ESTUDIO DE LA ONTOLOGÍA OWL-S PARA EL DESARROLLO DE SERVICIOS WEB SEMÁNTICOS

CARLOS HUMBERTO CLAVIJO PATIÑO
WILSON PINTO ROMERO

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FALCULTAD DE INGENIERIA DE SISTEMAS
BUCARAMANGA
2007

ESTUDIO DE LA ONTOLOGÍA OWL-S PARA EL DESARROLLO DE SERVICIOS WEB SEMÁNTICOS

CARLOS HUMBERTO PATIÑO CLAVIJO
WILSON PINTO ROMERO

Tesis de Maestría

Director
DR. EDUARDO CARRILLO ZAMBRANO
Director de Investigación

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FALCULTAD DE INGENIERIA DE SISTEMAS
BUCARAMANGA
2007

Nota de Aceptación

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

CONTENIDO

pág

INTRODUCCIÓN 8

1. OBJETIVOS 9

1.1 OBJETIVO GENERAL 9

1.2 OBJETIVOS ESPECÍFICOS 9

2. MARCO TEÓRICO 10

2.1 WEB SEMÁNTICA 10

2.1.1. Estructura 11

2.1.2. Ontologías 12

2.1.3. Sistemas de almacenamiento 16

2.1.4. Razonadores 16

2.1.5. Lenguajes de reglas 17

2.1.5.1 RQL (RDF Query Language) 17

2.1.5.2 SWRL (Semantic WEB Rule Language) 18

2.1.6. Lenguajes de consultas 18

2.1.6.1 RQL (RDF Query Language) 19

2.1.6.2 RDQL (RDF Data Query Language) 18

2.1.6.3 SeRQL (Sesame RDF Query Language) 19

2.1.6.4 DQL (Daml Query Language) 19

2.1.6.5 OWL-QL 20

2.1.6.6 SPARQL……21

2.1.7. Metodología para creación del modelo semántico formal 21

2.1.7.1 Adecuación de propósito y alcance…… 22

2.1.7.2 Normalización y modularidad creación de subdominios…… 22

2.1.7.3 Reutilización de ontologías…… 23

2.1.7.4 Traducción básica ……..… 23

2.1.7.4.1 Entidades…….. 23

2.1.7.4.2 Atributos de entidades……23

2.1.7.4.3 Relaciones…… 24

2.1.7.4.4 Grados en las relaciones……... 25

2.1.7.4.5 Definición completas /primitiva. uso de axiomas…… 25

2.1.7.5 Refinanciamiento……..25

2.1.7.6 Extensión del conocimiento……26

2.1.7.7 Pruebas o evaluación…… 26

2.1.7.8 Documentación……26

2.2. SERVICIOS WEB 27

2.3. SERVICIOS WEB SEMÁNTICOS.....30

2.3.1 Descubrimiento automático del servicio…… 30

2.3.2 Ejecución automática del servicio WEB……..30

2.3.3 Composición e interoperación del servicio WEB……31

2.3.4 Ejecución y monitorización……31

2.3.5. Arquitecturas para diseño de servicios web semánticos.....32

2.3.6. Ontología para la descripción de servicios WEB semánticos……33

2.3.6.1 Service Profile…… 35

2.3.6.2 Service Model……36

2.3.6.3 Service Grounding……37

3. METODOLOGÍA ..... **¡Error! Marcador no definido.**

3.1. CREACIÓN DE LA ONTOLOGÍA DE LOS DATOS 40

3.2. CREACIÓN DEL SERVICIO WEB.....41

3.3. DESCRIPCIÓN DEL SERVICIO WEB UTILIZANDO OWL-S.....42

4. DESCRIPCIÓN DEL PROTOTIPO 43

4.1. FUNCIONALIDAD DEL SERVICIO.....47

4.2. INSTALACIÓN DEL SERVICIO 48

5. CONCLUSIONES Y TRABAJOS FUTUROS.....50

BIBLIOGRAFÍA 51

ANEXOS 54
Anexo 1: Prototipo de Ontología para la consulta de inversiones.....54
Anexo 2: Descripción WSDL del servicio prototipo de inversiones 60
Anexo 3: Descripción del servicio prototipo de inversiones con OWL-S.....61
Anexo 4: Interfaces para la utilización de los objetos descritos por la ontología....64
Anexo 5: Código del Servicio Web.....74
Anexo 6: Código fuente del programa para probar la lógica del Servicio Web 77

LISTADO DE FIGURAS

pág

Figura 3.1. Aplicaciones de la Web Semántica.....11

Figura 3.2. Estructura de la Web Semántica [Ber98].....12

Figura 3.3. Evolución de tecnologías Web.....28

Figura 3.4. Tecnologías en los Servicios Web.....29

Figura 3.5. Acceso a los Servicios Web 29

Figura 3.6 Propiedades de la clase service.....34

Figura 4.1. Ontología e instancias de prueba del prototipo.....43

Figura 4.2. Relaciones de la ontología del documento cambiario.....44

Figura 4.3. Proceso del Servicio Web Semántico.....45

Figura 4.4. Profile del Servicio Web Semántico.....46

Figura 4.5. Interfaz cliente para prueba del prototipo del servicio.....47

Figura 4.6. Carpeta para instalación del servicio.....48

Figura 4.7. Definición de la URI del servicio con Protégé.....49

INTRODUCCIÓN

Ya desde 1989 Tim Berners-Lee tenía en mente la revolución que causarían los servicios Web implementados en Internet, gracias a los protocolos desarrollados por él y su grupo de trabajo CERN en Europa. La mayoría de las empresas del área tecnológica tuvieron que acoplarse de manera inmediata a dichas reglas para crecer de la mano de esta tecnología. En la actualidad, Internet es la mejor herramienta que existe para hacer búsquedas sobre cualquier tipo de información, desafortunadamente los resultados casi siempre son excesivos e ineficaces por la sobre oferta de información. Tim Berners-Lee, creador del WWW y actual director de W3C (consorcio internacional que produce estándares para Internet) propone la Web Semántica, que es una extensión de la Web actual en la que se proporciona la información con un significado bien definido (metadatos), y se mejora la forma en la que las máquinas y las personas trabajan en cooperación.

El dominio de la infamación financiera y económica es conceptualmente muy, amplio, complejo, voluminoso, y con un valor muy alto. Cada día se genera un enorme volumen de información, haciendo que no sea procesable por una única persona. Se necesitan mecanismos eficientes de clasificación, filtrados, búsqueda y navegación para los consumidores de dicha infamación con el fin de que puedan acceder a los contenidos más relevantes según el perfil de cada usuario, aportando así un alto valor añadido a la información. Esta necesidad evidencia porque la comunidad financiera sea uno de los mayores consumidores de tecnologías de la información. Las tecnologías emergentes de la Web Semántica están generando nuevas formas de representación y distribución de la información, y prometen soluciones al problema planteado inicialmente, estas soluciones pasan por el uso de agentes personales que entiendan nuestras preguntas y busquen las respuestas por la red. Esta nueva red, paralela a la tradicional, es la que se ha venido a denominar Web Semítica, en la que la información esta estructurada y dispuesta para su consumo. Naturalmente no son redes desacopladas, sino que la Web tradicional se enriquecerá por la existencia de la nueva. Por tanto, el primer paso consiste en preparar semánticamente la infamación, bien la nueva, generada día a día como en nuestro caso, bien la tradicional de las páginas Web.

De manera que la Web Semántica, los nuevos servicios Web y los nuevos lenguajes de etiquetado dejan ver la luz al final de túnel y abren muchas posibilidades para el manejo eficiente y seguro de la información por Internet.

1. OBJETIVOS

1.1 OBJETIVO GENERAL

Estudio aplicado del estado actual del desarrollo de Servicios Web Semánticos utilizando descripciones basadas en OWL-S, a partir de una revisión del estado actual de dicha tecnología y de la implementación de un prototipo función.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar el estudio del estado del arte de las tecnologías involucradas con los Servicios Web Semánticos.

- Crear un prototipo funcional de un Servicio Web Semántico, orientado al sector financiero.

2. MARCO TEÓRICO

2.1 Web Semántica

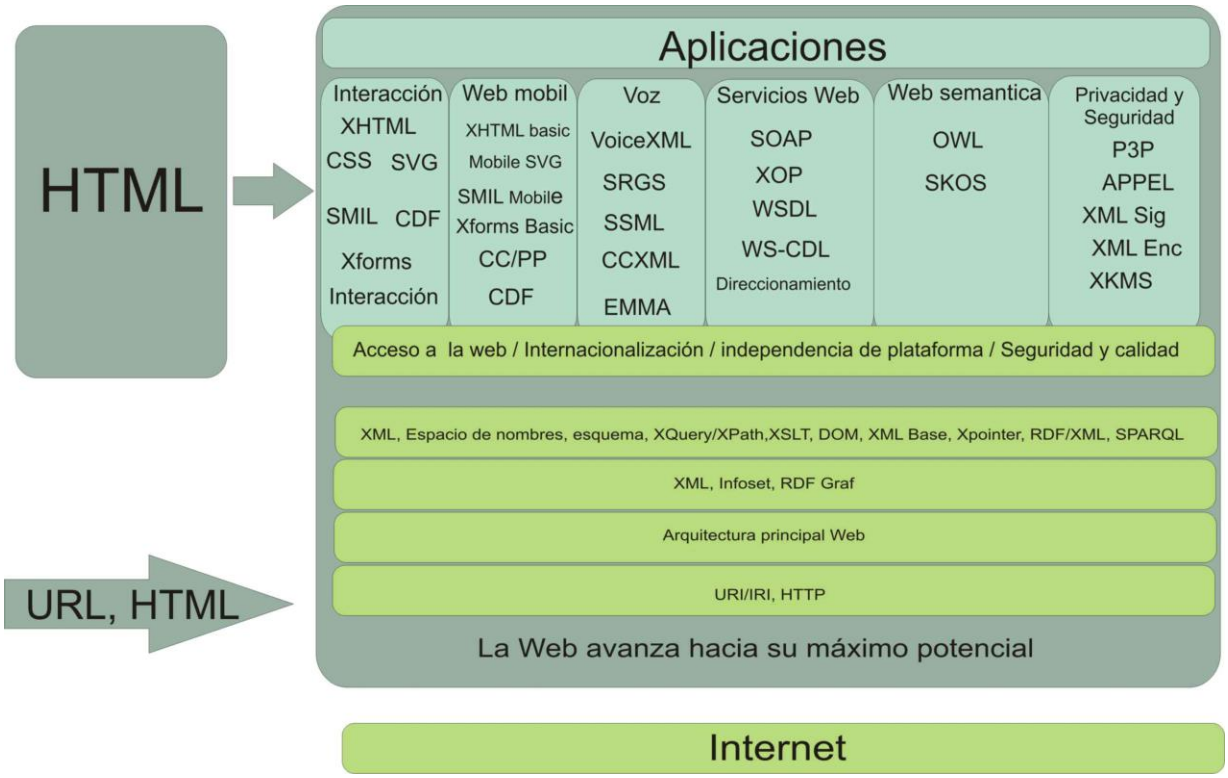
La Web Semántica *"es una extensión de la Web actual en la qué se proporciona la información con un significado bien definido, y se mejora la forma en la que las máquinas y las personas trabajan en cooperación"* [Ber01].

Esta Web extendida está basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados para una Web carente de Semántica en la que en ocasiones el acceso a la información se convierte en una tarea difícil. Esta técnica esta cimentada en las bases del conocimiento de las preferencias de los usuarios para realizar búsquedas y la relación entre la información disponible en la red y el conocimiento.

La base de la Web Semántica son las ontologías, extraídas de la inteligencia artificial y que comprenden una jerarquía de conceptos con atributos y relaciones que define una terminología aprobada por usuarios específicos para definir redes semánticas de unidades de información interrelacionada, dicha ontología proporciona una terminología de clases y relaciones para describir cualquier tipo de dominio siempre y cuando este compartido por todos los entes que quieran implementarlo, por otra parte facilitan un idioma dinámico a los diseñadores para describir las propiedades y capacidades de estos servicios, de tal manera que las descripciones pueden interpretarse por un sistema de manera automatizada, los lenguajes de Ontologías Web para Servicios (OWL-S, Ontology Weg lenguaje for services) describen la organización jerárquica de ideas que pueden analizarse y entenderse por un software.

Una de las aplicaciones más atractivas de la Web Semántica es la de agentes inteligentes en la Web, para conseguir aplicaciones eficaces en esta área, es necesario completar este tipo de arquitectura con tres capas de reglas, lógica, prueba y confianza, además de obtener un buen número de participantes que aporten metadata.

Figura 3.1. Aplicaciones de la Web Semántica

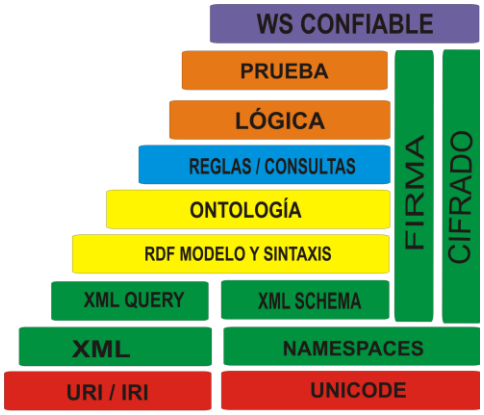


2.1.1 Estructura

Tim Berners-Lee [Ber98] ideó una infraestructura de lenguajes y mecanismos para poder llevar a cabo la idea de la Web Semántica. Esta infraestructura se puede esquematizar
en diferentes capas o niveles (ver Figura 3.1.2).

Las capas Unicode y URI (Uniform Resoure Identifer) aseguran que se usen conjuntos de caracteres internacionales y aporten significado para identificar los objetos en la Web Semántica. La capa XML junto al Namespace (Espacio de nombres) y XML Esquema aseguran que se pueda integrar la definición de Web Semántica con los demás estándares basados en]

Figura 3.2. Estructura de la Web Semántica [**Ber98**]



Con RDF (Resourse Description Frameworks) y RDFSchema es posible hacer declaraciones sobre objetos con URIs y definir los vocabularios que pueden ser referenciados por una URI, en esta capa se pueden definir los tipos de recursos a los enlaces.

- La capa de ontología da soporte a la evolución de vocabularios compartidos y permite definir relaciones entre conceptos.

- La capa lógica esta basada en la posibilidad de definir reglas de tipo “antecedentes a consecuentes” y hechos.

- Las ultimas capas, Prueba y Confianza están todavía en fase de investigación, la capa lógica tiene como objetivo la definición de las reglas, para que la capa de pruebas junto a la capa de confianza las evalúen y permitan establecer si son o no confiables.

2.1.2 Ontologías

El uso de ontologías proporciona una forma de representar y compartir conocimiento haciendo uso de un vocabulario común. Mediante esta representación permiten utilizar un formato de intercambio de este conocimiento, y a su vez brindan la posibilidad de ampliar e integrar otras ontologías o reutilizarlas en la aplicación de otros dominios, además separan el conocimiento del dominio

del conocimiento operacional, haciendo independientes las técnicas y algoritmos, así como hacen que las suposiciones hechas sobre el dominio se hagan explícitas, lo que facilita el replanteamiento de estas y ayuda a que otros puedan entender su descripción. Por último, permiten analizar el conocimiento del dominio utilizando métodos formales.

Las ontologías tienen los siguientes componentes que sirven para representar el conocimiento de algún dominio [GRU93b]:

- Conceptos: son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.

- Relaciones: representa la interacción y enlace entre los conceptos del dominio, suelen formar la taxonomía del dominio.

- Funciones: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.

- Instancias: se utilizan para representar objetos determinados de un concepto.

- Axiomas: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Permiten junto al mecanismo de la herencia de conceptos, inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos.

Según Van Heijst [VAN96] las ontologías se deben clasificar de acuerdo con la cantidad y tipo de conceptualización. Así se pueden diferenciar los siguientes tipos de ontologías:

- Terminológicas: especifican los términos que son usados para representar conocimiento en el universo de discurso. Suelen ser usadas para unificar vocabulario en un dominio determinado.

- De información: especifican la estructura de almacenamiento de bases de datos.

- De modelado del conocimiento: especifican conceptualizaciones del conocimiento. Contienen una rica estructura interna y suelen estar ajustadas al uso particular del conocimiento que describen.

Hay otras posibles clasificaciones de ontologías atendiendo a diversos criterios, por ejemplo si se atiende al asunto que conceptualizan, se distinguen tres tipos fundamentales de ontologías:

- Ontologías de un dominio: en las que se representa el conocimiento especializado pertinente de un dominio o subdominio, como la medicina, las aplicaciones militares, tráfico etc.

- Ontologías genéricas: en las que se representan conceptos generales y fundacionales del conocimiento como las estructuras parte/todo, la cuantificación, los procesos o los tipos de objetos, independientes de un dominio en particular.

- Ontologías representacionales: en las que se especifican las conceptualizaciones que subyacen a los formalismos de representación del conocimiento, por lo que también se denominan meta-ontologías.

En el diseño de una ontología se debe tener en cuenta que la ontología resultante cumpla ciertas características, si cumple estas características se puede hablar de una ontología de calidad eficiente en las aplicaciones en donde se emplee. Para especificar una ontología numerosos autores en un principio proponen 5 criterios que más tarde fueron ampliados a once, dichos autores afirman que no es necesario el cumplimiento de todos ellos, pero que se debe realizar un balance de cuales son los convenientes para cada caso en particular; las tareas identificadas para el desarrollo de cualquier ontología se pueden clasificar en tres así:

a) Actividades ligadas al manejo del proyecto: Planificación, control del seguimiento de la planificación y actividades que aseguren la calidad del producto.

b) Actividades orientadas al desarrollo de la ontología: especificación, conceptualización, formalización, implementación y mantenimiento.

c) Actividades integrales (necesarias para un buen desarrollo de la ontología): Adquisición de conocimiento, integración con otras ontologías, evaluación y documentación.

Hay dos métodos principales que permiten diferenciar dos tipos de ontologías según su construcción:

- Kactus: es un método de construcción de ontologías que se basa en tomar una base de conocimiento y a partir de esta, determinar y conceptualizar cuales son los términos y relaciones más importantes que representaran a la ontología.(Utilizado en el prototipo de este proyecto).

- Sensus: representa a las ontologías construidas a partir de una rama de una ontología más general y que es especializada para obtener una ontología nueva. Es decir consiste en crear ontologías específicas de dominio a partir de ontologías más generales.

Además de estos dos tipos de metodologías existen otras menos nombradas, Methonlogy basada en la especificación de tareas, adquisición de conocimiento, conceptualización, integración, evaluación y documentación de las ontologías para uso científico, y otras de tipo organizacional como On-To-Knowledge (OTK) que es una ontología orientada a procesos, incluye una identificación de metas, objetivos, que son logrados con herramientas de soporte para el manejo de conocimiento; maneja distintas fases que promueven la factibilidad, refinamiento, inferencia, evaluación y aplicación de proyecto y manejo de proceso.

Lenguaje para Ontologías Web (OWL (Ontology Web Language).ge). Surge del W3C [OWL04] como la búsqueda de un lenguaje de especificación de ontologías que sirve como estándar para todos los investigadores de la Web Semántica, está separado en tres niveles así:

- *OWL Lite*: la versión más simple para los programadores principiantes. Permite la jerarquía de clasificación y las restricciones simples.

- *OWL DL*: esta versión ya tiene todo el vocabulario *OWL* completo. Las limitaciones son que las clases no son instancias ni tipos y los tipos no son ni instancias ni clases, no permite restricciones de cardinalidad en propiedades transitivas, posee gran expresividad sin perder las propiedades de completitud y decidibilidad.

- *OWL Full*: esta versión también incluye todo el vocabulario de *OWL* pero en este caso no hay limitaciones para explotar todo su potencial.

OWL Full se considera la más completa de todas y se supone una extensión de DL que a su vez es una extensión de Lite, por lo que toda ontología correcta en OWL Lite es una ontología correcta en OWL DL, y toda conclusión correcta en OWL Lite es una conclusión correcta en OWL DL (pero no a la inversa). De la misma manera esto también ocurre con OWL DL y OWL Full (toda ontología correcta en OWL DL es una ontología correcta en OWL FULL, y toda conclusión correcta en OWL DL es una conclusión correcta en OWL FULL, pero no a la inversa).

Dentro de una ontología, las clases pueden formar jerarquías de herencia al igual que las clases de los lenguajes de programación Orientados a Objetos, de modo que cada clase hija heredará los atributos y relaciones de sus clases padre y se permiten distintos niveles de jerarquía, aunque tampoco se recomienda que esta herencia sea muy profunda dado que se puede perder cierta Semántica.

2.1.3 Sistemas de almacenamiento

Otra de las herramientas más desarrolladas por los investigadores de la Web Semántica han sido los sistemas de almacenamiento de ontologías. Mediante estos sistemas es posible mantener las ontologías en bases de datos e ir añadiendo nueva información, y con la ayuda de razonadores probar la consistencia de la ontología. La mayoría de los sistemas de almacenamiento están orientados a descripciones de conceptos escritas en RDF, aunque algunas han ido actualizándose a los últimos lenguajes de especificación.

2.1.4 Razonadores

Los sistemas de inferencia permite que las máquinas compartan un marco común de referencia, esto quiere decir que las máquinas pueden coleccionar información sobre el dominio o presentarlo a personas o a otras máquinas.

Lógica descriptiva. Es un lenguaje de representación de conocimiento adaptado para expresar conocimiento basado en conceptos y jerarquías; mediante la creación de ontologías formales, este tipo de formalismos es adecuado para dar una estructura sólida a la información a base de frames y redes semánticas con la ayuda de leguajes como KL-ONE, Krypton y Kris entre otros.

Las características más sobresalientes para cualquier tipo de lógica de tipo DL son:

- Operadores para formar conceptos, tales como conceptos primitivos, conjunción y cuantificadores existenciales.
- Operadores para desarrollar propiedades primitivas e inversas.
- Axiomas para el manejo de conceptos tales como igualdad e implicación en las relaciones.
- Inferencia en lógica descriptiva.

Todo sistema terminológico esta conformado por tres subsistemas:

- Base de conocimiento: se almacena la información sobre el dominio a base de jerarquías de clases y relaciones entre ambas, maneja dos partes, una que contiene el conocimiento terminológico (normas, definiciones,

conceptos y roles), y otra que contiene el conocimiento las definiciones de las instancias.

- Sistema de inferencia: se utiliza para razonar la base del conocimiento.

- Interfaz: para comunicar el usuario final con el sistema.

Una de las herramientas más utilizadas con las ontologías son los razonadores, que sirven para realizar inferencia, a través de los conceptos y en algunos casos las instancias, para obtener nuevo conocimiento. Generalmente difieren en el lenguaje formal en el que se especifica el conocimiento, así como los lenguajes de consulta que puedan utilizar:

En la actualidad, son varios los razonadores o sistemas de inferencia basados en *lógica descriptiva* que permiten el razonamiento y la inferencia en las ontologías. Los razonadores basados en *lógica descriptiva* deben de poseer la suficiente expresividad para poder permitir que el conjunto de constructores que forman parte de los lenguajes de ontologías sean soportados, y poder permitir ambos tipos de inferencia, tanto de clasificación como de chequeo de instancias.

2.1.5 Lenguajes de Reglas

Otro de los objetivos de la Web Semántica es la definición de un lenguaje de reglas semánticas. Las reglas en la Web Semántica pueden ser de utilidad a los razonadores como métodos para inferir sobre el conocimiento, obtener nueva información, y que además, puedan ser empleadas por los Servicios Web para especificar tarea.

Como lenguajes de reglas destacan dos, RuleML [RuleML03] y SWRL [SWRL04]. RuleML ha sido el primer lenguaje empleado para definir reglas en aplicaciones basadas en Web Semántica, aunque recientemente se ha propuesto el uso de SWRL.

2.1.5.1 RuleML (Rule Markup Language). Es una iniciativa de diferentes instituciones y grupos investigadores que tenían como objetivo definir un lenguaje de hechos y reglas semánticas que pudiese ser utilizado en los diferentes sistemas de inferencia comerciales de reglas. La primera especificación estable de este lenguaje es de 2001, y en ella se quiso dar una sintaxis para todas las formas de reglas existentes en la lógica, pero al ser demasiado extenso se decidió por proporcionar al lenguaje una sintaxis que permitiese especificar reglas de implicación y hechos.

2.1.5.2 SWRL (Semantic Web Rule Language). Se basa en la combinación de OWL y RuleML [SWRL04]. Las características principales del lenguaje son:

- Basado en OWL DL y OWL Lite y RuleML.
- Incluye sintaxis para expresiones de cláusulas de Horn.
- Se aporta un modelo teórico-semántico para incluir reglas en ontologías OWL.
- Aporta sintaxis para emplearse junto OWL-XML y OWL-RDF.

La propuesta de SWRL es la de permitir definir reglas de *Horn* en una base de conocimiento OWL. Para ello extiende el conjunto de axiomas de OWL, es decir, añade los axiomas adecuados para que el lenguaje de especificación de ontologías OWL permita definir las reglas.

Las reglas propuestas para este lenguaje tienen forma de implicación, con un antecedente (*body*) y un consecuente (*head*), igual a las definidas en RuleML. El significado que se quiere dar con estas reglas es el siguiente: si las condiciones especificadas en el antecedente son ciertas, entonces las condiciones del consecuente también lo son.

SWRL se ha convertido en el lenguaje de reglas recomendado por la comunidad de Web Semántica.

2.1.6 Lenguajes de consultas

2.1.6.1 RQL (RDF Query Language) [RQL] es un lenguaje de consulta para RDF y RDF Schema basado en OQL (Object Query Language). RQL nos permite navegar por los grafos que hay en el modelo RDF y proporciona un mecanismo para preguntar y seleccionar los nodos del modelo que queramos recuperar.

La característica más destacable de este lenguaje es que posee construcciones propias específicas para las relaciones semánticas dentro del RDF Schema, como pueden ser las relaciones de clase/instancia, clase/propiedad o el dominio y rango de una propiedad, por lo que resulta más fácil recuperar información de los nodos del modelo.

2.1.6.2 RDQL (RDF Data Query Language) [RDQL04] fue desarrollado por HP para que fuese el lenguaje de consulta para RDF en los modelos de Jena, con la idea de convertirse en un modelo de consulta orientado a datos por ser una aproximación más declarativa. Debido a esto, solo se pueden hacer consultas sobre la información que hay en el modelo, por lo que la inferencia o razonamiento

no es posible. RDQL deriva de SquishQL que es un lenguaje de consulta para RDF, que a la vez deriva de rdfDB [Dum00], y son una clase de lenguajes para RDF, no para el RDF Schema a no ser que esté explícitamente en el modelo que se maneje.

Como RDF provee una estructura de grafos, donde los nodos son recursos o literales, RDQL permite especificar el patrón de la tripleta <sujeto, predicado, objeto> que queremos buscar en el grafo del modelo para poder recuperar cualquier parte de la tripleta, devolviendo todas las tripletas que cumplan el patrón que le pasamos (muy parecido a como actúa RQL).

Además de la desventaja de no permitir realizar ninguna inferencia, la utilización de filtros para obtener resultados es muy limitada. Las ventajas son la sencillez de manejo, ya que solo hace falta tener claro que tripleta <sujeto, predicado, objeto> se quiere preguntar, y otra de las ventajas son la facilidades de integración con Java, debido a que ha sido implementado por los mismos desarrolladores de Jena [JENA05].

2.1.6.3 SeRQL (Sesame RDF Query Language) [SeRQL04], [Bro02] es un lenguaje de consulta desarrollado por los administradores de Sesame, combinando las características de RQL, RDQL, N-Triples y N3, añadiendo alguna más propia. Las principales características son:

- Transformación de grafo
- Soporte de RDF Schema
- Soporte de Datatype de XML Schema
- Sintaxis expresivas para los path (o URI)
- Matching opcional de path (o URI)

Este lenguaje de consultas posee tres componentes importantes: URIs, literales y variables.

Una de las diferencias de este lenguaje respecto al resto es la de presentar dos formas diferentes de hacer la consulta, la devolución de tablas con los posibles valores que pueden tomar las variables en nuestra consulta ("*Select*", común al resto de lenguajes), y devolviendo el resultado en la forma de subgrafo que nosotros le sugerimos, almacenando la información del resultado en un grafo RDF (son consultas conocidas como "*Construct*").

2.1.6.4 DQL (Daml Query Language) [Fik03] se trata de un lenguaje formal y protocolo para conducir el diálogo entre un agente cliente que realiza consultas y un agente que contesta los requerimientos, haciendo uso para ello de conocimiento representado en DAML + OIL.

La diferencia principal de este lenguaje con el resto reside en la sintaxis para la descripción de las consultas. En este caso se deben de seguir los patrones de *Query* y *Answer* descritos por una ontología *dql.daml*. Esta ontología describe como deben de estar formadas tanto las consultas como las respuestas.

Es un lenguaje de consulta y no de inferencia, y respecto a éstos tiene la ventaja de que está preparado para manejar conocimiento descrito en DAML + OIL. Sin embargo, las herramientas que implementan este lenguaje, han sido escasas. DQL es soportado por el razonador JTP, que está preparado para admitir consultas mediante mensajes SOAP y retornar los resultados en el correspondiente mensaje de respuesta SOAP.

2.1.6.5 OWL-QL [Fik04] es una actualización del lenguaje anterior (DQL). En este caso, se trata de un lenguaje formal y protocolo para conducir el diálogo entre un agente cliente que realiza consultas y un agente que contesta los requerimientos, haciendo uso para ello de conocimiento representado en OWL. Intenta convertirse en un estándar, y para ello tiene en cuenta una serie de factores dentro de la Web Semántica:

- La previsión de que incluirá muchos tipos de servicios petición-respuesta con acceso a muchos tipos de información representados en muchos formatos. Por lo que se debe lograr manejar esta heterogeneidad.

- El hecho de que algunos servidores contengan solo información parcial sobre algún tópico y por tanto serán incapaces de manejar o dar respuesta a ciertos tipos de requerimientos. Se pretende que los propios protocolos de consulta puedan proveer algún medio de transferencia de resultados parciales.

- La idea de que una consulta o requerimiento especificada en un lenguaje de la Web Semántica necesita dar soporte a requerimientos que no incluyan una especificación de la base(s) de conocimiento, de igual manera que los usuarios de un navegador en Internet, no necesitan describir que portales Web se han de considerar cuando realizan una búsqueda. Se pretende que los propios servidores sean capaces de encontrar las apropiadas bases de conocimiento.

- El hecho de que el conjunto de notaciones y sintaxis usadas en la Web se ya demasiado extenso, y varias comunidades tengan diferentes preferencias, y por tanto ninguna universal. Se debe lograr que los aspectos esenciales del lenguaje sean independientes del conjunto de su sintaxis.

- Por último, la premisa de que en la Web Semántica, los lenguajes declarativos usados para representar en la Web tenga una Semántica definida formal y lógica. Este es el caso de OWL y sus predecesores.

OWL-QL tiene en cuenta todos estos factores e intenta subsanar todas las deficiencias que hasta ahora se habían encontrado en los demás lenguajes de requerimientos.

2.1.6.6 SPARQL [SPARQL05] es un lenguaje de consultas para grafos RDF propuesto recientemente por W3C. Ofrece a los desarrolladores y usuarios finales un camino para presentar y utilizar los resultados de búsquedas a través de una gran variedad de información como puede ser datos personales, redes sociales y metadatos sobre recursos digitales como música e imágenes. SPARQL también proporciona un camino de integración sobre recursos diferentes. Permite:

- Extraer información en diversas formas, incluyendo URIs.
- Extraer subgrafos RDF.
- Construir nuevos grafos RDF basados en la información de los grafos consultados.

2.1.7 Metodología para creación del modelo semántico formal

La metodología propuesta por [SAM05] distingue 8 fases:

- Adecuación de propósito y alcance.
- Normalización y modularidad. Creación de subdominios.
- Reutilización de ontologías.
- Traducción básica.
- Refinamiento.
- Extensión del conocimiento: Adición de instancias.
- Pruebas o Evaluación.
- Documentación.

2.1.7.1 Adecuación de propósito y alcance. Adecuación del propósito y alcance dados para el desarrollo del modelo de datos a nuestro modelo semántico formal. Análisis de nuevos requerimientos que el nuevo modelo debe cumplir.

2.1.7.2 Normalización y modularidad. Creación de subdominios. Análisis del esquema conceptual y propuesta de creación de diferentes módulos o subdominios:

a) Identificar de todas las entidades existentes en el modelo ER, con sus atributos y con las relaciones que posean con el resto de entidades del modelo.

b) En base a la anterior clasificación, se construye una tabla, en la que son insertadas las entidades mediante un orden de prioridad establecido por el factor de número de relaciones, y en caso de empate por número de atributos. Esta tabla debe mostrar los enlaces entre cada una de las entidades.

c) Extraer de la tabla la entidad con mayor prioridad, esta primera extracción, en la entidad central del modelo, se constituye en la base para el desarrollo del principal dominio del modelo, a partir del cual gira el resto.

d) Intentar establecer que otras entidades pueden formar parte del dominio en cuestión y extraerlas de la tabla. Para ello se escogen de entre aquéllas que están relacionados con la entidad escogida en el paso c), las que por sus características y aplicando lógica común deban de permanecer juntas en el mismo dominio.

e) Volver al paso c) y extraer de la tabla la siguiente entidad por orden de prioridad para la creación del resto de subdominios periféricos, hasta que la tabla quede vacía.

El propósito de la normalización es la sencillez la cual permitirá la reutilización, mantenimiento y evolución. Ante la necesidad de evitar descripciones complejas, esta fase de normalización consiste en crear módulos simples que pueden ser combinados para alcanzar las descripciones complejas de interés. Se establecen por tanto las razones para clasificar y se impone una disciplina para establecer cada definición considerada relevante de forma explícita. La razón para clasificar bajo los distintos módulos debe darse de forma explícita para que el razonador pueda realizar las correctas inferencias.

Los módulos consisten generalmente en jerarquías simples donde los conceptos primitivos generalmente tienen un único padre. Las definiciones y descripciones permiten enlazar las diferentes jerarquías y permiten al razonador inferir la estructura compleja. A su vez debe existir una estructura taxonómica que relacione los diferentes módulos, especificando las diferentes categorías [Ste03] [Rec03].

La modularidad es un aspecto muy importante a tener en cuenta, ya que mantener ontologías independientes significa que cada una puede ser modificada separadamente.

Inclusive dentro de una misma ontología es sumamente conveniente modificar las jerarquías en base a sus estructuras, roles etc., de tal forma que se puedan tener diferentes tipos de clasificaciones dentro de una misma ontología (subtaxonomías).

2.1.7.3 Reutilización de ontologías. Una vez se determinan los diferentes subdominios, se considera la reutilización de ontologías ya disponibles y públicas que se puedan tener en cuenta en cada uno de éstos. El uso de éstas puede llegar a simplificar las siguientes fases de la metodología.

Una vez finalizada la fase de normalización y analizando el posible uso de ontologías ya creadas, se pasa a la siguiente fase en la metodología, que se aplica a cada uno de los subdomínios establecidos.

2.1.7.4 Traducción básica. Traducción de los diferentes elementos del esquema conceptual a modelo semántico formal.

2.1.7.4.1 Entidades. Transformación de las entidades en clases y/o individuales según criterio y establecimiento de su jerarquía.

Hay dos formas de representar los elementos de una clase, como individuales o como subclases [Cec02]. El uso de clases es semánticamente más rico y hace que la ontología se pueda extender más fácilmente. Incluso aún existiendo mayor complejidad, el uso de clases es mejor salvo en los casos en donde los individuales sean necesarios como por ejemplo, al establecer miembros de una clase enumerada.

2.1.7.4.2 Atributos de entidades. Transformación de los atributos de las entidades que aparecen en el modelo semántico de datos en propiedades, teniendo en cuenta que cada una de estas entidades posee una serie de atributos que a menudo solo tienen utilidad en el modelo de datos semántico, debido a las características intrínsecas de los modelos de datos relacionales. Por ejemplo, los atributos que hacen referencia a las versiones del Diccionario de Datos. Estos atributos quedan por tanto excluidos de la especificación formal. Pasos a seguir:

Estas propiedades pueden ser dos tipos: propiedades objeto (ObjectProperty) o propiedades de tipos de datos (DatatypeProperty), lo cual determina el rango de valores que puede tomar.

- Propiedades objeto son usadas para relacionar un recurso a otro recurso.

- Propiedades de tipos de datos únicamente relacionan un recurso a un *rdfs:literal* o a un tipo de datos perteneciente a *XML Schema*.

Se estudia según criterios dados la conveniencia de restringir estas propiedades globalmente: Rango y Dominio.

Más tarde en el proceso, es posible restringir el rango especificado en una propiedad de una clase padre, cuando dicha propiedad afecta a clases descendientes, siempre y cuando ese nuevo conjunto de valores que la propiedad pueda tomar sea un subconjunto del rango de valores original de la clase ascendente. Esto se logrará mediante operadores de cuantificación generalmente.

Definición de características de las propiedades: Transitividad, Inversa, Simétrica.

Hay que tener en cuenta que las características de transitividad o simétrica solo pueden ser aplicadas en propiedades de tipo objeto (relacionan dos recursos). Inclusión de dicha propiedad en una jerarquía de propiedades (elección de superpropiedad(es)) que ayuda en la aplicación de posibles métodos de inferencia.

2.1.7.4.3 Relaciones. Tratamiento de las relaciones que aparecen en el esquema, y distinción según el grado de la relación y la posesión o no de atributos propios, en:

a) Relaciones Binarias, las cuales directamente se pueden traducir al lenguaje de especificación formal mediante propiedades de tipo objeto; son introducidas como propiedades, al igual que son introducidos los atributos, con la salvedad de que ahora solo pueden ser de tipo objeto, puesto que deben de relacionar dos recursos o entidades. Se toman los mismos criterios que los tomados para especificar los atributos, en cuanto a las características de dichas propiedades.

b) Relaciones n-arias, cuya traducción no se puede realizar directamente. Debido a que en DL tradicional (y por tanto en los lenguajes formales basados en ella) únicamente son considerados relaciones unarias o binarias, se toma en consideración el método propuesto por Calvanese et al. [Sat03], el cual consiste en cambiar las relaciones, generalmente mediante la traducción de cada relación en un concepto cuyas instancias representan las tuplas de la relación. Mayor detalle de la definición de este tipo de relaciones y su uso

mediante individuales puede ser encontrado en el informe técnico de Alan Rector en W3C [Rec04a].

c) Relaciones con atributos propios. Para este caso se toma de nuevo en consideración si dicho atributo (propiedad), puede ser asumido por alguna de las entidades relacionadas, de lo contrario se procede a crear una nueva clase que establece el rango de una nueva relación (propiedad objeto), tomando como base este nuevo atributo.

2.1.7.4.4 Grados en las relaciones

- Observación del modelo para determinar los grados de cardinalidad existentes en cada una de las relaciones. La traducción al lenguaje formal se hará mediante el uso de restricciones de cardinalidad a cada una de las clases afectadas o en su lugar restringiendo globalmente la propiedad mediante las características funcionales o inversamente funcionales. Las características de funcional o inversamente funcional pueden ser aplicadas a cualquier tipo de propiedad (objeto y datatype).

- Especificación de las restricciones locales a cada una de las clases mediante el uso de operadores de cuantificación existencial y/o universal.

2.1.7.4.5 Definiciones completas/primitivas. Uso de axiomas.

- Determinación de que clases deberían formar parte en una definición completa o cuales simplemente deberían ser simplemente clases primitivas.

- Teniendo en cuenta el punto anterior, para aquella parte de la especificación de una clase que no deba formar parte de una definición (completa o parcial), se hará uso de axiomas de cubrimiento, disyunción etc. Los axiomas proveerán información adicional sobre las clases.

2.1.7.5 Refinamiento

a) Revisión de entidades y de propiedades, ya que quizás sea necesario una mayor expresividad o base de conocimiento mucho más elaborada, de tal forma que algunas de estas propiedades o entidades necesiten de la elaboración de nuevos dominios específicos no contemplados en principio en el modelo semántico de datos. Pensemos por ejemplo en atributos del modelo de datos los cuales toman valores de su diccionario de datos. En estos casos puede ser

conveniente la introducción de nuevos dominios correspondientes en principio a simples taxonomías que más tarde pueden convertirse en complejas ontologías con la adición de nueva información proveniente de otras fuentes, distintas al diccionario. Por ejemplo, de nuevo en el desarrollo de la ontología “Vías”, ciertos atributos como “carretera” de las entidades Punto y Tramo del modelo de datos semántico o en la ontología de Vehículos (atributo “tipo” de la entidad Vehículo) o Geografía (atributos provincia, población etc. de las entidades Área y Punto), fueron convertidos a ontologías.

b) Una vez, establecida la primera “traducción” se procederá a estudiar las posibles aplicaciones de nuestro modelo. Quizás tras un estudio exhaustivo de éstas se llegue a la conclusión de que el modelo formal actual no contemple ciertas propiedades o entidades que serán necesarias para un adecuado uso en dichas aplicaciones. (Por ejemplo, las propiedades específicas de elementos como Previsión que simplemente formaban parte del DD del modelo de datos original).

2.1.7.6. Extensión del conocimiento: *Adición de instancias.* Adición de conocimiento extensional es decir aporte de instancias o individuales. La adición no tiene porque ser en el mismo fichero físico que en el que resida el sistema terminológico (términos y relaciones), aunque tendrán que estar ligados. A partir de esta fase se considerará el sistema globalmente.

2.1.7.7 Pruebas o Evaluación. Pruebas de requerimientos sobre la base de conocimiento que muestren la eficacia de éste. En este punto se tratará de comprobar si la base de conocimiento es capaz de satisfacer todos los requisitos especificados en la fase inicial de propósitos y alcance. Es decir, responder a las interrogantes que inicialmente se abordaron. El uso de un sistema de inferencia, posiblemente mediante la elaboración de consultas permitirá resolver esta cuestión.

2.1.7.8 Documentación. La especificación ontológica debe ser ampliamente documentada, de tal forma que cualquiera puede entender fácilmente su composición y estructura. De esta forma las tareas de mantenimiento y escalabilidad del modelo podrán ser llevadas de forma más adecuada. En esta fase se considera útil aunque no necesario especificar la correspondencia entre ambos modelos.

A partir del paso 3 será útil comprobar en cada paso la consistencia del modelo e inclusive una vez determinadas aquellas clases que deban de poseer una definición completa, ayudarse del razonador para establecer posibles clasificaciones en la jerarquía.

2.2 SERVICIOS WEB

Los Servicios Web (SW) permiten a las compañías publicar componentes y servicios en un directorio en el que otras aplicaciones Web pueden buscar e implementar nuevos servicios a través de una llamada.

Las tecnologías previas al desarrollo de los SW fueron conocidas como objetos distribuidos, dentro de las que se encuentran DCOM de Microsoft, RMI de Sun. y CORBA de OMG (Object Management Group). Sin embargo, y a pesar de las alianzas de las compañías para conseguir interoperabilidad entre diferentes aplicaciones, y la no participación de Microsoft en el desarrollo de CORBA, ocasionó que se frenara el desarrollo de esta tecnología para dar paso a los SW, que aunque se fundamentan en nuevas características, siguen buscando la interacción entre aplicaciones.

Muchos esfuerzos se han hecho y se están realizando en la actualidad para definir las diferentes especificaciones y arquitecturas que posibiliten la expansión de este nuevo tipo de aplicaciones denominadas Servicios Web. Grandes empresas como Microsoft, IBM, Intel y consorcios como W3C están profundamente involucrados en esta investigación.

W3C aporta la siguiente definición en su grupo de investigación de SW: "*Un servicio de Web es un sistema de software diseñado para soportar la interacción entre dos máquinas a través de una red. Posee un interfaz descrito en un formato que puede ser procesado por una máquina (específicamente WSDL). Otros sistemas pueden interactuar con el Servicio Web en la manera prescrita por su descripción utilizando mensajes SOAP, típicamente transportados utilizando HTTP y serializados con XML, en conjunción con otros estándares relacionados con la Web*" [W3C04].

Los servicios Web son componentes software independiente de la plataforma hardware o software sobre la que esta implementado, y pueden ser:

- Descritos mediante un lenguaje de descripción de servicio, como el lenguaje WSDL.

- Publicados al someter las descripciones y políticas de uso en algún registro conocido, utilizando el método de registro UDDI u otro método.

- Encontrados al enviar peticiones al registro y recibir detalles de enlace del servicio que se ajusta a los parámetros de la búsqueda.

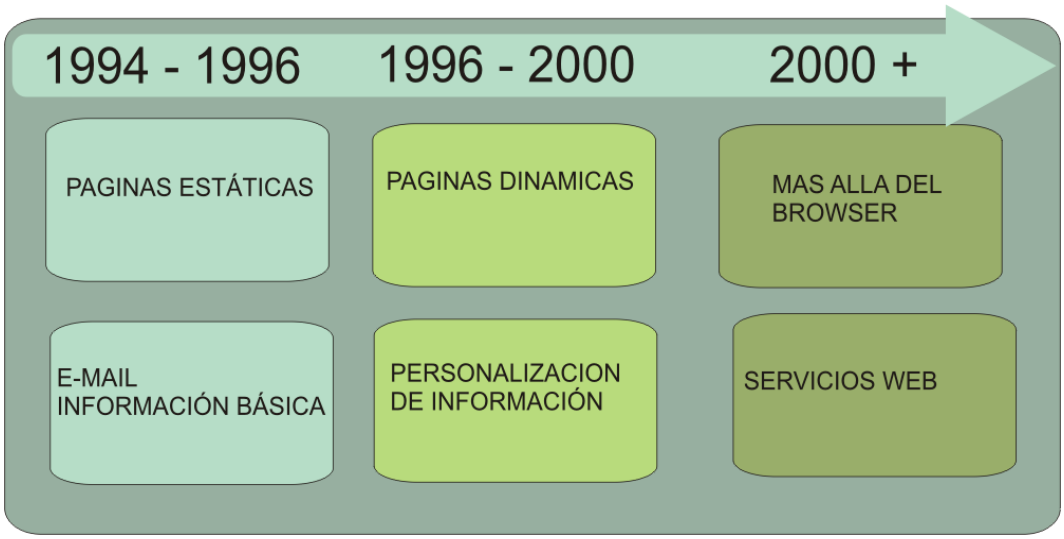
- Asociados al utilizar la información contenida en la descripción del servicio para crear una instancia de servicio disponible o Proxy.

- Invocados sobre la red de forma local o remota al utilizar la información contenida en la descripción del servicio o detalles de service binding.

- Compuestos con otros servicios para integrar servicios y aplicaciones nuevas.

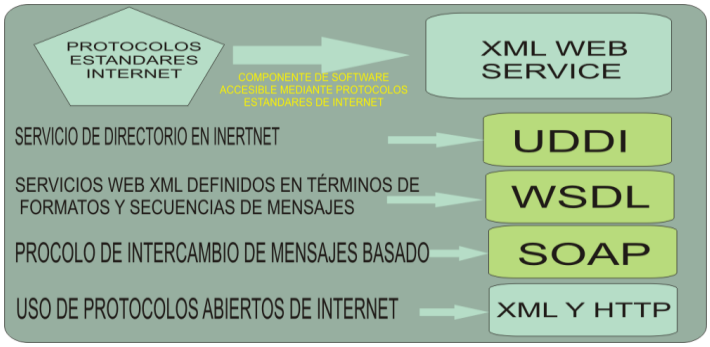
Durante décadas el desarrollo de software ha tenido diversas metodologías, no obstante siempre se ha buscado reducir la complejidad del software. Para ello, surgen las contracciones (funciones, componentes, clases) que permiten ocultar su implementación y proporcionan acceso controlado a su comportamiento mediante una interfase, que en lo posible sea amigable con el usuario final de cualquier tipo de aplicación; actualmente las tecnologías como EJB, .NET y CORBA se basan en la arquitectura de componen.

Figura 3.3. Evolución de tecnologías Web



Los servicios Web consisten de una clase particular de servicio, que es invocado usando la conjunción de cuatro tecnologías para su implementación: HTTP (invocación), SOAP/XML (mensaje), UDDI (descubrimiento) y WSDL (descripción). Este tipo de servicios especifica la forma en que el servicio es invocado, no como es implantada la lógica del servicio.

Figura 3.4. Tecnologías en los Servicios Web



Básicamente es la interacción de tres roles: el proveedor de servicio, el registro del servicio y el solicitante del servicio, los servicios software se registran utilizando los siguientes lenguajes y protocolos, muchos de estos estándares y otros están desarrollados en el proyecto UDDI. Un consorcio de industrias que coordina los esfuerzos de diseño y creación. Estas tecnologías encapsulan un conjunto de estándares que permiten a los desarrolladores implementar aplicaciones distribuidas.

Figura 3.5. Acceso a los Servicios Web

2.3 SERVICIOS WEB SEMÁNTICOSIS

Los SWS son una línea importante de la Web Semántica, que propone describir no solo información sino definir ontologías de funcionalidad y procedimientos para describir SW: sus entradas y salidas, las condiciones necesarias para que se puedan ejecutar, los efectos que producen, o los pasos a seguir cuando se trata de un servicio compuesto. Estas descripciones procesables por máquinas permitirían automatizar el descubrimiento, la composición, y la ejecución de servicios, así como la comunicación entre unos y otros. Las tareas que debe permitir el marcado semántico de SW ([OWLS03]) so:

2.3.1 Descubrimiento automático del servicio.

Está relacionado con la localización automática de los SW que provee un servicio particular y que se adhiere a las restricciones dadas en la petición.

a) Posibilidad de usar un motor de búsqueda para encontrar dicho servicio.

b) Lectura del sitio Web encontrado.

c) Ejecución del servicio manualmente (introducción de franja horaria, nomenclatura de la carretera etc.)

Con el marcado semántico de servicios, es posible especificar la información necesaria para el descubrimiento del servicio como marcado semántico interpretable por computadores, de manera que un servicio de registro o motor de búsqueda (mejorado con ontologías) pueda localizar automáticamente los servicios apropiados.

2.3.2 Ejecución automática del servicio Web.

Involucra la ejecución automática de un servicio Web identificado por un programa de computador o agente. La ontología de descripción del servicio debería proveer APIs declarativos para SW, necesarios par una correcta ejecución automática de éstos. Siguiendo con el ejemplo anteriormente expuesto, supongamos que quisiéramos finalmente suscribirnos al servicio de alertas para recibir información actualizada en la franja horaria y carretera establecida.

Posiblemente en este caso los pasos sean los siguientes:

- Acceder al sitio Web, rellenar el formulario de suscripción y presionar sobre la opción de aceptar para ejecutar el servicio.
- Alternativamente, enviar una petición “HTTP” directamente al servicio mediante la dirección URL con los parámetros apropiados

2.3.3 Composición e interoperación del servicio WEB.

Equivale a la selección automática, composición e interoperación de SW para realizar alguna tarea, dada una descripción detallada de un objetivo. Se denomina composición del SW a la técnica de componer funcionalidades de servicios relativamente más simples para producir una aplicación compleja y significante de forma arbitraria.

La ontología de descripción de servicio debe proveer especificaciones declarativas de los prerrequisitos y consecuencias del uso individual de un servicio, necesarias para la composición e interoperación automática. Imaginemos que el usuario quisiera planificar un viaje por carretera, en una fecha concreta. Las tareas a desempeñar podrían ser las siguientes:

a) Especificar la composición manualmente.

b) Asegurar que cualquier software para la interoperación está hecho a la medida.

c) Proveer la entrada en los puntos de elección (Por ejemplo seleccionar todas aquellas carreteras por las que se prevé pasar).

2.3.4 Ejecución y monitorización

Esta tarea involucra la habilidad de los usuarios / agentes para conocer el estado de su petición durante el periodo de ejecución o cualquier violación de las restricciones motivada por las acciones llevadas a cabo por los agentes software. En este caso la ontología de descripción de servicio debe proveer descriptores para la ejecución de servicios. En el ejemplo, podríamos querer saber el estado de nuestra suscripción o cualquier detalle adicional.

Con el marcado semántico de SW, la información necesaria para seleccionar, componer, y responder a servicios es codificada en los sitios del servicio. De esta forma es posible crear software para manipular este marcado, junto con los objetivos de la tarea, para conseguir la automatización de este proceso.

2.3.5 Arquitecturas para diseño de Servicios Web Semánticos

En [Gom04b] se han propuesto los tipos de características que permiten a los agentes o programas externos descubrir, invocar y componer SWS, así:

- Acceso (o comunicación): Está relacionada con el protocolo de comunicación requerido para invocar la ejecución del servicio (ejemplo SOAP o HTTP).

- Descriptivas: Detalla propiedades de los SWS tales como ubicación geográfica, clasificación comercial (ejemplo UNSPSC) o proveedor. Esas características definen el dominio (tal como minerales en UNSPSC) en el cual es desarrollada la operación del servicio. También pueden guiar el descubrimiento del servicio rechazando los servicios que operan en un dominio diferente.

- Funcionales: Especifican las capacidades de los SWS (sus datos de entrada y salida), efectos, pre y postcondiciones de ejecución. Una vez el dominio del servicio ha sido establecido, esas características permiten definir a un agente externo cuando la ejecución del servicio puede obtener el resultado solicitado.

- Estructurales: Describen la estructura interna del servicio, es decir, los componentes estructurales y cómo se combinan para ejecutar el servicio. Típicamente, los agentes usan esas características para composición del servicio debido a que ellas determinan si existen interacciones entre subservicios y otros servicios usados para componer un nuevo servicio.

En el desarrollo de SWS se requieren arquitecturas conceptuales a nivel de conocimiento y en forma independiente del lenguaje. Estas arquitecturas deben permitir especificar un conjunto de capas que cubran las características del servicio.

Las estructuras internas de los SW y de los SWS han sido modeladas tradicionalmente como procesos de negocio, teniendo en cuenta las actividades que se deben realizar para ejecutar el proceso. Este enfoque rompe el proceso en actividades cuyas interacciones pueden ser modeladas como patrones de flujo [Gom04b].

Otra propuesta para el desarrollo de arquitecturas conceptuales existentes es ODE SWS [ODESWS05] que utiliza modelado basado en métodos de solución de problemas (PSM: *Problem Solving Methods*) para la descripción de la estructura de los servicios.

En [Nar03] se plantea la traducción de la Semántica detrás de OWL-S en lógica de primer orden, obteniendo una serie de axiomas para describir las características de los servicios. Por otra parte, utilizan Redes de Petri para expresar la Semántica operacional distribuida para las ontologías DAML-S y OWL-S, permitiendo razonar a cerca de la interacción en el proceso de composición de la estructura de un servicio.

Posiblemente un aspecto que dificulta notablemente la existencia de propuestas concretas de arquitecturas se basa en la complejidad de integración tecnológica (agentes inteligentes, SW, sistemas de emparejamiento) y la diversidad de propuestas existentes que componen los SWS.

2.3.6 Ontología para la descripción de servicios Web semánticos

WSDL muestra como se debe solicitar un servicio Web, pero no específica que parámetros recibe ni tampoco cuales retorna y nunca habla sobre instancias tan importantes como el significado de cada parámetro, adema olvida explicar en detalle el servicio Web como tal. OWL-S por su parte permite publicar de forma declarativa las propiedades y cualidades de un servicio, brindando la posibilidad de descubrir e invocar servicios de forma automática así como componerlos teniendo en cuenta su descripción semántica.

OWL-S es una ontología específica para describir las propiedades y capacidades de un servicio Web. La estructura de esta ontología viene dada por la necesidad de responder a tres aspectos (manera similar a WSDL) básicos para este tipo de servicio:

- ¿Qué necesita el servicio del cliente que lo utiliza (humano/s o agente/s) y qué le proporcionará como respuesta el servicio? Esto se denomina el perfil del servicio.
- ¿Qué hace el servicio? Esto es el modelo del servicio
- ¿Cómo se usa el servicio? La respuesta a esta pregunta es el grounding del servicio.

De manera que el cliente podría descubrir dinámicamente el servicio gracias a su perfil y necesidades utilizarlo la información suministrada por el grounding, este modelo describe el flujo de tareas y un posible camino de ejecución, de manera que se dice que el servicio es descrito por su modelo, el cual permite a su agente consumidor:

- Realizar un análisis detallado de cómo el servicio cumple con sus necesidades.

- Componer la descripción de un servicio desde múltiples servicios Web que cumplen una tarea específica.

- Coordinar las tareas de múltiples agentes.

- Monitorear la ejecución del servicio Web.

Las metas de OWL-S son proporcionar:

- Desarrollo de ontologías robustas para servicios de Web.

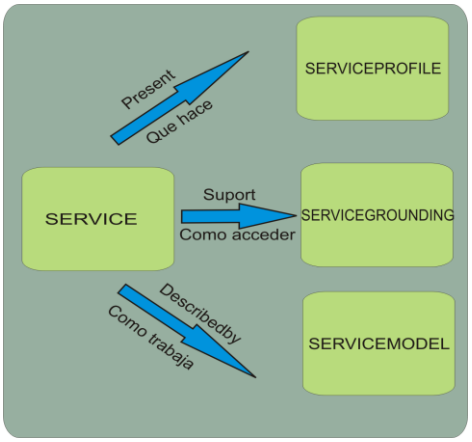
- El descubrimiento dinámico de servicios basados en preguntas ontológicas.

- La invocación transparente de esos servicios.

Desde el punto de vista de OWL, el lenguaje en el que está escrita esta ontología, el servicio se representa por la clase *Service*. Se usará una instancia de esta clase para representar cada servicio concreto. Las propiedades *presents, describedBy y supports* son propiedades de la clase *Service*. Las clases *ServiceProfile, ServiceModel* y *ServiceGrounding* son los respectivos rangos de estas propiedades, por lo que, en una instancia concreta de *Service*, el valor de la propiedad *presents* será una clase derivada de *ServiceProfile*, el valor de la propiedad *describedBy* será una clase derivada de *ServiceModel*, y el valor de la propiedad *supports* será una clase derivada de *Service-Grounding*.

La parte superior de la ontología se denomina *service* como se muestra en el siguiente gráfico.

Figura 3.6 Propiedades de la clase *Service*



La clase *Service* es el punto de referencia para poder anunciar un servicio Web. Esta clase *Service* tienen tres propiedades *presents*, *describedby*, *supports* donde los rangos de estas propiedades son las tres clases asociadas para representar cada parte de un servicio: *ServiceProfile*, *ServiceModel*, *ServiceGrounding* respectivamente. Cada vez que queramos anunciar un servicio, deberemos de instanciar cada una de estas clases.

- La clase *Service* es presentada (*presents*) por *Service Profile*, que nos sirve para definir qué es lo que el servicio necesita del usuario o agente.
- La clase *Service* es descrita (*describedby*) por *Service Model*, en el que explica cómo trabaja el servicio (cómo debe usar los inputs, el flujo de datos en un servicio compuesto etc.).
- La clase *Service* es soportada (*supports*) por *Service Grounding*, nos indica cómo debe ser invocado el servicio.

2.3.6.1 Service Profile. Responde a la primera de las preguntas, es decir a "qué es lo que el servicio requiere de los usuarios, u otros agentes y qué cosas proporciona el servicio a ellos". Service Profile provee una forma de describir los servicios ofrecidos por los proveedores y los servicios necesitados por los solicitantes. Esencialmente, es una pequeña descripción de lo que hace el servicio, describiendo el servicio como una función de tres tipos básicos de información:

- Especificación comprensible para un humano del servicio, que contiene información sobre la organización que provee el servicio y participantes,

- Especificación de las **funcionalidades** que proporciona el servicio en términos de parámetros (entradas, salidas, precondiciones y efectos).

- Conjunto de **atributos no funcionales** que proveen información adicional sobre las necesidades del servicio (*QoS*, región en la que se aplica el servicio etc.).

La descripción funcional básicamente describe qué hace el servicio en cuanto a qué datos de entrada necesita, qué datos genera, precondiciones que se deben satisfacer y efectos generados.

- *hasParameter*
- *hasInput*
- *hasOutput*
- *hasPrecondition*
- *hasEffect*.

Los rasgos que especifican las características del servicio son una lista en la que pueden aparecer datos tales como clasificación del servicio (como las taxonomías usadas en UDDI), una medida de la calidad del servicio ofrecido, o cualquier otra información (tiempo de respuesta, disponibilidad geográfica del servicio, etc.). Como se puede intuir estos parámetros podrían usarse para incluir nociones de calidad de servicio. Es importante aclarar que es responsabilidad del cliente el uso de esta información, su verificación, y decidir qué hacer con ella. Se concreta en:

- *serviceParameter*. Es una lista de propiedades. Cada propiedad tiene rango ServiceParameter. La clase *ServiceParameter* tiene las propiedades:
- *serviceParameterName.* El nombre del parámetro. Puede ser un literal o la URI del parámetro.
- *sParameter.* Apunta al valor del parámetro dentro de alguna ontología.
- *serviceCategory.* El rango de esta propiedad es la clase ServiceCategory, la cual se compone de:
    - *categoryName*. Es el nombre de la categoría, pudiendo ser un literal, o una URI.
    - *taxonomy*. Puede ser la URI de una taxonomía, la URL donde la taxonomía reside, o un literal cualquiera.
    - *value*. Apunta un valor concreto dentro de la taxonomía. Puede haber varios.
    - *code*. El código asociado al tipo de servicio en la taxonomía.

2.3.6.2 Service Model. OWL-S define una clase derivada de *ServiceModel* para describir el modelado de los procesos. Esta clase se llama *Proces*. Los procesos concretos son clases derivadas de esta clase. Cada proceso descrito es una especificación de cómo es la interacción entre el cliente y el servicio.

Las precondiciones y los efectos se representan mediante reglas lógicas. Un proceso no deberá ser ejecutado a menos que sus precondiciones se cumplan (sean ciertas), si se ejecutase sin cumplirse las precondiciones el resultado está indefinido. El lenguaje en el que se van a expresar estas reglas puede ser cualquiera que tenga una representación textual. Las expresiones lógicas serán de tipo literal o literales XML.

El resultado de la invocación de un servicio se puede ver como la salida y el efecto, se puede condicionar el resultado del servicio mediante las propiedades *inCondition, hasResultVar, withOutput, hasEffect*. La propiedad *inCondition* especifica la condición bajo la que se produce este resultado y no otro. Las propiedades *withOutput* y *hasEffect* establecen qué sucede cuando la condición se satisface. La propiedad *hasResultVar* declara variables usadas en *inCondition*. Estas variables, llamadas *ResultVars*, son análogas a las variables *Locals*, y sirven para un propósito similar, permitiendo ligar las variables a las

precondiciones y usarlas en las declaraciones de salida y efectos. En OWL-S, los procesos se distinguen en atómicos, simples y compuestos.

- Los procesos atómicos son aquellos directamente invocables, no tienen subprocesos en un solo paso, al menos desde la perspectiva de quien lo invoca.

- Los procesos simples no son directamente invocables. Como los procesos atómicos pueden ser concebidos como que tienen un solo paso de ejecución.

Los procesos simples son usados como elementos de un proceso abstracto, pueden ser para mostrar una vista de un proceso atómico, o simplificar un proceso complejo. Se puede decir que un proceso simple es realizado por un proceso atómico o que expanden un proceso compuesto. Los procesos compuestos son procesos que se pueden descomponer en otros subprocesos tanto compuestos como no.

2.3.6.3 ServiceGrounding. Las clases *ServiceProfile y ServiceModel* se consideran especificaciones abstractas en el sentido de que no dan detalles acerca de formato de los mensajes, protocolo usados, URL de acceso al servicio, etc. La misión de la clase *ServiceGrounding* es proporcionar todos esos detalles. Esta tarea de concretar mensajes y operaciones definidos de manera abstracta ya la realizan los *bindings* de WSDL. OWL-S aprovecha WSDL y define nuevos puntos de extensión. Además, de aprovechar la relación de WSDL con SOAP y UDDI, permitiendo una extensión sencilla, para añadir contenido semántico a los *web services* tradicionales.

Para poder hacer referencia a WSDL desde OWL-S se crea la clase *WSDLGrounding*, subclase de *ServiceGrounding*. Cada instancia de esta clase contiene una lista de instancias de la clase *WSDLAtomicProcessGrounding*. Cada una de las instancias de *WSDLAtomicProcessGrounding* se refiere a elementos de WSDL a través de las siguientes propiedades:

- *wsdlVersion*. Una URI que indica la versión de WSDL que se usa.
- *wsdlDocument*. La URI del documento WSDL.
- *wsdlOperation*. La URI de la operación WSDL correspondiente a ese proceso atómico.
- *wsdlnputMessage*. Un objeto que contiene la URI de la definición del mensaje WSDL que contiene los valores de entrada de ese proceso atómico.
- *wsdOutputMessage*. Como el anterior, pero para los valores de salida.
- *wsdlInputs*. Un objeto con una lista de pares de mapeo. Cada par es una instancia de *WsdlInputMessageMap*. Un elemento del par (expresado con

la propiedad wsdlMessagePart) identifica la *part* del *message* usando una URI. El otro elemento indica cómo obtener ese *part* del *message* a partir de una o más entradas del proceso atómico OWL-S. En el caso más sencillo, en el que corresponde directamente con una única entrada OWL-S que tiene un tipo WSDL, basta con poner en el atributo *owlsParameter* la URI del objeto. En el peor de los casos hay que usar el atributo *xsltTransformation* para indicar una transformación XSLT que genere el *part* del *message* a partir de una instancia del proceso atómico.

- *WsdOutputs*. Como el anterior, pero para los valores de salida. En vez de *WsdlInputMessagePart* se usa *WsdlOutputMessagePart*.

La especificación de estos dos lenguajes son complementarias, OWL-S no sustituye a WSDL. Los lenguajes se solapan en lo que en WSDL se designan como tipos abstractos, que caracterizan las entradas y salidas del servicio. OWL-S provee la capacidad de darle sentido semántico a las definiciones abstractas, mientras que por otro lado no tiene la capacidad de expresar la información de ligado (*binding*) que tiene WSDL. La muestra la relación entre los componentes del WSDL y los de OWL-S.

2.3.6.4 El *Service Grounding.* *D*e un servicio sirve para especificar como acceder al servicio (formato de mensajes, protocolo de interacción, direccionamiento y transporte). Se puede ver como una correspondencia o mapeo entre la especificación abstracta y la concreta, de aquellos elementos de descripción del servicio que son requeridos para la interacción con el servicio (entradas y salidas de los procesos atómicos).

En principio para describir un servicio Web se necesitarán las tres propiedades para hacerlo correctamente, pero no se indica ninguna restricción respecto una cardinalidad mínima para las propiedades *presents*, *described by*, ni máxima cardinalidad para las propiedades *presents*, *supports* (un servicio podrá ser representado de diferentes formas, así como ser usado o invocado de diferentes formas si se quiere).

El trasfondo (*grounding*) de OWL-S se construye a través de una descripción en un archivo WSDL clásico, donde se agregan las siguientes extensiones:

- En el elemento *message*, se añade el atributo *owl-s-parameter*. Se le debe asignar como valor un objeto OWL-S (instancia de la clase *Parameter*).

  Alternativamente también se puede declarar la clase OWL en el elemento types.

- En el elemento *binding*, cuando el elemento *part* de *message* hace referencia a un tipo OWL, se debe poner *encodingStyle* a http://www.w3.org/2002/07/owl (o la versión que corresponda).

- En el elemento *operation* de *binding*, se añade el atributo *owl-s-process*. Se le debe asignar como nombre del proceso atómico OWL-S.

3. METODOLOGÍA

Para el desarrollo del presente proyecto se empezó por la búsqueda del estado del arte, etapa realizada durante todo el tiempo de desarrollo del presente proyecto. Para esta tarea se tuvo en cuenta:

- Tecnologías relacionadas.
- Normas y recomendaciones W3C pertinentes.
- Tesis, trabajos de grado y libros relacionados con Web Semántica.
- Aplicaciones software existentes, incluyendo APIs de java, protégé, documentación de desarrollo y documentación de usuario.

Otra tarea realizada fue la apropiación del conocimiento, la cual incluyó la discusión con especialistas y la prueba de ejemplos de herramientas para desarrollo de ontologías y creación de Servicios Web. Por último se desarrolló el prototipo, para lo cual se siguieron los siguientes pasos:

3.1 CREACIÓN DE LA ONTOLOGÍA DE LOS DATOS

Se puede considerar que la tarea más importante a la hora de desarrollar una aplicación basada en la tecnología de la Web Semántica es la de definir la ontología que contendrá los conceptos del dominio manejado (financiero, literario, médico, etc.), sus propiedades y las relaciones entre dichos conceptos.

El primer paso a la hora de modelar la ontología será analizar el dominio que contempla, con objeto de identificar los conceptos más destacados, relativos a dicho dominio.

Para la construcción de la ontología se utilizo el lenguaje de modelado de ontologías RDF/RDFS por las facilidades que brinda al momento de construir la base para el resto de lenguajes de ontologías (OIL, DAM+OIL, OWL). Para las pruebas de creación de las ontologías se utilizaron dos herramientas, de las cuales se observo lo siguiente:

Protegé, editor desarrollado por la universidad de Stanford, basado en el lenguaje Java y en forma de plugins; actualmente soporta la edición de ontologías con RDF/RDFS y dispone de un plugin para el desarrollo de ontologías en OWL y OWL-S. Se caracteriza por disponer de varias pantallas de formularios, para incluir las propiedades o características de las clases, atributos, instancias y relaciones de una ontología. La creación de la ontología en esta aplicación se facilitan si se

aplican conceptos de programación orientada a objetos. Se debe hacer una lectura completa del manual de protegé para construcción de ontologías con OWL, para crear una ontología propia.

Librería Jena 2.5, se puede encontrar en el portal sourceforge.net, es necesario un conocimiento tanto del lenguaje java, como de la librería jena y adicionalmente de los conceptos de construcción de ontologías. Para construir una ontología es necesario leer documentación acerca de la construcción de ontologías, documentación del api de jena y tener experiencia con programación en java.

Una ventaja adicional de la utilización de Protegé como editor de ontologías es la generación de clases e interfaces en java, necesarias al momento de generar el servicio Web y los clientes del servicio Web, si se utilizan objetos descritos por la ontología como parámetros de entrada o salida.

Para la creación de la ontología de los datos primero se crearon las clases y luego se asignaron las propiedades, restricciones y relaciones entre ellas. Para aumentar la expresividad de la ontología se utilizaron tantas propiedades inversas, simétricas y transitivas como fue posible, con el fin de facilitar el escalamiento del prototipo y la búsqueda de información desde diferentes puntos de entrada.

Para ver la ontología creada vea el Anexo 1: Prototipo de Ontología para la consulta de inversiones.

3.2 CREACIÓN DEL SERVICIO WEB

Utilizando el editor de Java NetBeans 5.5 se creo el servicio Web y una operación con parámetros de entrada, de salida y la lógica de la operación necesaria, se utilizó TOMCAT (versión 5.5.17) como herramienta de servidor, y el protocolo SOAP (Simple Object Access Protocol) para el controlador de mensajes.

Entre las ventajas que presenta NetBeans en la creación de Servicios Web está la creación del controlador de mensajes SOAP, la generación de una descripción WSDL del servicio y el enlace con la aplicación de servidor Tomcat.

Se utilizó Jena (version 2.5.5) para la lógica y la inferencia sobre la ontología para sacar resultados, se tomaron instancias creadas en el mismo archivo de la ontología (esquema), pero trabajar con instancias en diferentes archivos no presenta mayor inconveniente, gracias al ModelFactory de Jena, que permite relacionar un esquema con una instancia (y en archivos diferentes).

Se tuvieron problemas inicialmente con la creación de servicios Web con versiones de java inferiores a 1.5. Para ver la descripción WSDL del servicio vea el Anexo 2: Descripción WSDL del servicio prototipo de inversiones.

Para crear un Servicio Web con Netbeans se debe leer la documentación de NetBeans para la creación de Servicios Web y conocer de programación de funciones en Java.

3.3 DESCRIPCIÓN DEL SERVICIO WEB UTILIZANDO OWL-S

A partir del la descripción del servicio por WSDL (Web Service Description Language), se realizo la descripción del servicio por medio de OWL-S (Ontology Web Language for Services).

Se intentó utilizar un WSDL2WOLS hecho en java para hacer la desricpion OWL-S a partir del archivo WSDL del servicio, pero no funcionó quizá por problemas de incompatibilidad entre la descripción WSDL y los parámetros del archivo de configuración que requiere WSDL2OWLS para funcionar (config.xml).

Posteriormente se hizo la descripción del servicio con la ayuda de Protege y el plugin de para la descripción de servicios Web con OWL-S, para lo cual es necesario realizar la lectura completa del manual del plugin, la utilización de éste hizo más eficiente el trabajo, debido a que se tiene el control de la descripción y es visible tal descripción gráficamente, mientras que con la aplicación WSDL2OWLS, la única información que se tiene para generar la descripción OWL-S es el archivo de configuración donde se describen algunos parámetros pero se debe hacer manualmente y se desconoce como definir tipos de datos de entrada y salida del Servicio Web que están descritos por algún esquema en particular.

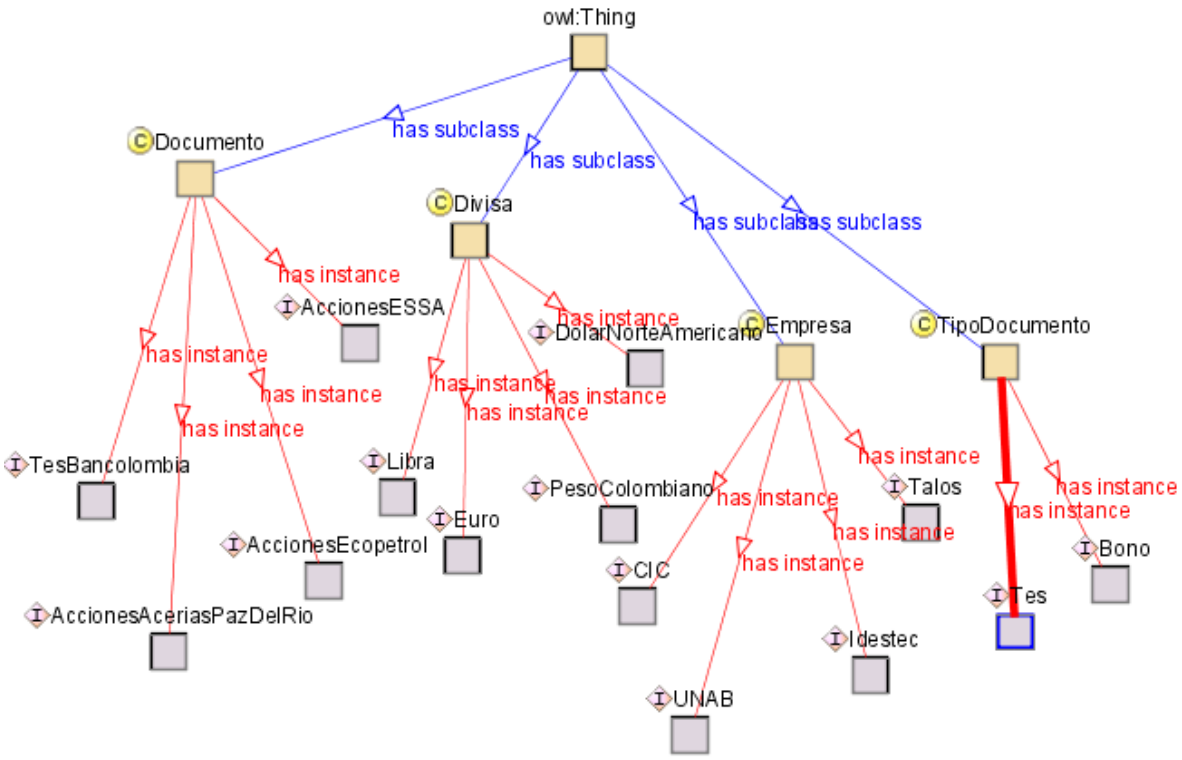
Para ver la descripción OWL-S del servicio vea el Anexo 3: Descripción del servicio prototipo de inversiones con OWL-

4. DESCRIPCIÓN DEL PROTOTIPOTIPO

El prototipo tiene 3 componentes que son: La ontología de los datos, descrita con OWL, el Servicio Web, descrito mediante WSDL y el servicio Web Semántico, escrito con OWL-S

Una vista de la ontología y las instancias de prueba creadas es la que provee el plugin de Protege denominado Jambalaya

Figura 4.1. Ontología e instancias de prueba del prototipo



Una vista de relaciones es la que nos muestra la figura 4.2

Para mostrar la estructura del Servicio Web Semántico también podemos utilizar el tab del plugin de OWL-S Editor, utilizado para hacer la descripción del servicio por medio de OWL-S (ver figura 4.3 y 4.4)

Figura 4.2. Relaciones de la ontología del documento cambiario

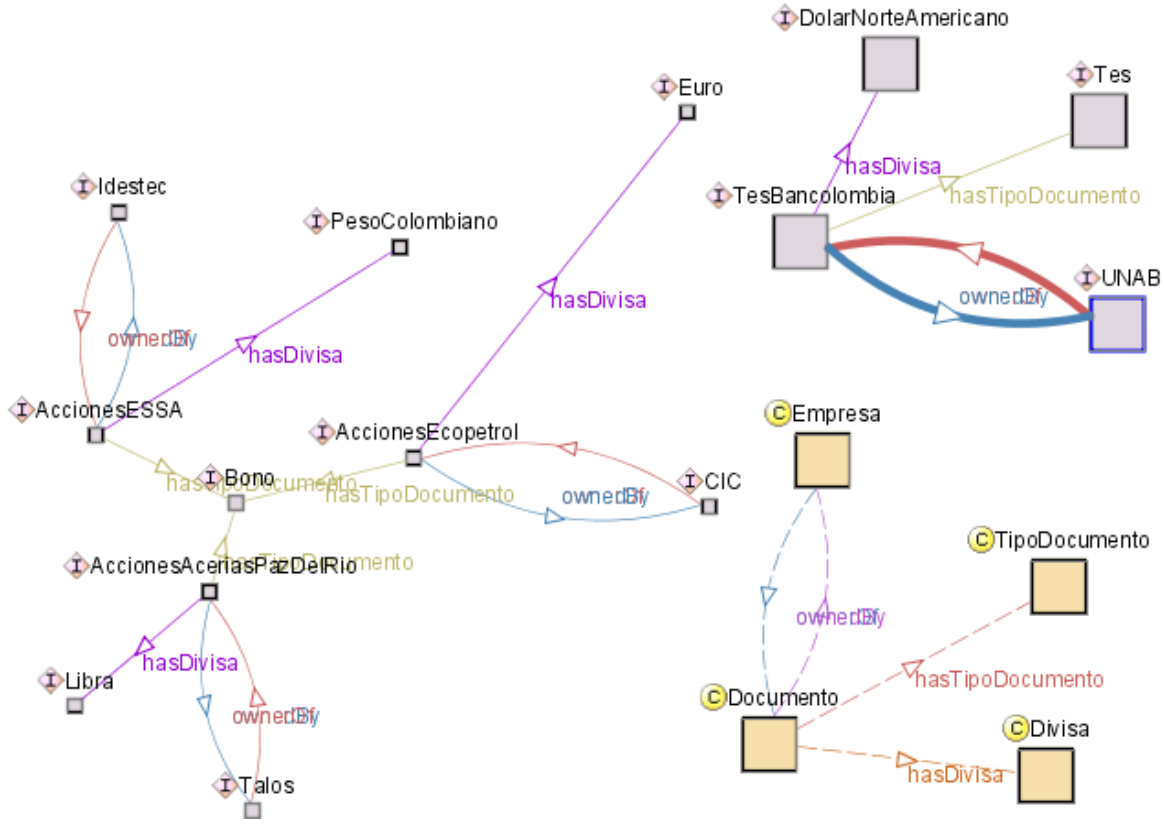


Figura 4.3. Proceso del Servicio Web Semántico

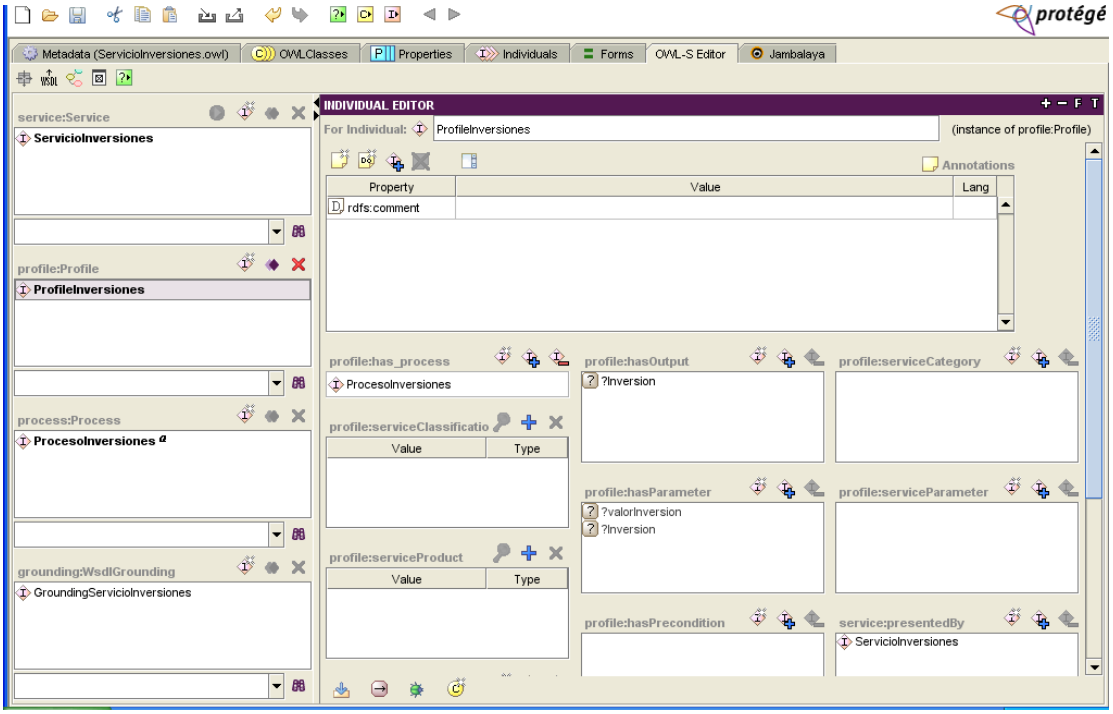
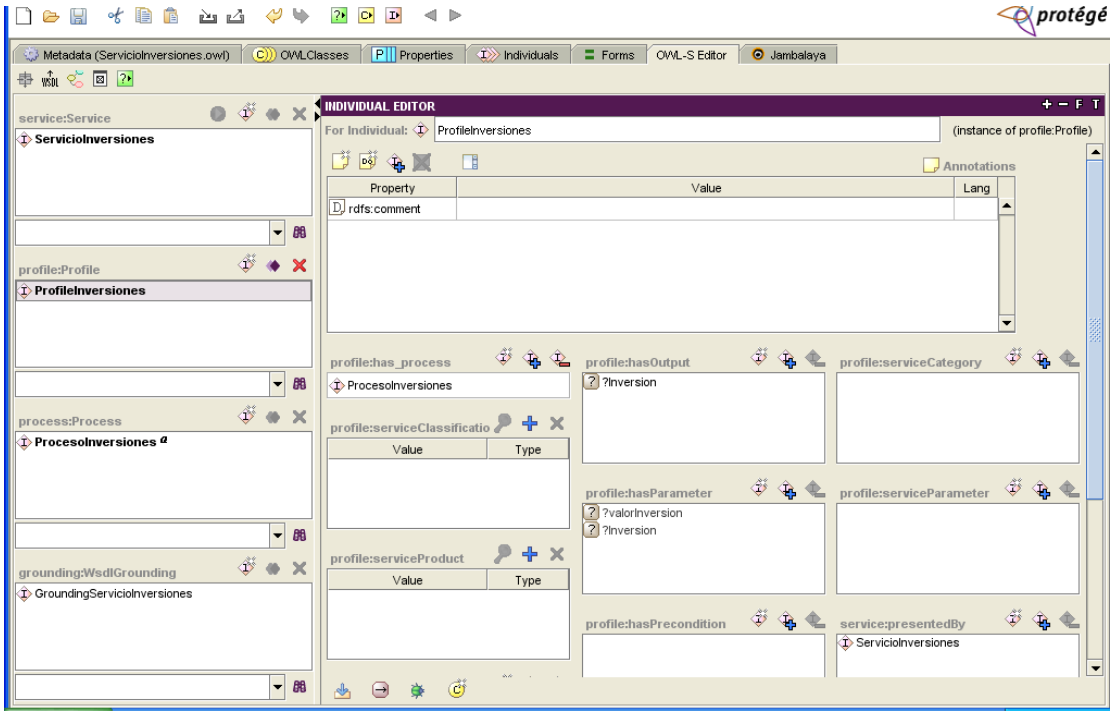


Figura 4.4. Profile del Servicio Web Semántico



Para las pruebas de la lógica del servicio Web se utilizó un programa corto en JAVA, esto con el fin de quitar posibles errores por cuestiones de conectividad e integración con el servidor Tomcat, el código fuente del programa de prueba puede verse en el anexo 6.

En las pruebas del servicio Web como tal, se observó que si se utilizan parámetros de entrada y/o de salida descritos por alguna ontología en especifico es importante que esté a disposición de los clientes potenciales del servicio las clases e interfaces que permiten crear (para ver las interfaces que describen los objetos de la ontología remítase al anexo 4) y manipular objetos de estos tipos, de lo contrario se pueden generar obstáculos para el uso del servicio, ya que cuando se utilizan servicios Web se espera que devuelvan información explícita acerca de lo que se busca y que no requiera de ningún tipo de tratamiento. Por lo anterior se sugiere, que aunque los Servicios Web pueden recibir y retornar cualquier tipo de dato, solamente se deben trabajar con tipos de datos primitivos para los parámetros de entrada y salida, esto con el fin de evitar trabajos al cliente final del servicio y mejorar rendimiento de los clientes.

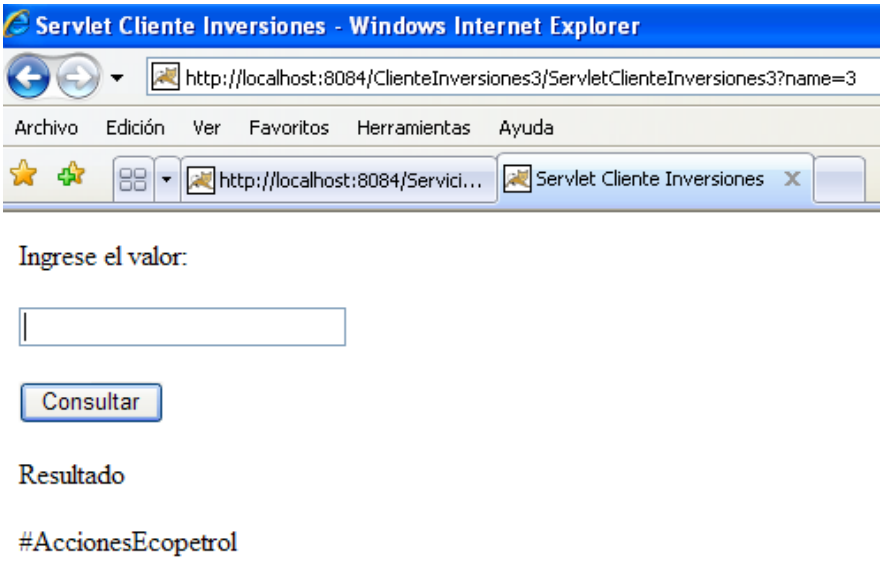
Las pruebas sobre el servicio Web fueron realizadas a nivel de red local, para ver código fuente del Servicio Web remítase al anexo 5.

Al momento de realizar las pruebas en el computador utilizado para el desarrollo del proyecto, se observó que al realizar cambios en el servicio Web estos no eran automáticamente tomados por el cliente, por lo cual se debía actualizar la referencia al servicio Web, pero este procedimiento en algunas ocasiones provocaba que el proyecto que contenía el Servicio Web se dañara, el servicio Web se volvía un método común o al momento de volver a correr el proyecto, Tomcat no podía iniciarse, La solución adoptada fue volver a crear un proyecto web y volver a crear un Servicio Web con el mismo nombre para poder utilizar el cliente generado y posteriormente en el cliente actualizar la referencia al nuevo Servicio Web.

4.1 FUNCIONALIDAD DEL SERVICIO

Con el fin de verificar la integración de los componentes utilizados en el prototipo (ontología, servicio, motor de inferencia) se creo un servlet que permite la búsqueda en las instancias de la ontología creada, basado en la propiedad de valor del documento.

Figura 4.5. Interfaz cliente para prueba del prototipo del servicio

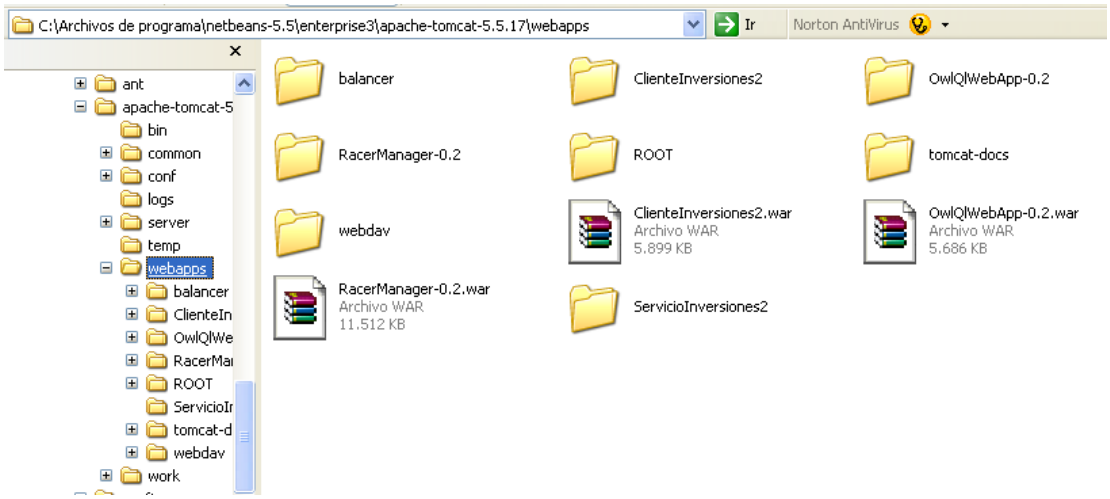


Para el prototipo del cliente solamente se devuelve la primera instancia de la clase Documento (definida en la ontología prototipo) que coincida con el parámetro de consulta que para este caso es el valor del documento

4.2 INSTALACIÓN DEL SERVICIO

Para la instalación del servicio es necesario descomprimir el archivo de distribución *ServicioInversiones2.war* en la carpeta webapps donde está instalado Tomcat y ubicar la ontología dentro de la carpeta *ServicioInversiones2* la cual fue creada al descomprimir el archivo de distribución.

Figura 4.6 Carpeta para instalación del servicio

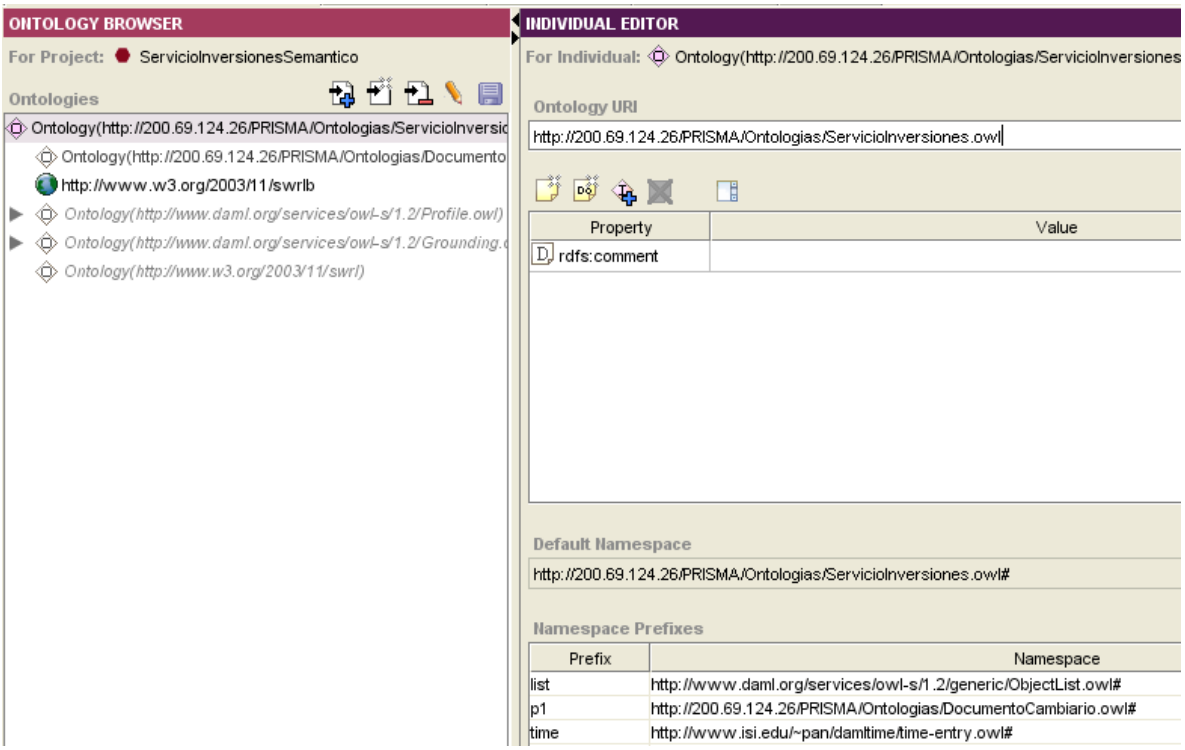


Para instalar el cliente del servicio se debe descomprimir el archivo de distribución *ClienteInversiones2.war* en la carpeta webapps donde está instalado Tomcat (Ver figura 4.5).

Una vez instalado el servicio y/o el cliente, debe reiniciarse el servidor Tomcat esto se hace por medio de los archivos *shutdown.bat* y *startup.bat* localizados en la carpeta *bin* dentro de la carpeta donde está instalado Tomcat (ubicar en figura 4.5 carpeta *bin*).

Es recomendable que tanto la URI (namespace) de la ontología como la del Servicio Web sean cambiadas a la ubicación en la cual van a estar disponibles para que puedan ser utilizados (esto se puede hacer con Protégé).

Figura 4.7 Definición de la URI del servicio con Protégé



Las pruebas fueron realizadas con la versión 5.5.17 de Tomcat, java 1.4.2_03 y sistema operativo Windows XP SP2.

5. CONCLUSIONES Y TRABAJOS FUTUROS

- Al realizar las pruebas de construcción de la ontología de los datos (con OWL) y la ontología del servicio (con OWL-S), se pudo determinar que Protege es una herramienta completa, en constante evolución, con buena documentación sobre la herramienta en sí y sobre los plugin, lo cual la convierte en punto de apoyo para la difusión de tecnologías relacionadas con Web Semántica.

- A partir de las pruebas realizadas se determinó que el plugin de Protege OWLS – Editor, permite una mejor descripción del servicio, comparado con WSDL2OWLS, ya que se tiene control de el tipo de datos de entrada y de salida del sistema, permite comentarios sobre los mismos y referencias a los *namespace* que los definen.

- A partir de la prueba del Servicios Web se sugiere que se utilicen tipos de datos primitivos para los datos de entrada y salida de los Servicio Web Semánticos, ya que un tipo de dato no primitivo, supone la utilización de una clase que permite manipular el objeto, lo cual aumenta la complejidad en la creación de los clientes del servicio y disminuye el rendimiento de los mismos.

Como trabajo futuro se propone la extensión y refinamiento de la ontología, así como del servicio aquí descrito, con el fin de hacer un servicio más aplicable en la realidad.

Se recomienda la utilización de OWL-S para proyectos aplicados, actualmente en desarrollo, tales como el sistema RUNT (Registro Único Nacional de Transito), cuyo objetivo es tener registro de los automotores nacionales y conocer el estado mecánico de los mismos, debido a que utilizará archivos provenientes de diferentes herramientas software. Para este proyecto en particular se propone la creación de un Servicio Web para el ministerio de transito, que provea la descripción de los vehículos a partir de la palca del mismo, un Servicio Web para que cada uno de los servidores de los CDA (Centros de Diagnóstico Automotor) para que publique los resultados de las inspecciones técnico mecánicas, y una ontología, para presentar los resultados de las inspecciones y las descripciones de los vehículos. Bajo esta arquitectura orientada a servicios, se evitarían problemas de compatibilidad que tendría que solucionar un sistema de información tradicional y mecanismos de cortafuegos (firewall) que impidan la normal comunicación de cada uno de los servidores de los CDA con el servidor del ministerio de tránsito.

BIBLIOGRAFÍA

[Ber98] Berners-Lee, Tim. Semantic Web Road map. Disponible en: http://www.w3.org/DesignIssues/Semantic.html. 1998.

[Ber01] Berners-Lee, Tim; Hendler, james and Lassila, Ora. "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", Scientific American.2001.

[Bro02] Broekstra, Jeen; Kampman, Arjohn and van Harmelen, Frank. "Sesame: a Generic Architecture for Storing and Querying RDF and RDF Schema". 1st International Semantic Web Conference (ISWC2002), June 9-12, 2002. Sardinia, Italy.

[Cec02] Ceccaroni, L. and Ribiere,M., "Experiences in Modeling Agentcities Utilily ontologies with a Collaborative Approach". In: Ontologies in Agent Systems Workshop, Autonomous Agents and Multi-Agents Systems Conference 2002, Bologna, Italy, July 2002.

[Dum00] Dumbill, Edd. "Putting RDF to Work". Article on XML.com. 08-09-2000.

[Fik03] Fikes, Richard; Hayes, Patrick and Horrocks, Ian. "Daml Query language abstract specification". Disponible en: http://www.daml.org/2003/04/dql

[Fik04] Fikes, Richard; Hayes, Patrick and Horrocks, Ian. "OWL-QL, A Language for Deductive Query Answering on the Semantic Web".2004

[GRU93b] Gruber, T.R.,"A Translation Approach to Portable Ontology Specifications".
Knowledge Acquisition Lournal, Vol.5 pp. 199-20, 1993.

[Gom04b] Gomez-Perez; Asunción; Gonzalez-Cabero, Rafael, and Lama, Manuel: "A Frameworks for Description, Compositions, and Evaluation of Semantic web services". IEEE Intelligent Systems. Special Issue on Semantic Web Services. 2004.

[JENA05] Jena 2. "A Semantic Web Framework" Disponible en: http://www.hpl.hp.com/semweb/jena.htm

[Nar03] Narayanan, S. and McIllarith, S. "Analysis and simulation of Web Services", Computers networks, Vol. 42, No 5. 2003, Pág. 675-695.

[OWLS03] “OWL-S 1.0 Release 2003-11”, Disponible en: http://www.daml.org/services/owl-s/1.0/

[OWL04] “OWL Web Ontology Language Reference”, W3C Recommendation 10 February 2004. Disponible en: http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[ODESWS05] ODE SWS A Toolset for Design and Composition of Semantic Web Service. Disponible en: http://kw.dia.fi.upm.es/odesws/

[RDQL04] RDQL Query de HP. En http://www.hpl.hp.com/semweb/rdql.htm

[Rec03] Rector Alan L. “Oiled Normalised Ontology Tutorial- Biomedical Version”, Oiled Ontology Editor web page (http://oiled.man.ac.uk/), 2004.

[Rec04a] Rector Alan L.”Defining N-ary Relations on the semantic Web: Use With Individuals”, W3C Working Draft 21 Julio 2004.

[RuleML03] “The Rule Markup Initiative”. Disponible en: http://www.ruleml.org/

[RQL] “The RDF Query Language” Disponible en: http://139.91.183.30:9090/RDF/RQL/ Forth Institute of Computer Sciences.

[SAM05] Samper Zapater José Javier, Ontologías para Servicios Web Semánticos de Información de Tráfico: Descripción y Herramientas de explotación, Universitat de València

[Sat03] Sattler, U.; Calvanese, D. and Molitor, R., The description Logic handbook, Theory, Implementation and Applications “relationships with other formalisms, semantic data Models”, Pág. 164, 2003.

[SeRQL04] Aduna B.V., “Chapter 6. The SeRQL query language, rev. 1.1” from User Guide for Sesame Updated for Sesame release 1.1 Copyright © 2002-2004. Sirma AI Ltd. Disponible en http://www.openrdf.org/doc/users/ch06.html.

[SWRL04] “SWRL 0.6: Semantic Web Rules Languages” Disponible en: http://www.daml.org/2004/04/swrl/rules-all.html

[SPARQL05] SPARQL W3C. Disponible en: http://www.w3.org/TR/ref-sparql-query/.

[Ste03] Stevens, Robert; Wroe, Chris; Bechhofer, Sean; Lord, Phillip; Rector, Alan and Globe, Carole. Conference Review, “Building ontologies in DAML + OIL, Comparative and Functional Genomics”, Comp Funct Genom 2003; 4: 133-141. Published online in wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cfg.233.

[W3C04] World Wide Web Consortium (W3C). Web Services Glossary, W3C Working Group Note 11. Disponible en: http://www.w3c.org/TR/ws-gloss/, Feb 2004.

[VAN96] Van Heijst, G.; Shereiber, A.T. and Wieling, B.J. “Using Explicit Ontologies in KBS Development” International Journal of Human and Computer Studies, 1996.

ANEXOS

Anexo 1: Prototipo de Ontología para la consulta de inversiones

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#"
   xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
  xml:base="http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Divisa">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >Identifica tipos de monedas</rdfs:comment>
   <owl:disjointWith>
    <owl:Class rdf:ID="TipoDocumento"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf:ID="Empresa"/>
   </owl:disjointWith>
   <owl:disjointWith>
    <owl:Class rdf:ID="Documento"/>
   </owl:disjointWith>
   <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >divisa</rdfs:label>
  </owl:Class>
  <owl:Class rdf:about="#Documento">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >Especifica las propiedades del documento cambiario</rdfs:comment>
   <rdfs:subClassOf>
    <owl:Restriction>
     <owl:someValuesFrom>
      <owl:Class rdf:about="#TipoDocumento"/>
     </owl:someValuesFrom>
     <owl:onProperty>
      <owl:FunctionalProperty rdf:ID="hasTipoDocumento"/>
     </owl:onProperty>
    </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
    <owl:Restriction>
     <owl:someValuesFrom>
```

```
        <owl:Class rdf:about="#Empresa"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="ownedBy"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#TipoDocumento"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Empresa"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Divisa"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >documento</rdfs:label>
</owl:Class>
<owl:Class rdf:about="#TipoDocumento">
  <owl:disjointWith rdf:resource="#Documento"/>
  <owl:disjointWith rdf:resource="#Divisa"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Tipo de documento cambiario</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >tipoDocumento</rdfs:label>
  <owl:disjointWith>
    <owl:Class rdf:about="#Empresa"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Empresa">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >empresa</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Descripcion de Empresas</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isOwnerOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Documento"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:disjointWith rdf:resource="#TipoDocumento"/>
  <owl:disjointWith rdf:resource="#Documento"/>
```

```
  <owl:disjointWith rdf:resource="#Divisa"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#ownedBy">
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:range rdf:resource="#Empresa"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isOwnerOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isOwnerOf">
  <rdfs:range rdf:resource="#Documento"/>
  <rdfs:domain rdf:resource="#Empresa"/>
  <owl:inverseOf rdf:resource="#ownedBy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasInteres">
  <rdfs:range>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="interes"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >0</owl:minCardinality>
    </owl:Restriction>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Documento"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasTipoDocumento">
  <rdfs:domain rdf:resource="#Documento"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#TipoDocumento"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#interes">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Interes actual del documento cambiario</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasDivisa">
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:range rdf:resource="#Divisa"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasValor">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
```

```
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:range>
   <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >0</owl:minCardinality>
    <owl:onProperty>
     <owl:FunctionalProperty rdf:ID="valor"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:range>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="direccion">
  <rdfs:domain rdf:resource="#Empresa"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="nombreEmpresa">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Empresa"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#valor">
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="nombreTipoDocumento">
  <rdfs:domain rdf:resource="#TipoDocumento"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="nombreDivisa">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Divisa"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >nombreDivisa</rdfs:label>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Especifica el nombre de la divisa</rdfs:comment>
</owl:FunctionalProperty>
<Divisa rdf:ID="Libra">
  <nombreDivisa rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Libra</nombreDivisa>
</Divisa>
<Documento rdf:ID="AccionesEcopetrol">
```

```
    <ownedBy>
      <Empresa rdf:ID="Talos">
        <direccion rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Giron</direccion>
        <nombreEmpresa rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >TALOS LTDA</nombreEmpresa>
        <isOwnerOf rdf:resource="#AccionesEcopetrol"/>
        <isOwnerOf>
          <Documento rdf:ID="AccionesTalos">
            <hasTipoDocumento>
              <TipoDocumento rdf:ID="Bono">
                <nombreTipoDocumento
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >BONO</nombreTipoDocumento>
              </TipoDocumento>
            </hasTipoDocumento>
            <valor rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
            >13000.0</valor>
            <hasDivisa>
              <Divisa rdf:ID="PesoColombiano">
                <nombreDivisa
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >PesoColombiano</nombreDivisa>
              </Divisa>
            </hasDivisa>
            <ownedBy rdf:resource="#Talos"/>
            <interes rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
            >0.3</interes>
          </Documento>
        </isOwnerOf>
      </Empresa>
    </ownedBy>
    <hasDivisa rdf:resource="#PesoColombiano"/>
    <valor rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
    >10000.0</valor>
    <interes rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
    >0.2</interes>
    <hasTipoDocumento>
      <TipoDocumento rdf:ID="Tes">
        <nombreTipoDocumento
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >TES</nombreTipoDocumento>
      </TipoDocumento>
    </hasTipoDocumento>
  </Documento>
```

```
  <Divisa rdf:ID="DolarEstadounidense">
    <nombreDivisa rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >DolarEstadounidense</nombreDivisa>
  </Divisa>
  <Divisa rdf:ID="Euro">
    <nombreDivisa rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Euro</nombreDivisa>
  </Divisa>
</rdf:RDF>
```

Anexo 2: Descripción WSDL del servicio prototipo de inversiones

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://inversiones.step.org/"
name="divisaDocumentoService" xmlns:tns="http://inversiones.step.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://inversiones.step.org/"
schemaLocation="DivisaDocumentoService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="hayDocumentosConDivisa">
    <part name="parameters" element="tns:hayDocumentosConDivisa"/>
  </message>
  <message name="hayDocumentosConDivisaResponse">
    <part name="parameters" element="tns:hayDocumentosConDivisaResponse"/>
  </message>
  <portType name="divisaDocumento">
    <operation name="hayDocumentosConDivisa">
      <input message="tns:hayDocumentosConDivisa"/>
      <output message="tns:hayDocumentosConDivisaResponse"/>
    </operation>
  </portType>
  <binding name="divisaDocumentoPortBinding" type="tns:divisaDocumento">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="hayDocumentosConDivisa">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="divisaDocumentoService">
    <port name="divisaDocumentoPort" binding="tns:divisaDocumentoPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

Anexo 3: Descripción del servicio prototipo de inversiones con OWL-S

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns="http://200.69.124.26/PRISMA/ServiciosWeb/ServicioInversiones#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:time="http://www.isi.edu/~pan/damltime/time-entry.owl#"
    xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:expr="http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:grounding="http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xml:base="http://200.69.124.26/PRISMA/ServiciosWeb/ServicioInversiones">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl"/
>
    <owl:imports rdf:resource="http://www.w3.org/2003/11/swrlb"/>
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports                    rdf:resource="http://www.daml.org/services/owl-
s/1.2/Grounding.owl"/>
    <owl:imports rdf:resource="http://www.w3.org/2003/11/swrl"/>
  </owl:Ontology>
  <profile:Profile rdf:ID="ServicioInversionesProfile">
    <profile:textDescription
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Servicio    en    linea    para    consultar    donde    hacer
inversiones</profile:textDescription>
    <profile:has_process>
      <process:AtomicProcess rdf:ID="ObtenerMejorInversion">
        <process:hasInput>
          <process:Input rdf:ID="DineroDisponible">
            <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#float</process:parameterType>
            <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Cantidad de dinero disponible para invertir</rdfs:comment>
          </process:Input>
        </process:hasInput>
```

```
        <process:hasInput>
          <process:Input rdf:ID="Divisa">
            <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

>http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Divisa</proc
ess:parameterType>
          </process:Input>
        </process:hasInput>
        <process:hasOutput>
          <process:Output rdf:ID="DocumentoAInvertir">
            <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

>http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Documento<
/process:parameterType>
            <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Documento en el que se sugiere hacer la inversion</rdfs:comment>
          </process:Output>
        </process:hasOutput>
        <service:describes>
          <service:Service rdf:ID="AgenteInversiones">
            <service:presents rdf:resource="#ServicioInversionesProfile"/>
            <service:describedBy rdf:resource="#ObtenerMejorInversion"/>
            <service:supports>
              <grounding:WsdlGrounding rdf:ID="Inversiones_Grounding">
                <service:supportedBy rdf:resource="#AgenteInversiones"/>
                <grounding:hasAtomicProcessGrounding>
                  <grounding:WsdlAtomicProcessGrounding
rdf:ID="ObtenerMejorInversionGrounding">
                    <grounding:owlsProcess rdf:resource="#ObtenerMejorInversion"/>
                    <grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

>http://200.69.124.26/PRISMA/ServiciosWeb/ServicioInversiones.wsdl</grounding:
wsdlDocument>
                    <grounding:wsdlOperation>
                      <grounding:WsdlOperationRef
rdf:ID="ObtenerMejorInversion_Operatior">
                        <grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

>http://200.69.124.26/PRISMA/ServiciosWeb/ServicioInversiones.wsdl#ObtenerMe
jorInversionOperation</grounding:operation>
```

```
                    <grounding:portType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
                    >ObtenerMejorInversionPortType</grounding:portType>
                  </grounding:WsdlOperationRef>
                </grounding:wsdlOperation>
              </grounding:WsdlAtomicProcessGrounding>
            </grounding:hasAtomicProcessGrounding>
          </grounding:WsdlGrounding>
        </service:supports>
      </service:Service>
    </service:describes>
  </process:AtomicProcess>
 </profile:has_process>
 <profile:hasInput rdf:resource="#Divisa"/>
 <profile:hasInput rdf:resource="#DineroDisponible"/>
 <service:presentedBy rdf:resource="#AgenteInversiones"/>
 <profile:serviceName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
 >AgenteInversiones</profile:serviceName>
 <profile:hasOutput rdf:resource="#DocumentoAInvertir"/>
</profile:Profile>
<grounding:WsdlAtomicProcessGrounding rdf:ID="Wsdl">
 <grounding:owlsProcess rdf:resource="#ObtenerMejorInversion"/>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Anexo 4: Interfaces para la utilización de los objetos descritos por la ontología

```
package org.step.inversiones;

import edu.stanford.smi.protegex.owl.model.*;

public class FactoryInversiones {

   private OWLModel owlModel;


   public FactoryInversiones(OWLModel owlModel) {
      this.owlModel = owlModel;
   }


   public RDFSNamedClass getDivisaClass() {
      final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Divisa";
      final String name = owlModel.getResourceNameForURI(uri);
      return owlModel.getRDFSNamedClass(name);
   }

   public Divisa createDivisa(String name) {
      final RDFSNamedClass cls = getDivisaClass();
      return (Divisa) cls.createInstance(name).as(Divisa.class);
   }

   public Divisa getDivisa(String name) {
      return (Divisa) owlModel.getRDFResource(name).as(Divisa.class);
   }


   public RDFSNamedClass getDocumentoClass() {
      final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Documento";
      final String name = owlModel.getResourceNameForURI(uri);
      return owlModel.getRDFSNamedClass(name);
   }

   public Documento createDocumento(String name) {
      final RDFSNamedClass cls = getDocumentoClass();
      return (Documento) cls.createInstance(name).as(Documento.class);
   }
```

```
    public Documento getDocumento(String name) {
        return (Documento) owlModel.getRDFResource(name).as(Documento.class);
    }


    public RDFSNamedClass getTipoDocumentoClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#TipoDocume
nto";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public TipoDocumento createTipoDocumento(String name) {
        final RDFSNamedClass cls = getTipoDocumentoClass();
        return (TipoDocumento) cls.createInstance(name).as(TipoDocumento.class);
    }

    public TipoDocumento getTipoDocumento(String name) {
        return (TipoDocumento)
owlModel.getRDFResource(name).as(TipoDocumento.class);
    }


    public RDFSNamedClass getEmpresaClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Empresa";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public Empresa createEmpresa(String name) {
        final RDFSNamedClass cls = getEmpresaClass();
        return (Empresa) cls.createInstance(name).as(Empresa.class);
    }

    public Empresa getEmpresa(String name) {
        return (Empresa) owlModel.getRDFResource(name).as(Empresa.class);
    }
}

//////////////////////////////////////////////////////////////////////


package org.step.inversiones;
```

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.*;

public class Shema {
    /** <p>The ontology model that holds the vocabulary terms</p> */
    private static OntModel m_model = ModelFactory.createOntologyModel(
OntModelSpec.OWL_MEM, null );

    /** <p>The namespace of the vocabulary as a string</p> */
    public static final String NS =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#";

    /** <p>The namespace of the vocabulary as a string</p>
     *  @see #NS */
    public static String getURI() {return NS;}

    /** <p>The namespace of the vocabulary as a resource</p> */
    public static final Resource NAMESPACE = m_model.createResource( NS );

    public static final ObjectProperty hasInteres = m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasInteres" );

    public static final ObjectProperty hasValor = m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasValor" );

    public static final ObjectProperty hasDivisa = m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasDivisa" );

    public static final ObjectProperty hasTipoDocumento =
m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasTipoDocu
mento" );

    public static final ObjectProperty isOwnerOf = m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#isOwnerOf" );

    public static final ObjectProperty ownedBy = m_model.createObjectProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#ownedBy" );

    /** <p>Especifica el nombre de la divisa</p> */
    public static final DatatypeProperty nombreDivisa =
m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#nombreDivisa
" );
```

```
    public static final DatatypeProperty nombreTipoDocumento =
m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#nombreTipoD
ocumento" );

    public static final DatatypeProperty valor = m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#valor" );

    public static final DatatypeProperty nombreEmpresa =
m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#nombreEmpr
esa" );

    public static final DatatypeProperty direccion =
m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#direccion" );

    /** <p>Interes actual del documento cambiario</p> */
    public static final DatatypeProperty interes = m_model.createDatatypeProperty(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#interes" );

    /** <p>Especifica las propiedades del documento cambiario</p> */
    public static final OntClass Documento = m_model.createClass(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Documento"
);

    /** <p>Descripcion de Empresas</p> */
    public static final OntClass Empresa = m_model.createClass(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Empresa" );

    /** <p>Tipo de documento cambiario</p> */
    public static final OntClass TipoDocumento = m_model.createClass(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#TipoDocume
nto" );

    /** <p>Identifica tipos de monedas</p> */
    public static final OntClass Divisa = m_model.createClass(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Divisa" );

    public static final Individual AccionesTalos = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#AccionesTalo
s", Documento );
```

```
    public static final Individual DolarEstadounidense = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#DolarEstadou
nidense", Divisa );

    public static final Individual PesoColombiano = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#PesoColombi
ano", Divisa );

    public static final Individual Bono = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Bono",
TipoDocumento );

    public static final Individual Talos = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Talos",
Empresa );

    public static final Individual AccionesEcopetrol = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#AccionesEco
petrol", Documento );

    public static final Individual Libra = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Libra", Divisa
);

    public static final Individual Tes = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Tes",
TipoDocumento );

    public static final Individual Euro = m_model.createIndividual(
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Euro", Divisa
);

}

//////////////////////////////////////////////////////////////////////

package org.step.inversiones;

import edu.stanford.smi.protegex.owl.model.*;

import java.util.*;

public interface Documento extends OWLIndividual {
```

```
    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasDivisa

    Divisa getHasDivisa();

    RDFProperty getHasDivisaProperty();

    boolean hasHasDivisa();

    void setHasDivisa(Divisa newHasDivisa);


    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasInteres

    Collection getHasInteres();

    RDFProperty getHasInteresProperty();

    boolean hasHasInteres();

    Iterator listHasInteres();

    void addHasInteres(Object newHasInteres);

    void removeHasInteres(Object oldHasInteres);

    void setHasInteres(Collection newHasInteres);


    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasTipoDocu
mento

    TipoDocumento getHasTipoDocumento();

    RDFProperty getHasTipoDocumentoProperty();

    boolean hasHasTipoDocumento();

    void setHasTipoDocumento(TipoDocumento newHasTipoDocumento);


    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#hasValor
```

```
    Object getHasValor();

    RDFProperty getHasValorProperty();

    boolean hasHasValor();

    void setHasValor(Object newHasValor);

    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#interes

    float getInteres();

    RDFProperty getInteresProperty();

    boolean hasInteres();

    void setInteres(float newInteres);

    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#ownedBy

    Collection getOwnedBy();

    RDFProperty getOwnedByProperty();

    boolean hasOwnedBy();

    Iterator listOwnedBy();

    void addOwnedBy(Empresa newOwnedBy);

    void removeOwnedBy(Empresa oldOwnedBy);

    void setOwnedBy(Collection newOwnedBy);

    // Property http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#valor

    float getValor();
```

```
    RDFProperty getValorProperty();

    boolean hasValor();

    void setValor(float newValor);
}

//////////////////////////

package org.step.inversiones;

import edu.stanford.smi.protegex.owl.model.*;

public interface TipoDocumento extends OWLIndividual {

    //                                                                  Property
http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#nombreTipoD
ocumento

    String getNombreTipoDocumento();

    RDFProperty getNombreTipoDocumentoProperty();

    boolean hasNombreTipoDocumento();

    void setNombreTipoDocumento(String newNombreTipoDocumento);
}

//////////////////////////////////////

package org.step.inversiones;

import edu.stanford.smi.protegex.owl.model.*;

public interface Divisa extends OWLIndividual {

    //                                                                  Property
http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#nombreDivisa

    String getNombreDivisa();

    RDFProperty getNombreDivisaProperty();

    boolean hasNombreDivisa();
```

```
    void setNombreDivisa(String newNombreDivisa);
}

package org.step.inversiones;


import edu.stanford.smi.protegex.owl.model.*;

public class FactoryInversiones {

    private OWLModel owlModel;


    public FactoryInversiones(OWLModel owlModel) {
        this.owlModel = owlModel;
    }


    public RDFSNamedClass getDivisaClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Divisa";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public Divisa createDivisa(String name) {
        final RDFSNamedClass cls = getDivisaClass();
        return (Divisa) cls.createInstance(name).as(Divisa.class);
    }

    public Divisa getDivisa(String name) {
        return (Divisa) owlModel.getRDFResource(name).as(Divisa.class);
    }


    public RDFSNamedClass getDocumentoClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Documento";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public Documento createDocumento(String name) {
        final RDFSNamedClass cls = getDocumentoClass();
```

```
        return (Documento) cls.createInstance(name).as(Documento.class);
    }

    public Documento getDocumento(String name) {
        return (Documento) owlModel.getRDFResource(name).as(Documento.class);
    }


    public RDFSNamedClass getTipoDocumentoClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#TipoDocume
nto";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public TipoDocumento createTipoDocumento(String name) {
        final RDFSNamedClass cls = getTipoDocumentoClass();
        return (TipoDocumento) cls.createInstance(name).as(TipoDocumento.class);
    }

    public TipoDocumento getTipoDocumento(String name) {
        return (TipoDocumento)
owlModel.getRDFResource(name).as(TipoDocumento.class);
    }


    public RDFSNamedClass getEmpresaClass() {
        final String uri =
"http://200.69.124.26/PRISMA/Ontologias/DocumentoCambiario.owl#Empresa";
        final String name = owlModel.getResourceNameForURI(uri);
        return owlModel.getRDFSNamedClass(name);
    }

    public Empresa createEmpresa(String name) {
        final RDFSNamedClass cls = getEmpresaClass();
        return (Empresa) cls.createInstance(name).as(Empresa.class);
    }

    public Empresa getEmpresa(String name) {
        return (Empresa) owlModel.getRDFResource(name).as(Empresa.class);
    }
}
```

Anexo 5: Código del Servicio Web

```
package org.step.inversiones;

import java.util.Collections;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

/**
 *
 * @author Wilson Carlos
 */
public class ManejadorServicio implements SOAPHandler<SOAPMessageContext> {

    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
        return true;
    }

    public Set<QName> getHeaders() {
        return Collections.EMPTY_SET;
    }

    public boolean handleFault(SOAPMessageContext messageContext) {
        return true;
    }

    public void close(MessageContext context) {
    }

}


//////////////////////////////////////////////////////

package org.step.inversiones;

import javax.jws.HandlerChain;
import javax.jws.WebMethod;
```

```
import javax.jws.WebParam;
import javax.jws.WebService;
import org.step.inversiones.impl.*;
import org.step.inversiones.*;
//import com.hp.hpl.jena.rdf.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.*;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.vocabulary.*;
 /**
 *
 * @author Wilson Carlos
 */
@WebService()
@HandlerChain(name = "divisaDocumento_handlerChain", file =
"divisaDocumento_handler.xml")
public class divisaDocumento {
    /**
     * Web service operation
     */
    @WebMethod
    public String hayDocumentosConDivisa(@WebParam(name = "valor") float
valor) {
        String s="No hay sugerencias";
        try{
            if(valor>0){
                Model
esquema=FileManager.get().loadModel("DocumentoCambiario.owl");
                Model data=FileManager.get().loadModel("DocumentoCambiario.owl");
                Reasoner reas=ReasonerRegistry.getOWLReasoner();
                reas=reas.bindSchema(esquema);
                InfModel inferenciaModelo=ModelFactory.createInfModel(reas,data);
                Resource
documentos=inferenciaModelo.getResource("DocumentoCambiario");
                s=documentos.getModel().toString();
            }
            else
                s="No existe ninguna sugerencia";
        }catch(Exception exc){
            s=exc.toString();
        }
        return s;
    }
```

```
}
```

Anexo 6: Código fuente del programa para probar la lógica del Servicio Web

```
package servicioinversionesconsola;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.*;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.vocabulary.*;
/**
 *
 * @author WilsonCarlos
 */
public class Main {

    /** Creates a new instance of Main */
    public Main() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Main m=new Main();
        m.analizarOWL(2);
    }
   public void analizarOWL(float valor){
       String s="No hay sugerencias";
        try{
            if(valor>0){
                s="PROYECTO WILSON CARLOS";
                Model
esquema=FileManager.get().loadModel("c:\\DocumentoCambiario.owl");
                Model
data=FileManager.get().loadModel("c:\\DocumentoCambiario.owl");
                Reasoner reas=ReasonerRegistry.getOWLReasoner();
                reas=reas.bindSchema(esquema);
                InfModel inferenciaModelo = ModelFactory.createInfModel(reas,data);
                Resource documentos = inferenciaModelo.getResource("#Documento");
                Property propiedad = inferenciaModelo.getProperty("ownerOf");
                Resource divisa = inferenciaModelo.getResource("PesoColombiano");
                Resource empresa = inferenciaModelo.getResource("#Idestec");
                //propiedad=null;
                //divisa=null;
```

```
//s=inferenciaModelo.getDeductionsModel().getProperty("Valor",Float.toString(valo
r)).toString();
                //String                                                    r
=inferenciaModelo.getProperty(documentos,propiedad).toString();
                //s="\""+Boolean.toString(b)+"\"";
                //s="\""+documentos.toString()+"\"";
                //s="\""+data.listStatements(documentos,propiedad,2).toString()+"\"";
                for(StmtIterator
i=data.listStatements(documentos,propiedad,divisa);i.hasNext();){
                    Statement stmt=i.nextStatement();
                    System.out.println(" - "+PrintUtil.print(stmt));
                }
                //s=documentos.getModel().toString();
            }
            else
                s="No existe ninguna sugerencia";
        }catch(Exception exc){
            s="Exception controlada "+exc.toString();
        }

    }

}
```