho algoritmo, cuyo proceso se explicará más adelante.

A pesar de que hRESTS soporta algunas características propias de REST, mencionadas en el anterior párrafo, no considera una característica fundamental en los servicios Web RESTful: el soporte para que cada recurso se pueda serializar en múltiples representaciones, a través de una negociación del tipo de contenido y formato con el servidor. De esta manera, el hecho de que hRESTS no tenga esta característica nos motivó a realizar un aporte en este sentido. En este orden de ideas, *se propone adicionar dos propiedades al modelo de servicio hRESTS original* (mostrado en la Figura 11). Estas propiedades las hemos llamado *Payload Format* y *Content Type* y su explicación se da a continuación:

Payload Format: hace referencia al formato en el cual el servidor codifica la respuesta enviada al cliente cuando recibe una petición; dándole la opción la cliente de seleccionar la que más se adapte a sus necesidades. Esta propiedad puede tener dos valores (usualmente los más comunes en los servicios REST): XML o JSON. Valga como ejemplo el caso en que el cliente desea que la respuesta esté codificada en JSON ya que el parser que posee sólo procesa las respuestas codificadas en éste estándar.

Content Type: hace referencia al tipo de contenido enviado como respuesta al cliente cunado recibe una petición. En teoría, su valor puede ser cualquiera de los tipos de media MIME[17] (debido a que los servicios REST pueden incluir en su respuesta cualquier tipo de dato), peros los más comunes son: text/plain, text/html, application/pdf, o image/jpeg. De esta manera, el cliente en la petición puede negociar el tipo de contenido que desea que el servicio le retorne en la respuesta.

Teniendo en cuenta lo anterior, en la Figura 25 se muestra el modelo de servicio de hRESTS extendido con las dos propiedades definidas en este trabajo de grado. Como se puede apreciar, el espacio de nombres para la definición del modelo de servicio de hRESTS ahora apunta a un archivo alojado localmente, que contiene la extensión propuesta. Adicionalmente, se aprecia que las propiedades se definen para cada operación, ya que cada una pues tener diferentes representaciones. El modelo extendido está disponible en formato rdf/xml, en xml, en texto plano y en notación n3. En la Figura 26 se aprecia una vista del espacio de nombres del modelo de servicio hRESTS extendido accedida desde un navegador Web, en donde se presentan los links a las diferentes versiones del modelo.

Las dos nuevas propiedades definidas para hRESTS, como se aprecia en la Figura 25 se especifican para cada operación en la descripción del servicio, ya que cada una puede retornar un tipo de contenido diferente, o tener un formato distinto.

[17] Disponibles en: http://www.iana.org/assignments/media-types/index.html

```
@prefix nhr:  <http://localhost:8080/hrests/#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix wsl:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

# hRESTS properties added to the WSMO-Lite model (copied below)
nhr:hasAddress  a  rdf:Property ;
    rdfs:domain  wsl:Operation ;
    rdfs:range  nhr:URITemplate .
nhr:hasMethod  a  rdf:Property ;
    rdfs:domain  wsl:Operation ;
    rdfs:range  xsd:string .

#hRESTS extensions proposed
nhr:hasPayloadFormat a rdf:Property ;
    rdfs:domain a wsl:Operation ;
    rdfs:range a xsd:string .

nhr:hasContentType a rdf:Property ;
    rdfs:domain a wsl:Operation ;
    rdfs:range a xsd:string .

# a relevant subset of the classes and properties of the WSMO-Lite minimal service model,
# copied here for completeness
wsl:Service  a  rdfs:Class .
wsl:hasOperation  a  rdf:Property ;
    rdfs:domain  wsl:Service ;
    rdfs:range  wsl:Operation .
wsl:Operation  a  rdfs:Class .
wsl:hasInputMessage  a  rdf:Property ;
    rdfs:domain  wsl:Operation ;
    rdfs:range  wsl:Message .
wsl:hasOutputMessage  a  rdf:Property ;
    rdfs:domain  wsl:Operation ;
    rdfs:range  wsl:Message .
wsl:Message  a  rdfs:Class .

# a datatype for URI templates
nhr:URITemplate  a  rdfs:Datatype .
```

Figura 25. Modelo de servicio HRESTS extendido con las propiedades definidas.

Figura 26. Modelo de servicio extendido de hRESTS propuesto para el desarrollo de este trabajo

4.1.3. Descripción de los servicios con el formato hRESTS extendido

A partir del modelo de servicio extendido de hRESTS, se describieron los servicios Web RESTful disponibles en la colección de prueba utilizada para la evaluación del prototipo funcional desarrollado en este proyecto, y que se presenta en el Capítulo 5. A todos los servicios disponibles en esta colección de prueba se les añadió, de forma manual y aleatoria, las nuevas propiedades hRESTS propuestas en la sección anterior con fines evaluativos. En el Capítulo 5 se profundiza en este aspecto.

4.2. Adaptación del algoritmo de descubrimiento

El algoritmo de descubrimiento definido en este trabajo se basa en los presentados en (Lampe, Schulte et al. 2010) y (Bellur and Kulkarni 2007). Nuestro algoritmo utiliza algunos componentes presentados en los trabajos mencionados anteriormente, y adiciona otros, con el fin de dar solución a las necesidades planteadas en el planteamiento del problema de este trabajo. Hay que tener presente que, como mencionamos anteriormente, aunque *hRESTS* modela los servicios Web de manera muy similar a WSDL, tiene algunas propiedades especificas propias de REST que hacen que los algoritmos de descubrimiento para servicios Web SOAP no puedan ser directamente aplicados en servicios descritos en este formato. De ahí que es necesario un proceso de adaptación de dichos algoritmos. A continuación se presenta un resumen explicativo de cada algoritmo presentado en ambos trabajos:

- El algoritmo de (Lampe, Schulte et al. 2010) está basado en una noción introducida en (Schulte, Lampe et al. 2010) llamada “emparejamiento centrado en las operaciones” (*operations-focused matching*), mediante la cual los autores expresan que las operaciones proporcionan la funcionalidad necesaria que un consumidor de un servicio busca. De esta forma, valores de similitud de todos los niveles de abstracción en un servicio (niveles de servicio, operación, entradas y salidas) son agregadas al nivel de operación. Luego, pares de operaciones son comparadas, independientemente de cómo éstas están organizadas (Lampe, Schulte et al. 2010; Schulte, Lampe et al. 2010).

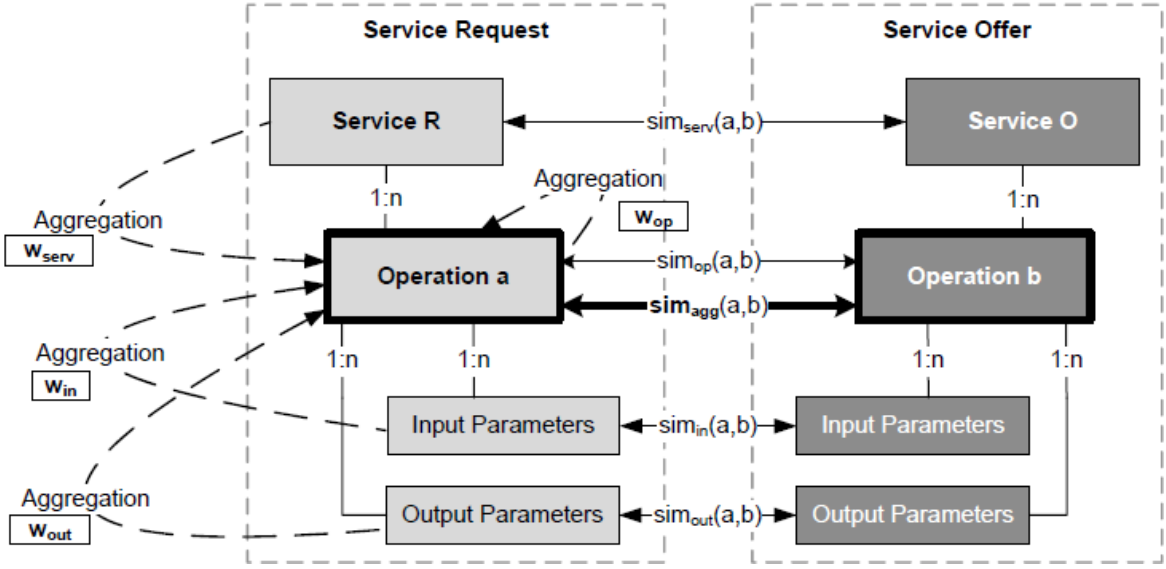


Figura 27. Proceso de emparejamiento para el *matchmaker* XAM4SWS (Lampe, Schulte et al. 2010)

En la Figura 27 se muestra el proceso de emparejamiento para el prototipo XAM4SWS propuesto en (Lampe, Schulte et al. 2010) y que será utilizado en este trabajo, susceptible a una adaptación, en donde para cada par de operaciones en la petición del servicio (*service request*) y la oferta del mismo (*service offer*) se determina la similitud para los respectivos niveles de entrada (sim_{in}), salida (sim_{out}) y operaciones nativas (sim_{op}). Posteriormente se calcula una similitud agregada (sim_{agg}) para cada par de operaciones, la cual es producto de la combinación de las similitudes individuales, multiplicadas por unos factores w_{op}, w_{in}, w_{out} (un factor de peso para cada similitud). Para la implementación de este algoritmo, y por aspectos como la complejidad y el tiempo definido en el proyecto, la similitud a nivel de servicio no se considera. Así,

$$w_{op} + w_{in} + w_{out} = 1 \qquad (ecuación\ 1)$$

Y,

$$sim_{agg}(a,b) = sim_{op}(a,b) * w_{op} + sim_{in}(a,b) * w_{in} \qquad (ecuación\ 2)$$

$$+sim_{out}(a,b) * w_{out}$$

Igualmente, la similitud completa de servicio (sim_{ov}) está dada por:

$$sim_{ov}(R,O) = \frac{1}{|A|} \sum_{a \in A, b \in B} x_{ab} * sim_{agg}(a,b) \qquad (ecuación\ 3)$$

Donde,

$A\ y\ B$: conjuntos de operaciones en la petición del servicio R, y en la oferta del servicio O respectivamente.

x_{ab}: variable binaria que indica si $a \in A$ ha sido emparejado con $b \in B$.

En el trabajo de (Lampe, Schulte et al. 2010), el proceso de emparejamiento para un conjunto de componentes (operaciones, entradas, salidas) es manejado utilizando grafos bipartitos. Los conjuntos de componentes de la petición y la oferta de servicio constituyen cada uno una partición de nodos en el grafo. Cada nodo en la primera partición es conectada a cada nodo en la segunda partición a través de una arista con un peso asignado (*weighted edge*). El peso corresponde a la similitud entre los dos componentes. De acuerdo al proceso de emparejamiento, los pesos de arista de todas las aristas emparejadas o comparadas son sumados y divididos por la cardinalidad de los conjuntos de componentes originales. Es probable que la cardinalidad de los conjuntos sea diferente, ya que un servicio puede realizar la misma funcionalidad pero con menos entradas que el servicio requerido. En ese caso, en general la cardinalidad del conjunto de componente de petición del servicio es decisiva. No obstante, en el caso de las entradas del servicio, la cardinalidad del conjunto de componentes de la oferta de servicio es determinante; así, si un servicio candidato requiere o le faltan algunas entradas o salidas con respecto al servicio pedido, su similitud total disminuye. De esta manera no se excluye

ningún servicio candidato cuyas entradas o salidas no coincidan exactamente con las del servicio requerido; en lugar de eso, estas posibles ofertas son "castigadas" con una reducción en la similitud.

- En el trabajo de (Bellur and Kulkarni 2007) se presenta un algoritmo de emparejamiento semántico, que se ejecuta sobre servicios Web SOAP, y que realiza el proceso de emparejamiento utilizando para ello grafos bipartitos, dando solución así a los problemas identificados en (Paolucci, Kawamura et al. 2002) en donde se propone el algoritmo de emparejamiento semántico original. El pseudocódigo del algoritmo definido por los autores se muestra en la Figura 28.

Algorithm 2 search($Query$)

```
1:  Result = Empty List
2:  for each Advt in Repository do
3:     outMatch = matchLists(Query_out, Advt_out)
4:     inMatch = matchLists(Advt_in, Query_in)
5:     if (outMatch = Fail OR inMatch = Fail) then
6:        Skip Advt. Take next Advt.
7:     else
8:        Result.append(Advt, outMatch, inMatch)
9:     end if
10: end for
11: return sort(Result)
```

Algorithm 3 matchLists($List_1$, $List_2$)

```
1:  Graph G = Empty Graph (V_0 + V_1, E)
2:  V_0 ← List_1, V_1 ← List_2
3:  (w_1, w_2, w_3) ← computeWeights(|V_0|)
4:
5:  for each concept a in V_0 do
6:     for each concept b in V_1 do
7:        degree = match(a, b)
8:        if degree ≠ Fail then
9:           Add edge (a, b) to G
10:          if (degree = Exact) then w(a, b) = w_1
11:          if (degree = Plugin) then w(a, b) = w_2
12:          if (degree = Subsume) then w(a, b) = w_3
13:       end if
14:    end for
15: end for
16:
17: Graph M = hungarianMatch(G)
18: if (M = null) then
19:    No complete matching exists. return Fail.
20: end if
21:
22: Let (a, b) denote Max-Weight Edge in G
23: degree ← match(a, b)
24: return degree
```

Figura 28. Algoritmos de emparejamiento definidos en (Bellur and Kulkarni 2007)

En la Figura 28 se aprecia el procedimiento principal del algoritmo definido en ese trabajo, el cuál recibe como consulta un servicio Web descrito a través de

WSDL y retorna el conjunto de servicios que más se asemejan a éste. Adicionalmente se hace un doble llamado al procedimiento *matchLists* que se encarga de determinar el grado de correspondencia entre dos conjuntos de conceptos especificados en una ontología de dominio; este grado de correspondencia se enmarca en una escala que se aprecia en la Figura 29. En síntesis, si las dos salidas hacen referencia al mismo concepto semántico, el grado de correspondencia es *exact;* si la salida de la consulta superpone o subsume a la salida de la oferta, el grado de correspondencia se clasifica como *plugin;* si la salida de la oferta subsume a la de la consulta el grado de correspondencia se clasifica como *subsume;* y en otro caso, el grado de correspondencia se clasifica como *fail,* ya que no existiría similitud entre ambos conceptos. El procedimiento para el emparejamiento de las entradas es análogo.

Algorithm 1 PROCEDURE match(outA, outQ)

```
1:  if outA = outQ then
2:      return Exact
3:  else if outQ SuperClass of outA then
4:      return Plugin
5:  else if outQ Subsumes outA then
6:      return Plugin
7:  else if outA Subsumes outQ then
8:      return Subsumes
9:  else
10:     return Fail
11: end if
```

Figura 29. Procedimiento para determinar en grado de similitud en (Bellur and Kulkarni 2007)

Como se mencionó anteriormente, el algoritmo implementado en nuestro trabajo de grado utiliza algunos componentes de los dos trabajos mencionados anteriormente. En síntesis, éstos son:

- Del trabajo de (Lampe, Schulte et al. 2010) se adoptaron los cálculos de las similitudes entre componentes, agregada y total, definidas dentro del proceso de emparejamiento que realizan los autores en el proyecto.
- Del trabajo de (Bellur and Kulkarni 2007) se adoptó el flujo principal de búsqueda (Figura 28), como esquema principal de nuestro mecanismo de descubrimiento.

Las modificaciones hechas a los componentes, y todos los detalles que cada fase del algoritmo de descubrimiento propuesto en nuestro trabajo de grado se presentan en la siguiente sección.

4.2.1. Descripción del algoritmo.

Inicialmente hay que aclarar que los componentes utilizados fueron adaptados para que se ajusten a las necesidades planteadas en este proyecto. Este proceso de adaptación, debido a que los algoritmos originales se utilizaban para servicios Web SOAP, trae consigo muy seguramente una pérdida en la exactitud. Es decir, que el hecho de ajustar los componentes para que se ejecuten sobre servicios Web REST trae la indeseable consecuencia de una pérdida en la precisión en el proceso de búsqueda de los servicios que más se acercan a los requerimientos del usuario. Sin

embargo, esta consecuencia no deseada está medianamente compensada a través de los módulos de filtrado y ordenamiento de servicios, en donde se priorizan los servicios que más se adapten al dispositivo desde el cuál se hace la petición. En la Figura 30 se muestra un diagrama en bloques del algoritmo adaptado y a continuación se presenta una explicación de cada uno de los componentes.

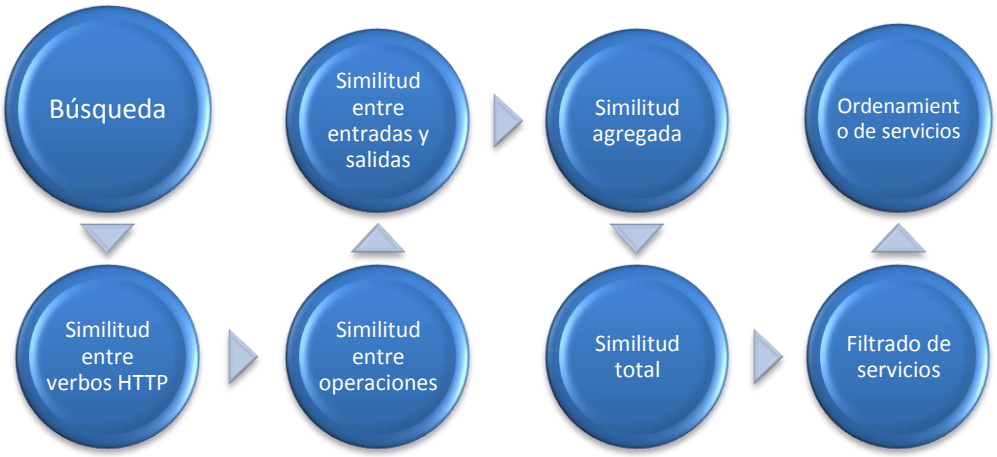


Figura 30. Diagrama en bloques del algoritmo de descubrimiento adaptado

- ***Búsqueda:*** es el flujo principal del algoritmo de descubrimiento, cuyo pseudocódigo se presenta a continuación. El algoritmo 1 conforma el flujo principal de nuestro mecanismo de descubrimiento. En él se recibe una consulta, se recorren todas las operaciones de todos los servicios ofrecidos en el repositorio (Advt) y el resultado es una lista de los servicios ordenados que satisfacen esa consulta. Adicionalmente, se invocan los diferentes algoritmos que se encargan de precisar la consulta y de clasificarlos teniendo en cuenta el contexto de entrega y las nuevas propiedades definidas para *hRESTS.* Este algoritmo fue adaptado teniendo en cuenta el propuesto en (Bellur and Kulkarni 2007) y que se presenta en la Figura 28. A éste se le adicionaron los procedimientos específicos que operan las descripciones *hRESTS* (*computeVerbSim, matchOperationName, matchLists, filterResult, sort*) obteniendo a partir de éstos las características propias de REST especificadas en la descripción de los servicios. Estos procedimientos se detallan en los siguientes párrafos.

Algoritmo 1

```
procedure search (Query)
Result = Empty List
FilteredResult = Empty List
for each Advt in Repository do
        for each Operation in Advt do
                verbSim = computeVerbSim (Query , Advt)
                opNMatch = matchOperationName (Query, Advt)
                outMatch = matchLists (Query_out, Advt_out)
                inMatch = matchLists (Advt_in, Query_in)
                if (opNMatch = Fail AND outMatch = Fail AND inMatch = Fail)
```

skip *Advt. Take next Advt.*
else
sim_{agg} = outMatch * w_{out} + inMatch * w_{in} + opNMatch * w_{op}
sim_{agg} = verbSim * sim_{agg}
end if
end for
servMatchOv = ($\sum sim_{agg}$)/|Query|
Result.*append*(Advt, servMatchOv)
end for
FilteredResult = ***filterResult***(Query,Result)
return sort(FilteredResult)

- ***Similitud entre verbos HTTP (computeVerbSim):*** en este procedimiento se aprovecha la característica propia de REST, implementada en las descripciones *hRESTS,* la cual especifica el método HTTP con el cuál se debe invocar el recurso que se desea acceder. Así, comparando el método HTTP del servicio ofertado presente en su descripción *hRESTS* con el de la petición, se puede calcular una similitud entre la petición y la oferta. Este cálculo se realiza como se observa en la Tabla 7 y su ponderación se utiliza para establecer un grado de similitud total entre la petición y el posible servicio candidato, ya que un recurso de un servicio que deba invocarse con el método HTTP GET no sería compatible con una petición que use el método HTTP PUT.

		Request Verb			
Offer verb		POST	GET	PUT	DELETE
	POST	1	0.8	0.5	0
	GET	0.8	1	0.2	0
	PUT	0.5	0.2	1	0
	DELETE	0	0	0	1

Tabla 7. Valores de similitudes entre los verbos HTTP

- ***Similitud entre operaciones:*** en este procedimiento se calcula la similitud entre dos operaciones haciendo uso de WordNet, de la manera especificada en (Lin 1998). El valor de similitud se encuentra en el rango de 0 a 1, en donde 0 significa que no existe ningún tipo de similitud y 1 significa que existe una similitud total. La similitud entre dos conceptos se calculó de la siguiente manera:

$$sim = \frac{2 * IC(lcs)}{IC(synset_1) + (synset_2)} \qquad (ecuación\ 4)$$

Donde:
lcs*:* concepto mínimo común por debajo del conjunto de sinónimos 1 y 2.
IC*:* función que retorna el contenido de información de un conjunto de sinónimos.
En (Lin 1998) y como se refleja en la ecuación 4, la similitud se obtiene utilizando el contenido de información (*information content*) en WordNet. Específicamente, se escala el contenido de información del *lcs* con el contenido de información de cada concepto individualmente. La aproximación

especificada por los autores se basa en la *Hipótesis de Distribución de Harris*, la cuál afirma que: "las palabras que acontecen en contextos iguales tienen significados similares". Así, los autores manejan la hipótesis de que "si dos caminos pertenecientes a árboles de dependencias de dos textos tienden a unir conjuntos iguales de palabras, los significados de ambos textos son similares" (Herrera de la Cruz 2005). El pseudocódigo de este procedimiento se detalla a continuación:

Algoritmo 2

```
procedure matchOperationName (Query , Advertisement)
        opNMatch = computeWordNetSim(QueryName, AdvertisementName)
return opNMatch
```

- ***Similitud entre entradas y salidas:*** en este procedimiento se calcula la similitud entre las entradas y salidas de los servicios candidatos. Se reciben como entradas dos listas de conceptos y se retorna la similitud total entre las dos listas como el promedio de las similitudes individuales. El esquema de este procedimiento se adaptó del trabajo presentado en la Figura 28; sin embargo, como se aprecia en el algoritmo 2, su implementación es significativamente diferente, ya que en el trabajo de (Bellur and Kulkarni 2007) se emparejaban los servicios haciendo uso de grafos bipartitos utilizando ontologías de dominio. Nosotros descartamos esta opción ya que el uso de grafos bipartitos incrementa el tiempo de procesamiento de las peticiones, lo cual es inviable ya que en el marco de nuestro trabajo el mecanismo de descubrimiento debía ser ejecutado desde un dispositivo móvil. Además, en nuestro trabajo no se tienen en cuenta ontologías de dominio ya que es muy difícil que una sola ontología agrupe los conceptos pertenecientes a un dominio general al cuál pertenecen los servicios Web REST; en cambio, se adicionó semántica al proceso a través del uso de WordNet, calculando la similitud como se muestra en la ecuación 4. A continuación se presenta el pseudocódigo de este procedimiento.

Algoritmo 3

```
procedure matchLists (List1, List2)
similarity =0
for each element a in List1 do
        for each element b in List2 do
                similarity = similarity + computeWordNetSim(a, b)
        end for
end for
return similarity/( List1+List2)
```

- ***Similitud agregada:*** la similitud agregada se calcula como la suma de las similitudes de operación, entradas y salidas, multiplicadas por unos pesos, como se observa en la ecuación 2. Esta similitud agregada es multiplicada por la similitud entre los verbos HTTP calculados según la Tabla 7, lo que hace que las peticiones en un servicio candidato a los recursos con un método HTTP

diferente al especificado en la descripción de dicho servicio serán "castigadas" con una reducción significativa en su similitud. Así, se está teniendo en cuenta la característica REST del uso de interfaz uniforme para acceder a los recursos del servicio.

- ***Similitud total:*** la similitud total entre dos servicios *servMatchOv* se calcula como la sumatoria de la similitud agregada de todas las operaciones de los mismos, dividida por la cardinalidad de la consulta, como se propone en (Lampe, Schulte et al. 2010). Como salida de este procedimiento se obtiene una lista inicial de servicios Web candidatos que satisfacen los requerimientos proporcionados por el cliente.

- ***Filtrado de servicios:*** en este procedimiento se tiene en cuenta la extensión propuesta en este trabajo de grado a la descripción *hRESTS* (*PayloadFormat* y *ContentType*). Se recibe en esta etapa una lista inicial de servicios Web candidatos; si el usuario no especificó ningún tipo de formato o tipo de contenido, la lista de resultados se retorna como estaba. Pero, si especificó que deseaba los servicios que cumplan con cierto contenido o formato, se retorna una lista (FilteredResult) con los servicios Web que cumplan con ello. De esta manera, se tiene en cuenta la característica propia de REST de permitir múltiples representaciones para cada recurso del servicio, y se le permite al cliente seleccionar la representación que más se adecúe a sus necesidades. El pseudocódigo de este procedimiento se presenta a continuación:

Algoritmo 4

```
procedure filterResult (Query, Result)
        FilteredResult = Empty List
        payLoadFormat = getDesiredpayLoadFormat (Query)
        contentType = getDesiredContentType (Query)
if (payLoadFormat = null AND contentType = null) then
        return Result
else
        for each Advt in Result do
                payLoadFormatA = getPayLoadFormat (Advt)
                contentTypeA = getContentType (Advt)
                if (payLoadFormatA = payLoadFormat OR contentTypeA = contentType) then
                        FilteredResult.append(Advt)
        end for
        return FilteredResult
end if
```

- ***Ordenamiento de servicios:*** finalmente, en este procedimiento se ordenan los servicios teniendo en cuenta el contexto de entrega del usuario. Es decir, que se presentarán al usuario prioritariamente los servicios que mejor se adapten a las características del dispositivo desde el cuál se realiza la petición. En el prototipo de prueba realizado, y debido a que la finalidad es demostrar que es posible utilizar características del dispositivo móvil para seleccionar los servicios Web que mejor se adapten al mismo, se utilizó una sola característica del dispositivo: el tamaño de la pantalla. Sin embargo es posible implementar el ordenamiento de los servicios utilizando cualquier otra característica de

dispositivos móviles (como la conexión, o la capacidad de procesamiento, o el sistema operativo) modificando sólo este módulo del algoritmo. Así que, en esta etapa se clasifican los servicios dándole prioridad a los que mejor se adapten al tamaño de la pantalla del dispositivo móvil. El pseudocódigo de este procedimiento se presenta en el algoritmo 5.

Algoritmo 5

---

procedure *sort* (FilteredResult, UserAgent)
DeviceRequirements = *getDeviceRequirements* (UserAgent)
return *classifyServices* (FilteredResult, DeviceRequirements, ServicesRequirements)

Los componentes implementados que definen el algoritmo de descubrimiento propuesto en este trabajo de grado permiten realizar el descubrimiento semántico (a través del cálculo de la similitud entre conceptos haciendo uso de WordNet, como se muestra en la ecuación 4) de servicios Web RESTful (descritos en *hRESTS*) y adicionalmente tiene en cuenta una característica del dispositivo desde el cuál se realiza la petición.

4.2.2. Aportes del algoritmo de descubrimiento adaptado

En esta sección se presentan explícitamente los aspectos que diferencian a este mecanismo de descubrimiento de los algoritmos presentados en los trabajos relacionados. De esta manera, los algoritmos definidos se diferencian en los siguientes aspectos:

- La adición de las propiedades *Content type y Payload format*, que obviamente son tenidas en cuenta en el mecanismo de descubrimiento de servicios (específicamente en el algoritmo 4 y en el procedimiento de filtrado de servicios), hacen que éste sea único, y que ofrezca la posibilidad de filtrar los servicios por el tipo de contenido que retornan. Obviamente los algoritmos definidos en los trabajos relacionados no tienen en cuenta esta propiedad. Sin embargo, se tiene conciencia de que proponer extensiones a un lenguaje de descripción es algo arriesgado ya que existe la posibilidad de que el modelo de servicio extendido no sea adoptado por la comunidad. No obstante, creemos que la realización de esta propuesta dentro del proyecto de grado llevado a cabo es una oportunidad ideal para divulgarla a la comunidad relacionada con el tema y fomentar una discusión al respecto.
- En nuestro mecanismo de descubrimiento es posible también filtrar los servicios automáticamente según el dispositivo desde el cuál se accede (utilizando para este fin la propiedad del tamaño de la pantalla del dispositivo desde el cuál se realiza la petición). Es decir, se tiene en cuenta el contexto de entrega en entornos móviles. Los algoritmos en los trabajos relacionados no tienen en cuenta el entorno móvil. El aporte de nuestro algoritmo en este sentido es sólo con respecto a los trabajos relacionados (a la fecha de la realización de la base de conocimiento no se encontraron proyectos que trabajaran el contexto de entrega sobre servicios Web RESTful) **y NO sobre la técnica**, ya que existen diversos trabajos que ya han abordado el tema del contexto de entrega e incluso de personalización sobre servicios Web SOAP.

- En este trabajo se tiene en cuenta el método HTTP utilizado para realizar la petición al recurso y el especificado literalmente en la descripción *hRESTS* para determinar el grado de correspondencia entre la petición realizada por el usuario y los servicios Web disponibles en el repositorio.
- En los trabajos relacionados, en el proceso de adición de semántica al descubrimiento de servicios, y debido a que no se tiene en cuenta el entorno móvil, se especificaban algoritmos de descubrimiento complejos que demandaban bastante procesamiento. En este trabajo, sólo se utilizó WordNet para añadir semántica (se descartó el uso de ontologías de dominio y de grafos bipartitos para el proceso de emparejamiento), lo que se traduce en tiempos de respuesta aceptables para dispositivos móviles; como se especifica en el capítulo 5. El cálculo de distancias semánticas de conceptos en una ontología o incluso el proceso mismo de cargar las ontologías sobrellevaría un incremento bastante considerable en los tiempos de respuesta (ya que para los servicios de prueba en los trabajos relacionados se cargaban 38 ontologías de dominio, muchas de ellas con muchísimos conceptos) lo cuál es inviable en entornos móviles.

4.3. Descripción de la plataforma "*What do you do?"*

En este capitulo se presenta el resultado de la investigación representado en una plataforma que permite al usuario o agente que la usa recuperar una lista de servicios ordenados que cumplen con los requerimientos solicitados, por este motivo hemos decidido llamar "*What do you do?"* a la aplicación que permite esto.

La plataforma permite el descubrimiento semántico de servicios Web RESTful alojados en un repositorio haciendo uso del diccionario lexico WordNet, además permite ampliar la consulta con el uso de Wordnet y meta datos en la descripción de los servicios para finalmente ordenar el resultado considerando el contexto de entrega, el cual es determinado haciendo uso de WURFL y un data model del que se obtiene los requerimientos mínimos para que un servicio pueda ser consumido eficientemente.

Tener en cuenta el contexto de entrega hace referencia a considerar tanto las condiciones temporales, espaciales y características especiales del dispositivo desde donde se consumirán los servicios; la plataforma solo tiene en cuenta las condiciones del dispositivo desde donde se realiza la consulta, aunque esto no quiere decir que no se puedan desarrollar módulos que se integren a los ya implementados y que permitan considerar los aspectos restantes (temporal y espacial). La plataforma obtiene características del dispositivo desde donde llega la solicitud de búsqueda de servicios, tales como ancho de banda, dimensión de la pantalla, software y versión, entre otras características, para que sirva como un parámetro para ordenar los servicios que son recuperados, por ejemplo, si un usuario realiza una petición desde un celular con limitaciones importantes en su pantalla, se ubicarán de primero los servicios que puedan ser consumidos eficientemente en su dispositivo teniendo en cuenta su limitación en este aspecto.

Para el desarrollo de la plataforma se siguió el patrón recomendado en Modelo de construcción de Soluciones (Serrano Cartaño 2008) el cual propone que sea un desarrollo iterativo e incremental, previo a un análisis y construcción de una arquitectura inicial.

4.3.1. Establecimiento de responsabilidades

Lo primero es determinar la viabilidad y el alcance del proyecto, y para esto se define a continuación unas consideraciones iniciales.

4.3.1.1. Consideraciones iniciales

Basado en la investigación documental y análisis del estado de arte (principalmente trabajos relacionados identificados) se consideran los siguientes aspectos como importantes:

- **Contexto.**

Se tomó la decisión de considerar el contexto de entrega, específicamente las características del dispositivo móvil desde donde se solicitan los servicios, debido a que actualmente más personas utilizan su dispositivo móvil para sus labores diarias y a que poco a poco se están eliminando las restricciones propias de este medio (como la duración de batería, el almacenamiento, procesamiento, entre otras) lo que augura un futuro prometedor para este campo. Al considerar el entorno móvil, es sumamente importante determinar con precisión las características del dispositivo; realizando una investigación documental se decidió tomar a WURFL como la mejor opción para recuperar dichas características, ya que por su amplio uso por parte de la mayoría de compañías productoras de dispositivos móviles y desarrolladores, se puede considerar como estándar, además de que existe una amplia documentación sobre su uso. WURFL trabaja con un archivo XML que hace las veces de repositorio de dispositivos, en donde se encuentra de forma detallada los especificaciones técnicas de la mayoría de terminales del mercado; en caso de que el dispositivo no se encuentre en el XML original, WURFL permite expandirse mediante el uso de otro archivo auxiliar, que puede ser modificado a conveniencia siguiendo la sintaxis del original. Su funcionamiento es bastante sencillo: se basa en obtener el *user-agent* de la petición HTTP que llega y obtener las características del repositorio mediante el procesamiento de dicho campo por parte de una API.

El contexto puede ser usado para filtrar servicios, que aunque cumplen con los requerimientos del usuario o agente, no podrían ser utilizados en un dispositivo específico debido a sus restricciones en el hardware o software. Del mismo modo, se puede usar para ordenar los servicios de forma que en las primeras posiciones aparezcan los que pueden ser consumidos eficientemente por un dispositivo determinado.

Las alternativas diferentes a WURFL no fueron muchas, y la elección de WURFL no fue difícil ya que ha alcanzado el nivel de estándar, tanto de la industria, como de los desarrolladores.

- **Semántica.**

El descubrimiento de servicios se puede hacer de tres formas posibles: sintáctico, semántico y basado en comportamiento.

El descubrimiento sintáctico, aunque mas ágil, presenta algunas falencias causadas principalmente porque se basa en comparar la forma como se escribe la petición contra la descripción de los servicios, lo que podría permitir que se descarten servicios

que cumplan con los requerimientos del usuario o agente por no coincidir sintácticamente.

En cambio, el descubrimiento semántico permite ampliar la búsqueda al considerar el significado de las palabras. Basados en la investigación documental, se hace uso de WordNet para determinar la similitud de palabras y para ampliar la consulta inicial obteniendo sinónimos de dicha consulta y repitiendo el proceso de búsqueda por cada sinónimo; debido a que este proceso hace que la consulta sea más lenta, será una opción activada por el usuario; las pruebas realizadas a la plataforma muestran el cambio y eficiencia de cada búsqueda.

JAWS es la API utilizada para extender la consulta obteniendo sinónimos de la original.

- **REST**

La decisión de usar REST se da principalmente a que es una arquitectura que actualmente está tomando fuerza y que empresas importantes la están usando en sus plataformas para que usuarios consuman sus servicios. Además, en comparación con los servicios SOAP, el descubrimiento de servicios es de mayor valor investigativo debido a los pocos trabajos relacionados.

La aplicación realiza la búsqueda de servicios RESTful descritos en RDF, descripción que fue obtenida después de realizar la transformación de un conjunto de servicios descritos originalmente en hREST mediante el uso de un archivo XSLT que fué amablemente proporcionado por los doctores *Jacek Kopecky*[18] *y Ulrich Lampe*[19] y al cuál se le adicionaron los procedimientos necesarios para procesar las nuevas propiedades hRESTS propuestas en este proyecto.

4.3.2. Captura y Análisis de Requisitos Funcionales

4.3.2.1. Alcance del sistema

La plataforma permite el descubrimiento de servicios Web RESTful considerando algunas características del dispositivo móvil como el tamaño de su pantalla. Para realizar un ranking de los servicios recuperados se presenta al usuario una interfaz web para solicitar la búsqueda de un servicio y en esta misma presenta los resultados.

En caso de que la búsqueda se realice desde un agente, las clases son tan modulares que permiten exponer una interfaz para solicitar el servicio de búsqueda.

El uso de WordNet con JAWS es opcional y es el usuario quien decide usar este modulo para extender la búsqueda con sinónimos de la petición original. A través de la interfaz se puede activar la funcionalidad descrita anteriormente (mirar el Anexo A para obtener información sobre el uso de la interfaz).

En síntesis *What do you do?* es una aplicación web diseñada para ser accedida desde dispositivos móviles (pero esto no imposibilita que pueda ser accedida desde

[18] http://jacek.cz/
[19] http://www.kom.tu-darmstadt.de/en/kom-multimedia-communications-lab/people/staff/ulrich-lampe/

computadores de escritorio) que recupera las características del dispositivo desde donde es accedida. Ofrece una interfaz minimalista con opciones muy claras, en donde la elección principal es la búsqueda de un servicio. En esta, mediante un simple formulario se pide al usuario que ingrese cuatro parámetros de búsqueda: tipo de respuesta que se espera obtener (*content type*), formato en el cual se codifica la respuesta (*payload format*), si desea extender su consulta con sinónimos y la petición (compuesta de un verbo y un sujeto), siendo solo esta ultima de carácter obligatorio. De esta forma se crean solicitudes muy precisas como: rentar carro, pagar factura, dibujar círculo, etc.

A partir de la solicitud del usuario la aplicación realiza una búsqueda de los servicios que cumplen con lo que desea el consumidor. Los resultados son mostrados como una lista de servicios ordenados, y cada uno de éstos tiene asociado un enlace para ser accedidos.

La composición de servicios es un tema que esta investigación no aborda, pero el servicio puede ser tomado como base para poder realizarla.

4.3.2.2. Requisitos funcionales del sistema

Obtener características del dispositivo. Este requisito es fundamental si queremos ordenar los servicios bajo estas características. El sistema permite visualizar mediante una interfaz web dichas características, solo sirve como información para el usuario.

Obtener petición del usuario. Se realiza mediante un simple formulario que recoge lo que desea hacer el usuario para buscar el servicio que mejor se adapte para esto.

Obtener sinónimos de la petición. Esto sirve para expandir la consulta de tal forma que si un usuario quiere por ejemplo “rentar carro”, dicha consulta puede ser expandida a “alquilar automotor”, “arrendar auto” etc. Como se ha mencionado con anterioridad esta es una opción adicional ya que representa que los tiempos de respuesta aumenten.

Publicar servicios. Solo se presenta la forma en que se debe realizar, no se ofrece una interfaz para hacerlo.

4.3.2.3. Requisitos no funcionales del sistema

Interfaces agradables y de fácil navegación considerando que van a ser accedidas desde dispositivos móviles en donde se tienen limitaciones en el tamaño de la pantalla. Sin embargo, tampoco se puede abusar en el diseño de las buenas interfaces debido al poco procesamiento de estos dispositivos.

Por el momento no son tenidos en cuenta aspectos como la accesibilidad.

4.3.3. Identificación de casos de uso

La Figura 31 muestra los casos de uso del sistema.

Publish services: permite alimentar el repositorio de servicios RESTful. Este modulo no fué desarrollado ya que se adoptó el uso de Fuseki (Servidor SPARQL) y éste cuenta con una opción de subir los servicios a través de una interfaz web.

Get Device Information: recupera la información más relevante del dispositivo móvil desde donde se accede al servicio. Ésta es almacenada y usada para ordenar servicios teniendo en cuenta el contexto de entrega. Es indispensable obtener esta información. Solo se realiza una vez por cada ingreso al sistema.

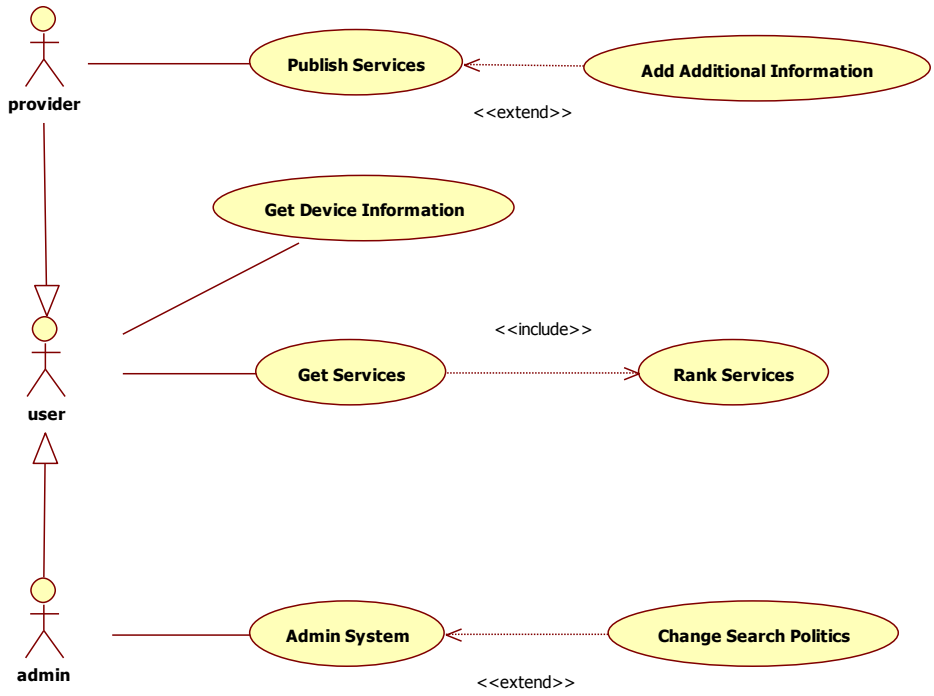


Figura 31. Casos de uso del sistema

Get services: toma la consulta o las consultas que se forman (a través de controles en la interfaz gráfica que le permiten al usuario seleccionar el tipo de contenido y el formato en el que desea la respuesta, y además configurar si desea ampliar la consulta original con sinonimos) y las ejecuta para obtener un conjunto de servicios respuesta. Este es el caso de uso principal del sistema.

Rank services: basado en las características del dispositivo se realiza un ordenamiento de los servicios ofreciendo al usuario una lista en donde las primeras posiciones ocuparan servicios que sean más eficientes para ser consumidos, considerando las limitaciones propias de cada dispositivo. También se discriminan los servicios por el tipo de respuesta que se desea obtener o el formato en el que se codifican, según lo haya dispuesto el usuario.

4.4. Descripción de la solución

A continuación de forma detallada se presenta el desarrollo de la herramienta que permite el descubrimiento semántico de servicios web RESTful teniendo en cuenta el contexto. Como se podrá apreciar, los servicios no necesariamente deben ser consumidos desde un dispositivo móvil, se puede hacer desde cualquier navegador (aunque se perdería las bondades adquiridas gracias a la detección de las características del dispositivo solicitante de servicios).

4.4.1. Diagrama de Clases del sistema

Este es el diagrama de clases de la aplicación. Algunas clases propias de las API usadas en el sistema no aparecen en este diagrama ya que, aunque importantes para la solución, no hacen parte fundamental del desarrollo realizado y simplemente se usaron como una herramienta.

Para explicar las clases de una forma mas clara se divide la explicación por funcionalidades destacadas del sistema.

- **Recuperación de la información del dispositivo**

  Agent: esta clase es la entidad que representa al dispositivo que esta accediendo a la aplicación y que quiere encontrar un servicio. Los atributos de ésta representan las características principales que se desean obtener del agente y que son usadas por el sistema para ordenar los servicios encontrados. Los métodos de esta clase son los que permiten fijar sus atributos o conocerlos desde otras clases (métodos *get y set*), y por este motivo no se muestran en el modelo. Esta clase hace uso del método *getCapability* de *Device* para obtener la información que necesita.
  Esta clase hace parte de la interfaz gráfica de usuario ya que de ésta se obtienen los atributos para ser desplegados en la interfaz (esto solo se usa para dar información sobre el dispositivo al usuario final).

  Device: es la clase que usa la API de WURFL para almacenar un dispositivo como un objeto Java. El método que el sistema usa de esta clase es *getCapability* que recibe como parámetro *Capability Name* y regresa la información solicitada.

  WURFLManager: hace parte de las clases de la API oficial de WURFL para Java y permite mapear una petición HTTP a *Device*, esto quiere decir que esta clase es la que procesa el *User-Agent* que se encuentra en los mensajes HTTP enviados.

  WURFLHolder: se utiliza como punto de entrada para las clases de la API de WURFL.

  WurflServlet: detecta cuando un mensaje HTTP llega a la aplicación, y a partir de este obtiene la IP del solicitante y *User-Agent*, el campo mas importante para que se pueda determinar las características del dispositivo mediante el uso de WURFL. Este servlet provee a las clases de la API de WURFL la información que requieren para crear una instancia de *Device* para que luego se pueda obtener las capacidades que la aplicación requiera.

- **Enriquecimiento de la consulta**

  JAWS: permite realizar consultas en la base de datos léxica WordNet, su parámetro más importante de configuración es *databaseDirectory* (que indica la ubicación de la base de datos WordNet). Su método más importante es

getSynonymous, este recibe como parámetros la palabra a la cual se le desea obtener los sinónimos, *synsetType* que se puede entender como el tipo de resultado que se desea obtener como verbos, nombres, adverbios, etcétera, y finalmente un booleano para fijar si se desea obtener resultados para palabras que se escriban similar a la que se envía como petición. El resultado es una lista de sinónimos que se presenta como un hashmap.

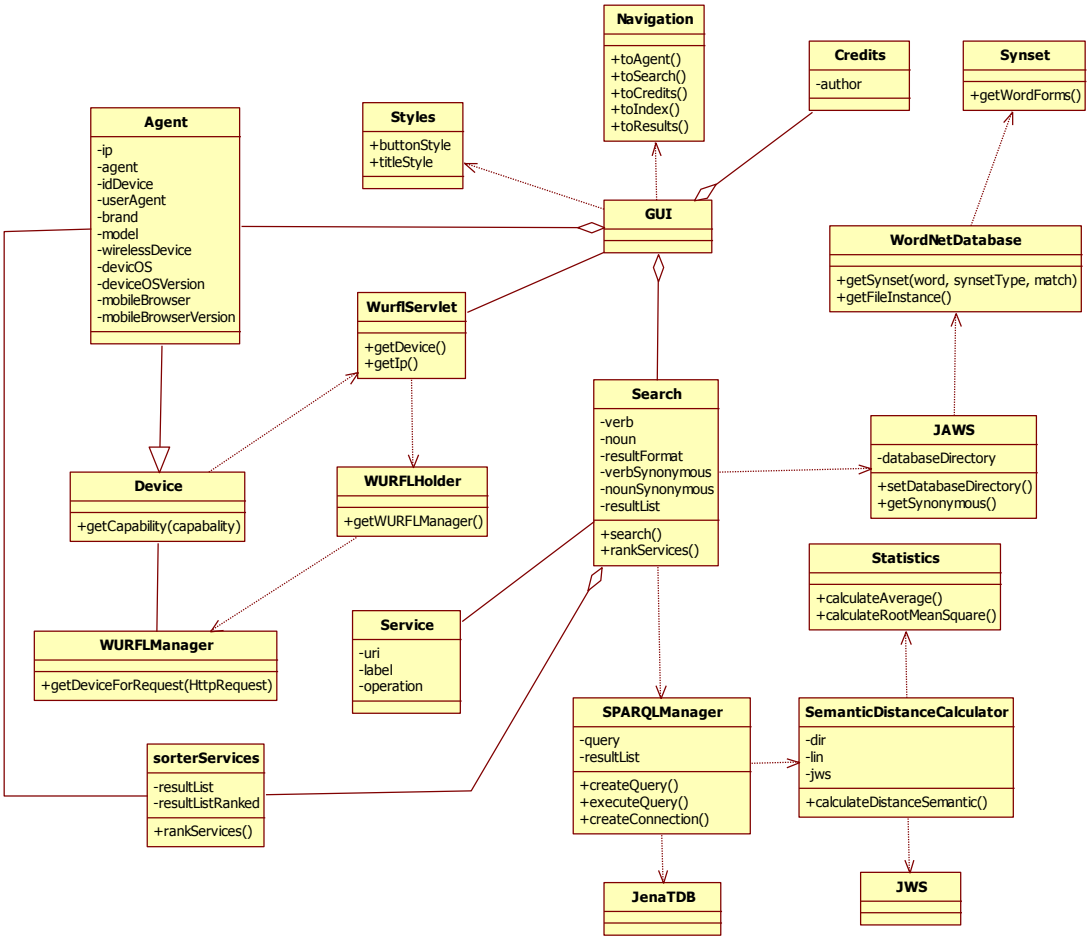


Figura 32. Diagrama de clases del sistema

WordNetDatabase: permite obtener una instancia de la base de datos WordNet para que se pueda operar sobre ésta y obtener la información que se necesite. El método principal para el sistema es *getSynsets,* que obtiene los sinónimos de la palabra.

Synset: es la entidad principal que usa JAWS, un *synset* son los sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos, es decir los sinónimos que se pueden obtener de WordNet.

- **Búsqueda del servicio**

Search: se conecta directamente con la interfaz grafica de usuarios porque esta clase es la encargada de obtener los parámetros que necesita para

realizar la búsqueda de los servicios. Se obtiene información como por ejemplo el tipo de respuesta que se desea obtener, el verbo y nombre que componen lo que el usuario desea hacer, por ejemplo “pagar facturas”, “alquilar automovil”. Esta clase, haciendo uso de una instancia de *JAWS* es capaz de obtener sinónimos de la consulta inicial, y haciendo uso de una instancia de *SPARQLManager* es capaz de crear la consulta para obtener los servicios y luego enviarlos a la clase *sorterServices* para que sean ordenados teniendo en cuenta la información del dispositivo desde donde se está accediendo al sistema.

SPARQLManager: esta clase se usa para generar la consulta SPARQL, ejecutarla y almacenar el resultado de ésta.

JenaTDB: implementa los métodos de las clases de la API de JenaTDB que permiten realizar consultas sobre el repositorio de servicios.

SemanticDistanceCalculator: calcula la similitud semántica entre dos palabras, que corresponden a la petición y a los posibles servicios. El cálculo de esta similitud se hace en base a una aproximación expuesta en (Lin 1998), en dónde se presenta un mecanismo para calcular la similitud entre dos palabras.

Statistics: clase auxiliar que realiza los cálculos matemáticos para el correcto funcionamiento de los métodos de *SemanticDistanceCalculator.*

- **Ordenamiento de los resultados**

  sorterServices: a partir de la lista de servicios recuperados se analizan las condiciones mínimas necesarias en cuanto a las características del dispositivo que se requiere para que el servicio funcione satisfactoriamente y se compara con la información almacenada en *Agent* para determinar si el dispositivo cumple con los requerimientos mínimos. Los servicios que sean más adecuados para consumir desde el dispositivo se ubicarán en la parte más alta del ranking. Finalmente se entrega a la interfaz gráfica la lista de los servicios ordenados.

- **Interfaz gráfica**

  GUI: implementa la lógica de presentación de toda la aplicación.

  Credits: almacena la información básica sobre los desarrolladores para que se puedan desplegar como una opción de un menú principal.

  Navigation: controla la navegación por la aplicación, permite navegar entre diferentes vistas como la que muestra la información del dispositivo, el formulario que recoge los datos para realizar la consulta de los servicios, los créditos y la interfaz de configuración para administradores.

Styles: controla el estilo de presentación de los elementos de las diferentes vistas, como el estilo de los botones, títulos, cajas de texto, entre otros elementos.

4.4.2. Diagrama de Paquetes del sistema

La Figura 33 muestra la distribución lógica de la aplicación en paquetes.

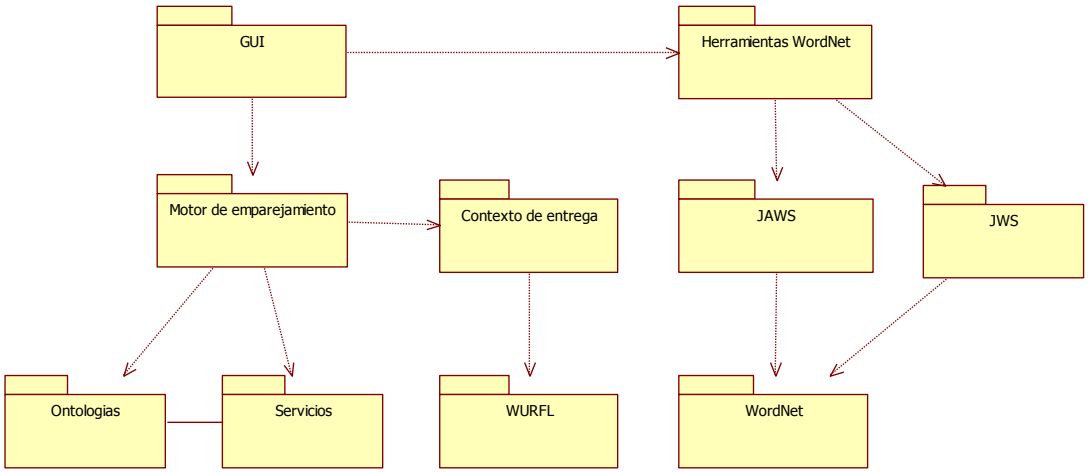


Figura 33. Diagrama de paquetes del sistema

GUI: agrupa las clases que se encargan de mostrar la interfaz al usuario a través de un navegador, controlar la navegación entre las diferentes vistas y llamar a los métodos de las diferentes clases cuando el usuario interacciona con las diferentes opciones del sistema. Además, se encarga de recuperar la información de los diferentes procesos (principalmente el de búsqueda) para dar a conocer al usuario sobre los resultados que tienen sus interacciones.

Herramientas WordNet: agrupa las clases que se encargan de procesar las peticiones de obtener sinónimos y similitudes de palabras. Para hacer esto, estas clases se valen de implementaciones de la API JAWS.

JAWS: agrupa a las clases que vienen dentro de la API del mismo nombre y se utilizan principalmente para obtener sinónimos de palabras (*synsets*).

WordNet: agrupa las clases que se encargan de la configuración de la conexión a la base de datos léxica WordNet, y también las encargadas de procesar y ejecutar las peticiones que lleguen desde la API JAWS.

Motor de emparejamiento: agrupa las clases encargadas de generar la búsqueda de los servicios y crear la consulta formal; el núcleo de estas clases es el algoritmo de búsqueda. Este paquete hace uso de otros para mejorar sus funcionalidades, como el de contexto de entrega que se usa para ordenar los servicios.

Servicios: agrupa las clases encargadas de configurar el enlace con el repositorio de servicios, de ejecutar las sentencias de búsqueda y de presentar los resultados.

Contexto de entrega: agrupa las clases encargadas de determinar las características principales del dispositivo de entrada haciendo uso del repositorio de dispositivos WURFL. Además, se encargan de ordenar los servicios de tal forma que aparezcan primero los que son más idóneos para ser consumidos desde un dispositivo teniendo en cuenta una serie de restricciones propias de un entorno móvil.

WURFL: agrupa las clases encargadas de la configuración de la conexión al repositorio de dispositivos, así como de determinar con exactitud el dispositivo móvil desde el cual se esta realizando la petición de búsqueda de servicios.

4.5. Definición de la arquitectura

Aunque *"What do you do?"* es una forma de implementar la arquitectura mostrada en la Figura 34, se podrían realizar variantes al seleccionar los componentes de la arquitectura.

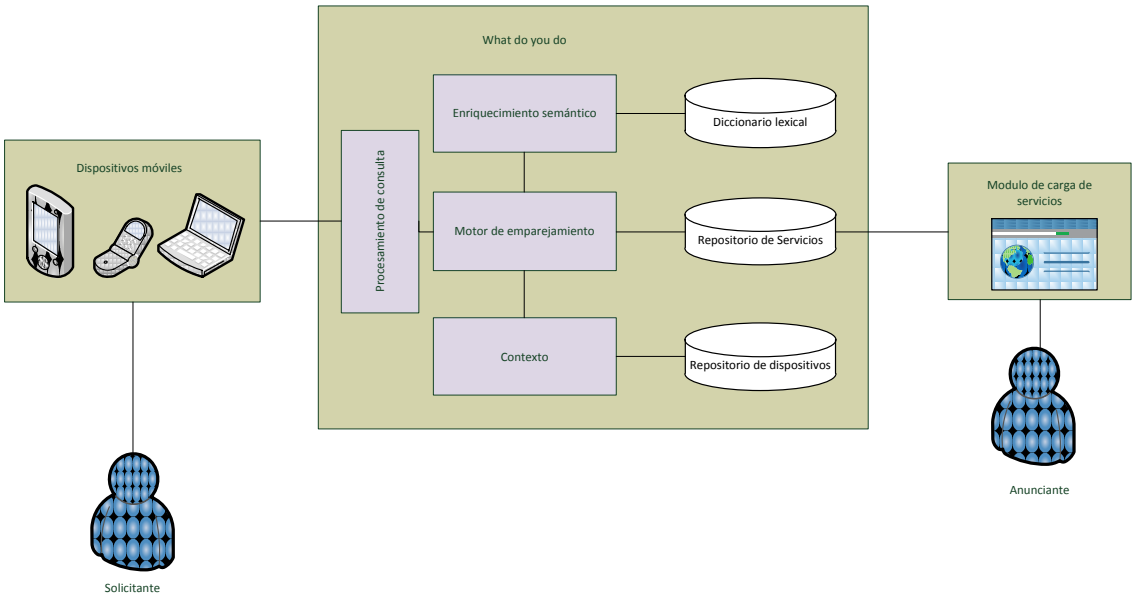


Figura 34. Arquitectura de referencia del sistema

A continuación se describe cada uno de los módulos que componen la arquitectura de referencia:

Procesamiento de consulta: en este módulo se procesa la solicitud en lenguaje natural de un usuario y se transforma a una consulta en lenguaje formal.

Motor de emparejamiento: en este módulo se procesa el requerimiento formal del solicitante, y se realizan los procesos necesarios para encontrar los servicios que mejor cumplan con las expectativas del usuario. Se podría considerar como un modulo central ya que éste hace uso de otros módulos para completar su funcionalidad, como el modulo de contexto de entrega que se usa para ordenar los servicios o el modulo de enriquecimiento semántico que puede ser empleado para extender la consulta; por este motivo, en este modulo se concentra el mayor procesamiento del sistema.

Repositorio de servicios: en este se almacenan los servicios que están disponibles para ser recuperados. Además, provee un mecanismo para que se puedan almacenar nuevos servicios por parte de un anunciante o Publisher.

Repositorio de Dispositivos: en este modulo se almacena la información sobre el mayor número de dispositivos del mercado.

Contexto: es el encargado de determinar las características del dispositivo que esta accediendo al sistema, esto lo hace mediante un procesamiento que involucre al repositorio de dispositivos.

Diccionario léxico: almacena el significado de palabras, relaciones con otras, sinónimos, antónimos entre otras relaciones lexicales.

Enriquecimiento semántico: este módulo se encarga de obtener los sinónimos y similitudes de palabras, puede ser usado para ampliar una consulta simple.

4.5.1. Diagrama de Despliegue de la Plataforma

La Figura 35 presenta la configuración adoptada que se basó en la arquitectura de referencia para crear un sistema capaz de descubrir servicios Web RESTful de forma semántica y considerando el contexto, además del hardware asociado a la solución.

El funcionamiento básico del sistema es el siguiente:

Un solicitante a través de un dispositivo móvil accede al servidor de descubrimiento a través de un browser haciendo uso del protocolo HTTP. El servidor de descubrimiento tiene toda la lógica del negocio y le presenta al usuario las diferentes funcionalidades del sistema en forma de un menú de opciones:

- *Search:* recupera la solicitud del usuario a través de un formulario.
- *Device Information:* despliega la información del dispositivo que está accediendo a la aplicación. Esta información es recuperada haciendo uso de WURFL.
- *Credits:* despliega información sobre los desarrolladores.
- *Admin/Publisher:* sin implementar.

El servidor de descubrimiento se encarga de obtener las características del dispositivo que está accediendo al sistema mediante el uso de la API de WURFL, el repositorio de dispositivos de WURFL y el *User-Agent;* éste último es obtenido de las cabeceras del mensaje HTTP que llega desde el dispositivo. Dichas características son mapeadas a una clase entidad desde la cuál se podrán recuperar para diferentes finalidades.

Cuando un usuario realiza una solicitud de búsqueda (a través de un formulario), el servidor de descubrimiento crea una consulta formal en SPARQL para ejecutarla sobre el repositorio de servicios. Adicionalmente a esto, se pueden obtener variaciones de la consulta original haciendo uso del diccionario léxico WordNet y JAWS. Los resultados que se obtienen con la consulta son mapeados a clases entidad y almacenados como un arreglo de objetos, un servicio es considerado como candidato solución si el algoritmo que determina la similitud entre la solicitud y los métodos del serivicio asi lo determina, y para esto se debe alcanzar un nivel de similitud de 0,5 en una escala en donde 0 indica que los términos no son nada similares y 1,0 que indica que los términos son idénticos.

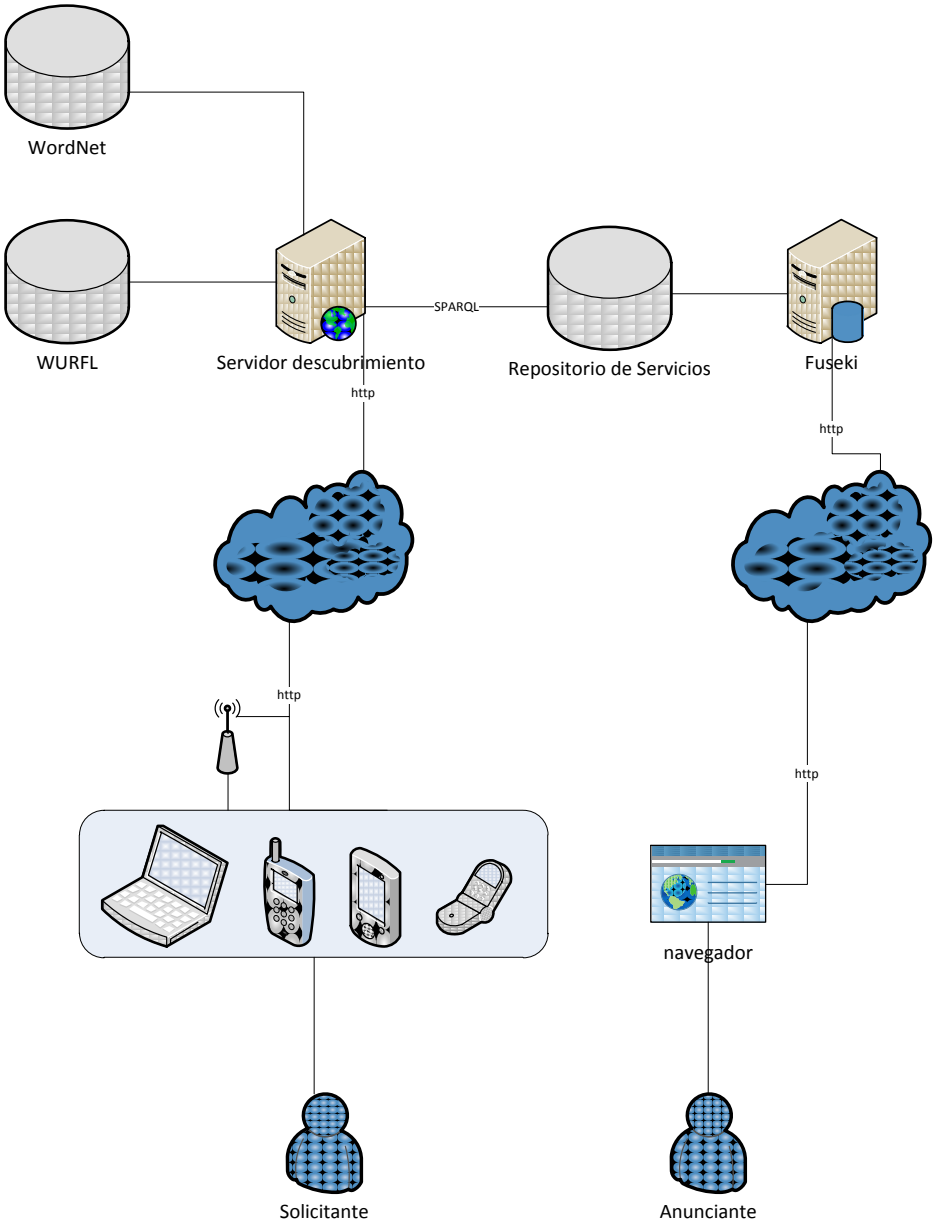


Figura 35. Diagrama de despliegue del sistema

Los servicios recuperados son ordenados teniendo en cuenta las capacidades del dispositivo solicitante, de tal forma que ocupan los primeros puestos de una lista los servicios que son más aptos para ser consumidos por el dispositivo. Para hacer esto se recupera la información del dispositivo que fue almacenada cuando el usuario ingresó por primera vez al sistema y se compara con un modelo de datos que contiene información sobre los requerimientos mínimos que cada servicio demanda para ser consumido con satisfacción. Para el sistema desarrollado solo se consideró procesar la información sobre las dimensiones de pantalla óptimas para que un servicio pueda ser desplegado correctamente. De esta forma, los servicios recuperados que se ajustan a las dimensiones del dispositivo de forma mas adecuada son listados de primero y presentados al usuario de dicha forma. Los servicios que no cumplen con los requerimientos mínimos no son descartados, simplemente se ubican más abajo en la lista.

Por otro lado, el anunciante puede subir servicios descritos con RDF al repositorio haciendo uso del servidor de SPARQL Fuseki.

4.6. Resumen

En este capítulo se describen todos aspectos técnicos referentes a la solución propuesta al planteamiento del problema de esta investigación. En la primera sección se muestra el lenguaje de descripción para servicios REST seleccionado y además se describen las extensiones propuestas a dicho lenguaje. Posteriormente se presenta todo lo referente al algoritmo de descubrimiento: la realización del proceso de emparejamiento, los procesos para filtrar los servicios según el tipo de contenido deseado por el usuario, entre otros. Adicionalmente, se presenta el modelo de la plataforma desarrollada para soportar el mecanismo de descubrimiento, que incluye: el establecimiento de responsabilidades, el análisis de los requisitos, los diagramas de clases, despliegue, entre otros.

Capítulo 5

5.1. Desempeño del sistema

Antes de evaluar el algoritmo adaptado se realizan pruebas al sistema en general con el fin de determinar la forma en que afecta a los tiempos de respuesta el desarrollo de la solución. En este punto es importante resaltar que el servidor de aplicaciones se encuentra en un equipo de prestaciones medias.

El equipo en el cual se encuentra alojado el servidor JBoss 5.1.0.GA es un Dell Inspiron N4010 con Windows Home Basic de 64 bits, tiene un procesador Intel Core i3 @2,53GHz y una memoria RAM de 3GB DDR3. La configuración utilizada de JBoss es *default,* esta versión inicia con módulos básicos suficientes para soportar la aplicación "*What do you do?*", no permite configuraciones especiales como *cluster*.

Plan de pruebas

El siguiente plan de pruebas cubre los aspectos que hemos considerado que son fundamentales para el funcionamiento completo del sistema que soporta al algoritmo adaptado. Se calculan los tiempos de respuesta ya que el servicio es consumido por un usuario final a través de un servicio web y este experimenta diferentes grados de satisfacción al usar el servicio dependiendo, principalmente, del tiempo de espera por una respuesta del servidor de aplicación.

NOMBRE	DESCRIPCIÓN
Conexión WURFL	Determina el tiempo que le toma a la aplicación obtener una instancia de las clases encargadas de obtener la información de contexto a través del repositorio de servicios WURFL.
Conexión WordNet	Determina el tiempo que le toma a la aplicación crear una conexión a la base de datos lexical WordNet y crear instancias de las clases que se emplean para obtener sinónimos y similitudes de dicha base.
Conexión al repositorio de servicios	Determina el tiempo que le toma a la aplicación conectarse al repositorio de servicios y crear una instancia de las clases que son capaces de ejecutar una consulta sobre el repositorio y recuperar el resultado
Cargar información de contexto	Determina el tiempo que le toma a la aplicación cargar el archivo service_requirements.xml , archivo que es empleado para determinar los requerimientos mínimos que cada aplicación necesita para que pueda ser consumida eficientemente
Calidad del algoritmo	Se determina la calidad del algoritmo de descubrimiento a través de unas medidas de desempeño calculadas dentro de una metodología de evaluación detallada más adelante

Tabla 8. Plan de pruebas

Según (Menasce 2002) la calidad de la prestación de un servicio se mide como la combinación de los siguientes factores de calidad:

- Disponibilidad: porcentaje de tiempo en el que el servicio esta funcionando.
- Seguridad: mecanismo que provee el servicio para guardar la confidencialidad, la integridad de los datos y resistencia a ataques.
- Tiempo de respuesta: tiempo que el servicio emplea para responder a diferentes tipos de solicitudes. El tiempo de respuesta puede variar dependiendo de la carga a la cual se somete el servicio.
- Rendimiento: velocidad con que un servicio puede responder las solicitudes. El rendimiento varía con la carga.

Aunque no se encuentra dentro de los cuatro ítems anteriores, la primera medida que se puede obtener de la implementación de la solución es el tiempo que tarda en iniciar el servidor, con el equipo descrito anteriormente y una versión de JBoss sin aplicaciones desplegadas (a excepción de los que tiene por defecto: consola de administración, monitor de conexiones, etc.). Éste inicia en un tiempo de 58 segundos en promedio. Cuando se despliega *What do you do?,* el servidor entra en funcionamiento después de 1 minuto y 20 segundos (en promedio). Analizando los *logs* del servidor se puede determinar que este tiempo extra es empleado en cargar el archivo `wurfl.zip` que se usa como repositorio de dispositivos.

La decisión de cargar el archivo `wurfl.zip` cuando el servidor esta iniciando se toma debido a que este proceso toma alrededor de 13 a 15 segundos, por lo que es inviable que este se cargue cada vez que un usuario quiera usar la aplicación, esto produciría tiempos de respuesta superiores a los 15 segundos.

A continuación se presenta la evaluación detallada de los tiempos de respuesta.

5.1.1. Conexión a repositorios, WordNet y archivos auxiliares

Como se puede leer en capítulos anteriores, el algoritmo de búsqueda se apoya en WURFL para obtener las características de un dispositivo, WordNet para agregar semántica a la búsqueda, y un modelo para obtener los requerimientos mínimos que los servicios necesitan. Para el correcto funcionamiento del algoritmo se debe cargar todo lo mencionado anteriormente, así que es importante calcular los tiempos que toman esta labor y que afectan el tiempo de respuesta del sistema. A continuación se detalla cada aspecto.

5.1.1.1. WURFL

El archivo wurfl.zip que contiene el archivo XML que usa la API de WURFL para determinar con exactitud que dispositivo accede al sistema, este archivo se carga cuando el servidor arranca, de no ser así, los tiempos de respuesta pasarían los 13 segundos. Pero determinar el agente que quiere buscar un servicio si es un proceso que se realiza por cada usuario que ingresa a la aplicación, y para esto se necesita crear una instancia de *WURFLHolder y WURFLManager* para luego procesar el *user-agent* de la petición HTTP que llega. El tiempo que tarda la aplicación en determinar el dispositivo que esta accediendo a la aplicación es de 0.03 segundos (promedio de 20 muestras).

5.1.1.2. WordNet

Para hacer uso del diccionario lexical se debe crear una conexión a este cada vez que un nuevo usuario accede a la aplicación. Realizando un conjunto de 20 pruebas se calculó que en promedio el sistema emplea 3,554 segundos en realizar la conexión a la base de datos y en instanciar las clases que van a realizar las consultas sobre WordNet, aunque es un tiempo elevado, se tiene la ventaja de que por cada usuario solo se debe realizar una vez, es decir que solo se apreciara este retardo una vez. Luego de obtener este retardo se tomó la decisión de cargar el diccionario cuando el usuario ingresa a la opción "*search*", de esta forma no siente el retardo cuando se ejecuta su consulta sino cuando se ingresa al formulario de búsqueda de servicios (y solo una vez).

5.1.1.3. Repositorio de servicios

Para poder realizar las consultas es necesario que se cree una conexión con el repositorio de dispositivos, dicha conexión tarda 0.33 segundos en promedio (20 pruebas) y solo se debe realizar una vez por cada usuario que ingresa al sistema. Se determino que el número de servicios alojados no afecta el tiempo que le toma a la aplicación conectarse con el repositorio; el número de servicios alojados afecta directamente al algoritmo adaptado.

5.1.1.4. Requerimientos

El archivo service_requiriments.xml almacena en forma de *data model* información sobre los requerimientos mínimos que necesita cada servicio para que pueda ser consumido eficientemente por un dispositivo móvil. Este archivo crece a medida que el repositorio de servicios crece, la aplicación lee este archivo y los datos recuperados de WURFL y así ordena los servicios dependiendo del contexto. Las Figura 36 y Figura 37 ilustran el tiempo que le toma a la aplicación cargar el archivo a medida que el número de servicios crece.

La Figura 36 muestra el comportamiento de la aplicación a medida que se agregan los servicios que se utilizaron en las pruebas, se puede determinar que los tiempos de respuesta no son muy altos y que no se presenta un crecimiento lineal, esto evidencia la eficiencia del XML *parser* empleado (SAX).

Para determinar si el sistema es escalable considerando solo este aspecto, se realizó una segunda prueba en donde se llegan a agregar 800 servicios, los resultados obtenidos se presentan en la Figura 37 y en esta ya se puede apreciar que el crecimiento es lineal, por lo que se concluye que llegará un instante en el cual los tiempos de respuesta procesando este archivo serán un problema y generará retardos significativos.

De la Figura 37 se puede determinar que el comportamiento es lineal, así que realizando un cálculo de la pendiente se puede determinar el tiempo de respuesta *t* dependiente del número de servicios *n* con la siguiente función:

$$t = 1{,}186x10^{-4}n$$

La anterior ecuación sirve para determinar hasta que punto es conveniente usar el método planteado en este proyecto para ordenar los servicios teniendo en cuenta el contexto del usuario.

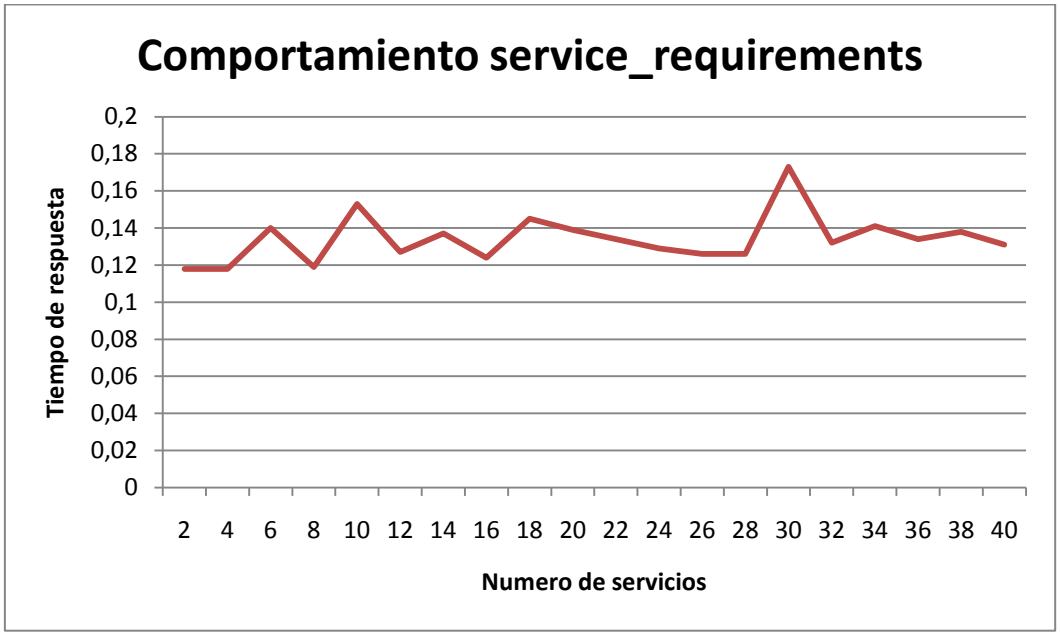


Figura 36. Comportamiento service_requirements (40 servicios)

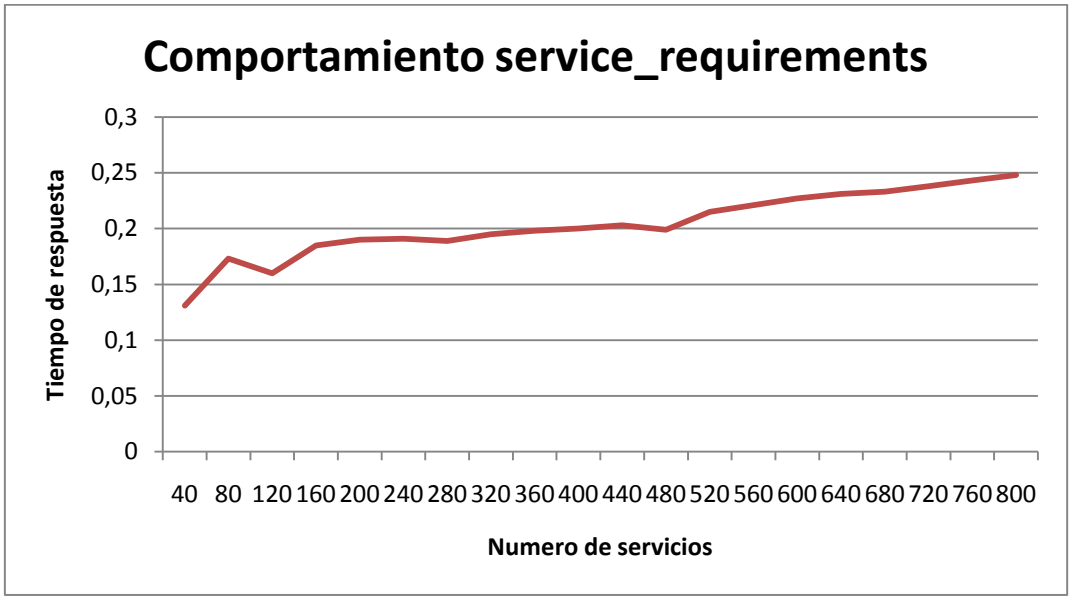


Figura 37. Comportamiento service_requirements (800 servicios)

Los tiempos de respuesta son aceptables ya que se encuentran alrededor de los 5 segundos. La valoración se da de acuerdo al estudio realizado en (Joines, Willenborg et al. 2003) en donde establece como tiempo de respuesta:

- Óptimo: un tiempo menor a 0.1 segundos, el cual es imperceptible por las personas.
- Bueno: tiempos que estén entre 0.1 y 1 segundos. Esta respuesta no genera inconvenientes para el usuario.
- Aceptable: entre 1 y 10 segundos; aunque este tiempo es notorio, el usuario puede esperar este tiempo, y más considerando que la búsqueda semántica le ahorrará tiempo obteniendo respuestas que con una búsqueda sintáctica las tendría que hacer con varias consultas.

- Deficiente: tiempo de respuesta mayor a 10 segundos; con este tiempo el usuario pierde interés y hasta tiende a pensar que la aplicación dejó de procesar su solicitud.

5.1.2. Estabilidad del sistema

Finalmente se realizaron pruebas de estrés a la aplicación, sometiéndola a muchas peticiones en paralelo, con el objetivo de determinar el comportamiento en condiciones más reales.

Para realizar estas pruebas se uso JMeter. Se creó un proxy encargado de capturar la navegación por la aplicación y así tener una navegación piloto o de prueba; ésta es almacenada en un hijo de ejecución de JMeter para posteriormente ejecutarlos en repetidas ocasiones de forma paralela. Los resultados fueron graficados por un plugin desarrollado por un grupo de desarrollo de google code.

En el anexo D se encuentran todas las pruebas que se realizó al servicio, y mediante las cuales se pudo demostrar que el sistema y la aplicación son medianamente robustos (teniendo como servidor de aplicaciones un computador de características medias), ya que empezó a presentar errores cuando se enviaban 40 peticiones en paralelo, como lo muestra la Figura 38.

De la Figura 38 se puede concluir que con 40 peticiones en paralelo el servicio solo respondió los primero 4 segundos y luego dejó de enviar respuesta a los clientes porque estaba tratando de procesar las solicitudes iniciales, después de 45 segundos se presenta el primer error (HTTP 500) y analizando los *logs* del servidor JBoss se pudo determinar que la mayoría de los errores se presentaron en el momento en el que se lanzaba la consulta SPARQL al repositorio de servicios.

Figura 38. Prueba de estrés

La disponibilidad del servicio es aceptable, aunque 40 peticiones no representen una gran numero, se debe considerar que están corriendo sobre un computadores de características medias que no esta optimizado para funcionar como servidor de aplicaciones, además de que no se empleó ningún mecanismo de robustez o un cluster que ayude a mejorar la robustez y disponibilidad del sistema en general.

5.2. Evaluación de la calidad del algoritmo

Para llevar a cabo la evaluación del algoritmo de descubrimiento adaptado, se realizó una búsqueda de una cantidad suficiente de servicios Web descritos con el lenguaje hRESTS. Esta tarea demandó bastante tiempo y esfuerzo, pero finalmente se encontró una colección de servicios de prueba denominada hRESTS-TC, la cuál es utilizada para la ejecución de la evaluación de *matchmakers* basados en hRESTS, y que fue construida por: Ulrich Lampe, Stefan Schulte, Martin Prodanov, y Martin Pinto. Esta colección de servicios está disponible en el siguiente enlace: http://semwebcentral.org/projects/hrests-tc/.

hRESTS-TC cuenta con un total de 1080 servicios disponibles para ser utilizados por algoritmos de emparejamiento. A estos servicios se les añadió las nuevas propiedades definidas en este trabajo para hRESTS y especificadas en el capítulo 4, de manera manual y aleatoria (en la Figura 39 se muestra un ejemplo de un servicio enriquecido con las nuevas propiedades). Los 1080 servicios del repositorio están divididos en 9 dominios: *communication, economy, education, food, geography, medical, simulation, travel y weapon.* El dominio seleccionado para realizar las pruebas es *weapon,* el cual cuenta con un total de 40 servicios.

Figura 39. Ejemplo de un servicio descrito con hRESTS, incluyendo las nuevas propiedades Payload Format y Content Type

El mecanismo de evaluación de calidad para el algoritmo desarrollado en este proyecto es algo complejo, ya que no sólo se consideró la tarea de emparejamiento o determinación del grado de correspondencia entre dos servicios a través de un algoritmo, sino que se construyó toda una plataforma para el descubrimiento de servicios REST, que incluye un módulo de procesamiento de la consulta digitada por un usuario en lenguaje no formal (SPARQL). De esta manera, la consulta no se define como un servicio requerido por un usuario sino que ésta se compone de una serie de palabras claves que proporciona el usuario y que son transformadas a una consulta formal.

Hay que decir que los servicios hRESTS presentes en la colección de prueba tienen las siguientes limitaciones (Lampe, Schulte et al. 2010):

- Todos los servicios sólo tienen una operación, con múltiples entradas y salidas

- Se ha asignado el métodos POST por defecto a todas las descripciones de los servicios

En (Kopecký, Vitvar et al. 2009) los servicios son transformados a RDF con el fin de guardarlos en un repositorio y posteriormente ser consultados. Esta transformación se hace utilizando la tecnología XSLT mediante un archivo de transformación[20] definido por el *PhD(c)* Jacek Kopecky quien nos entrego este archivo para realizar nuestras pruebas. Con el fin de procesar las nuevas propiedades definidas y en general para guardar los servicios enriquecidos con las nuevas propiedades en un repositorio, fue necesario modificar el archivo de transformación XSLT para agregar los métodos para procesar las nuevas propiedades (el archivo de transformación modificado se encuentra en el anexo C).

Se utilizó la herramienta Saxon[21] para transformar todos los servicios enriquecidos a RDF. Saxon permite transformar archivos XML o XHTML a través de XSLT no sólo individualmente sino a partir de directorios mediante un comando simple desde una consola. De esta manera se construyó el repositorio de servicios del dominio *weapon* transformados a RDF. Una vez transformados a RDF y guardados en el repositorio, los servicios están disponibles para soportar consultas realizadas por el usuario.

5.2.1. Metodología de evaluación

Debido a que no se cuenta con trabajos que implementen un motor de búsqueda semántico de servicios sobre el repositorio seleccionado, para la evaluación de la calidad del algoritmo se utilizó una metodología mencionada en el trabajo de (Bastidas and Ordoñez 2010) denominada Benchmark. Benchmark puede definirse como un punto de referencia para evaluar comparativamente el rendimiento de un sistema con respecto a otro, el cuál generalmente representa las mejores prácticas en su domino (Bastidas and Ordoñez 2010).

En nuestro caso el Benchmark de referencia está definido en la colección de prueba hRESTS-TC y fue realizado por los autores de los servicios de prueba: Stefan Schulte, Ulrich Lampe, Matthias Klusch, Patrick Kapahnke, Martin Prodanov, y Martin Pinto. Los autores establecieron unas consultas predefinidas y para cada una de ellas determinaron manualmente cuáles servicios deberían ser recuperados. Para el dominio seleccionado (*Weapon*) los autores definieron tres consultas; la Tabla 10 muestra dichas consultas y el número de servicios agrupados en cuatro categorías que se deben obtener o descartar. Los evaluadores expertos clasificaron cada una de las posibles parejas según unas ponderaciones, que se muestran en la Tabla 9.

Nombre	Grado	Valor
Altamente Relevante	1	3
Relevante	2	2
Potencialmente relevante	3	1
No relevante	4	0

[20] El archivo está disponible en: http://members.sti2.at/~jacekk/hrests/hrests.xslt y nos fue amablemente proporcionado por los doctores *Jacek Kopecky y Ulrich Lampe*
[21] Disponible en: http://www.saxonica.com/

Tabla 9. Ponderaciones declaradas por los expertos para cada par de servicios que conforma el Benchmark

Estas ponderaciones realizadas por los expertos (como se muestra en la Tabla 9) clasifican los servicios recuperados en cuatro categorías: desde *altamente relevantes*, es decir, que existe una similitud máxima entre la consulta y el servicio ofertado, hasta *no relevante*, lo que quiere decir que la conulta no tiene ninguna similitud con el servicio ofertado. Los ítems de grado y valor, descritos en la Tabla 9, se usan para identificar cuantitativamente la correspondencia entre el servicio de consulta y el ofertado. Por ejemplo, una correspondencia de grado 1 es equivalente a una de valor 3, que hacen referencia a que la relación entre la consulta y la oferta es *Altamente relevante.*

Así, para una consulta hecha sobre los servicios pertenecientes al domino seleccionado, los autores definen la siguiente reseña:

Servicio Web de consulta	Número de Servicios Altamente relevantes	Número de Servicios Relevantes	Número de Servicios potencialmente relevantes	Número de servicios no relevantes
Government Missile Funding Service	14	22	4	0
Government Missile Financing Service	10	22	7	1
Missile Government lending range	13	19	4	4

Tabla 10. Similitudes establecidas por los expertos para las tres consultas predefinidas en el dominio Weapon

Por ejemplo, los autores de la colección de prueba definieron que para la consulta *Government Missile Funding Service* los servicios clasificados como *Altamente Relevantes* son 14, los *Relevantes* 22, los *Potencialmente Relevantes* 4 y los *No Relevantes* 0. Así, es deseable que un mecanismo apropiado de descubrimiento recupere los servicios clasificados en las dos primeras categorías, ya que los potencialmente relevantes y obviamente los no relevantes, no cumplirían con el requerimiento del cliente.

Los valores de la Tabla 10 se tomarán como referencia para la determinación de indicadores de calidad que demuestren la eficiencia del algoritmo. En ese orden de ideas, estos valores constituyen el *benchmark* de referencia. La tabla anterior solo presenta el numero total de servicios, en el Anexo D se encuentra de forma detallada los servicios que deben ser recuperados para cada consulta. Finalmente hay que decir que en el benchmark de referencia establecido por los expertos en la colección de servicios de prueba no se propone un ranking de servicios. Como ejemplo, en la Tabla 10, los 14 servicios clasificados como altamente relevantes son tomados por igual; se considera que el servicio 1 satisface la consulta igual que el servicio 14.

Teniendo en cuenta todo lo anterior, el experimento realizado para la evaluación de la calidad del algoritmo es el siguiente:

Se ejecutaron las tres consultas establecidas en el benchmark de referencia (Tabla 10) utilizando la aplicación desarrollada que implementa el algoritmo de descubrimiento propuesto en nuestro trabajo de grado, y a partir de los resultados se calcularon unas medidas de desempeño, que nos permiten determinar la precisión del algoritmo a partir de la comparación entre los servicios recuperados por nuestro sistema y los que debieron haber sido recupertados según el benchmark de referencia. Estas pruebas fueron hechas en el equipo de cómputo especificado al inicio de este capítulo. A continuación se presentan las medidas de desempeño calculadas.

5.2.2. Medidas de desempeño

Seguidamente se definen unas medidas estadísticas que van a ser aplicadas con el fin de determinar la calidad de los resultados obtenidos con el mecanismo de descubrimiento propuesto. Estas medidas estadísticas son usualmente empleadas en el campo conocido como "*Information Retrieval*" o Recuperación de información, y son: *precision (p), recall (r) y overall (o) (Yatskevich 2003; Bastidas and Ordoñez 2010).*

La medida ***precision*** se define como la proporción de servicios clasificados como verdaderos positivos respecto al número total de servicios publicados recuperados por el algoritmo, y se calcula como:

$$p = \frac{|tp|}{|tp| + |fp|}$$

Donde,
tp: servicios positivos recuperados
fp: servicios negativos recuperados

El indicador ***recall*** identifica el porcentaje de servicios clasificados como verdaderos positivos respecto al número total de servicios publicados considerados como relevantes, es decir, aquellos que deben ser recuperados según el *benchmark* de referencia.

$$r = \frac{|tp|}{|tp| + |fn|}$$

Donde,

fn: servicios que debieron pero que no fueron recuperados

El indicador ***overall*** es quien determina la calidad del proceso mediante la siguiente ecuación que emplea los conceptos anteriores.

$$o = r * \left(2 - \frac{1}{p}\right)$$

5.2.3. Resultados obtenidos

Conociendo las ecuaciones necesarias para determinar la calidad del emparejamiento se procedió a realizar las consultas descritas en la Tabla 10 sobre el repositorio de prueba haciendo uso del algoritmo adaptado. Tomando como referencia los resultados de la Tabla 10 se obtiene los valores tp, fp y fn de la Tabla 11 de la siguiente forma:

- Si un servicio aparece en el benchmark de referencia como altamente relevante o relevante y este es recuperado con el algoritmo adaptado, se cataloga como servicio positivo recuperado (tp).
- Si un servicio aparece en el benchamark de referencia como potencialmente relevante o nada relevante y este es recuperado con el algoritmo adaptado, se cataloga como servicio negativo recuperado (fp).
- Si un servicio aparece en el benchmark de referencia como altamente relevante o relevante y este no es recuperado con el algoritmo adaptado, se cataloga como un servicio fn (que debió ser recuperado).

El número de servicios que se obtuvieron en cada categoría son los siguientes:

Consulta	tp	fp	fn
Government Missile Funding Service	32	2	4
Government Missile Financing Service	29	3	6
Missile Government lending range	28	4	6

Tabla 11. Resultados obtenidos

En el Anexo D se encuentra de forma detallada los servicios que fueron recuperados. Haciendo uso de las ecuaciones para determinar las medidas de desempeño se calcularon los valores de *precision, overall y recall* para cada consulta, los resultados son presentados en la tabla 12.

Consulta	p	r	o
Government Missile Funding Service	0.94	0.88	0.83
Government Missile Financing Service	0.90	0.82	0.72
Missile Government lending range	0.87	0.82	0.69

Tabla 12. Medidas de desempeño del sistema

Conociendo *precision* y *recall* para cada una de las consultas se puede determinar F_1 *(F-score).* F_1 es una medida que se utiliza para determinar la precisión de una prueba, el valor de esta variable va desde 0 hasta 1, siendo este ultimo valor el deseable. La formula general para determinar el valor de F_1 es la siguiente:

$$F_\beta = (1+\beta^2).\frac{p.r}{(p.\beta^2)+r}$$

Existe la posibilidad de que el proceso de descubrimiento tenga un buen valor de *p* pero uno no tan bueno de *r* o viceversa. β es un valor preestablecido que se usa para definir a que se da mayor importancia, *precision o recall*. Así, cuando β es mayor que 1 se le da más importancia a *r*, y cuando es menor que 1 se da mas importancia a *p.* En nuestro caso le damos la misma importancia a *p y r* asi que β=1 y la ecuación para determinar *F* seria la siguiente:

$$F = 2.\frac{p.r}{p+r}$$

De esta manera, para cada consulta y obviamente teniendo en cuenta los resultados establecidos en la Tabla 12, se determinaron tienen los siguientes resultados:

Consulta	F
Government Missile Funding Service	0.90
Government Missile Financing Service	0.85
Missile Government lending range	0.84

Tabla 13. Resultados F1-score

Análisis

De la Tabla 12 se puede realizar el siguiente análisis (promediando los resultados):

- De todos los servicios que deben ser recuperados para una consulta determinada, se recupera el 84% de estos. Es importante notar que se recuperan servicios que con un método de búsqueda sintáctico no hubiesen sido recuperados. En un ejemplo práctico se puede decir que los expertos determinaron que para una necesidad específica se deberían recuperar 10 servicios, y el algoritmo propuesto en este trabajo de grado recupera 8.
- Los 90% de los servicios recuperados satisfacen el requerimiento del usuario, quedando un margen muy pequeño para falsos positivos o servicios recuperados que debieron ser recuperados.
- Tomando los cálculos realizados de precision y recall se calcula que la calidad del emparejamiento (overall) es en promedio de 0.73.
- Los valores obtenidos son altos debido a que los servicios presentan restricciones como, por ejemplo, una sola operación por descripción del servicio. Es deseable por lo tanto que se complemente la colección de servicios de prueba añadiendo más operaciones por cada servicio, con el fin de obtener resultados más eficientes.

En cuanto a los resultados de la Tabla 13, se destaca que la relación entre *p y r* se acerca mucho a 1 (mejor valor) lo que indica que tanto *precision* y *recall* están en "armonia". Esto significa que el algoritmo recupera un buen número de servicios relevantes o altamente relevantes y además descarta un gran porcentaje de los servicios catalogados como potencialmente relevantes o no relevantes. Sin embargo, es deseable que se desarrolle otro prototipo de descubrimiento que utilice la misma colección de prueba con el fin de contar con otro ítem de comparación real. Para ello, en este trabajo se conformó un repositorio de servicios de prueba descritos en hRESTS que estará disponible para futuros proyectos enmarcados en este contexto.

Adicionalmente, este prototipo de descubrimiento requiere de bastantes trabajos futuros, ya que ésta fue una primera aproximación para proponer un mecanismo de descubrimiento sobre servicios Web RESTful y existe bastante trabajo para hacer, sobre todo con el fin de mejorar los resultados de precisión del algoritmo, por ejemplo a través del desarrollo de un módulo avanzado de procesamiento de consultas en lenguaje natural, o a través de la definición de una sola ontología de dominio para referenciar en esta a todos los servicios de prueba.

5.3. Resumen

En este capítulo se presentaron las pruebas realizadas al prototipo de descubrimiento de servicios Web RESTful desarrollado. Estas pruebas están divididas en dos partes: en la primera sección se presentan pruebas de rendimiento del sistema (tiempos de respuesta, estabilidad, entre otras) y en la segunda sección se presentan las pruebas de calidad del algoritmo de descubrimiento, realizadas a través de una metodología expuesta en esta sección. Finalmente se presenta un análisis de las medidas de desempeño del sistema.

Capítulo 6

Conclusiones, contribuciones y trabajos futuros

6.1. Contribuciones

Las principales contribuciones del proyecto son:

- Una base de conocimiento que puede servir para enmarcar futuros proyectos de investigación en el área de los servicios Web RESTful en la Universidad del Cauca.
- Una extensión del modelo de servicio de hRESTS, que soporta el ordenamiento de los servicios Web según el tipo de contenido que el usuario requiere.
- Un módulo de filtrado de servicios, que permite seleccionar los que retornen el tipo de contenido o formato que el cliente desee.
- La adaptación de un algoritmo que es capaz de realizar una búsqueda semántica de servicios Web RESTful; dicha búsqueda cumple con los requisitos o el servicio que espera alcanzar un usuario o un agente.

- El análisis de los mecanismos y procedimientos necesarios para hacer posible una búsqueda de servicios web RESTful de forma semántica. Se presentan temas como la descripción del servicio, el enriquecimiento de dicha descripción, mecanismos semánticos de búsqueda.

- El análisis de las principales contribuciones de otros autores que fueron empleadas para alcanzar los objetivos de este proyecto.

- Un repositorio de servicios Web RESTful que puede ser usado en proyectos futuros relacionados con el tema. Una de las actividades que demandó mas tiempo fue la obtención de servicios para realizar las pruebas. Con un número total de 1080 servicios, futuras investigaciones podrán hacer uso de nuestro repositorio.

- Un prototipo funcional que recoge todo el esfuerzo investigativo y demuestra que el descubrimiento semántico de servicios Web RESTful es posible.

- Artículo que recoge de forma detallada el proceso y las contribuciones de este proyecto de grado.

6.2. Conclusiones

- Se construyó con éxito una base de conocimiento relacionada con el tema de descubrimiento semántico de servicios REST en entornos móviles, a partir de la cual se logró orientar esta investigación y se identificaron los principales problemas en ese contexto en los cuales se podía hacer un aporte investigativo. Teniendo en cuenta lo anterior, podemos concluir que la base de conocimiento generada cumple con los interrogantes planteados en el capítulo

1 de esta monografía. De esta manera, la elaboración de la base de conocimiento fue exitosa.

- Para la construcción del estado del conocimiento referenciado, en los primeros capítulos de esta monografía, se utilizó la metodología del "modelo para la investigación documental" descrita por el ingeniero Carlos Serrano, en donde se especifican claramente cada uno de los pasos a seguir para este fin. Así, podemos decir que la metodología utilizada es eficiente y efectiva, ya que nos permitió abordar de manera apropiada todo el tema de investigación. Específicamente se resalta la directriz que sugiere dividir el tema central en subtemas o núcleos temáticos; esto permite un manejo eficiente de la complejidad de la investigación. Adicionalmente, es bastante importante la declaración del alcance de la base de conocimiento ya que, según estipula la metodología, sirve como indicador de calidad para determinar si el conjunto de unidades de análisis consideradas responde a los interrogantes planteados al inicio de la investigación.

- A partir de la investigación documental realizada y la base de conocimiento generada, se lograron identificar las principales iniciativas involucradas en tarea de definir un mecanismo de Descubrimiento de servicios Web RESTful, seleccionando hRESTS como el lenguaje más idóneo para describir los servicios, teniendo en cuenta las necesidades particulares del problema definido en el marco de este proyecto. Sin embargo, se concluyó que hRESTS no representaba fielmente la característica típica de los servicios REST de tener múltiples representaciones de cada recurso (permitiendo al usuario negociar con el servidor el tipo de contenido que desea).En este tema específico se identificó la posibilidad de realizar un aporte. En ese orden de ideas, se propusieron dos extensiones al formato: las propiedades *Content type* y *Payload format.* En síntesis, podemos concluir que las extensiones propuestas permiten que el lenguaje de descripción se adapte más a las características de REST, y que las mismas actúan con éxito al permitir la clasificación de los servicios según las especificaciones del usuario, como se menciona en el capítulo 5.

- La propuesta de adaptación del algoritmo de descubrimiento (reflejada en los algoritmos 1, 2, 3 y 4) permite satisfacer con los requerimientos establecidos en el planteamiento del problema para un mecanismo ideal de descubrimiento de servicios REST ya que:
    - El algoritmo se ejecuta sobre servicios Web REST, descritos con el formato hRESTS
    - El algoritmo ejecuta el descubrimiento de forma semántica a través del cálculo de la similitud entre operaciones, entradas y salidas haciendo uso de WordNet
    - El algoritmo soporta la clasificación de los servicios Web REST según el tipo de contenido y formato que el cliente seleccione, haciendo uso de las extensiones propuestas al formato de descripción hRESTS
    - El algoritmo tiene en cuenta un aspecto que implica el entorno móvil: los servicios recuperados se ordenan según el tamaño de la pantalla del

dispositivo. De esta manera, los servicios más apropiados para ser consumidos en determinados terminales móviles aparecerán primero en la interfaz. Para el prototipo se utilizó solamente la propiedad del tamaño del dispositivo, pero se pueden adicionar fácilmente otras propiedades relacionadas con el entorno móvil.

- Para el entorno en el cual se realizaron las pruebas, el algoritmo no es tan robusto como se espera ya que con 40 peticiones en menos de 10 segundos se logra saturar el servidor de aplicaciones. Obviamente este rendimiento empeora si aumentan las peticiones.

- Los tiempos de respuesta de la aplicación son aceptables para el entorno web. La primera consulta siempre es la que más se demora debido a que en esta se deben obtener la conexión a los diferentes repositorios y archivos auxiliares. Las pruebas dieron como resultado que a veces se presentan retardos de 12 segundos para la primera consulta (lo cual no es deseable para una aplicación web) pero estos no son abundantes y por lo general la respuesta del sistema para buscar servicios se da en menos de 5 segundos.

- El porcentaje de servicios recuperados que pueden satisfacer las necesidades del usuario y son efectivamente recuperados es del 90% y el porcentaje de servicios útiles recuperados es del 84%. Trabajos que tengan como objetivo mejorar el sistema desarrollado deben partir de que se debe mejorar estos indicadores y así mejorar la calidad del proceso en general (***overall***).

- Aunque el número de consultas para evaluar la calidad del algoritmo es bajo, estas son precisas, ya que los servicios recuperados se comparan contra los servicios que debieron ser recuperados según el criterio de expertos en el tema.

6.3. Trabajos futuros

- El repositorio de servicios utilizado para implementar la solución en este proyecto de grado se alimenta de servicios que son subidos desde Fuseki mediante una aplicación web. Un trabajo futuro es desarrollar un módulo que permita a los proveedores subir servicios desde una sola plataforma lo que permite tener un control más centralizado.

- Los servicios que se consiguieron para realizar las pruebas están descritos en hRESTS, pero se alojan en el repositorio de servicios como RDF. Para realizar la transformación de hRESTS a RDF se hace uso de Saxon a través de una consola. Saxon cuenta con API en Java, así que también es posible realizar como trabajo futuro una implementación de la API de Saxon para que se puedan subir servicios en formato hRESTS. Un sistema ideal recibiría cualquier formato de descripción y realizaría una transformación a partir de archivos XSLT apropiados.

- Considerar una ontología de dominio general que permita un mecanismo diferente de descubrimiento semántico con el objetivo de compararlo con el presente y determinar cual es la mejor solución.
- Desarrollar un módulo de personalización. Éste se encargaría de “aprender” sobre las preferencias de los usuarios a través de las consultas que realizan. De esta forma se podría refinar la búsqueda.

- Desarrollar un módulo que soporte el componente espacial y temporal del contexto ya que en este trabajo sólo se considero las características del dispositivo.

- En este trabajo se utilizó hRESTS como formato de descripción. Sin embargo, es probable que otros lenguajes como ReLL puedan ser igualmente adecuados para describir los servicios. Por lo tanto sería interesante adaptar el algoritmo definido para que opere sobre descripciones ReLL. Un trabajo obligatorio en este campo sería conformar una colección de servicios REST descritos con ReLL para poder realizar pruebas al algoritmo.

- Mejorar la colección de servicios de prueba utilizada para la evaluación del algoritmo (adicionando más operaciones por servicio), con el fin de realizar pruebas más exigentes.

- Realizar un módulo avanzado de procesamiento de consulta para permitir mejorar la experiencia del usuario final y poder enmarcar la aplicación a un contexto real.

- Desarrollar un módulo, o alguna técnica o estrategia que permita mejorar los resultados de precisión del algoritmo, ya que éstos no son los ideales. Esto se podría lograr a través de la implementación de alguna táctica de emparejamiento semántico de servicios más sofisticada. Sin embargo, hay que tener en cuenta que eso incrementaría los tiempos de procesamiento, lo cuál es indeseable en entornos móviles. Además, habría que conformar una colección de servicios adecuada para lograr este trabajo.

Referencias

Aijaz, F., M. A. Chaudhary, et al. (2010). Performance Comparison of a SOAP and REST MobileWeb Server. Alemania, RWTH Aachen University.

Alarcon, R. and E. Wilde (2010). "Linking Data from RESTful Services."

Alarcón, R. and E. Wilde (2010). RESTler: crawling RESTful services, ACM.

Algermissen, J. (2010). "Using DNS for REST Web Service Discovery." InfoQ. Retrieved 15 de Junio, 2010, from http://www.infoq.com/articles/rest-discovery-dns.

Alowisheq, A., D. E. Millard, et al. (2009). EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. Universidad de Southampton. Reino Unido**:** 8.

Bachlechner, D., K. Siorpaes, et al. (2006). Web service discovery-a reality check.

Bastidas, A. and L. Ordoñez (2010). Comparación semántica de tareas entre dos procesos de negocio de telecomunicaciones. Departamento de Telemática. Popayan, Universidad del Cauca.

Bellur, U. and R. Kulkarni (2007). Improved matchmaking algorithm for semantic web services based on bipartite graph matching, Ieee.

Bianchini, D., V. De Antonellis, et al. (2006). Lightweight Ontology-based Service Discovery in Mobile Environments. Universidad de Brescia. Brescia, Italia**:** 6.

Booth, D., H. Haas, et al. (2004). Web Services Architecture. W3C**:** 98.

Boss, R. W. (2002). Client Server Technology.

Bray, T., J. Paoli, et al. (1997). "Extensible markup language (XML)." World Wide Web Journal **2**(4): 27-66.

Burstein, M., C. Bussler, et al. (2005). "A semantic web services architecture." Internet Computing, IEEE **9**(5): 72-81.

Carrión Jiménez, G. M. (2010). Tecnologías de Web Semántica orientada al desarrollo de Servicios Web. Loja, Ecuador, Universidad Técnica Particular de Loja. **Ingeniero en Sistemas Informáticos y Computación**.

Cavanaugh, E. (2006). Web services: Benefits, challenges, and a unique, visual development solution. Altova. Beverly, EEUU.

Clark, J. (1999). "XSL transformations (XSLT) version 1.0." W3C recommendation **16**(11).

Cowan, J. (2005). An introduction to building Web Services without tears, Universidad West Chester.

Curbera, F., M. Duftler, et al. (2002). "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI." Internet Computing, IEEE **6**(2): 86-93.

Chinnici, R., M. Gudgin, et al. (2003). "Web services description language (WSDL) version 1.2 part 1: Core language." World Wide Web Consortium, Working Draft WD-wsdl12-20030611.

Doulkeridis, C., N. Loutas, et al. (2006). "A system architecture for context-aware service discovery." Electronic Notes in Theoretical Computer Science **146**(1): 101-116.

Farrell, J. and H. Lausen (2007). "Semantic Annotations for WSDL and XML Schema." W3C Recommendation.

Fensel, D., F. Fischer, et al. (2010). "WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web." W3C Member Submission **23**.

Fensel, D., U. Keller, et al. (2005). WWW Or What is Wrong with Web service Discovery. W3C. Austria.

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. California, EEUU, Universidad de California. **Doctor of Philosophy in Information and Computer Science**.

García Hervás, J. M. (2008). Redes sociales en el móvil. Análisis estratégico desde una comunidad virtual y un operador móvil. Madrid, España.

Gomadam, K., A. Ranabahu, et al. (2010). SA-REST: Semantic Annotation of Web Resources. W3C.

Gutiérrez, N. (2009). "Servicios Web Basados en REST." SoftwareGuru. Retrieved 11 de Agosto, 2010, from http://www.sg.com.mx/content/view/919.

Hamad, H., M. Saad, et al. (2009). Performance Evaluation of RESTful Web Services for Mobile Devices. Palestina, Universidad de Gaza.

Hermida, V. A. (2010). Arquitectura de Referencia para la Composición de Servicios en Ambientes de Computación Ubicua. Popayán, Colombia, Universidad del Cauca.

Hermida, V. A., J. C. Corrales, et al. (2009). Plataforma para Descubrimiento de Servicios en Ambientes Ubicuos. GIT, Universidad del Cauca. Popayán, Colombia**:** 20.

Herrera de la Cruz, J. (2005). Un Modelo Fundamentado en Análisis de Dependencias y WordNet para el Reconocimiento de Implicación Textual. Departamento de Lenguajes y Sistemas Informáticos, Universidad Nacional de Educaci´on a Distancia.

ilico.net (2011). "OOTAY." from http://www.ootay.com/QuiSommesNous.asp.

Joines, S., R. Willenborg, et al. (2003). Performance analysis for Java Web sites, Addison-Wesley Professional.

Kalin, M. (2009). Java Web Services: Up and Running, O'Reilly Media, Inc.

Keller, U., R. Lara, et al. (2005). "Semantic Web Service Discovery." WSMX Working Draft.

Keller, U., R. Lara, et al. (2004). "Wsmo web service discovery." WSML Working Draft D **5**.

Kiljander, H. (2004). Evolution and usability of mobile phone interaction styles, Citeseer.

Kopecký, J., K. Gomadam, et al. (2008). hrests: An html microformat for describing restful web services, IEEE.

Kopecký, J., T. Vitvar, et al. (2009). hRESTS & MicroWSMO, Technical report.

Kreger, H. (2001). Web Services Conceptual Architecture (WSCA 1.0). IBM Software Group**:** 21.

Lampe, U., S. Schulte, et al. (2010). Adaptive matchmaking for RESTful services based on hRESTS and MicroWSMO, ACM.

Lathem, J. (2007). SA-REST: Adding Semantics to REST-Based Web Services. Athens, Georgia, Universiad de Georgia. **Master of Computer Science:** 52.

Lin, D. (1998). An information-theoretic definition of similarity, San Francisco.

Maleshkova, M., C. Pedrinaci, et al. (2009). "Supporting the creation of semantic RESTful service descriptions."

Maleshkova, M., C. Pedrinaci, et al. (2010). "Semantic annotation of Web APIs with SWEET."

Manola, F. and E. Miller (2004). "RDF Primer." W3C recommendation.

Márquez Avendaño, B. M. and J. M. Zulaica Rugarcía (2004). Implementación de un reconocedor de voz gratuito a el sistema de ayuda a invidentes Dos-Vox en español. Puebla, Mexico Universidad de las Americas.

McGuinness, D. L. and F. Van Harmelen (2004). "OWL web ontology language overview." W3C recommendation **10**: 2004-2003.

Menasce, D. A. (2002). "QoS issues in Web services." Internet Computing, IEEE **6**(6): 72-75.

Nandigam, J., V. N. Gudivada, et al. (2005). "Semantic web services." Journal of Computing Sciences in Colleges **21**(1): 50-63.

Paolucci, M., T. Kawamura, et al. (2002). "Semantic matching of web services capabilities." The Semantic Web—ISWC 2002: 333-347.

Pautasso, C. and E. Wilde (2009). Why is the web loosely coupled?: a multi-faceted metric for service design, ACM.

Pautasso, C., O. Zimmermann, et al. (2008). Restful web services vs. big'web services: making the right architectural decision, ACM.

Quin, L. (2003). "Extensible Markup Language (XML)." from http://www.w3.org/XML/.

Roman, D., U. Keller, et al. (2005). "Web service modeling ontology." Applied Ontology **1**(1): 77-106.

Schulte, S., U. Lampe, et al. (2010). LOG4SWS. KOM: Self-Adapting Semantic Web Service Discovery for SAWSDL, IEEE.

Serrano Cartaño, C. E. (2008). Anexo 2: Elaboración de la Base Inicial de Conocimiento. Modelo Integral para el Profesional en Ingeniería. U. d. Cauca. Popayán, Universidad del Cauca**:** 105-107.

Serrano Cartaño, C. E. (2008). Modelo para la Construccion de Soluciones. Modelo Integral para el Profesional en Ingenieria. U. d. Cauca. Popayan, Universidad del Cauca**:** 43-58.

Serrano Cartaño, C. E. (2008). Modelo para la Investigacion Documental. Modelo Integral para el Profesional en Ingenieria. U. d. Cauca. Popayán, Colombia, Universidad del Cauca**:** 12-20.

Seung-Jun Cha, Y.-J. C., Kyu-Chul Lee (2010). Development of Retrieval Methods for RESTful Web Services using Semantic Technologies. 9th IEEE/ACIS International Conference on Computer and Information Science.

singh, T. (2009) REST vs. SOAP – The Right WebService.

Sotolongo, R., C. Kobashikawa, et al. (2007). Algorithm forWeb Service Discovery Based on Information Retrieval UsingWordNet and Linear Discriminant Functions**:** 8.

Sycara, K., M. Paolucci, et al. (2003). "Automated discovery, interaction and composition of Semantic Web services." Web Semantics: 27.

Toma, I., N. Steinmetz, et al. (2008). "D5. 1.1–State of the Art Report On Service Description and Existing Discovery Techniques." SOA4All–FP7–215219.

Toshiro Takase, S. M., Shinya Kawanaka, Ken Ueno, Christopher Ferris, Arthur Ryman (2008) Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0. Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0

Tramullas, J. (1997) Introducción a la Documática, 1: Teoría.

Vitvar, T. and D. Fensel (2009). "Semantic Web Service Automation with Lightweight Annotations."

Yatskevich, M. (2003). "Preliminary evaluation of schema matching systems."

zur Muehlen, M., J. V. Nickerson, et al. (2005). "Developing web services choreography standards--the case of REST vs. SOAP." Decision Support Systems **40**(1): 9-29.

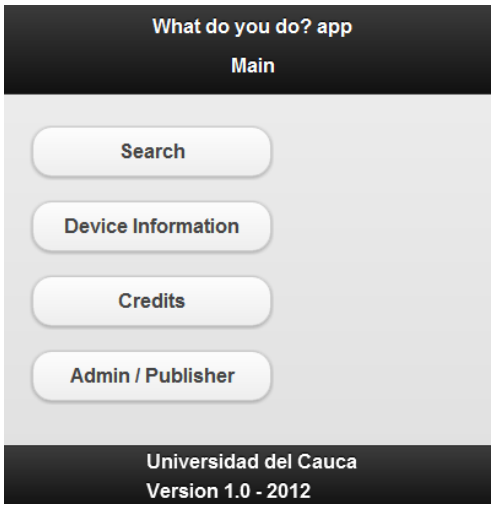
ANEXO A

INTERFAZ Y MANUALES DE USUARIO

A.1. Interfaz de usuario.

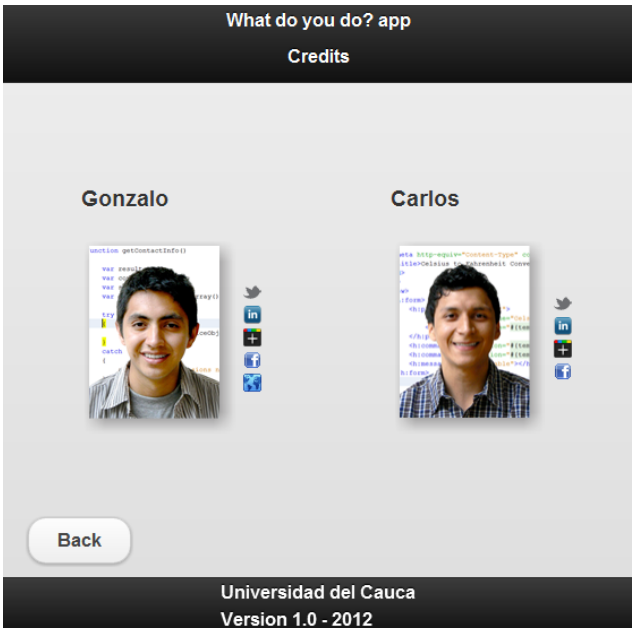
Pensando en que la aplicación se usa desde dispositivos móviles, la interfaz del usuario esta implementada pensando en este entorno, por este motivo predominan los botones grandes y textos de corta extensión. Para cumplir con lo anterior y tener una interfaz agradable al usuario, se hizo uso del framework jQuery Mobile, el cual es un sistema que permite crear interfaces basadas en HTML5 para las plataformas mas conocidas (las plataformas soportadas pueden consultarse en http://jquerymobile.com/gbs/). En el caso de que un dispositivo no soporte las características de jQuery Mobile la interfaz de usuario aparecerá como si fuera un html sin un estilo asociado (CSS), esto se veria como una pagina tipo Web1.0, pero lo importante es que la aplicación sigue siendo funcional.

El usuario ingresa a la aplicación mediante una URL (para nuestro entorno de pruebas http://localhost:8080/WDYDApp/) y le aparece la siguiente interfaz:

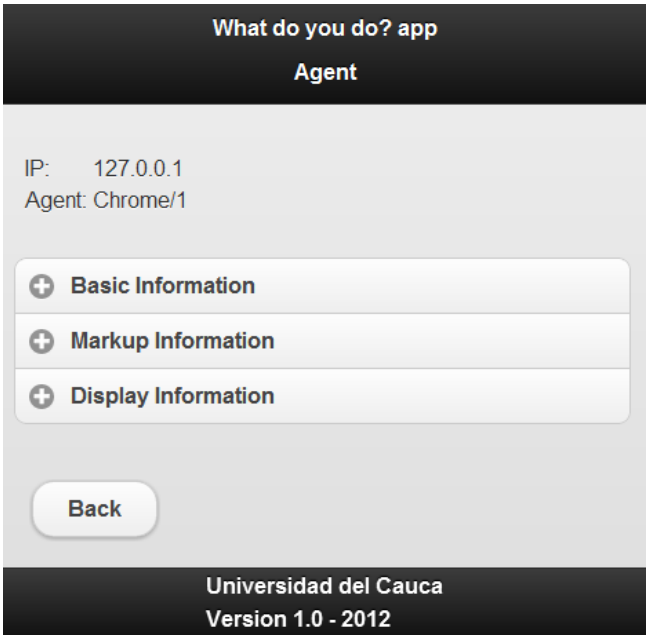


La opción de *Admin/Publisher* no esta implementada y esta pensada como un trabajo futuro del proyecto que permitirá subir los servicios desde la misma aplicación, así como modificar parámetros que afecten la búsqueda de los servicios.

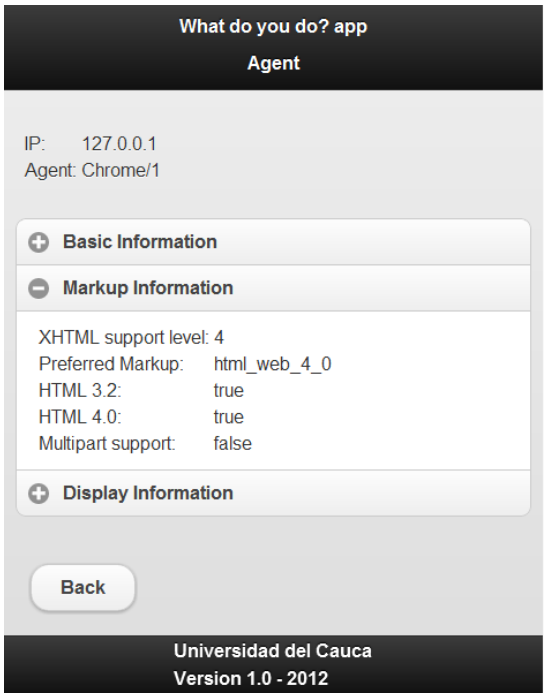
Al dar clic sobre la opción *Credits* se despliega información sobre los desarrolladores de la aplicación e imágenes de las redes sociales en las cuales se los puede encontrar (twitter, linkedin, google plus, facebook), estas imágenes funcionan como enlaces, y finalmente un botón *Back* que se usa para regresar al menú principal de opciones. La interfaz que aparece cuando se selecciona la opción *Credits* es la siguiente:



Al dar clic en la opción *Device Information* aparece la siguiente interfaz:

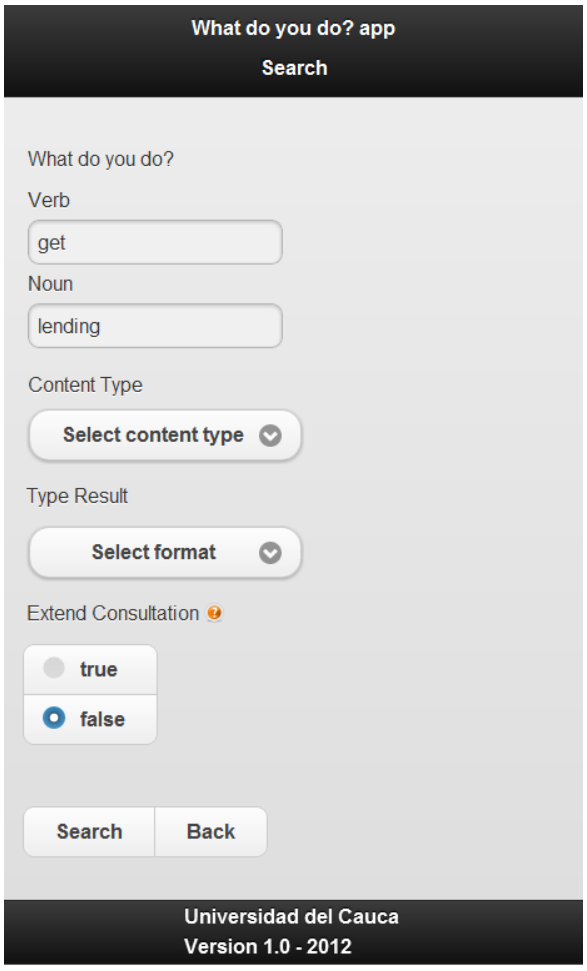


La información que se presenta en la anterior interfaz es la que se recupera con el uso de la API de WURFL, dicha información es recuperada cuando el usuario accede al menú principal o index de la aplicación, la interfaz de *Device Information* se usa simplemente para mostrar la información al usuario. En la parte de arriba se puede ver la IP desde donde se esta accediendo al servicio y el ID del agente, luego aparecen tres categorías de características del dispositivo: *Basic Information, Markup Information y Display Information.* Al dar clic sobre cada campo se despliega la información mas detallada de cada categoría, como se puede ver en la siguiente imagen:



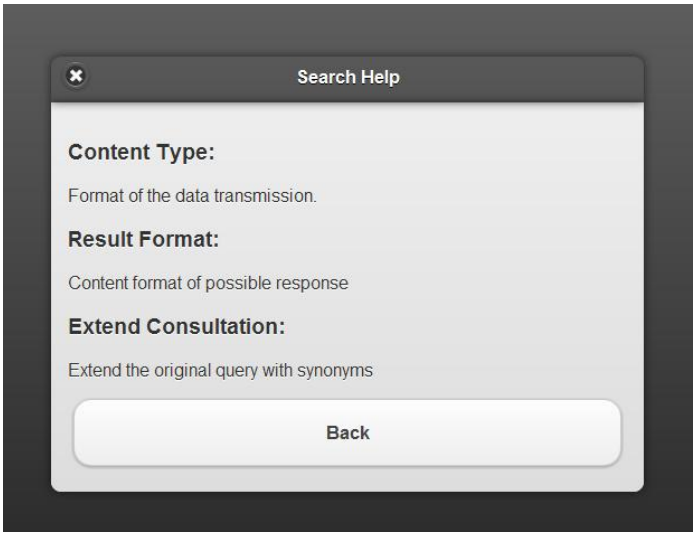
Finalmente aparece el botón *Back* que se usa para regresar al menú principal.

La opción principal y que tiene asociado la mayor cantidad de código es *Search,* opción que despliega la siguiente interfaz:



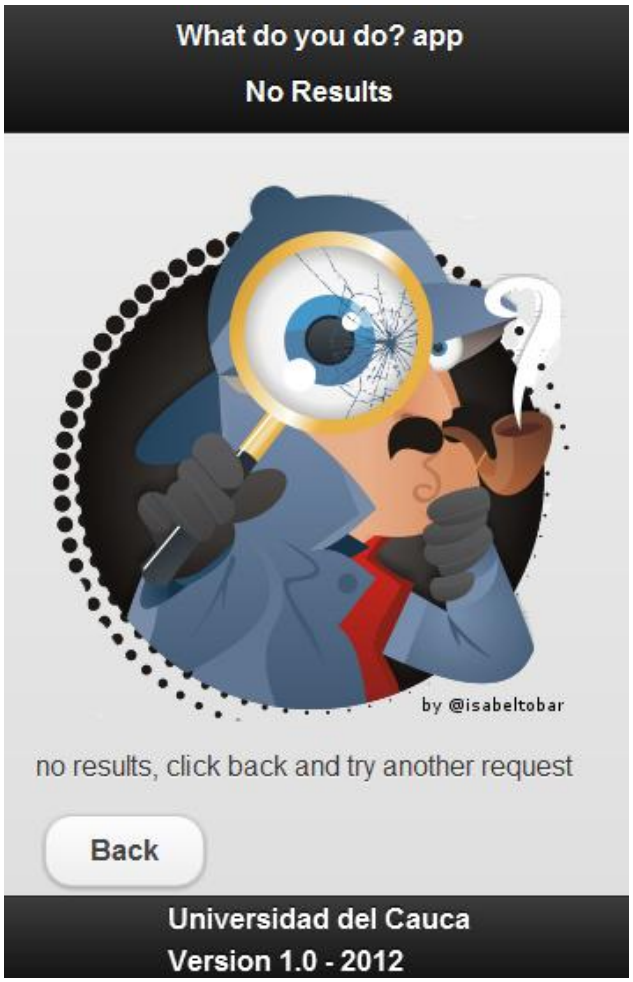
Se puede ver que es un formulario que recupera la solicitud de un usuario. Se le pide ingresar de forma obligatoria *verb* y *noun,* estos dos campos son suficientes para crear una solicitud formal y los servicios que pueden cumplir con los requerimientos del usuario. Las otras opciones sirven para filtrar los servicios desde la consulta inicial o para extender la consulta. *Content type* se usa para establecer el formato del contenido de la posible respuesta entre las siguientes posibilidades: text/html, text/plain, image/jpeg y aplicación/pdf; *Type Result* se usa para fijar el tipo de codificación que se realizara entre: XML o JSON, estas opciones requieren de cierto conocimiento técnico y por tal motivo su uso es opcional (se puede obtener información sobre el uso de estos campos no obligatorios dando clic en el botón). Finalmente se puede seleccionar si se desea extender la consulta, esta opción genera peticiones sinónimo de la original.

La interfaz que provee ayuda es un popup como el siguiente:



Finalmente aparecen dos botones: *Back* que se usa para regresar al menú principal y *Search* que se usa para lanzar la aplicación. En caso de que existan inconsistencias al momento de llenar el formulario (campos obligatorios vacíos principalmente) la aplicación notificará al usuario sobre esto.

Si la aplicación no encuentra servicios para la consulta aparece una interfaz en donde se notifica esto:



Si encuentra resultados, estos se despliegan en una tabla en donde aparece el nombre del método que cumplirá con los requerimientos del usuario y el botón que funciona como un enlace al sitio donde esta alojado el servicio, como lo muestra la siguiente figura.

ANEXO B

MANUALES DE INSTALACIÓN

A continuación se presenta la instalación de servidores y base de datos necesarias para el correcto funcionamiento de la plataforma desarrollada.

B.1. Fuseki

Fuseki es un servidor SPARQL que provee al administrador de funciones como actualización de servicios a través de métodos HTTP o haciendo uso de SPARQL, ejecución de consultas SPARQL sobre diferentes directorios que alojen servicios y la funcionalidad que para este trabajo de grado es de mayor interés: subir servicios a un directorio o repositorio.

B.1.1. Descargar servidor

Para descargar el servidor se accede a la siguiente dirección:

https://repository.apache.org/content/repositories/snapshots/org/apache/jena/jena-fuseki/

Dentro de esta dirección seleccionamos en enlace "*0.2.2-incubating-SNAPSHOT*" y posteriormente el archivo "*jena-fuseki-0.2.2-incubating-20120402.155625-10-distribution.zip*". Este archivo es el usado en el trabajo de grado, aunque no es la única forma de obtener el instalador.

B.1.2. Instalación y comienzo

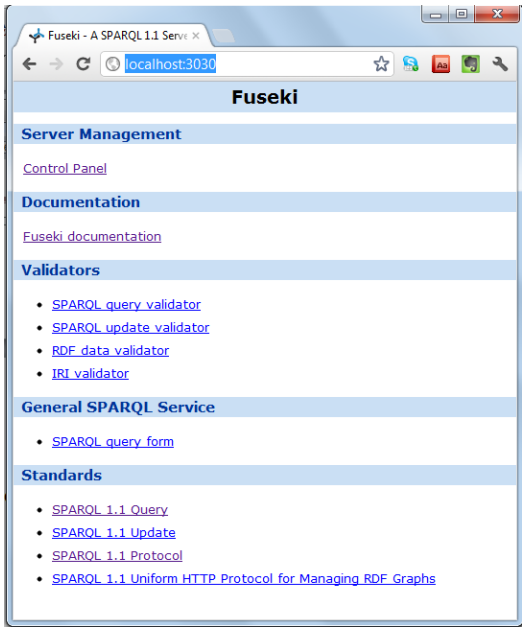
La instalación es bastante sencilla ya que solo se necesita descomprimir el archivo que se descargo previamente en cualquier directorio del computador (aunque es recomendable hacerlo en la carpeta en donde se instalan la mayoría de aplicaciones). Para arrancar el servidor es necesario ubicarse en la carpeta que al descomprimir el archivo se crea y hacer uso del siguiente comando:

java -jar fuseki-server.jar –loc = [repositorio] --mem /[ds]

repositorio es la dirección en donde se alojará el repositorio de servicios, esta dirección puede corresponder a cualquier carpeta del sistema, *ds* es el nombre que se le asigna al dataset que se crea al subir servicios. Cuando el servidor arranca con éxito aparecen las siguientes líneas en la consola (la cual queda funcionando como terminal de logs y transacciones).

```
11:05:16 INFO  Server              :: Dataset: in-memory
11:05:16 INFO  Server              :: TDB dataset: directory=C:\Users\SEGA\Desktop\Jena\sr
11:05:16 INFO  Server              :: Dataset path = /ds
11:05:16 INFO  Server              :: Fuseki 0.2.2-incubating-SNAPSHOT 20120402-1555
11:05:16 INFO  Server              :: Jetty 7.x.y-SNAPSHOT
11:05:16 INFO  Server              :: Started 2012/04/10 11:05:16 COT on port 3030
```

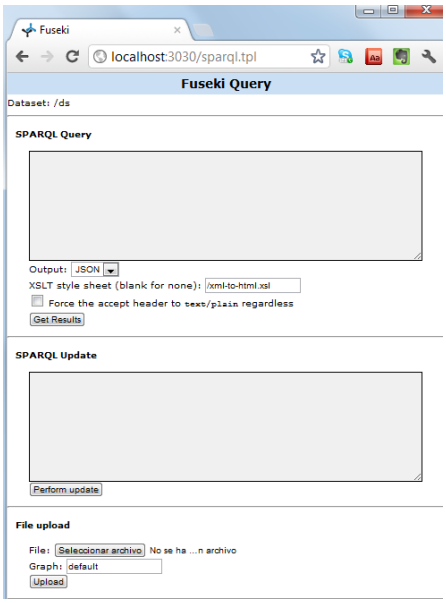
Otra forma de comprobar que el servidor arrancó correctamente es abrir un navegador e ingresar a la siguiente dirección http://localhost:3030/. La siguiente figura muestra lo que se debería desplegar.



Lo anterior aplica tanto para equipos con Windows como equipos con Linux.

B.1.3. Subir servicios

Para subir servicios y alimentar el repositorio que es usado desde la aplicación se debe ingresar a la siguiente dirección http://localhost:3030/ y dar clic en "*Control Panel*". Aparecen los *dataset* que están creados, se selecciona el adecuado y se da clic en "*Select*", se despliega lo siguiente:



En la parte superior se notifica que *dataset* se esta usando, luego aparece un cuadro de texto para escribir peticiones en SPARQL y poder ejecutarlas (esta herramienta puede ser usada para probar que los servicios fueron subidos con éxito), luego

aparece otro cuadro de texto en donde se puede escribir sentencias SPARQL para actualizar el repositorio y finalmente aparece una sección en donde se selecciona el servicio que se desea subir y el botón "*upload*" para poder hacerlo.

No es necesario que el servidor Fuseki este funcionando cuando la aplicación "*what do you do*" realiza las consultas, Fuseki solo se usa para alimentar el repositorio de servicios que usa la aplicación desarrollada para este trabajo de grado.

B.2. Jboss

JBoss 5.1.0.GA se eligió como el servidor que aloja la aplicación debido a que es muy usado actualmente, se puede obtener documentación sobre este fácilmente, es robusto, de uso es libre, fácil de configurar y de modificar.

B.2.1. Descargar servidor

Los paquetes de instalación de JBoss que se emplean en este trabajo de grado se obtienen de la siguiente dirección:

http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.1.0.GA/jboss-5.1.0.GA.zip

En caso de que no se tenga interfaz gráfica para obtener los paquetes, se puede hacer mediante wget en distribuciones linux:

```
wget <dirección del paquete de instalación>
```

B.2.2. Instalación

Para la instalación de JBoss solo se necesita descomprimir el archivo que se descargó en cualquier carpeta del sistema (se recomienda la carpeta /usr/src/ en Linux y Archivos de Programas en Windows). Para Linux podemos ejecutar los siguientes comandos en una consola:

```
cd /usr/src/
unzip jboss-5.1.0.GA.zip
```

B.2.2.1. Inicio y comprobación de la instalación

Arrancamos el servidor JBoss con los siguientes comandos en Linux:

```
cd /usr/src/jboos-5.1.0.GA/bin/
./run.sh -c all -b <dirección ip del nodo>
```

El comando en Windows es similar, solo se necesita remplazar *run.sh* por *run.bath*
El valor que se le da a –c corresponde a la configuración que se utiliza, en este caso *all* es una configuración que JBoss tiene por defecto y que carga todos los modulos. Las configuraciones se pueden encontrar el la carpeta *server*, en esta se puede crear configuraciones especiales.

Si no se ingresa el parámetro –b el servidor escucha peticiones por defecto en *localhost*

En un navegador se puede ingresar a la siguiente dirección para comprobar la instalación:

http://ip:8080/

Si el servidor está instalado y corriendo de forma correcta debe aparecer lo siguiente:

B.2.3. Desplegar aplicaciones

Desplegar aplicaciones en JBoss es bastante sencillo, se puede hacer por la interfaz de administración, aunque no es necesario, se parte del hecho de que la aplicación tiene su respectivo .war. Este archivo se debe copiar en la siguiente carpeta:

JBOSS_HOME\server\{conf}\deploy

En donde *JBOSS_HOME* es la dirección en donde quedo instalado JBoss y *conf* es el tipo de configuración que se uso al iniciar el servidor (parámetro -c). Por ejemplo la configuración *all* en la siguiente dirección:

C:\jboss-5.1.0.GA\server\all\deploy

B.2.3. Referencias

http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html/clustering-http.html#clustering-http-modjk

B.3. WordNet

La instalación de WordNet es bastante sencilla, solo se necesita descargar el .exe o .tar.gz de la siguiente dirección:

http://wordnet.princeton.edu/wordnet/download/current-version/

En nuestro caso, WordNet fue instalado en Windows, asi que solo se necesita ejecutar el archivo .exe.

ANEXO C

ARCHIVOS

Ejemplo de un servicio descrito con hRESTS (incluyendo las propiedades definidas):

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wsdl20="http://www.w3.org/ns/wsdl"
     xmlns:xhtml="http://www.w3.org/1999/xhtml"
     xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
     xmlns:nhr="http://localhost:8084/RESTWSDiscovery/hrests#"
     xml:lang="en"
     lang="en">
   <head profile="http://www.w3.org/2003/g/data-view">
     <title>MissileFinancing service API</title>
     <link rel="transformation" nhref="nhrests.xslt"/>
   </head>
   <body>
     <div class="service" id="MissileFinancing">
       <h1>
         <span class="label">MissileFinancing</span>
                                                    service API
                    </h1>
       <div class="operation" id="get_FINANCING">
         <h2>
          Operation
          <code class="label">get_FINANCING</code>
         </h2>
         <p>
          Invoked using the
          <span class="method">POST</span>
          at
<code class="address">
http://127.0.0.1/services/sawsdl_wsdl11/MissileFinancing/get_FINANCING</code>
         </p>
         <p>
           <strong>Parameters:</strong>
           <br/>
           <span class="input" id="MissileType">
             <a rel="model" nhref="http://127.0.0.1/ontology/Mid-level-ontology.owl#Missile">
               <code>MissileType</code>
             </a>
           </span>
           <br/>
         </p>
         <p>
           <strong>Output value(s):</strong>
           <br/>
           <span class="output" id="FinancingType">
             <a rel="model" nhref="http://127.0.0.1/ontology/Mid-level-ontology.owl#Financing">
               <code>FinancingType</code>
             </a>
           </span>
           <br/>
         </p>
              <!-- New nhrESTS Properties -->
```

```
                                <p>
            <strong>Payload format:</strong>
            <br/>
            <span class="payload">JSON</span>
            <br/>
                                    <strong>Content type:</strong>
            <br/>
            <span class="contentType">text/html</span>
            <br/>
          </p>
          <!-- End -->
        </div>
      </div>
    </body>
</html>
```

Archivo de descripción XSLT para transformar descripciones hRESTS a RDF con el objetivo de guardarlas en un repositorio:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xml="http://www.w3.org/XML/1998/namespace"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xml:space="default"
   xmlns:xhtml="http://www.w3.org/1999/xhtml"
   xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
   xmlns:nhr="http://localhost:8084/RESTWSDiscovery/hrests#"
   xmlns:wsl="http://www.wsmo.org/ns/wsmo-lite#"
   xmlns:msm="http://cms-wg.sti2.org/ns/minimal-service-model#"
   >
<xsl:output indent="yes" />
   <!-- this template parses hRESTS and MicroWSMO microformats into RDF              -->
   <!-- it is intended to be used as a GRDDL transformation.                        -->
   <!-- this template does not do much input validation, so garbage-in-garbage-out  -->
   <xsl:template match="/">
      <rdf:RDF>
         <xsl:choose>
            <xsl:when test="//*[contains(concat(' ',normalize-space(@class),' '),' service ')]">
               <xsl:for-each select="//*[contains(concat(' ',normalize-space(@class),' '),' service ')]">
                  <msm:Service>
                     <xsl:if test="@id">
                        <xsl:attribute name="rdf:ID"><xsl:value-of select="@id"/></xsl:attribute>
                     </xsl:if>
                     <!-- rdfs:isDefinedBy rdf:resource=""/ -->
                     <xsl:apply-templates mode="servicelabel" select="*" />
                     <xsl:apply-templates mode="microwsmo" select="*" />
                     <xsl:apply-templates mode="localmicrowsmo" select="." />
                     <xsl:apply-templates mode="operation" select="*"/>
                  </msm:Service>
               </xsl:for-each>
            </xsl:when>
            <xsl:otherwise>
               <msm:Service>
                  <rdfs:isDefinedBy rdf:resource=""/>
                  <xsl:apply-templates mode="operation" select="*"/>
               </msm:Service>
            </xsl:otherwise>
         </xsl:choose>
      </rdf:RDF>
   </xsl:template>
   <xsl:template match="*[contains(concat(' ',normalize-space(@class),' '),' operation ')]" mode="operation">
      <msm:hasOperation>
         <msm:Operation>
            <xsl:if test="@id">
               <xsl:attribute name="rdf:ID"><xsl:value-of select="@id"/></xsl:attribute>
            </xsl:if>
            <xsl:apply-templates mode="operationlabel" select="*"/>
                                          <xsl:choose>
               <xsl:when test=".//*[contains(concat(' ',normalize-space(@class),' '),' method ')]">
```

```
            <xsl:apply-templates mode="operationmethod" select="*"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates mode="reverseservicemethod" select="."/>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:choose>
          <xsl:when test=".//*[contains(concat(' ',normalize-space(@class),' '),' address ')]">
            <xsl:apply-templates mode="operationaddress" select="*"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates mode="reverseserviceaddress" select="."/>
          </xsl:otherwise>
        </xsl:choose>
                                        <!--This parses the new payload hRESTS property  -->
                                        <xsl:choose>
          <xsl:when test=".//*[contains(concat(' ',normalize-space(@class),' '),' payload ')]">
            <xsl:apply-templates mode="operationpayload" select="*"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates mode="reverseservicepayload" select="."/>
          </xsl:otherwise>
        </xsl:choose>
                                        <!-- End of the new hRESTS property process -->
                                        <!--This parses the new contentType hRESTS property  -->
                                        <xsl:choose>
          <xsl:when test=".//*[contains(concat(' ',normalize-space(@class),' '),' contentType ')]">
            <xsl:apply-templates mode="operationcontentType" select="*"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates mode="reverseservicecontentType" select="."/>
          </xsl:otherwise>
        </xsl:choose>
                                        <!-- End of the new hRESTS property process -->
        <xsl:apply-templates mode="microwsmo" select="*" />
        <xsl:apply-templates mode="localmicrowsmo" select="." />
        <xsl:apply-templates mode="operationinput" select="*"/>
        <xsl:apply-templates mode="operationoutput" select="*"/>
      </msm:Operation>
    </msm:hasOperation>
  </xsl:template>
          <!-- New template operationpayload -->
          <xsl:template match="*" mode="operationpayload">
                    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                contains(concat(' ',normalize-space(@class),' '),' output ') or
                contains(concat(' ',normalize-space(@class),' '),' operation ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' payload ')">
        <xsl:call-template name="payload"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="operationpayload" select="*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
          <!-- New template reverseservicepayload -->
          <xsl:template match="*" mode="reverseservicepayload">
                    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' service ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' payload ')">
        <xsl:call-template name="payload"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="reverseservicepayload" select="parent::*"/>
        <xsl:apply-templates mode="operationpayload" select="preceding-sibling::*"/>
        <xsl:apply-templates mode="operationpayload" select="following-sibling::*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
          <!-- New template payload -->
          <xsl:template name="payload">
                    <hr:hasPayloadFormat>
      <xsl:choose>
        <xsl:when test="@title"><xsl:value-of select="@title"/></xsl:when>
        <xsl:when test="@href"><xsl:value-of select="@href"/></xsl:when>
        <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
```

```
        </xsl:choose>
    </hr:hasPayloadFormat>
</xsl:template>
            <!-- New template operationcontentType -->
            <xsl:template match="*" mode="operationcontentType">
                        <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                    contains(concat(' ',normalize-space(@class),' '),' output ') or
                    contains(concat(' ',normalize-space(@class),' '),' operation ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' contentType ')">
            <xsl:call-template name="contentType"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="operationcontentType" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
            <!-- New template contentType -->
            <xsl:template name="contentType">
                        <hr:hasContentType>
        <xsl:choose>
            <xsl:when test="@title"><xsl:value-of select="@title"/></xsl:when>
            <xsl:when test="@href"><xsl:value-of select="@href"/></xsl:when>
            <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
        </xsl:choose>
    </hr:hasContentType>
</xsl:template>
            <!-- New template reverseservicepayload -->
            <xsl:template match="*" mode="reverseservicecontentType">
                        <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' service ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' contentType ')">
            <xsl:call-template name="payload"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="reverseservicecontentType" select="parent::*"/>
            <xsl:apply-templates mode="operationcontentType" select="preceding-sibling::*"/>
            <xsl:apply-templates mode="operationcontentType" select="following-sibling::*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template match="*" mode="operation">
    <xsl:apply-templates mode="operation" select="*"/>
</xsl:template>
<xsl:template match="*" mode="servicelabel">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' operation ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' label ')">
            <xsl:call-template name="label"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="servicelabel" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template match="*" mode="operationlabel">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                    contains(concat(' ',normalize-space(@class),' '),' output ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' label ')">
            <xsl:call-template name="label"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="operationlabel" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template name="label">
    <rdfs:label>
        <xsl:choose>
            <xsl:when test="@title"><xsl:value-of select="@title"/></xsl:when>
            <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
        </xsl:choose>
    </rdfs:label>
</xsl:template>
<xsl:template match="*" mode="operationmethod">
```

```
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                      contains(concat(' ',normalize-space(@class),' '),' output ') or
                      contains(concat(' ',normalize-space(@class),' '),' operation ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' method ')">
        <xsl:call-template name="method"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="operationmethod" select="*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="node()" mode="reverseservicemethod">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' service ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' method ')">
        <xsl:call-template name="method"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="reverseservicemethod" select="parent::*"/>
        <xsl:apply-templates mode="operationmethod" select="preceding-sibling::*"/>
        <xsl:apply-templates mode="operationmethod" select="following-sibling::*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template name="method">
    <hr:hasMethod>
      <xsl:variable name="value">
        <xsl:choose>
          <xsl:when test="@title"><xsl:value-of select="@title"/></xsl:when>
          <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:choose>
        <xsl:when test="$value='GET'    or $value='get'    or $value='Get'"   >GET</xsl:when>
        <xsl:when test="$value='PUT'    or $value='put'    or $value='Put'"   >PUT</xsl:when>
        <xsl:when test="$value='POST'   or $value='post'   or $value='Post'"  >POST</xsl:when>
        <xsl:when test="$value='DELETE' or $value='delete' or $value='Delete'">DELETE</xsl:when>
        <xsl:otherwise>
          <xsl:message terminate="no">Unknown HTTP method: <xsl:value-of select="$value"/></xsl:message>
          <xsl:value-of select="'GET'"/>
        </xsl:otherwise>
      </xsl:choose>
    </hr:hasMethod>
  </xsl:template>
  <xsl:template match="*" mode="operationaddress">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                      contains(concat(' ',normalize-space(@class),' '),' output ') or
                      contains(concat(' ',normalize-space(@class),' '),' operation ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' address ')">
        <xsl:call-template name="address"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="operationaddress" select="*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="node()" mode="reverseserviceaddress">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' service ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' address ')">
        <xsl:call-template name="address"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="reverseserviceaddress" select="parent::*"/>
        <xsl:apply-templates mode="operationaddress" select="preceding-sibling::*"/>
        <xsl:apply-templates mode="operationaddress" select="following-sibling::*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template name="address">
    <hr:hasAddress rdf:datatype="http://www.wsmo.org/ns/hrests#URITemplate">
      <xsl:choose>
        <xsl:when test="@title"><xsl:value-of select="@title"/></xsl:when>
        <xsl:when test="@href"><xsl:value-of select="@href"/></xsl:when>
```

```
            <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
        </xsl:choose>
    </hr:hasAddress>
</xsl:template>
<xsl:template match="*" mode="operationinput">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' output ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ')">
            <xsl:call-template name="input"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="operationinput" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template match="*" mode="operationoutput">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ')" />
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' output ')">
            <xsl:call-template name="output"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="operationoutput" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template name="input">
    <msm:hasInput>
        <msm:MessageContent>
            <xsl:if test="@id">
                <xsl:attribute name="rdf:ID"><xsl:value-of select="@id"/></xsl:attribute>
            </xsl:if>
            <xsl:apply-templates mode="messagelabel" select="*"/>
            <xsl:apply-templates mode="parameter" select="*"/>
            <xsl:apply-templates mode="microwsmo" select="*" />
            <xsl:apply-templates mode="localmicrowsmo" select="." />
        </msm:MessageContent>
    </msm:hasInput>
</xsl:template>
<xsl:template name="output">
    <msm:hasOutput>
        <msm:MessageContent>
            <xsl:if test="@id">
                <xsl:attribute name="rdf:ID"><xsl:value-of select="@id"/></xsl:attribute>
            </xsl:if>
            <xsl:apply-templates mode="messagelabel" select="*"/>
            <xsl:apply-templates mode="parameter" select="*"/>
            <xsl:apply-templates mode="microwsmo" select="*" />
            <xsl:apply-templates mode="localmicrowsmo" select="." />
        </msm:MessageContent>
    </msm:hasOutput>
</xsl:template>
<xsl:template match="*" mode="messagelabel">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' parameter ')"/>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' label ')">
            <xsl:call-template name="label"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="messagelabel" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template match="*" mode="parameter">
    <xsl:choose>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' label ')"/>
        <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' parameter ')">
            <xsl:call-template name="parameter"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:apply-templates mode="parameter" select="*"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<xsl:template name="parameter">
    <xsl:choose>
```

```
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' mandatory ')">
        <msm:hasMandatoryPart>
          <xsl:call-template name="parameterbody"/>
        </msm:hasMandatoryPart>
      </xsl:when>
      <xsl:otherwise>
        <msm:hasOptionalPart>
          <xsl:call-template name="parameterbody"/>
        </msm:hasOptionalPart>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template name="parameterbody">
    <msm:MessagePart>
      <xsl:if test="@id">
        <xsl:attribute name="rdf:ID"><xsl:value-of select="@id"/></xsl:attribute>
      </xsl:if>
      <xsl:apply-templates mode="parameterlabel" select="*"/>
      <xsl:apply-templates mode="microwsmo" select="*" />
      <xsl:apply-templates mode="localmicrowsmo" select="." />
    </msm:MessagePart>
  </xsl:template>
  <xsl:template match="*" mode="parameterlabel">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' label ')">
        <xsl:call-template name="label"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="parameterlabel" select="*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="*" mode="microwsmo">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@class),' '),' input ') or
                      contains(concat(' ',normalize-space(@class),' '),' output ') or
                      contains(concat(' ',normalize-space(@class),' '),' parameter ') or
                      contains(concat(' ',normalize-space(@class),' '),' operation ')" />
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' model ')">
        <xsl:call-template name="model"/>
      </xsl:when>
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' lifting ')">
        <xsl:call-template name="lifting"/>
      </xsl:when>
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' lowering ')">
        <xsl:call-template name="lowering"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates mode="microwsmo" select="*"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="*" mode="localmicrowsmo">
    <xsl:choose>
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' model ')">
        <xsl:call-template name="model"/>
      </xsl:when>
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' lifting ')">
        <xsl:call-template name="lifting"/>
      </xsl:when>
      <xsl:when test="contains(concat(' ',normalize-space(@rel),' '),' lowering ')">
        <xsl:call-template name="lowering"/>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
  <xsl:template name="replace">
    <xsl:param name="data"/>
    <xsl:param name="pattern"/>
    <xsl:param name="replacement"/>
    <xsl:choose>
      <xsl:when test="contains($data,$pattern)">
        <xsl:value-of select="concat(substring-before($data,$pattern),$replacement)"/>
        <xsl:call-template name="replace">
          <xsl:with-param name="data" select="substring-after($data,$pattern)"/>
          <xsl:with-param name="pattern" select="$pattern"/>
          <xsl:with-param name="replacement" select="$replacement"/>
```

```
            </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$data"/>
          </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
    <xsl:template name="model">
      <sawsdl:modelReference>
        <xsl:attribute name="rdf:resource">
          <xsl:call-template name="replace">
            <xsl:with-param name="data" select="@href"/>
            <xsl:with-param name="pattern" select="string(' ')"/>
            <xsl:with-param name="replacement" select="string('%20')"/>
          </xsl:call-template>
        </xsl:attribute>
      </sawsdl:modelReference>
      <!-- <sawsdl:modelReference rdf:resource="{replace(@href, ' ', '%20')}"/>  - this would work with xslt 2.0 -->
    </xsl:template>
    <xsl:template name="lifting">
      <sawsdl:liftingSchemaMapping rdf:resource="{@href}"/>
    </xsl:template>
    <xsl:template name="lowering">
      <sawsdl:loweringSchemaMapping rdf:resource="{@href}"/>
    </xsl:template>
    <!-- todo add support for link and form operations -->
</xsl:stylesheet>
```

Finalmente, el servicio en RDF que se obtiene al transformar la descripción hRESTS:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:xhtml="http://www.w3.org/1999/xhtml"
        xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
        xmlns:nhr="http://localhost:8084/RESTWSDiscovery/hrests#"
        xmlns:wsl="http://www.wsmo.org/ns/wsmo-lite#"
        xmlns:msm="http://cms-wg.sti2.org/ns/minimal-service-model#">
  <msm:Service rdf:ID="MissileFinancingSoap">
    <rdfs:label>MissileFinancing</rdfs:label>
    <msm:hasOperation>
      <msm:Operation rdf:ID="get_FINANCING">
        <rdfs:label>get_FINANCING</rdfs:label>
        <nhr:hasMethod>POST</nhr:hasMethod>
        <nhr:hasAddress
rdf:datatype="http://www.wsmo.org/ns/hrests#URITemplate">http://127.0.0.1/services/sawsdl_wsdl11/
MissileFinancing/get_FINANCING</nhr:hasAddress>
        <nhr:hasPayloadFormat>JSON</nhr:hasPayloadFormat>
        <nhr:hasContentType>text/html</nhr:hasContentType>
        <msm:hasInput>
          <msm:MessageContent rdf:ID="MissileType">
            <sawsdl:modelReference rdf:resource="http://127.0.0.1/ontology/Mid-level-
ontology.owl#Missile"/>
          </msm:MessageContent>
        </msm:hasInput>
        <msm:hasOutput>
          <msm:MessageContent rdf:ID="FinancingType">
            <sawsdl:modelReference rdf:resource="http://127.0.0.1/ontology/Mid-level-
ontology.owl#Financing"/>
          </msm:MessageContent>
        </msm:hasOutput>
      </msm:Operation>
    </msm:hasOperation>
  </msm:Service>
</rdf:RDF>
```

ANEXO D

PRUEBAS

A continuación se presentan tabulados los resultados de las pruebas descritas en el capitulo 5.

D.1. Tiempos de respuesta

- Tiempo que tarde la aplicación en crear una conexión con WURFL e instanciar las clases necesarias para obtener la información de los dispositivos móviles.

Prueba	Tiempo (seg)
1	0,030
2	0,040
3	0,030
4	0,040
5	0,030
6	0,040
7	0,030
8	0,030
9	0,030
10	0,040
11	0,040
12	0,040
13	0,030
14	0,040
15	0,030
16	0,030
17	0,040
18	0,030
19	0,030
20	0,030
Promedio	0,03

- Tiempo que tarda la aplicación en crear una conexión a WordNet e instanciar las clases que se necesitan para ejecutar las consultas sobre esta base de datos lexical.

Prueba	Tiempo (seg)
1	3,523
2	3,907
3	3,353
4	3,701

5	3,356
6	3,376
7	3,520
8	3,706
9	3,710
10	3,700
11	3,600
12	3,520
13	3,630
14	3,389
15	3,520
16	3,643
17	3,615
18	3,398
19	3,521
20	3,397
Promedio	3,554

- Tiempo que tarda la aplicación en crear una conexión con el repositorio de servicios e instanciar las clases que son capaces de ejecutar una consulta y recuperar información de ésta.

Prueba	Tiempo (seg)
1	0,343
2	0,415
3	0,333
4	0,390
5	0,403
6	0,320
7	0,356
8	0,296
9	0,289
10	0,389
11	0,321
12	0,289
13	0,345
14	0,330
15	0,311
16	0,307
17	0,298
18	0,296
19	0,299
20	0,291

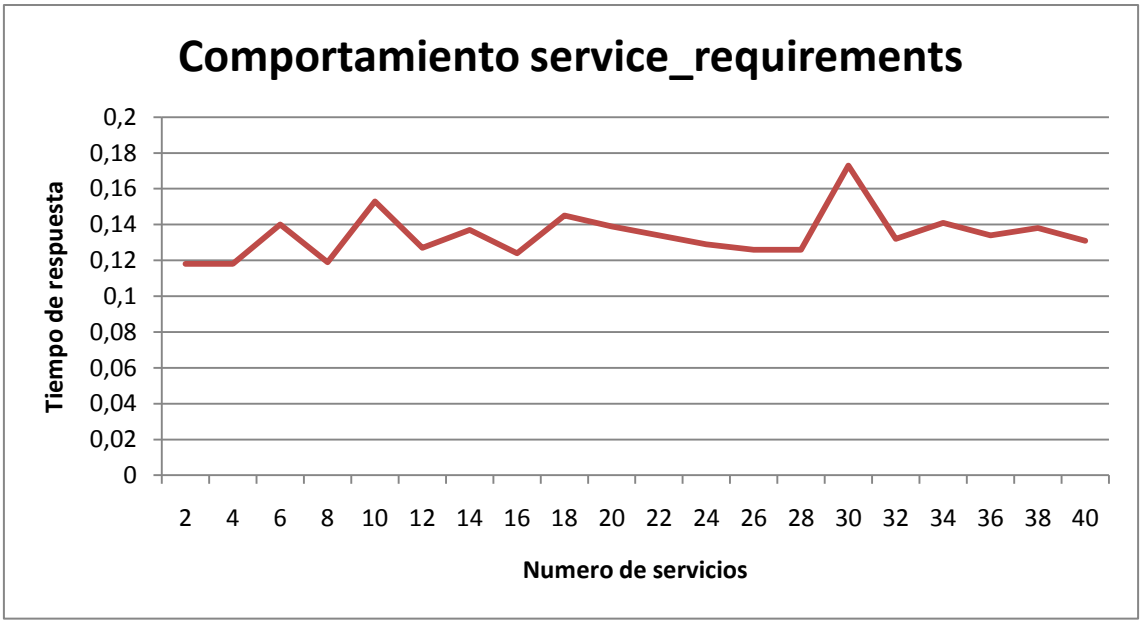
Promedio	0,331

- Tiempo que la aplicación tarda en cargar el archivo services_requirements.xml, archivo usado para obtener los requerimientos mínimos para cada servicio. Como este archivo crece a medida que crece el numero de servicios alojados en el repositorio de servicios. Se obtiene los tiempos de respuesta a medida que el número de servicios crece.

  En la siguiente tabla se presenta los resultados para un número total de servicios igual al número de servicios empleados en el resto de pruebas.

Prueba	Numero de servicios	Tiempo (seg)
1	2	0,118
2	4	0,118
3	6	0,14
4	8	0,119
5	10	0,153
6	12	0,127
7	14	0,137
8	16	0,124
9	18	0,145
10	20	0,139
11	22	0,134
12	24	0,129
13	26	0,126
14	28	0,126
15	30	0,173
16	32	0,132
17	34	0,141
18	36	0,134
19	38	0,138
20	40	0,131

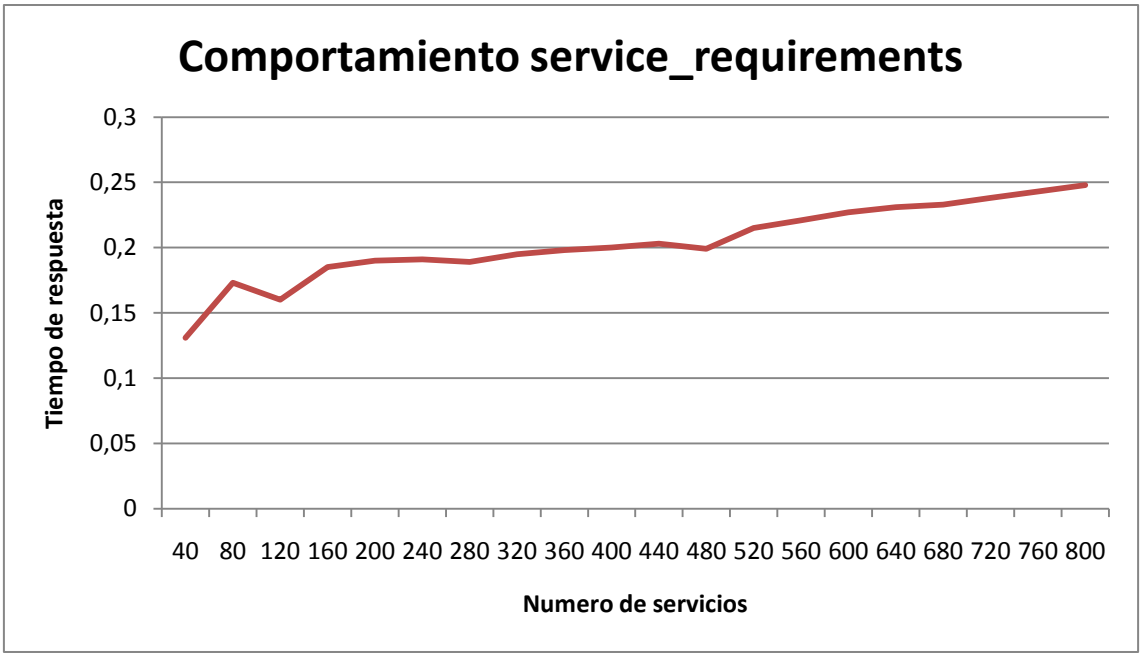
  La siguiente figura muestra la distribución de los anteriores resultados.



Finalmente se determino el crecimiento tomando como número limite 800 servicios, los datos obtenidos son los siguientes.

Prueba	Numero de servicios	Tiempo (seg)
1	40	0,131
2	80	0,173
3	120	0,16
4	160	0,185
5	200	0,19
6	240	0,191
7	280	0,189
8	320	0,195
9	360	0,198
10	400	0,2
11	440	0,203
12	480	0,199
13	520	0,215
14	560	0,221
15	600	0,227
16	640	0,231
17	680	0,233
18	720	0,238
19	760	0,243
20	800	0,248

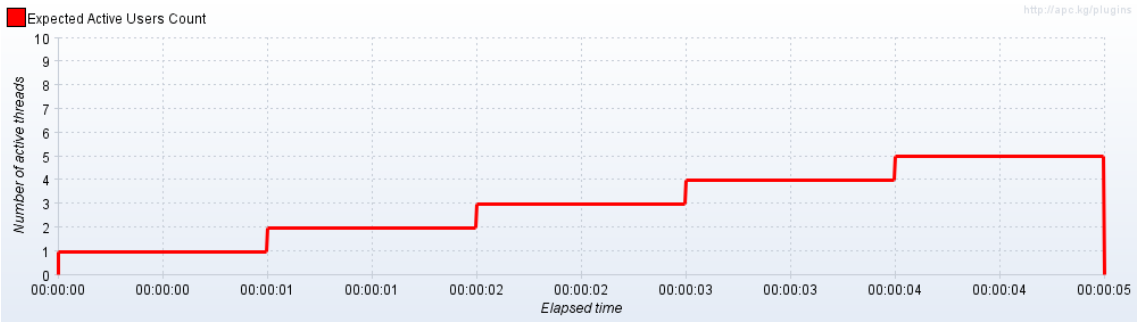
La figura que representa los anteriores datos es la siguiente.



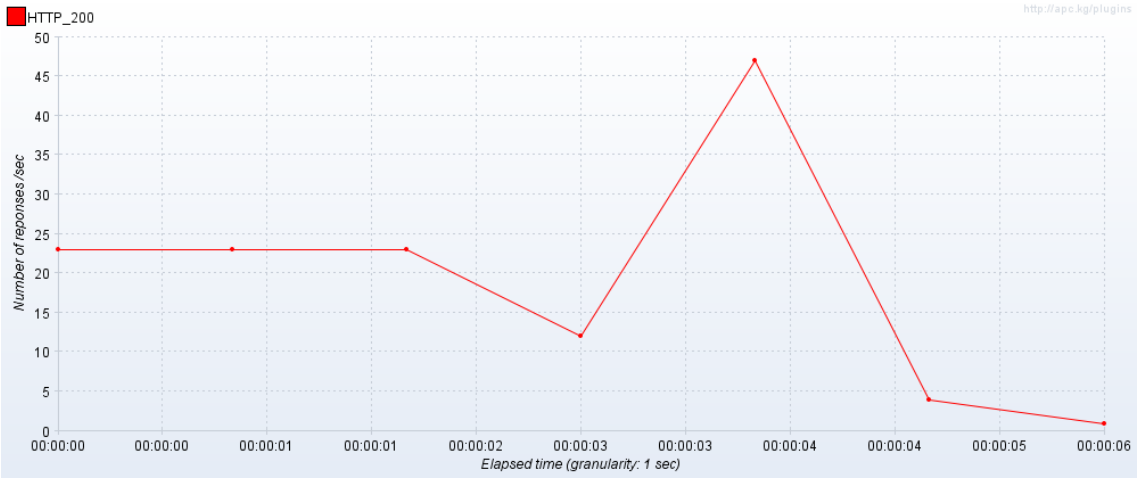
D.2. Pruebas de estrés

También se realizaron pruebas de estrés con JMeter, los resultados son los siguientes.

Para una distribución de 5 peticiones lanzadas a la aplicación en un lapso de tiempo de 5 segundos, como lo muestra la siguiente figura:

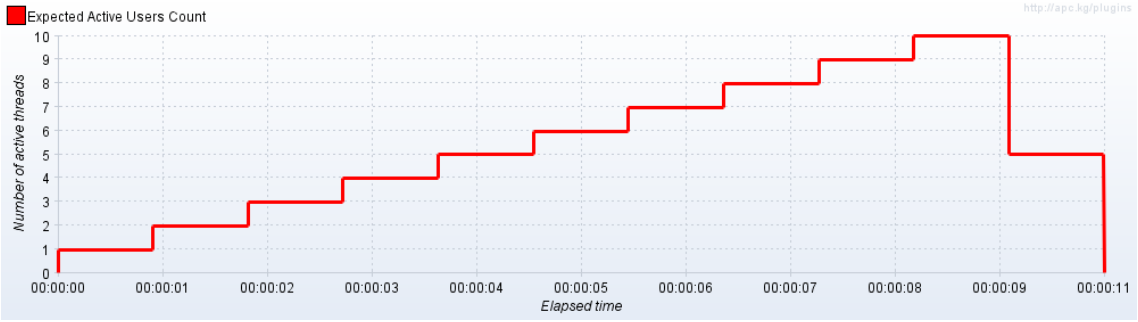


Se obtuvo el siguiente resultado:

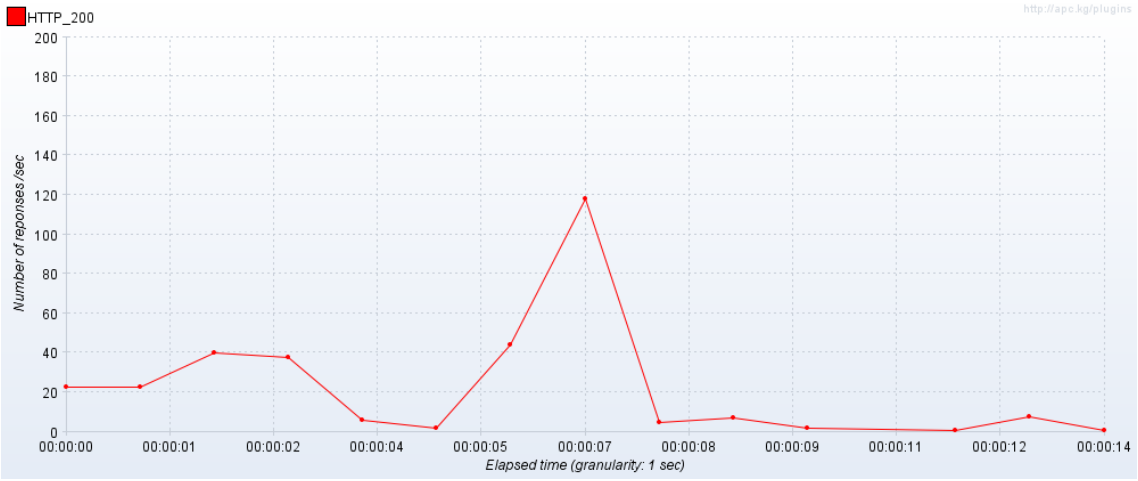


Una solicitud es una navegación completa por la aplicación y finaliza con en lanzamiento de una consulta SPARQL

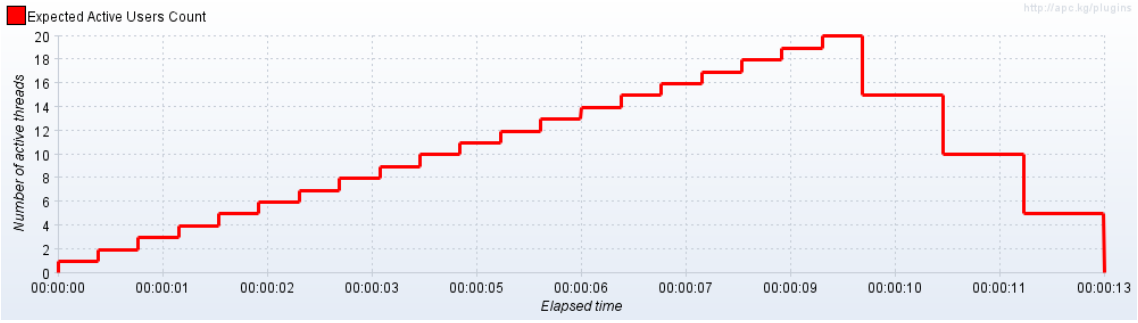
En el siguiente caso el número de peticiones sube hasta 10, y son lanzadas en menos de 10 segundos.



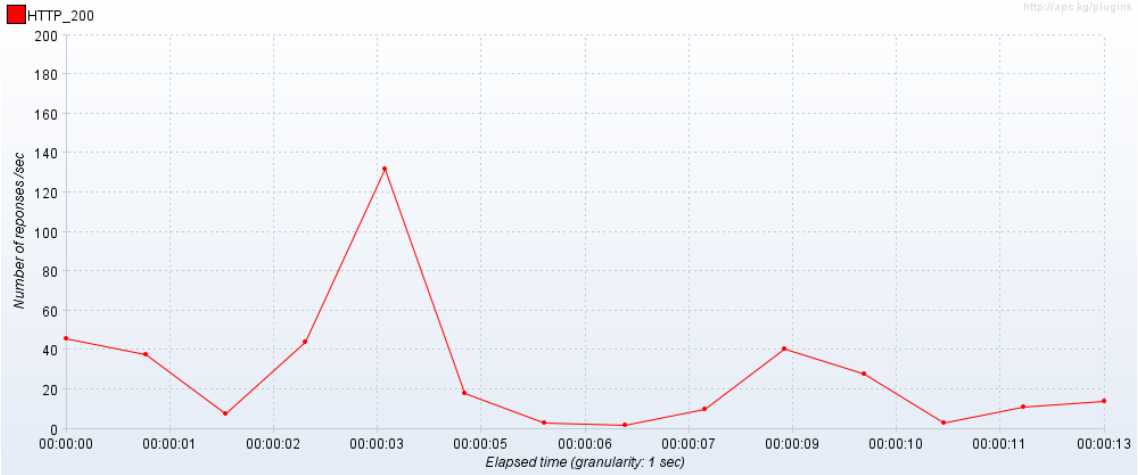
La respuesta obtenida por parte del servidor es la siguiente:



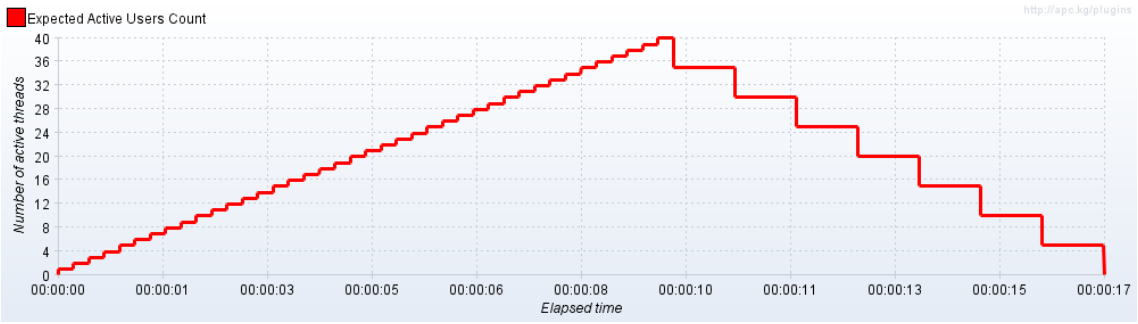
Ahora el numero de peticiones es de 20 en los mismos 10 segundos.



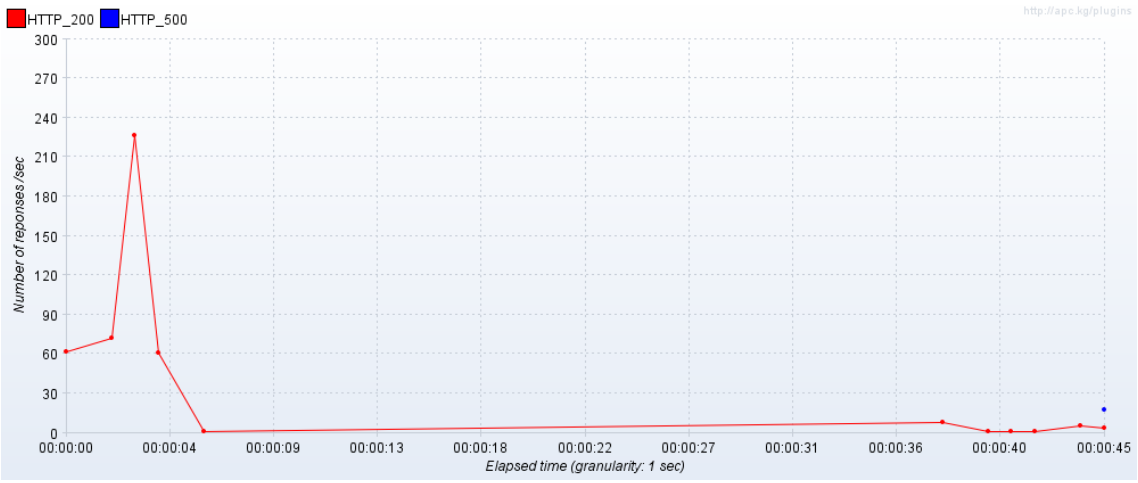
La respuesta obtenida es la siguiente:



Finalmente con 40 peticiones se hace que el servicio empiece a fallar (aparece el primer HTTP 500)



El resultado obtenido es el siguiente:



D.3. Resultados del algoritmo

El benchmark de referencia es el siguiente:

Servicios Ofertados	Consultas		
	Government Missile Funding Service	Government Missile Financing Service	Missile Government lending range
Government organization missile funding range service	Altamente Relevante	Altamente Relevante	Altamente Relevante

Government funding service	Altamente Relevante	Altamente Relevante	Altamente Relevante
Missile government giving borrow service	Altamente Relevante	Altamente Relevante	Relevante
Government missile funding range service	Altamente Relevante	Relevante	Altamente Relevante
Gov. Missiles and weapons funding service	Altamente Relevante	Altamente Relevante	Relevante
Gov. Missile funding reliable service	Altamente Relevante	Altamente Relevante	Altamente Relevante
Missile gov. giving service	Altamente Relevante	Altamente Relevante	Relevante
Gov. Missile giving range service	Altamente Relevante	Relevante	Altamente Relevante
Missile gov. Organization funding service	Altamente Relevante	Altamente Relevante	Relevante
Gov. organization self powered device funding service	Altamente Relevante	Relevante	No relevante
Gov. Missiles and weapons funding service	Altamente Relevante	Altamente Relevante	Relevante
Gov. Organization missile giving range service	Altamente Relevante	Potencialmente Relevante	Altamente Relevante
Gov. Projectile weapon funding service	Altamente Relevante	Altamente Relevante	Relevante
Gov. Missile funding service	Altamente Relevante	Altamente Relevante	Relevante
Asian missiles funding service	Relevante	Relevante	Relevante
Gov. Ballistic missile financing service	Relevante	Relevante	Relevante
Gov. Organization missile financing range service	Relevante	Relevante	Altamente Relevante
Gov. Organization ballistic missile funding range service	Relevante	Potencialmente Relevante	Altamente Relevante
Gov. Ballistic missile lending service	Relevante	Relevante	Relevante
Gov. Organization ballistic missile financing range service	Relevante	No Relevante	Altamente Relevante
Iraq missiles and mass destruction weapons funding service	Relevante	Potencialmente Relevante	No Relevante
Russian missiles financing service	Relevante	Relevante	Relevante
Gov. Organization ballistic missile giving range service	Relevante	Relevante	Altamente Relevante
Us missiles financing service	Relevante	Relevante	Relevante
Gov. Missile financing range service	Relevante	Relevante	Altamente Relevante
China missiles financing service	Relevante	Relevante	Relevante
Gov. Ballistic missile giving service	Relevante	Relevante	Relevante
Pakistan missiles funding service	Relevante	Relevante	Relevante
National gov. Weapon funding service	Relevante	Relevante	Potencialmente Relevante
Gov. Funding abomb service	Relevante	Potencialmente Relevante	No Relevante
N. Korea missiles funding service	Relevante	Relevante	Relevante

Government weapon funding service	Relevante	Relevante	Potencialmente Relevante
India missiles funding service	Relevante	Relevante	Relevante
Gov. Missiles financing service*	Relevante	Relevante	Relevante
Gov. Funding Ball. Missile service	Relevante	Potencialmente Relevante	Potencialmente Relevante
Gov. Ballistic missile funding service	Relevante	Relevante	Potencialmente Relevante
Gov. Missile lending range service	Potencialmente relevante	Relevante	Altamente Relevante
Gov. Organization ballistic missile lending range service	Potencialmente relevante	Relevante	Relevante
Gov. Organization missile lending range service	Potencialmente relevante	Potencialmente Relevante	Altamente Relevante
Gov. Org. Missiles uniliteral giving service	Potencialmente relevante	Potencialmente Relevante	No Relevante

Benchmark de referencia

Los resultados obtenidos con el algoritmo adaptado son los siguientes (lo deseable es recuperar los servicios relevantes y altamente relevantes):

Servicios Ofertados	**Consultas**		
	Government Missile Funding Service	**Government Missile Financing Service**	**Missile Government lending range**
Government organization missile funding range service	recuperado	recuperado	recuperado
Government funding service	recuperado	recuperado	recuperado
Missile government giving borrow service	recuperado	no recuperado	no recuperado
Government missile funding range service	recuperado	Relevante	recuperado
Gov. Missiles and weapons funding service	recuperado	recuperado	recuperado
Gov. Missile funding reliable service	recuperado	recuperado	recuperado
Missile gov. giving service	no recuperado	recuperado	no recuperado
Gov. Missile giving range service	recuperado	no recuperado	no recuperado
Missile gov. Organization funding service	recuperado	recuperado	recuperado
Gov. organization self powered device funding service	recuperado	recuperado	no recuperado
Gov. Missiles and weapons funding service	recuperado	recuperado	recuperado
Gov. Organization missile giving range service	no recuperado	no recuperado	no recuperado
Gov. Projectile weapon funding service	recuperado	recuperado	recuperado
Gov. Missile funding service	recuperado	recuperado	recuperado
Asian missiles funding service	recuperado	recuperado	recuperado
Gov. Ballistic missile financing service	recuperado	recuperado	recuperado
Gov. Organization missile financing range service	recuperado	recuperado	recuperado
Gov. Organization ballistic missile funding range service	recuperado	no recuperado	recuperado
Gov. Ballistic missile lending service	recuperado	no recuperado	recuperado
Gov. Organization ballistic missile	recuperado	no recuperado	recuperado

financing range service			
Iraq missiles and mass destruction weapons funding service	recuperado	no recuperado	no recuperado
Russian missiles financing service	recuperado	recuperado	recuperado
Gov. Organization ballistic missile giving range service	no recuperado	no recuperado	no recuperado
Us missiles financing service	recuperado	recuperado	recuperado
Gov. Missile financing range service	recuperado	recuperado	recuperado
China missiles financing service	recuperado	recuperado	recuperado
Gov. Ballistic missile giving service	no recuperado	recuperado	no recuperado
Pakistan missiles funding service	recuperado	recuperado	recuperado
National gov. Weapon funding service	recuperado	recuperado	recuperado
Gov. Funding abomb service	recuperado	no recuperado	no recuperado
N. Korea missiles funding service	recuperado	recuperado	recuperado
Government weapon funding service	recuperado	recuperado	recuperado
India missiles funding service	recuperado	recuperado	recuperado
Gov. Missiles financing service*	recuperado	recuperado	recuperado
Gov. Funding Ball. Missile service	recuperado	no recuperado	recuperado
Gov. Ballistic missile funding service	recuperado	recuperado	recuperado
Gov. Missile lending range service	recuperado	no recuperado	recuperado
Gov. Organization ballistic missile lending range service	recuperado	no recuperado	recuperado
Gov. Organization missile lending range service	no recuperado	recuperado	recuperado
Gov. Org. Missiles uniliteral giving service	no recuperado	recuperado	no recuperado

Resultados obtenidos