PROYECTO FIN DE CARRERA

GODO: Descubrimiento basado en Objetivos para Servicios Web Semánticos

Facultad de Informática

Universidad de Murcia

Septiembre 2009

Director: Francisco García Sánchez

Autora: Mª Antonia Parreño González

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El proyecto sobre el que trataremos a lo largo de este documento, nace como ampliación del proyecto GODO (Generación inteligente de Objetivos para el DescubrimientO de servicios web semánticos) financiado con las ayudas concedidas por el Ministerio de Industria, Turismo y Comercio. El objetivo del proyecto GODO consiste fundamentalmente en desarrollar una plataforma para facilitar el descubrimiento de Servicios Web gracias a la incorporación de semántica a los mismos y utilizando mecanismos de procesamiento del lenguaje natural e ingeniería ontológica.

Este documento se encuentra dividido en cuatro capítulos:

- **Introducción y Referencias Históricas:** En este capítulo se realiza una pequeña introducción para situar al lector en el contexto del proyecto. Además, se introducen de forma muy sencilla las tecnologías base sobre las que se fundamenta, y se trata brevemente acerca de los antecedentes al proyecto GODO.

- **Análisis de Objetivos y Metodología:** En este capítulo se puntualizan claramente cuáles son los objetivos del proyecto. Posteriormente, se comenta de forma muy detallada las tecnologías empleadas desde el punto de vista más técnico, y los patrones de diseño empleados.

- **Diseño y Resolución del trabajo:** En esta parte del documento, se hace una introducción sobre el trabajo realizado con anterioridad referente a GODO. Además, se detalla el proyecto de forma más precisa y extensa. Posteriormente, se realiza un análisis de requisitos. También se describe la arquitectura de la plataforma con las modificaciones realizadas, la arquitectura de la aplicación y se describe el modo en que el módulo OWLPath ha sido integrado en la herramienta. Por último, se explica el funcionamiento de la aplicación.

- **Conclusiones y Vías futuras:** En este último capítulo, se exponen las conclusiones obtenidas tras la elaboración del proyecto. Finalmente, se detallan cuáles serán las actuaciones futuras para continuar con el desarrollo del trabajo realizado que mejoren el funcionamiento de la plataforma actual.

CAPÍTULO I. INTRODUCCIÓN Y REFERENCIAS HISTÓRICAS

1.1. INTRODUCCIÓN

Internet se ha convertido, a día de hoy, en uno de los mayores repositorios de información que existen a nivel mundial. Además, se ha transformado en una plataforma tecnológica, donde las organizaciones pueden desplegar, compartir y explorar procesos de negocio a través de diversas tecnologías como los Servicios Web.

El problema principal es que la Web ha crecido desmesuradamente en los últimos años y sigue creciendo de forma exponencial, lo que implica que la búsqueda de información concreta se convierta en una tarea ardua e incluso imposible en algunos casos. Para solucionar este problema, Tim Berners-Lee propuso en 2001 (Berners-Lee et al., 2001) la idea de Web semántica, consistente en añadir información semántica a la World Wide Web, mejorando la interoperabilidad entre sistemas.

Gracias a la aceptación de esta idea, la Web está evolucionando hacia un modelo semántico (ver figura 1). Los Servicios Web que presentaban un carácter meramente sintáctico, han evolucionado incorporando este paradigma, dando lugar a lo que hoy se conoce como Servicios Web Semánticos.

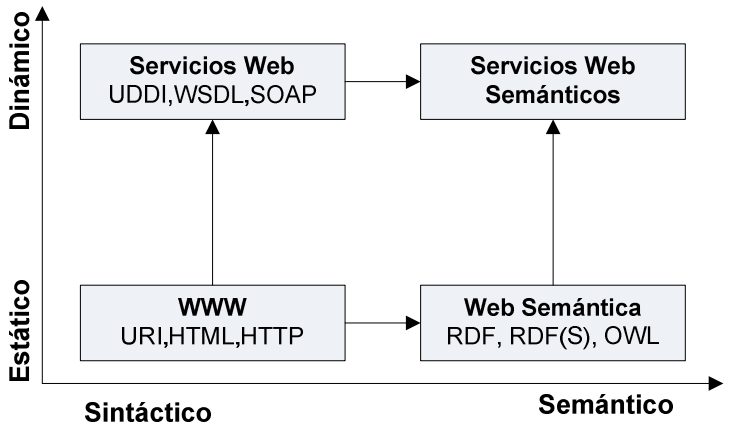

Figura 1. Evolución de la Web (Fensel et al., 2002).

Los Servicios Web Semánticos se fundamentan principalmente en el empleo de las tecnologías de la Web Semántica a disposición de los Servicios Web, dando lugar a Servicios Web inteligentes, de forma que su descubrimiento, composición e invocación pueda realizarse de forma automática. Actualmente el W3C está examinando cinco aproximaciones a los Servicios Web Semánticos con la finalidad de obtener un estándar para esta tecnología: OWL-S (Martin et al., 2004), WSMO (Roman et al., 2004), SWSF

(Battle et al., 2005), WSDL-S (Akkiraju et al., 2005) y SAWSDL (Farrell et al., 2007). Algunas de estas aproximaciones vienen acompañadas de entornos de ejecución de Servicios Web Semánticos que las soportan: WSMX como entorno de ejecución de WSMO, para OWL-S existe OWL-S Virtual Machine, IRS-III y METEOR-S.

El problema que nos surge con estos entornos de ejecución es que los usuarios que los utilizan tienen que tener un alto grado de conocimiento informático y, por lo tanto, no son herramientas globalmente accesibles.

Este proyecto se encuentra enmarcado en la tecnología de los Servicios Web Semánticos y tiene como propósito conseguir una infraestructura basada en la aplicación de técnicas de análisis de lenguaje natural y ontologías, de forma que el usuario sea capaz, a través de una sencilla interfaz, de expresar los objetivos de búsqueda y obtener como resultado el descubrimiento de los Servicios asociados a éstos.

El sistema, basándose en tecnologías de la Web Semántica y del Procesamiento del Lenguaje Natural, deberá ser capaz de interpretar la entrada del usuario, ya sea introducida por él mismo o guiado por la ontología, y realizar de forma automática todo el proceso de selección y descubrimiento de Servicios Web Semánticos que satisfagan las necesidades y deseos de éste.

1.2. ANTECEDENTES

Durante el desarrollo del proyecto GODO se obtuvieron importantes logros tanto a nivel científico, técnico como empresarial. Además, se avanzó en el estudio de técnicas y tecnologías relacionadas con los Servicios Web Semánticos. También se realizaron estudios de las herramientas de generación del lenguaje natural controlado (CNL, *Controlled Natural Language*) que se han desarrollado hasta el momento.

En base a este estudio, uno de los logros más importantes obtenidos ha sido con respecto a la extracción de ontologías a partir de consultas en lenguaje natural, y al procesamiento de este lenguaje, utilizado por el usuario "no experto" para realizar su consulta.

Otro de los estudios ya realizados tenía por objeto la elección de una de las plataformas para el descubrimiento y ejecución de Servicios Web. Entre las herramientas analizadas se encontraban WSMX (Bussler et al., 2005), IRS-III (Domingue et al., 2008), OWL-S Virtual Machine (Paolucci et al., 2003), METEOR-S (Patil et al., 2004) e INFRAWEBS (Atanasova et al., 2005). Este último es un proyecto desarrollado por Atos Research & Innovation, nodo de Atos Origin, con la finalidad de crear una plataforma que permita el análisis, diseño y ejecución de Servicios Web Semánticos en un entorno colaborativo. El estudio dio como resultado la fusión de dos componentes WSMX e INFRAWEBS, de forma que se subsanasen las carencias que tienen ambas plataformas cuando se aplican por separado.

En las siguientes dos sub-secciones se describe de forma precisa cada una de las tecnologías principales en las que se fundamenta este trabajo: la Web Semántica y los Servicios Web Semánticos. De este modo, se obtiene una visión inicial de los

inconvenientes que existen actualmente y cómo se subsanan gracias a la incorporación de información de carácter semántico a las mismas.

1.2.1. WEB SEMÁNTICA

La World Wide Web (WWW) surgió en 1989 por Tim Berners-Lee como medio de comunicación a través de Internet. Actualmente podemos encontrar en la Web más de 25 billones (WWWS, 2009) de páginas indexadas y su crecimiento se estima en diez millones de páginas Web diarias (HT, 2009), convirtiéndolo en uno de los medios de comunicación más importantes, equiparable a la televisión o a la radio. El exceso de información ha provocado que el proceso de búsqueda se convierta en una tarea muy compleja. Como ejemplo, si deseamos obtener información referente a "león" como animal en uno de los buscadores más influyentes como es Google, obtenemos como resultado información de León como ciudad española y no como lo que realmente estamos buscando. Para solucionar estos problemas basta con que el usuario redefina, afinando la búsqueda aún más, pero ¿qué ocurriría si la búsqueda es más compleja? Por ejemplo, si deseamos buscar información relativa a "artículos sobre García Lorca" la mayoría de buscadores nos ofrecerán como resultado artículos del propio García Lorca, no sobre éste, complicando la extracción de resultados.

Para solventar estos problemas, ocasionados en parte por el crecimiento desmesurado de contenidos en Internet, su desorden y la carencia de significado de los mismos, nació la Web Semántica tras la publicación del artículo "*The Semantic Web*" de Tim Berners-Lee en la prestigiosa revista "*Scientific America*" en el año 2001 (Berners-Lee et al., 2001).

La Web semántica propone superar las barreras que posee la Web actual mediante la descripción formal de los contenidos y servicios disponibles en la WWW. La idea principal consiste en dotar de más significado a la Web actual, convirtiendo la información a un lenguaje formal de modo que la máquina pueda entenderlo y sea capaz de procesar su contenido, razonar con él, combinarlo y realizar deducciones lógicas para resolver problemas automáticamente, como el que se nos planteaba en el apartado anterior (W3CWS, 2008).

Los buscadores Web, tal y como los conocemos actualmente, utilizan un proceso de búsqueda por palabra clave y no tiene en cuenta el significado de la oración completa que introdujo el usuario. Por ello dotamos de semántica a la información para que sea compresible, no sólo por humanos, sino también por las máquinas.

En la figura 2 se puede observar un intento fallido de búsqueda de vuelos a Praga para mañana por la mañana, mostrándonos información errónea.

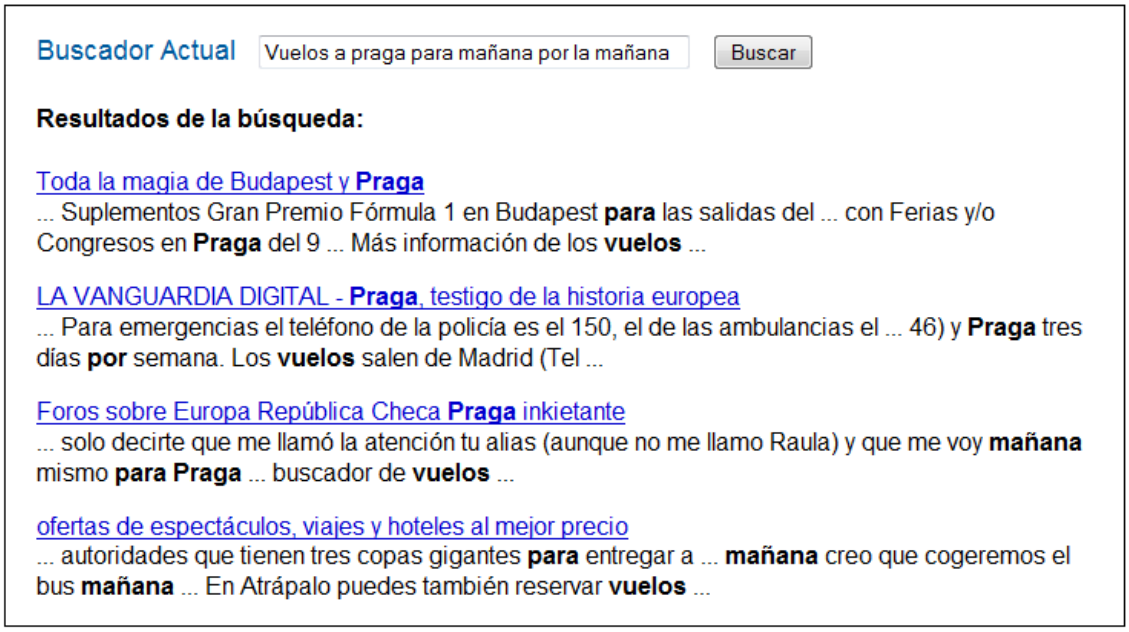

Figura 2. Buscador Web actual (W3CWS, 2008).

Si dotamos a la Web de significado accederemos a la información deseada de manera más ágil y directa, como podemos ver en la siguiente imagen en la que se nos muestra un buscador semántico (ver figura 3).

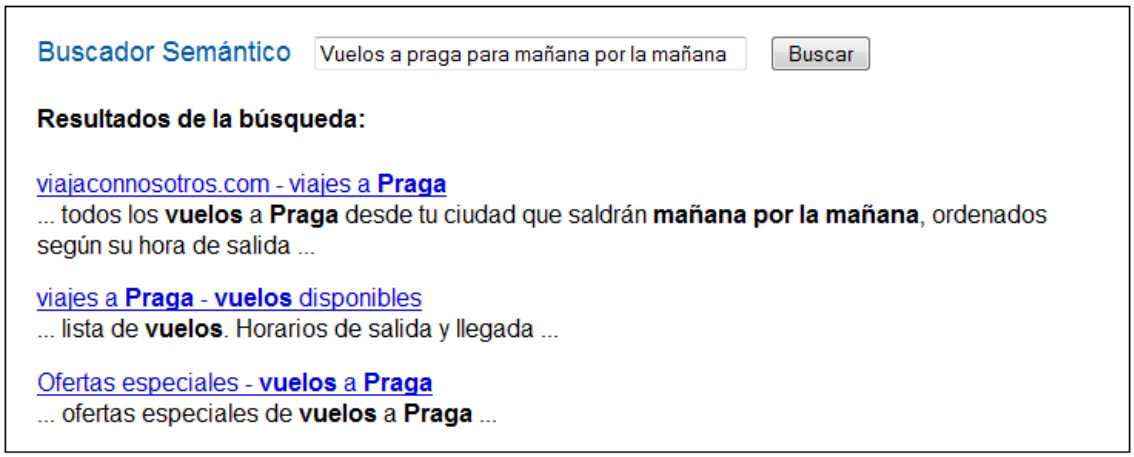

Figura 3. Buscador Web semántico (W3CWS, 2008).

1.2.2. SERVICIOS WEB SEMÁNTICOS

Internet no es sólo un espacio de información, sino que también lo es de interacción entre el usuario y la máquina (Olaya et al., 2006). A través de la Web el usuario, de forma remota, puede solicitar un servicio ofrecido por algún proveedor de Internet. Para que se produzca esta interacción debe existir un mecanismo de comunicación entre aplicaciones, que las permita interactuar entre sí. Gracias a los Servicios Web conseguimos invocar funciones independientemente del sistema operativo o la plataforma en la que se ejecuten.

La incorporación de semántica a estos Servicios, facilita la automatización del proceso de descubrimiento, composición, invocación, interoperabilidad y ejecución de los mismos. Algunos de los mecanismos más importantes para la incorporación de semántica a la descripción de los servicios, se fundamentan en el uso de ontologías, tales como las citadas OWL-S o WSMO.

Las principales características en el uso de los Servicios Web Semánticos son las siguientes (Valledor, 2006):

- Publicación de servicios en un registro semántico.
- Descubrimiento automático en función de la petición y la descripción semántica publicada en el servicio.
- Selección de servicios en caso de conflicto entre varios que satisfagan la petición.
- Composición de Servicios, de modo que el resultado de varios servicios formen la solución a la petición del usuario.
- Invocación del Servicio. Validando los parámetros del servicio.
- Despliegue.
- Gestión de las ontologías.

Entre las características que hemos mencionado destacamos las herramientas que han surgido para la composición e invocación de servicios. Uno de los entornos de ejecución más completos para OWL-S es el "*OWL-S Virtual Machine*" que proporciona un cliente de Servicios Web de propósito general para la invocación de servicios provistos por OWL-S. WSMX (*Web Service Execution Environment*) es un entorno de ejecución para el descubrimiento, selección, composición, mediación e invocación de Servicios Web Semánticos basados en WSMO, al igual que IRS-III (*Internet Reasoning Service*), un framework para Servicios Web Semánticos que da soporte a la publicación, localización, composición y ejecución de Servicios Web Semánticos. Por último, el proyecto METEOR-S tiene como propósito definir y dar soporte al ciclo de vida completo de los Servicios Web Semánticos, es decir, desarrollo de Servicios Web Semánticos, publicación y descubrimiento de servicios, composición de procesos Web, optimización y análisis de restricciones y finalmente la ejecución de Servicios Web Semánticos.

CAPÍTULO II. ANÁLISIS DE OBJETIVOS Y METODOLOGÍA

2.1. OBJETIVOS

Como se ha comentado anteriormente, este proyecto tiene por objetivo fundamental ampliar y rediseñar el estado actual del proyecto GODO mejorando, por un lado, la adaptación de los módulos a diferentes tecnologías consiguiendo, de este modo, que GODO pueda interactuar con otros entornos de ejecución además de WSMX y, por otro, añadiendo un nuevo método de consulta y búsqueda basado en ingeniería ontológica.

Para satisfacer este objetivo, es necesario definir una nueva capa de abstracción que haga posible a cada uno de los módulos de GODO usar diferentes tecnologías semánticas, permitiendo que la evaluación de las mismas sea más sencilla. Una vez realizado este proceso, será preciso incorporar al proyecto un módulo de definición de consultas guiadas por la ontología y un nuevo mecanismo de selección de objetivos más avanzado y perfeccionado.

Para llevar a cabo estos objetivos generales, se han identificado los siguientes sub-objetivos:

1. Evaluación de las diferentes tecnologías para la Web Semántica y las herramientas que existen.

2. Evaluación de las diferentes tecnologías y herramientas para los Servicios Web Semánticos.

3. Estudio de la arquitectura y funcionalidad de GODO en su versión anterior.

4. Definición, desarrollo y pruebas de la capa de abstracción para cada uno de los módulos de GODO.

5. Estudio y evaluación de la herramienta OWLPath para la introducción de texto guiado por la ontología.

6. Diseño, desarrollo y puesta en marcha del nuevo módulo para la entrada de consultas guiadas por la/s ontología/s del dominio.

7. Desarrollo de un nuevo módulo de selección de objetivos desde dos perspectivas diferenciadas:

    a. Transformando los objetivos y las ontologías del dominio definidos en la plataforma de Servicios Web Semánticos a un lenguaje ontológico estándar y razonar sobre éste para obtener los objetivos.

b. Adaptar OWLPath para que genere consultas adaptadas al lenguaje de definición de ontologías y objetivos de la propia plataforma de Servicios Web Semánticos.

2.2. METODOLOGÍA

El primer paso antes de comenzar el proyecto fue seleccionar una metodología de desarrollo de software entre las más empleadas. La metodología utilizada está basada en el modelo incremental de Pressman (Pressman, 2005) (ver figura 4).

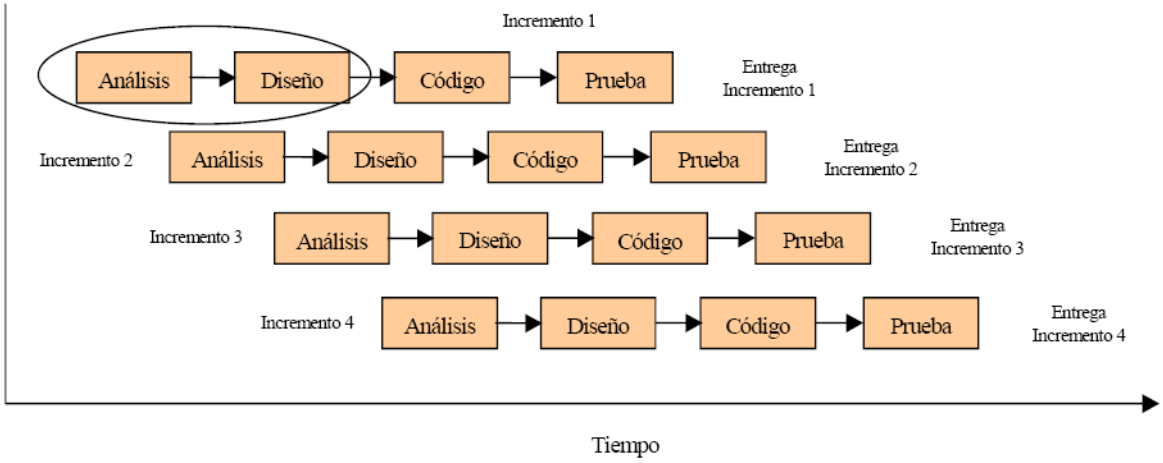

Figura 4. Metodología del Modelo Incremental (adaptado de Pressman, 2005).

Esta metodología, como su propio nombre indica, consiste en la realización de pequeños progresos durante el desarrollo del software. El primer incremento es fundamental puesto que se afrontan los requisitos básicos, las funciones extras se realizarán en los siguientes incrementos. Una vez comprobado el funcionamiento de la primera entrega, se desarrolla un plan para el incremento siguiente. Este proceso se repite hasta alcanzar el objetivo deseado. Este modelo tiene la ventaja de que en cada incremento, se produce una entrega del sistema operacional.

Como se puede apreciar en la figura 4, se realiza una etapa previa de análisis del módulo a desarrollar, posteriormente se diseña y se codifica, y por último se prueba. Si durante esta etapa se verifica su correcto funcionamiento se produce un incremento y se realiza la misma operación.

Esta metodología es la que más se ajusta al proyecto puesto que se realizan pequeños progresos en el desarrollo y por lo tanto se adapta mejor a las modificaciones realizadas sobre los módulos del sistema GODO.

2.3. HERRAMIENTAS Y TECNOLOGÍAS PARA EL DISEÑO

En cuanto a la metodología de diseño, se ha optado por una metodología orientada a objetos, que facilita la tarea de modelado gracias a la proximidad entre los conceptos del mundo real con los conceptos a modelar. La orientación a objetos facilita la transición de una fase a otra del modelado, la construcción, mantenimiento y reutilización del mismo.

En cuanto al lenguaje de modelado se ha utilizado UML (UML), y como herramienta para el desarrollo de modelos hemos utilizado StarUML (STARUML). A continuación daremos unas pinceladas sobre UML y la herramienta StarUML.

2.3.1. UML

UML (*Unified Modeling Language*) es un lenguaje para el modelado, construcción y documentación de elementos de un sistema orientado a objetos. Gracias a UML existe una herramienta compartida entre todos los ingenieros de software que trabajan en el desarrollo orientado a objetos, convirtiéndose en un estándar de facto apoyado por el OMG (*Object Management Group*). UML es un lenguaje para especificar el sistema y no un proceso, es decir ofrece al desarrollador un "plano" del sistema que se va a construir.

Para realizar el modelado de un sistema orientado a objetos podemos utilizar diferentes diagramas categorizados jerárquicamente. Algunos de los diagramas más importantes y que se han aplicado en este proyecto son:

- Los diagramas de clases, utilizados para la descripción de la estructura del sistema, mostrando sus clases, atributos y las relaciones entre ellos. Estos diagramas son usados durante el proceso de análisis y diseño del sistema.
- Los diagramas de secuencia, se utilizan para modelar la interacción entre los objetos del sistema.
- Los diagramas de colaboración, utilizados para mostrar las interacciones entre objetos. Es una alternativa a los diagramas de secuencia, proporcionando una representación principal del escenario, ya que las colaboraciones se organizan en torno a las interacciones de unos objetos con otros.

2.3.2. STARUML

StarUML es una herramienta de modelado UML de código abierto. Esta herramienta está basada en los estándares UML y MDA (*Model-Driven Architecture*).

Como características más importantes es posible destacar las siguientes:

- Soporte para once tipos diferentes de diagrama de UML versión 1.4. (diagramas de clases, de casos de uso, actividades, colaboración, secuencia, etcétera.). También acepta notación UML 2.0.
- Exportación de diagramas a formato gráfico (BMP, JPG).
- Soporte OCL (*Object Constraint Language*) para clases. Permite definir restricciones a aplicar en los diagramas UML mediante el OCL.
- Ingeniería inversa. Permite a partir de un conjunto de clases en un lenguaje orientado a objetos, generar su diagrama de clases.
- Soporte para la generación de código, Java, Visual Basic, Delphie, C++ y C#.
- Interfaz de módulos extensible basada en mecanismos plug-in.

2.3.3. PATRONES DE DISEÑO EMPLEADOS

2.3.3.1 Patrón Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador (MVC) tiene como finalidad desacoplar la vista del modelo de la aplicación para mejorar la reutilización. De este modo un cambio en la vista provoca el menor impacto posible en la lógica de negocio de la aplicación.

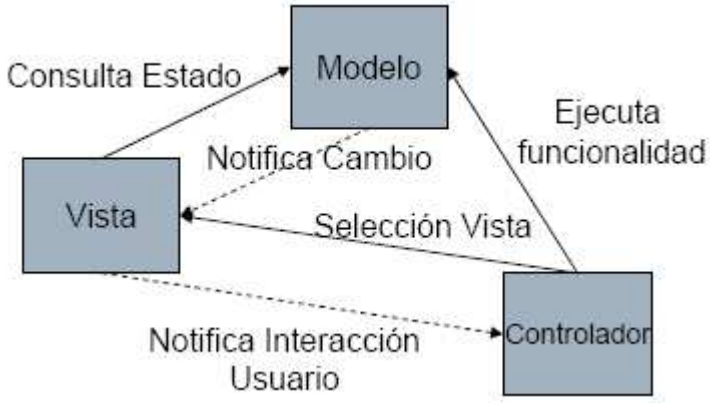

Figura 5. Patrón Modelo-Vista-Controlador (Bermúdez, 2009b).

Como podemos ver en la figura 5, el modelo es el responsable de acceder a la capa de almacenamiento de los datos, manipulándolos y controlando sus transformaciones. El modelo es el encargado de notificar los cambios a la vista.

La vista, por su parte, es la responsable de recibir los datos del modelo y mostrarlos al usuario. Interactúa con el modelo a través del controlador.

El controlador es el responsable de definir el comportamiento de la aplicación. Actúa sobre la información representada en el modelo. Su funcionamiento dentro de una aplicación Web se describe en dos ámbitos:

- Cuando se produce algún cambio en la vista o en el modelo el controlador informa a ambos para que refresquen la información. Además, se encarga de seleccionar la vista para la respuesta.

- El controlador es el responsable de seleccionar la vista que corresponde a una determinada respuesta, es decir, se encarga de gestionar la navegación de la aplicación.

2.3.3.2. Patrón Singleton

El patrón Singleton tiene como propósito el asegurar una única instancia de una clase, de modo que sólo exista un punto de acceso global a ella. A continuación mostramos el esquema de clases típico de este patrón (ver figura 6).

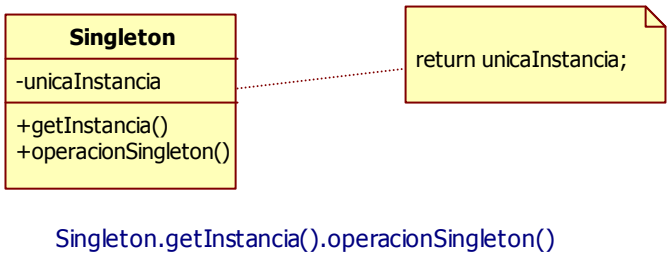

Figura 6. Patrón Singleton (adaptado de Larman, 2004).

A través de este patrón se consigue obtener un acceso controlado a una única instancia, y sólo la propia clase puede crear la instancia del objeto. El patrón presenta un atributo privado y estático que representa el objeto único a instanciar. Además tiene un método estático para devolver la instancia del objeto si ya está creado y en caso de que no exista lo crea y lo devuelve. Normalmente, el constructor de la clase Singleton posee un nivel de visibilidad privado, para que sólo pueda ser instanciado a través del método público que devuelve la instancia única.

2.3.3.3. Patrón Método Factoría

El patrón Método Factoría permite a una clase delegar la instanciación a sus subclases. Este patrón se utiliza cuando una clase no puede anticiparse a la clase de objetos que debe crear o se desea que sean las subclases las que especifiquen los objetos a instanciar.

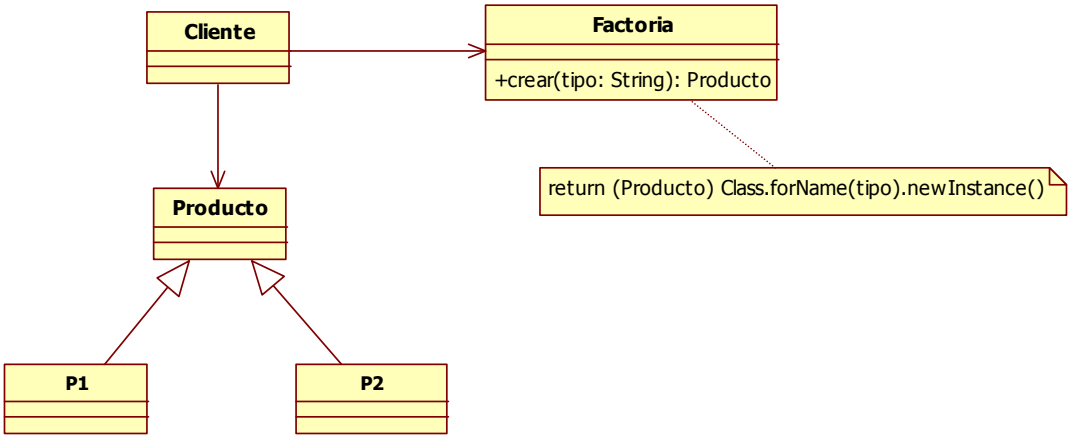

Figura 7. Patrón Método Factoría (adaptado de Larman, 2004).

En este proyecto se hace uso del Método Factoría con metaclases, lo que significa que la instanciación del objeto se realiza a través del nombre de la clase que queremos crear. En

la figura anterior (ver figura 7), se muestra un ejemplo de creación de productos, donde es la factoría la encargada de crear el producto P1 o P2, según el valor de la variable tipo con el que se realice la llamada. Este valor es una cadena de texto con el nombre de la clase a instanciar. De este modo, si tipo es P1, se creará un objeto de la clase P1.

2.4. HERRAMIENTAS Y TECNOLOGÍAS PARA LA IMPLEMENTACIÓN

Para la implementación y desarrollo de GODO, se ha hecho uso del paradigma de Programación Orientada a Objetos (Meyer, 1999). Este paradigma proporciona las siguientes características:

- **Herencia:** Las clases pueden incorporar propiedades y métodos de otras clases formando una jerarquía de clasificación. La herencia facilita el polimorfismo y el encapsulamiento permitiendo la construcción de objetos a partir de otros ya predefinidos sin necesidad de reescribir todo el código.

- **Encapsulamiento:** Reuniendo los elementos pertenecientes a una misma entidad, al mismo nivel de abstracción.

- **Abstracción:** Cada objeto en el sistema puede definirse de manera abstracta, de forma que puede realizar su trabajo, informar, cambiar su estado y "comunicarse" con otros objetos sin que éstos conozcan cómo se implementan sus características.

- **Polimorfismo:** Comportamientos diferentes asociados a objetos distintos, pueden compartir un mismo nombre y semejanza. Al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto concreto que se esté utilizando.

2.4.1. JAVA

Tal y como se acaba de mencionar, para el desarrollo del proyecto se ha utilizado como paradigma de programación la orientación a objetos y, en concreto, el lenguaje de programación Java.

Este lenguaje presenta una serie de ventajas que se enumeran a continuación (Horstmann et al., 2006):

- **Soporte multiplataforma.** Dado que no se ejecuta sobre el sistema operativo sino sobre la máquina virtual, funciona sobre cualquier ordenador independientemente del sistema operativo que utilice.

- **Robusto.** Gracias a que se encuentra tipado estáticamente. Además el uso de excepciones evita que el programa se cuelgue ante un fallo.

- **Seguro.** Ya que se realizan comprobaciones tanto de privilegios como de bytecodes ante la ejecución del programa Java, evitando así excepciones cuando, por ejemplo se intenta acceder a un determinado recurso para el que se carece de privilegios.

- **Distribuido.** Es distribuido, ofreciendo mecanismos para interactuar con otras máquinas a través de sockets TCP/IP, o con publicación de objetos distribuidos a través de plataformas como RMI, CORBA o Servicios Web.

- **Comunidad.** En definitiva, gracias a las ventajas que presenta, Java es uno de los lenguajes más populares y más empleados en la actualidad contando con una gran comunidad de desarrolladores.

2.4.2. IDE DE DESARROLLO DE SOFTWARE

Eclipse (ECLIPSE) es una plataforma de desarrollo de software de código abierto, que proporciona un entorno integrado de desarrollo (IDE) para la construcción de aplicaciones con tecnología JAVA, y con una gran cantidad de componentes que facilitan el desarrollo, tanto visual como funcional, de aplicaciones Web.

Como características principales es posible destacar las siguientes: compilación en tiempo real, ayuda en línea, editor de texto, resaltado de sintaxis, depuración, asistente para la creación de proyectos, clases, control de versiones, etcétera.

Existe una gran variedad de razones por las que utilizar este software libre como herramienta de desarrollo. Además de su potencia y flexibilidad, es sencilla de utilizar y permite ampliar su funcionalidad de forma muy sencilla gracias a que ha sido construida bajo el motor OSGi Equinox, que permite emplear o desarrollar nuevas herramientas mediante plug-in.

2.4.3. SERVLETS Y JSP

En el desarrollo del proyecto se han utilizado Servlets y JSP (*Java Server Pages*) (Falkner et al., 2004) para la creación de páginas Web dinámicas usando Java como lenguaje. Una de las ventajas de utilizar JSP y Servlets es que se ejecutan sobre una máquina virtual Java, permitiendo su funcionamiento en cualquier ordenador que posea dicha máquina virtual.

Un Servlet es un componente que puede ser instalado en un servidor para ampliar su funcionalidad y han sustituido en la arquitectura Java a la tecnología CGI (*Common Gateway Interface*). Cada una de las peticiones HTTP recibidas por un Servlet es tratada por un hilo diferente, y puede generar como respuesta cualquier tipo de recurso tanto de cualquier estándar Web como multimedia. Los Servlets utilizan la API Servlet que suele estar incluida en cualquier servidor de aplicaciones Web Java.

JSP es un lenguaje para el desarrollo de contenido Web dinámico que permite la utilización de código Java embebido en páginas Web mediante scripts. A través de JSP es posible utilizar algunas acciones predefinidas mediante etiquetas, incluso pueden ser enriquecidas y personalizadas mediante la utilización de *TagLibs* o librerías de etiquetas externas.

Partiendo del Patrón MVC que se mencionó anteriormente, en la aplicación Web desarrollada en el ámbito de este proyecto, los controladores se han implementado haciendo uso de Servlets y las vistas son páginas JSP.

2.4.4. CSS

CSS (*Cascading Style Sheets*) (Meyer, 2000) es un lenguaje empleado para la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). Su principal función es la de separar la presentación del contenido.

Las ventajas de utilizar CSS son:

- Control total de la presentación de un sitio Web completo y facilidad de actualización y mantenimiento del mismo.
- Un documento HTML es más sencillo de entender al reducirse su tamaño considerablemente.
- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o la elección del usuario.
- Un usuario puede especificar su propia hoja de estilo local que será aplicada a un sitio Web.

2.4.5. JAVASCRIPT

JavaScript (Flanagan, 2006) es un lenguaje no tipado e interpretado orientado a páginas Web, con una sintaxis similar al lenguaje Java.

El lenguaje JavaScript se utiliza en las páginas Web HTML para realizar operaciones en el marco de la aplicación cliente aliviando así el hecho de realizar determinadas consultas al servidor, como validación de campos, carga de determinadas secciones de la página, etcétera.

2.4.6. AJAX

Ajax (*Asynchronous JavaScript And XML*) (Zakas, 2006) es una técnica de desarrollo Web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el navegador cliente mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. Gracias a Ajax se pueden realizar cambios sobre las páginas sin necesidad de recargarlas completamente, lo que implica que aumente la interactividad, velocidad y usabilidad en las aplicaciones.

2.4.7. SERVIDOR TOMCAT

Tomcat (TOMCAT) es un contenedor de aplicaciones Web con soporte para Servlets y JSP desarrollado por la *Apache Software Foundation*. Es un desarrollo de código abierto y multiplataforma al encontrarse implementado en Java.

Tomcat puede utilizarse como contenedor en solitario o como plug-in para un servidor Web existente como *Apache Web Server* e *Internet Information Services*. En este proyecto, se ha empleado únicamente como contenedor de aplicaciones Web basadas en tecnologías Java, permitiendo publicar de forma sencilla los servicios que ofrece la plataforma GODO.

2.4.8. SERVICIOS WEB

Para comprender qué es un servicio Web hay que tener claro el concepto de objeto distribuido (Bermúdez, 2009a). Un objeto distribuido es la instanciación de una clase cuyos métodos o servicios públicos pueden ser invocados de forma remota y, en algunos casos, por diferentes tecnologías de desarrollo. Con el auge de Internet, la red se ha convertido en un mecanismo de comunicación universal. Los servicios Web nacen con la idea de fusionar el mundo de los objetos distribuidos e Internet. Es decir, un servicio Web se define básicamente como un servicio público implementado bajo cualquier tecnología que puede ejecutarse remotamente a través de Internet.

Durante el proceso de comunicación intervienen una serie de tecnologías que posibilitan la circulación de información:

- **SOAP** (*Simple Object Access Protocol*) es un protocolo basado en XML que posibilita la interacción entre varios dispositivos y presenta la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etcétera.

- **WSDL** (*Web Services Description Language*) es un lenguaje que se utiliza para la descripción de los servicios Web. Está basado en XML y describe la forma de comunicación. WSDL se usa a menudo con SOAP y XML Schema.

- **UDDI** (*Universal Description, Discovery and Integration*) es un estándar que permite el registro y descubrimiento de los Servicios Web.

- **HTTP** (*HiperText Transfer Protocol*) es un protocolo de transporte que utilizan los servicios Web.

El funcionamiento estándar de un servicio Web es el siguiente (ver figura 8):

1. Un proveedor de servicio crea un servicio Web.

2. El proveedor de servicio utiliza WSDL para describir el servicio.

3. El proveedor de servicio registra el servicio en un registro UDDI.

4. Otro servicio o usuario localiza y solicita el servicio registrado al consultar los registros UDDI.

5. El cliente liga el servicio registrado utilizando SOAP.

6. Se produce el intercambio de datos y mensajes XML sobre HTTP.

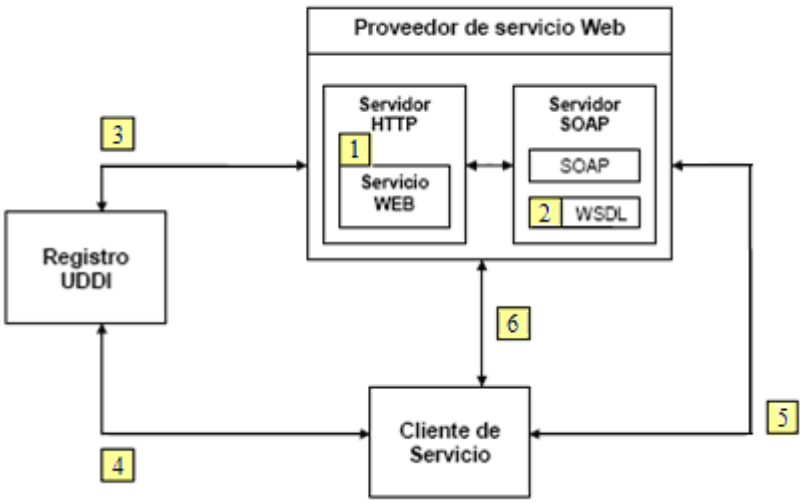

Figura 8. Funcionamiento estándar de un Servicio Web (Bermúdez, 2009a).

2.5. HERRAMIENTAS Y TECNOLOGÍAS PARA LA WEB SEMÁNTICA

2.5.1. ARQUITECTURA DE LA WEB SEMÁNTICA

La arquitectura de la Web Semántica está formada por una serie de capas o niveles, tal y como se puede apreciar en la figura 9.

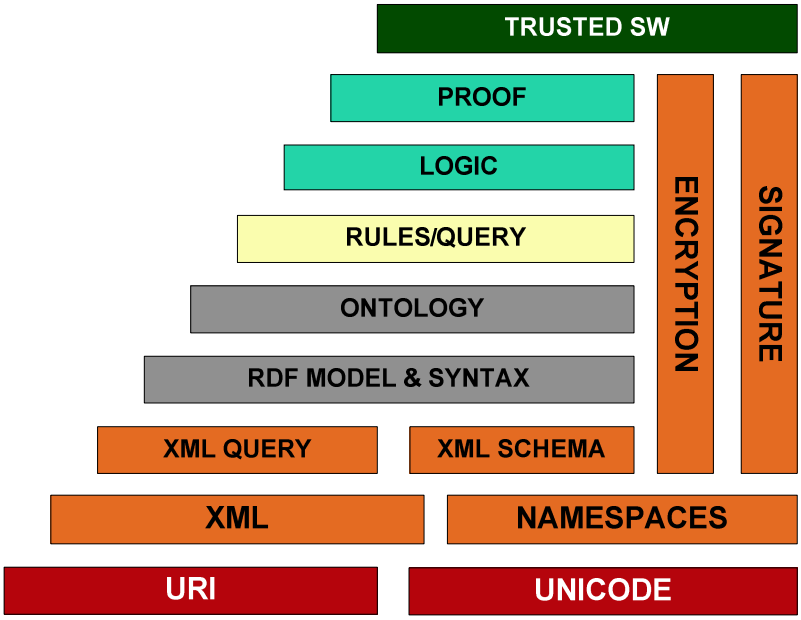

Figura 9. Arquitectura de la Web Semántica en forma de pastel según el W3C. (Berners-Lee, 2003).

En la capa más baja de la estructura en forma de pastel, se sitúa Unicode y URI. Unicode es un estándar que permite codificar y manipular un texto en la mayoría de los idiomas sin ningún problema. A través de URIs identificamos de manera unívoca los recursos Web.

En el nivel sintáctico, se encuentra XML (*eXtensible Markup Language*) (XML). Este metalenguaje de etiquetado proporciona un método para calificar elementos y atributos asociándolos con espacios de nombres identificados a través de URIs. De este modo se consigue solucionar el problema de asociar semántica a las páginas Web. A través de los XML Schema se puede definir la estructura de los documentos XML.

El modelo RDF (*Resource Description Framework*) (RDF) permite convertir la descripción y las relaciones entre recursos en expresiones de la forma sujeto-predicado-objeto. En otras palabras, estandariza el lenguaje con el que podemos describir datos en la Web Semántica.

A nivel de ontologías se consigue clasificar la información. Esta capa permite añadir nuevas propiedades a la funcionalidad de la Web Semántica, tales como cardinalidad y

relaciones entre clases. El lenguaje más utilizado para llevar a cabo esta función es OWL (*Ontology Web Language*).

A continuación, se comenta en mayor profundidad, las capas que se consideran de más interés en la arquitectura de la Web Semántica, como son RDF, RDFS (McBride, 2004) y OWL (OWL) como lenguajes de creación de ontologías. Antes, se describe qué son las ontologías, sus características y de qué se componen.

2.5.2. ONTOLOGÍAS

El concepto de ontología (Gruber, 2008) procede del ámbito de la inteligencia artificial, y define la terminología a emplear para describir y representar un determinado dominio. Formalmente, en informática una ontología se define como "una especificación explícita y formal de una conceptualización compartida" (Studer et al., 1998). Se dice que es explícita porque los objetos del dominio están definidos explícitamente, formal porque puede ser interpretado no sólo por humanos sino también por máquinas , conceptualizada porque se encuentra representado a través de un modelo abstracto y, por último, compartida porque el conocimiento representado ha sido consensuado previamente.

Las ontologías aparecen en la Web Semántica para catalogar la información de los recursos Web. A través de la información almacenada y organizada en ontologías, las aplicaciones podrán extraer automáticamente datos de las páginas Web, procesarlos y tomar decisiones a partir de ellos.

Una característica principal para el uso de ontologías es que puede ser comprendida tanto por humanos como por máquinas gracias a su representación mediante términos y relaciones. Para un ser humano la representación de significado viene especificada mediante palabras en lenguaje natural así como las relaciones semánticas entre ellas.

Las ontologías están formadas por tres elementos, que servirán para la representación del conocimiento:

- **Conceptos:** conjunto de entidades que presentan cierto interés para el dominio de representación. Los conceptos pueden organizarse en jerarquías de conceptos y subconceptos, de forma que los subconceptos son más específicos que los conceptos padre en la jerarquía, ya que heredan sus características y además pueden añadir otras. Pueden establecerse diferentes restricciones de carácter lógico entre conceptos.

- **Atributos:** los atributos representan las características de los conceptos a los que se encuentran asociados. Pueden ser descritos a través de una relación cuyo dominio es el propio concepto y cuyo rango puede ser un valor primitivo (int, float, string, boolean, etcétera). De la misma manera que los conceptos, los atributos pueden presentar una jerarquía de forma que sus sub-atributos presentan un rango más específico que los atributos padre. Al igual que los conceptos pueden existir relaciones de equivalencia y desigualdad entre atributos.

- **Relaciones:** las relaciones representan un vínculo común entre dos o más conceptos. Se modela mediante una relación cuyo domino es un concepto, denominado sujeto, y cuyo rango es otro concepto, denominado objeto. Del mismo modo que ocurre con conceptos y atributos podemos establecer jerarquías entre relaciones y sub-relaciones más concretas. También pueden existir relaciones de equivalencia y desigualdad entre dos relaciones.

Un concepto, atributo o relación puede tener asociadas etiquetas que especifican un nombre o expresión en cierto idioma. Este tipo de extensiones también nos ayudarán a establecer anotaciones que es una característica muy importante dentro de la Web Semántica (Davies et al., 2003) pero que no es un objeto de este proyecto.

2.5.3. LENGUAJES DE DEFINICIÓN DE ONTOLOGÍAS

2.5.3.1. RDF

RDF (*Resource Description Framework)* nació con la finalidad de proporcionar un lenguaje para la descripción de recursos y las relaciones existentes entre los mismos. Cada recurso viene definido a través de expresiones de la forma Sujeto-Predicado-Objeto denominadas 'tripletas' representadas mediante etiquetas XML. A continuación, se explica cada uno de los elementos que componen dichas tripletas:

- El Sujeto identifica al recurso que se va a describir. Puede tratarse de cualquier recurso: una página Web, una persona, una cosa, y se identifica mediante una URI (*Uniform Resource Identifier)* unívoca.

- Un Predicado identifica una propiedad del sujeto mediante una URI. Cada sujeto puede poseer una o más propiedades.

- Un Objeto especifica el valor de la propiedad del sujeto.

Cada elemento de la tripleta viene representado por una URI única (ver figura 10).

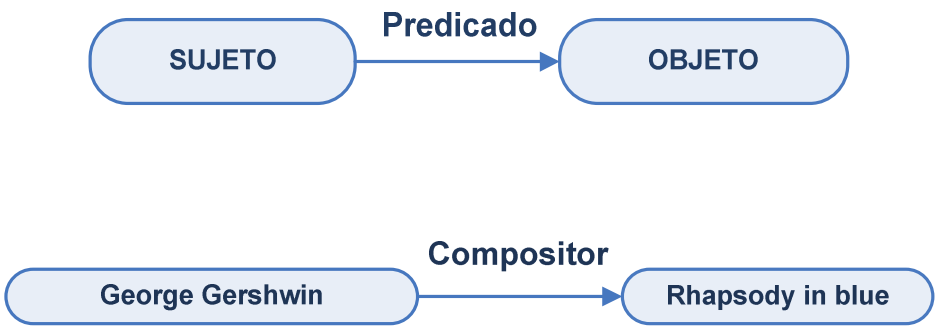

Figura 10. Grafo RDF.

Como se puede apreciar en la figura 10, RDF constituye un modelo de datos sencillo para la descripción de recursos y sus relaciones.

2.5.3.2. RDFS

El lenguaje RDFS (*RDF Schema*) extiende el estándar RDF visto con anterioridad siendo un vocabulario para la descripción de propiedades y clases de los recursos RDF. Este lenguaje permite mejorar la Web añadiendo contenido semántico legible por máquinas de forma automática. RDFS consigue realizar esto, mediante la definición de conceptos como *Clases, subClaseDe y Propiedad*.

Las expresiones RDFS son también expresiones RDF válidas. La diferencia entre ambos radica en que en RDFS se especifica un consenso en el significado de ciertos términos y, por lo tanto, en la interpretación de determinadas sentencias.

2.5.3.3. OWL (*Web Ontology Language*)

OWL (*Web Ontology Language)* es un lenguaje para la creación de ontologías que facilita en mayor medida, la interpretación del contenido Web que los estándares XML, RDF, y RDF Schema, proporcionando un vocabulario adicional con enriquecimiento semántico. De esta manera, OWL permite organizar los recursos en clases y crear instancias de esas clases. También permite definir propiedades, tanto de tipo primitivo como de tipo objeto, conocidos popularmente como atributos y relaciones respectivamente.

En OWL existen tres variantes con diferente expresividad y capacidad a la hora de inferir conocimiento (ver figura 11). Un grupo de desarrolladores o usuarios puede elegir la forma más apropiada de representación en OWL según las características siguientes (W3COWL, 2004):

- **OWL Lite:** indicado cuando se necesita una clasificación jerárquica y restricciones simples. Esta variante es menos compleja que OWL DL, pero su expresividad es más reducida.

- **OWL DL**: indicado en casos donde se requiera mayor expresividad, conservando la computacionalidad y resolubilidad, es decir, que cualquier conclusión pueda alcanzarse mediante reglas de inferencia y alcanzables en un tiempo finito respectivamente.

- **OWL Full**: indicado en casos de máxima expresividad y libertad sintáctica de RDF, pero no ofrece garantías computacionales. OWL Full permite a una ontología aumentar el significado de los vocabularios predefinidos en RDF u OWL. Como consecuencia, resulta imposible diseñar un sistema que sea capaz de realizar razonamiento sobre un lenguaje de estas características. OWL Full puede ser visto

como una extensión de RDF, mientras OWL Lite y OWL DL puede ser visto como extensiones de vistas restringidas de RDF.

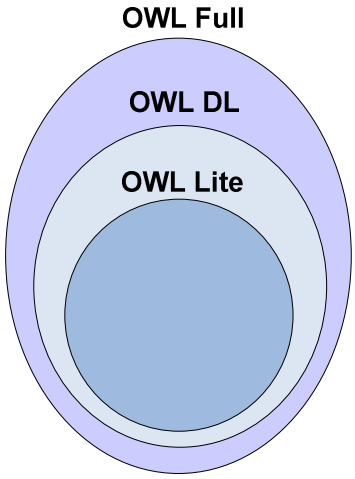

Figura 11. Versiones del lenguaje OWL (Vicente Torres, 2007).

Las ontologías empleadas en el proyecto se encuentran definidas en el lenguaje OWL, más concretamente en OWL-DL.

2.5.4. IDE DE DESARROLLO DE ONTOLOGÍAS

Protégé (PROTEGE) es una herramienta basada en Java, extensible, mediante plug-in, y de código abierto que permite editar ontologías y bases de conocimiento. Esta herramienta implementa un conjunto amplio de estructuras de modelado de conocimiento y operaciones de creación, visualización y manipulación de ontologías en varios formatos de representación. Además, permite exportar una ontología a múltiples formatos como RDF(S), OWL y XML Schema.

Esta plataforma soporta dos vías de modelado de ontologías, una a través del editor Protégé-Frames y otra a través del editor Protégé-OWL. El editor Protégé-Frames permite a los usuarios construir ontologías de acuerdo con el modelo Open OKBC (*Knowledge Base Connectivity protocol*). En este modelo una ontología está formada por un conjunto de clases jerárquicas que representan conceptos. Estas clases pueden ser descritas a través de propiedades y las relaciones con otras clases. El editor Protégé-OWL permite el usuario construir ontologías de acuerdo con el modelo de la Web Semántica, a través del lenguaje OWL. En el desarrollo del proyecto se ha utilizado este IDE para la creación y edición de ontologías.

2.5.5. PROCESAMIENTO DEL LENGUAJE NATURAL

El PLN (*Procesamiento del Lenguaje Natural*) (Fernández Breis, 2007) es una sub-disciplina de la Inteligencia Artificial y de la lingüística computacional. El PLN investiga mecanismos eficaces computacionalmente para la comunicación entre personas o entre personas y máquinas por medio del lenguaje natural.

El PLN puede ser visto como un intento de simular el comportamiento lingüístico humano, de forma que el sistema de signos que constituye la lengua sea procesado por el computador, siendo capaz de comprender, interpretar y generar lenguaje humano, ya sea escrito o hablado.

En sus comienzos, el PLN se enfocó fundamentalmente en tres áreas: la traducción automática, el reconocimiento del habla y el acceso a bases de datos. Aunque estos elementos se encuentran hoy en día en investigación, se han incorporado nuevos usos con el desarrollo de las tecnologías actuales como es Internet, tales como la recuperación de información, la búsqueda de respuestas, extracción de información o el resumen automático.

A pesar de los avances, continúa sin resolverse uno de los problemas principales de la Inteligencia Artificial, esto es, cómo proporcionar a las máquinas de conocimiento suficiente de forma que sean capaces de producir oraciones con sentido completo e inferir conocimiento de ellas.

La relación que se establece entre las ontologías y el PLN es bidireccional: por un lado son herramientas para la representación de redes semánticas y por otro el PLN es una técnica importante en la construcción automática de ontologías.

2.5.6. SPARQL

SPARQL (*SPARQL Protocol And RDF Query Language*) (W3CSPARQL, 2008) define un lenguaje de consulta estándar para modelos de datos RDF(S). Actualmente está recomendado por el W3C como lenguaje de consulta para la Web Semántica. La funcionalidad de SPARQL se encuentra definida en tres especificaciones diferentes :

- SPARQL Query Language: en esta especificación se explica la sintaxis para la creación de sentencias y su concordancia.
- SPARQL Protocol for RDF: utiliza WSDL 2.0 para definir protocolos HTTP y SOAP para consultar remotamente bases de datos RDF.
- SPARQL Query Results XML Format: describe el formato XML de cómo se devuelven los resultado de una consulta.

El lenguaje SPARQL posee tres componentes importantes: URIs, literales y variables procedentes del lenguaje RDF.

- URIs: sirven para especificar las URLs que identifican a los elementos de la ontología.
- Literales: se describen como una cadena de caracteres encerradas entre " ".
- Variables: estas variables son globales, además deben de ser prefijadas por "?" o "$" no formando parte del nombre de la variable.

2.5.7. JENA ONTOLOGY API

Jena (JENA) es un Framework Java de código abierto utilizado para el desarrollo de aplicaciones basadas en Web Semántica. Esta API define una serie de interfaces para acceder y manipular grafos RDF. A través de Jena se puede crear, escribir y leer un modelo RDF y navegar a través de él a partir de la URI del recurso. Jena permite trabajar tanto con RDF como OWL a través de sus API's, y procesar ontologías provenientes desde distintas fuentes de datos tales como bases de datos o simples ficheros. Además, proporciona una implementación del lenguaje de consulta SPARQL (ARQ) e incluye un motor de inferencia basado en reglas.

2.6. HERRAMIENTAS Y TECNOLOGÍAS PARA LOS SERVICIOS WEB SEMÁNTICOS

Desde los inicios de los Servicios Web Semánticos han aparecido numerosas tecnologías para añadir información semántica a los servicios. Entre las propuestas que están siendo evaluadas actualmente por parte del consorcio W3C se pueden destacar las siguientes:

- OWL-S (*Web Ontology Language for Services*) es una ontología expresada en OWL-DL que describe los conceptos de un servicio.
- WSMO (*Web Service Modeling Ontology*) define una ontología y un lenguaje para la descripción de Servicios Web Semánticos, denominado WSML.
- SWSF (*Semantic Web Service Framework*) define un modelo ontológico llamado SWSO y un lenguaje para la descripción de Servicios Web Semánticos, denominado SWSL.
- WSDL-S (*Web Service Semantics*) es una extensión del lenguaje de especificación de Servicios Web WSDL para permitir describir de forma semántica los componentes de éstos.
- SAWSDL (*Semantic Annotations for WSDL*) es un mecanismo que permite definir anotaciones semánticas a los componentes de Servicio Web descritos en WSDL.

Las tres primeras propuestas (OWL-S, WSMO y SWSF) se basan en una filosofía muy parecida consistente en un modelo ontológico y en un lenguaje de definición de los servicios. En el caso de OWL-S se emplea como lenguaje, el estándar OWL. Tanto WSMO como SWSF utilizan su propio lenguaje ya que OWL presentaba problemas a la hora de definir reglas en la composición entre los diferentes Servicios Web.

Las siguientes dos aproximaciones (WSDL-S y SAWSDL) se basan en mecanismos para añadir información semántica a las herramientas ya existentes para la definición de Servicios Web Semánticos, como WSDL. Estas propuestas son compatibles con los anteriores por lo que en muchas ocasiones se emplean combinados.

Actualmente, los modelos más empleados son OWL-S y WSMO. Ambos presentan una arquitectura semejante. La principal diferencia entre ellos es que OWL-S se basa en OWL-DL y WSMO en un lenguaje propio denominado WSML que se adapta mejor a los requerimientos semánticos de los Servicios Web.

OWL-S es más maduro en algunos aspectos, tales como la definición del modelo de proceso y la composición de los Servicios Web. Sin embargo, WSMO presenta ventajas importantes con respecto a OWL-S:

- Su modelo conceptual presenta una mejor separación entre el consumidor y el proveedor de servicios, incluyendo la composición de éstos estableciendo, tanto estática como dinámicamente, la reutilización de los servicios logrando obtener funcionalidades más complejas.

- Proporciona mecanismos semánticos formales para describir la composición entre Servicios Web.

- Permite definir diferentes formas de interactuar con un mismo servicio.

- Proporciona un lenguaje mejor adaptado a los Servicios Web, ya que, a diferencia de OWL, no presenta problemas al combinarlo con diferentes lenguajes de reglas.

Aunque OWL-S es un estándar que ha sido desarrollado durante más tiempo y que define algunos aspectos que pueden ser empleados en un entorno real, WSMO es el estándar que más se está empleando debido a las ventajas que comentamos anteriormente. Además, WSMO está soportado por diferentes implementaciones como: IRS-III, Semantic Web Fred y WSMX, que actualmente es la más utilizada y la que se emplea en este proyecto.

El objetivo principal de WSMO consiste en estandarizar y proporcionar un framework que permita definir todo el ciclo de vida de los Servicios Web Semánticos, desde su descripción hasta su ejecución automática.

En esta sección, se presenta una vista general de los elementos que acompañan a WSMO (ver figura 12). WSMO (*Web Service Modeling Ontology*) como modelo conceptual para los Servicios Web Semánticos, WSML (*Web Service Modeling Language*) como lenguaje que provee a WSMO de una sintaxis y semántica formal, y WSMX (*Web Service Execution Environment*), un entorno de ejecución que facilita la interacción con los Servicios Web Semánticos.

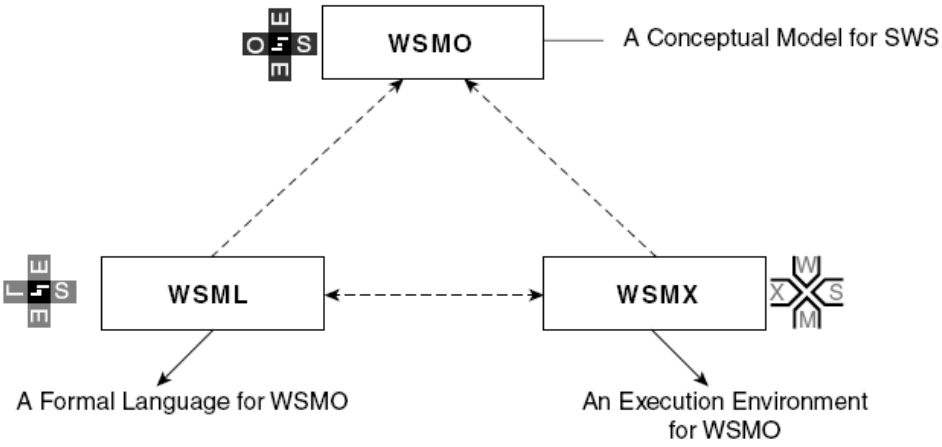

Figura 12. El Modelo conceptual WSMO (adaptado de Dumitru, 2004).

2.6.1. WSMO

WSMO (*Web Service Modeling Ontology*) (Dumitru, 2004) es un modelo conceptual para la descripción de Servicios Web de forma semántica, facilitando de este modo tareas de descubrimiento, combinación e invocación de estos servicios a través de la Web o, dicho de un modo más coloquial, es una ontología para la descripción de Servicios Web Semánticos. WSMO utiliza el concepto de URI (*Universal Resource Identifier*) para identificar de manera unívoca los recursos a describir. En WSMO existen cuatro elementos principales para la descripción de Servicios Web Semánticos: ontologías, objetivos, servicios Web y mediadores (ver figura 13).

- **Ontologías**: proveen de una terminología utilizada por los restantes elementos de WSMO. Estas ontologías proporcionan una semántica procesable por máquinas para la descripción, e intercambio de información utilizado por los actores implicados en el uso de Servicios Web.

- **Objetivos**: representa el deseo que un usuario desea satisfacer a través de la consulta a un servicio Web.

- **Servicio Web**: representa la descripción del servicio Web en sí, en cuanto al comportamiento y funcionalidad del servicio.

- **Mediadores**: WSMO se encuentra estrictamente desacoplado, es decir, los recursos están especificados independientemente sin considerar interacciones entre ellos. Los mediadores resuelven estos problemas de interoperabilidad entre servicios.

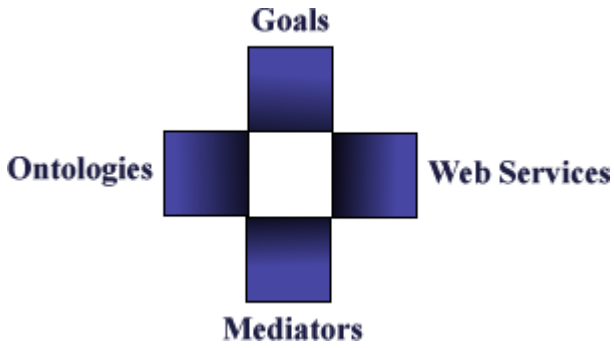

Figura 13. Elementos de WSMO (Dumitru et al., 2004).

WSMO es un metamodelo para aspectos relacionados con los Servicios Web Semánticos. Se encuentra especificado siguiendo el modelo MOF (*Meta-Object Facility*). MOF define un lenguaje abstracto y un framework para la especificación y construcción de metamodelos.

MOF define una arquitectura de metadatos de cuatro capas:

- La capa de información contiene los datos que serán descritos.
- La capa de modelo contiene los metadatos que describen los datos en la capa de información.
- La capa de metamodelo contiene las descripciones que definen la estructura y la semántica de los metadatos.
- La capa de meta-metamodelo contiene la descripción de la estructura y semántica del metamodelo.

En el caso de WSMO, las cuatro capas de MOF se definen del siguiente modo:

- El lenguaje definido por WSMO se corresponde con la capa de meta-metamodelo.
- El propio WSMO también constituye la capa de metamodelo.
- Las ontologías, los Servicios Web, los objetivos (goals) y los mediadores constituyen la capa de modelo.
- Por último, los datos descritos por las ontologías e intercambiados entre los Servicios Web constituyen la capa de información.

En la figura 14, se puede observar la relación que existe entre WSMO y la arquitectura de capas de MOF.

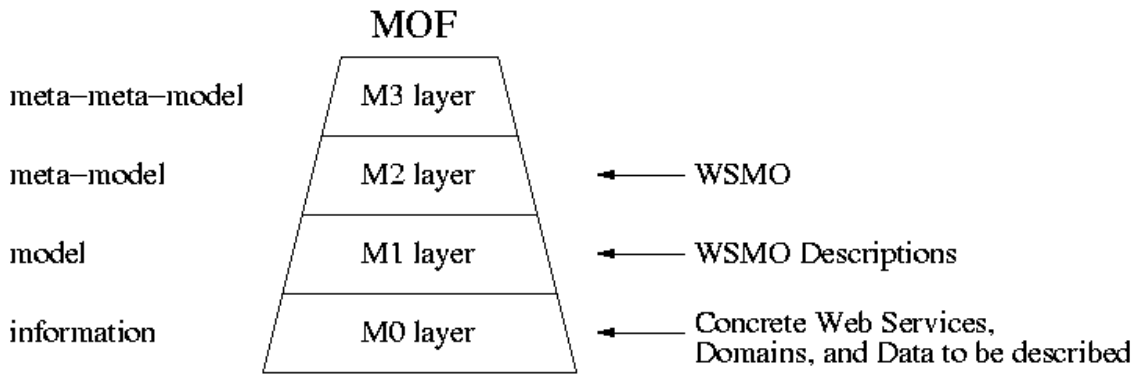

Figura 14. Relación entre WSMO y MOF (Brujin et al., 2005a).

2.6.2. WSML

WSML (*Web Service Modeling Language*) (Lausen et al., 2005) es un lenguaje de representación para la Web Semántica que permite la descripción de ontologías, objetivos, servicios Web y mediadores. WSML provee de una sintaxis y semántica formal para WSMO. WSML está basado en los siguientes formalismos lógicos (ver figura 15): WSML-Core, WSML-DL, WSML-Flight, WSML-Rule y WSML-Full. Esto permite al usuario elegir la expresividad y complejidad que necesitan en sus aplicaciones, de manera que es el propio usuario el que determina qué variante de WSML elegirá. En la figura 15 se pueden apreciar esos formalismos y las relaciones existentes entre cada uno de ellos. Cada variación difiere de otra, principalmente en la expresividad lógica que ofrece y en el paradigma del lenguaje subyacente.

El lenguaje básico de WSML está definido en WSML-Core, que se ramifica en dos direcciones denominadas, WSML-DL o Lógica de Descripción y WSML-Flight o Lógica de Programación.

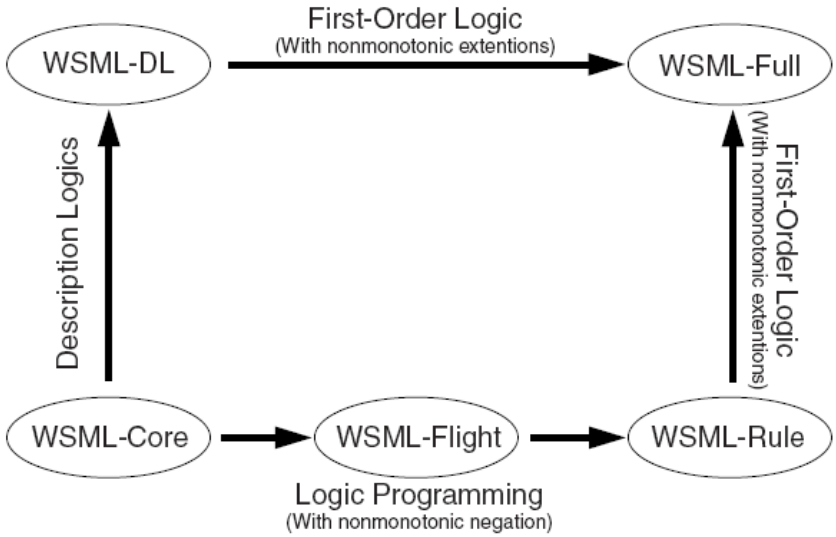

Figura 15. Variantes de WSML (Lausen et al., 2005).

Entrando un poco más en detalle en cada una de estas variantes:

- **WSML-Core**: se define por la intersección entre la Lógica Descriptiva y la Lógica en la Descripción de Programas. Este lenguaje soporta el modelado de clases, atributos, relaciones binarias, instancias, herencia de clases y relaciones jerárquicas.

- **WSML-DL**: es una extensión de WSML-Core, y contiene la mayor parte de la expresividad en la definición de ontologías que proporciona el lenguaje OWL.

- **WSML-Flight**: es también una extensión de WSML-Core e incluye la posibilidad de incluir restricciones. Basado en una variante de la programación lógica F-Logic. WSML-Flight proporciona un lenguaje muy potente para la definición de reglas.

- **WSML-Rule**: es una extensión de WSML-Flight añadiendo más posibilidades dentro de la lógica de programación (uso de símbolos de función y posibilidad de definir reglas no seguras).

- **WSML-Full**: unifica los lenguajes WSML-DL y WSML-Rule de modo que presenta las ventajas de ambos.

En la figura 16 se puede apreciar cómo WSML-Core es un lenguaje menos expresivo a diferencia de WSML-Full que presenta la mayor expresividad.

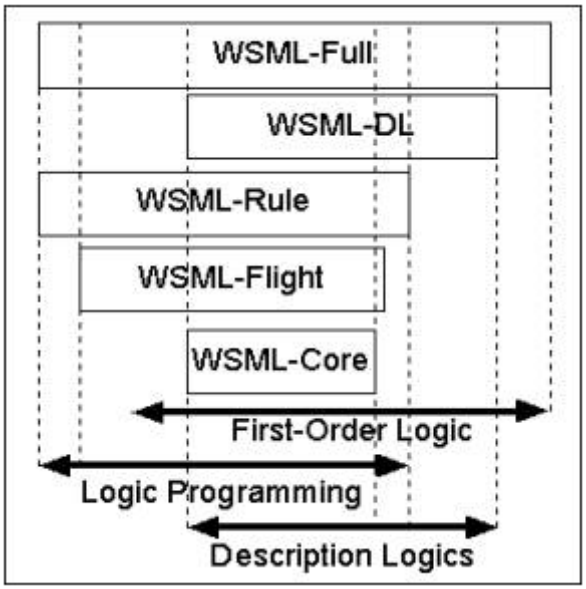

Figura 16. Formalismos lógicos en las variantes de WSML (Lausen et al., 2005).

2.6.3. WSMX

WSMX (*Web Service Execution Environment*) (Herold, 2008) es un entorno de ejecución que permite el descubrimiento, selección, mediación e invocación de Servicios Web Semánticos. WSMX está basado en el modelo conceptual proporcionado por WSMO.

La construcción de WSMX incluye la definición de su modelo conceptual, la definición de la semántica necesaria para el entorno de ejecución, el diseño de la arquitectura y la construcción de la implementación.

WSMX puede actuar como repositorio de servicios, realizando el proceso de descubrimiento de servicios utilizando su descripción semántica, y como proveedor de los mismos, realizando procesos de mediación e invocación.

- **Registro de servicios:** los Servicios Web descritos pueden ser registrados dentro del entorno WSMX, definidos mediante WSML.

- **Proveedor de servicios:** WSMX también funciona como proveedor de Servicios Web Semánticos. Un cliente que desee invocar un servicio, no ha de conocer los servicios que están almacenados dentro de WSMX ni sus capacidades. El cliente simplemente se limitará a enviar el objetivo que quiere alcanzar al entorno WSMX y éste es el responsable de buscar el servicio que satisfaga los objetivos del cliente. WSMX compara el objetivo con las descripciones semánticas de los servicios que tiene almacenados en sus registros. Además, WSMX es capaz de mediar para adaptar uno de los servicios de los que dispone a la petición del cliente. Podrá realizar una mediación de datos utilizando equivalencia entre las ontologías del solicitante y las ontologías del proveedor. WSMX también puede realizar tareas de arbitraje entre los procesos de invocación realizada por el solicitante y coreografía del servicio.

La arquitectura de WSMX está formada por un conjunto de componentes bajamente acoplados. A continuación describiremos los componentes principales de la arquitectura de WSMX.

- **WSMX Manager:** controla el flujo de operaciones del sistema. Es el coordinador de eventos de WSMX.

- **Matchmarker:** este componente es el responsable de comparar el objetivo del cliente con las capacidades de los Servicios Web conocidos por WSMX.

- **Selector:** este componente selecciona los Servicios Web que mejor encajan con el objetivo del usuario de entre los Servicios devueltos por el componente *Matchmarker*.

- **Data mediator:** realiza la mediación entre la ontología usada por el objetivo y la ontología usada por el Servicio Web seleccionado, a través de un conjunto de reglas de mapeo entre las dos ontologías.

- **Invoker:** es el último componente de la infraestructura. Realiza la invocación del Servicio Web seleccionado usando la mediación de datos.

2.6.4. RAZONADORES WSML

Algunos de los razonadores más importantes para WSML son (Brujin et al., 2005b):

- Fact++ (Fact++) es un razonador DL (*Description Logic*). Utiliza una implementación del razonador FaCT en C++, implementado con anterioridad en Lisp. Además de la mejora en cuanto a eficiencia y portabilidad debido al cambio en el lenguaje de implementación, presenta una arquitectura interna diferente, nuevas optimizaciones y características. Fact++ está disponible para Windows, Linux y Mac, y se distribuye bajo licencia GPL.
    - WSML-Core: Fact++ puede utilizarse como razonador de WSML-Core, pese a las limitaciones en cuanto al tipo de datos soportados, ya que no puede trabajar con enteros ni cadenas de texto.
    - WSML-Flight: Fact++ no es apropiado para el razonamiento de WSML-Flight ya que no soporta *DataLog* (Lenguaje lógico para el modelo relacional) ni metamodelado.
    - WSML-Rule: Fact++ no soporta símbolos de funciones, ni ningún tipo de "default negation", por lo que no es recomendable para el razonamiento de WSML-Rule.
    - WSML-DL: Fact++ está recomendado para el razonamiento de WSML-DL, aunque presenta la desventaja de no permitir cadenas de texto ni enteros.
    - WSML-Full: debido a que WSML-Full unifica WSML-DL y WSML-Rule, no está recomendado el uso de Fact++ para razonar, ya que WSML-Rule no lo recomienda.
- RACER es un razonador DL (*Description Logic*). Al igual que Fact++ posee limitaciones para dominios WSML concretos.
    - WSML-Core: RACER puede manejar enteros, reales, números complejos y cadenas de texto a diferencia de Fact++, por lo que está recomendado como razonador de WSML-Core.
    - WSML-Flight: RACER no está recomendado para WSML-Flight ya que no soporta "default negation" ni metamodelado.
    - WSML_Rule: RACER no soporta símbolos de funciones, además de "default negation" ni negación bajo semántica bien fundada, por lo que no está recomendado para WSML-Rule.

  - WSML-DL: RACER está recomendado para el razonamiento completo de WSML-DL.
  - WSML-Full: RACER no puede emplearse como razonador de WSML-Full ya que no está soportado por WSML-Rule.

- Pellet (Pellet) es un razonador sólido y completo para DL. La principal ventaja de Pellet es que soporta tipos de datos de XML Schema, otra ventaja es que ha sido desarrollado en Java.
  - WSML-Core: Pellet razona sobre tipos de datos de XML Schema. También soporta la definición de nuevos tipos de datos derivados de los tipos de datos del XML Schema, por lo tanto está recomendado cómo razonador de WSML-Core.
  - WSML-Flight: puesto que Pellet no soporta "default negation" ni metamodelado, no está recomendado para satisfacer todos los requisitos de razonamiento de WSML-Flight.
  - WSML-Rule: Pellet tampoco soporta símbolos de funciones ni negación bajo semántica bien fundada, por lo tanto, no está recomendado como razonador de WSML-Rule.
  - WSML-DL: dado que Pellet es un razonador para DL (*Description Logic*) sí está recomendado para WSML-DL.
  - WSML-Full: Pellet no puede emplearse como razonador de WSML-Full ya que no está soportado por WSML-Rule.

- KAON2, proporciona un razonador hibrido capaz de razonar con un subconjunto amplio de OWL DL.
  - WSML-Core: KAON2 permite *DataLogs* y maneja restricciones de integridad, por lo que es capaz de satisfacer todos los requisitos para que las tareas de consulta sean más eficientes con WSML-Core.
  - WSML-Flight: KAON2 es capaz de manejar símbolos de igualdad y metamodelado, por lo que está especialmente recomendado para razonar sobre WSML-Flight.
  - WSML-Rule: KAON2 no es capaz de soportar símbolos de funciones ni reglas inseguras, por lo que no está considerado como razonador de WSML-Rule.
  - WSML-DL: desde que KAON2 es un razonador DL es capaz de razonar sobre WSML-DL.
  - WSML-Full: no es capaz de manejar aquellas características añadidas por WSML-Full.

Fact++ y RACER presentan muchas limitaciones en cuanto al tipo de datos soportado, por eso Pellet y KAON2 son los razonadores más empleados para el procesamiento de la información semántica.

En la tabla 1, se encuentra un resumen donde es posible observar las diferencias existentes entre estos tipos de razonadores.

	Fact++	RACER	Pellet	KAON2
WSML-Core	No Recomendado (No soporta enteros ni cadenas de texto)	Recomendado (Puede manejar enteros y cadenas)	Recomendado (Soporta tipos de datos del XML Schema)	Recomendado(Permite *Datalog*)
WSML-Flight	No Recomendado (No soporta *DataLogs* ni metamodelado)	No Recomendado (No soporta "default negation" ni metamodelado)	No Recomendado(No soporta "default negation" ni metamodelado)	Recomendado(Permite símbolos de igualdad y metamodelado)
WSML-Rule	No Recomendado (No soporta símbolos de funciones)	No Recomendado (No soporta "default negation" ni negación bajo semántica bien fundada)	No Recomendado (No soporta símbolos de funciones ni negación bajo semántica bien fundada)	No Recomendado(No soporta símbolos de funciones)
WSML-DL	Recomendado	Recomendado	Recomendado	Recomendado
WSML-Full	No Recomendado	No Recomendado	No Recomendado	No Recomendado

Tabla 1. Comparativa razonadores.

2.6.5. WSMO4J API

WSMO4J (Dimitrov, 2006) es una API Java y una implementación de referencia para la construcción de aplicaciones de Servicios Web Semánticos basadas en WSMO. Esta API está disponible puesto que es de código abierto.

Esta API presenta:

- Interfaces que se corresponden con entidades WSMO.
- API para expresiones lógicas que forman parte de la especificación WSML.
- Interfaces que ofrecen funcionalidades de infraestructura (*parsers, validators, factories, etc*).

CAPÍTULO III. DISEÑO Y RESOLUCIÓN DEL TRABAJO

3.1. TRABAJO RELACIONADO

GODO es un proyecto de investigación financiado a cargo del proyecto Avanza, del Ministerio de Industria, Turismo y Comercio. Este proyecto continúa con el desarrollo realizado durante estos últimos dos años de investigación, gracias al consorcio liderado por la empresa Atos Origin S.A.E y donde colaboran activamente la Universidad de Murcia y la Universidad Carlos III de Madrid. Este grupo ha obtenido importantes resultados desde el punto de vista científico, técnico y empresarial, en el desarrollo de GODO, aunque todavía queda mucho trabajo por investigar y desarrollar.

Durante la ejecución del proyecto, se realizó un exhaustivo estudio de las técnicas, herramientas y tecnologías relacionadas con los Servicios Web Semánticos. Entre los avances más significativos obtenidos en el transcurso del proyecto, cabe destacar los siguientes. En primer lugar, se realizó un estudio de las herramientas de generación de lenguaje natural controlado (CNL, *Controlled Natural Language*), realizando progresos sobre la extracción de ontologías a partir de consultas al lenguaje natural. Además, se propuso una arquitectura del sistema para permitir la entrada de texto guiada por ontologías.

Tal y como se indicó en la introducción de este proyecto, el sistema GODO pretende acercar al usuario final no experto a una plataforma sencilla e intuitiva para la búsqueda y ejecución de Servicios Web Semánticos.

En cuanto al trabajo relacionado cabe destacar dos módulos principales que se tratarán a continuación en dos sub-secciones diferenciadas. Uno de ellos es el proyecto GODO en sí, su arquitectura funcional, el procesamiento de la consulta, etcétera. Por otro lado se encuentra OWLPath, un sistema capaz de ayudar a los usuarios durante la fase de generación de la consulta, gracias a que se encuentra guiado por la ontología.

3.1.1. PROYECTO GODO

GODO está formado por 5 módulos principales (ver Figura 17), *Control Manager, Language Analizer, Goal Loader,Goal Matcher* y *Goal Sender*.

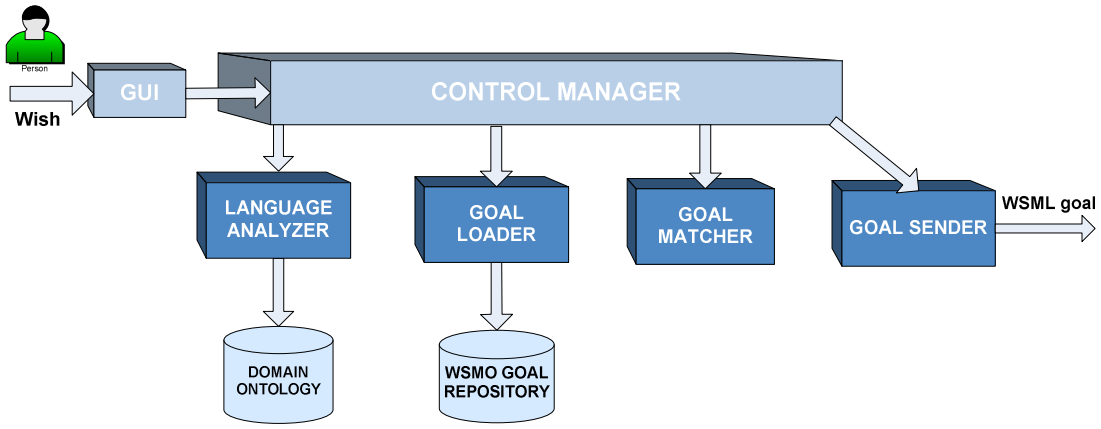

Figura 17. Arquitectura funcional de GODO.

El componente principal de GODO es el *Control Manager*, encargado de supervisar e interactuar con los demás módulos del sistema. El módulo *GUI* está situado entre el usuario y el Controlador. Este componente se encarga de recibir el texto del usuario y pasarlo directamente al *Control Manager,* que pregunta al *Language Analyzer* el significado de la sentencia.

El *Language Analyzer* tiene por objetivo filtrar y procesar la entrada introducida por el usuario en lenguaje natural y determinar los conceptos (atributos y valores) y relaciones que ésta contiene. Este componente necesita de una fase previa de entrenamiento, en la que el experto establece todos los patrones que usará el analizador para obtener los conceptos y relaciones de la sentencia introducida por el usuario. En esta fase, el experto indica los conceptos, sus atributos, relaciones entre conceptos y valores de los atributos y el sistema los almacenará en la base de datos. De este modo cuando un usuario introduce la sentencia, GODO será capaz de obtener los conceptos, atributos, valores y relaciones gracias a lo aprendido en el proceso de entrenamiento. Para ello, en primer lugar se identifica el verbo en la frase, luego el sistema busca la relación semántica asociada a ese verbo. Una vez establecida la relación semántica del verbo se aplica el subsistema MCRDR (que se describe a continuación) para extraer conocimiento a través del significado de la categoría gramatical de las palabras, su posición en la sentencia actual y el tipo de relación asociada al verbo.

Supongamos que el usuario introduce la sentencia "*I want to buy plane ticket from Galway to Madrid, then book a hotel in the center and rent a C class car*". Una vez analizado por el *Language Analyzer*, obtenemos la ontología que se muestra en la figura 18.

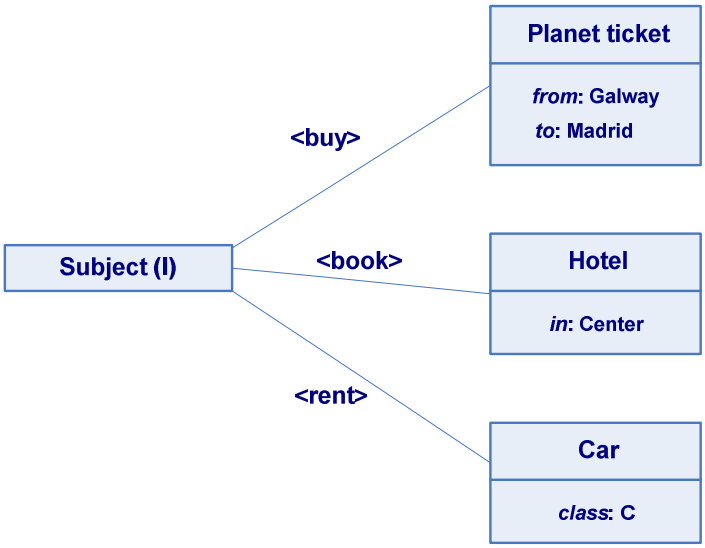

Figura 18. Ontología derivada del Language Analyzer (Gómez et al., 2007).

En esta figura, se distinguen cuatro conceptos, "*Subject*", "*Planet ticket*", "*Hotel*" y "*Car*", tres relaciones "*buy*", "*book*" y "*rent*" y atributos como "*from*", "*to*" cuyo valor es Galway y Madrid respectivamente, "*in*" atributo de "*Hotel*" cuyo valor es Center y por último "*class*", atributo de "*Car*", cuyo valor es C. Esta ontología se utilizará para comprobar si existe algún objetivo que la satisfaga.

El *Goal Loader* se encarga de buscar los objetivos en el repositorio de WSMO, o en su defecto, en el sistema de ficheros local donde los objetivos estén almacenados. Los objetivos (goals) se encuentran expresados en WSML.

En una fase posterior, el *Goal Matcher* compara los elementos de la ontología obtenida en la fase de análisis de la sentencia introducida por el usuario y los objetivos WSML extraídos del repositorio. En particular, sólo se comparan algunas partes: postcondiciones y efectos. Dependiendo de la entrada del usuario, uno o más objetivos se seleccionarán del repositorio y se enviarán al *Control Manager*.

Siguiendo con el ejemplo anterior, ahora el módulo *Goal Matcher* compara cada una de las relaciones ("*buy*", "*book*" y "*rent*") con diferentes objetivos. De este modo, tres objetivos (goals) estarán involucrados en la búsqueda de la solución.

Finalmente, el *Goal Sender* envía los diferentes goals a WSMX para ser ejecutados. Los objetivos son enviados secuencialmente sin tener en cuenta cualquier otro flujo de trabajo. En el siguiente diagrama de secuencia (ver figura 19) podemos observar cómo interactúan los diferentes componentes del sistema GODO.

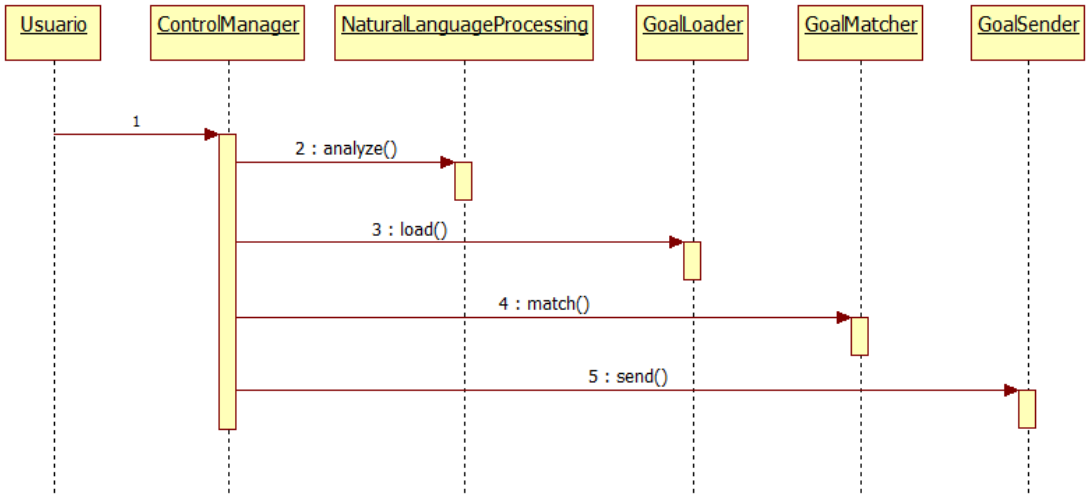

Figura 19. Diagrama de secuencia de GODO.

3.1.1.1. Procesamiento de la Consulta

La aplicación debe ser capaz de procesar el texto introducido por el usuario en lenguaje natural. En nuestro caso, la herramienta de procesado de texto en lenguaje natural debe ser capaz de procesar y filtrar la entrada del usuario, diferenciando aquellos elementos ontológicos que aparecen en el texto (conceptos, atributos y relaciones).

El reconocedor de lenguaje natural utilizado en el proyecto está basado en KAText, la metodología propuesta en (Valencia-García et al., 2004; Valencia-García, 2005) para extraer conocimiento a partir de un texto. Esta metodología, emplea ontologías y una técnica para la adquisición de conocimiento denominada MCRDR (Kang, 1996). MCRDR tiene su fundamento en la idea de que las relaciones entre conceptos, están asociadas a verbos en el lenguaje natural.

El proceso de adquisición de conocimiento de KAText está dividido en tres fases secuenciales: *POS-Tagging*, *Concept search* e *Inference*. En la fase de *POS-Tagging* se obtiene la categoría gramatical de cada palabra en la frase. Para llevar a cabo esta tarea se utiliza el etiquetador sintáctico (*POS-Tagger*) descrito en (Ruiz-Sánchez et al., 2003).

En la fase de *Concept search* se identifican las expresiones lingüísticas que representan conceptos. Previamente durante la fase de entrenamiento, deben haber sido almacenadas en una base de conocimiento las asociaciones entre las expresiones lingüísticas y los conceptos. Finalmente, como resultado de esta fase, se obtienen todas las expresiones del fragmento que se encontraban en la base de conceptos.

En la última fase *Inference* se utiliza la teoría de que las relaciones entre conceptos coinciden con el verbo de la oración en lenguaje natural.

El componente MCRDR, que se utiliza para obtener las relaciones entre conceptos, está formado por una base de conocimiento que contiene expresiones lingüísticas que

representan relaciones conceptuales y por un subsistema que infiere los participantes en dichas relaciones.

Hay que tener en cuenta que debe existir una fase previa de entrenamiento para que la herramienta funcione correctamente. En esta fase inicial el experto es el encargado de establecer los patrones que se utilizarán por parte del sistema para obtener los conceptos y relaciones de la frase introducida por el usuario. Esta fase se basa en el estudio de un conjunto de sentencias de un texto en el dominio de la aplicación en el que se quiera entrenar la herramienta. El experto indica los conceptos, atributos de esos conceptos, valores de los atributos y las relaciones existentes entre conceptos de cada una de las oraciones del texto. De este modo, cuando un usuario introduce un texto, el sistema debe ser capaz de obtener de forma automática los conceptos, relaciones, atributos y valores asociados a la oración.

KAText se ha utilizado como una librería en el proyecto GODO para el procesado del texto del usuario en el módulo *NaturalLanguageProcessing*.

3.1.2. OWLPATH

OWLPath (Chirlaque-Medrano et al., 2008) tiene la misión de ayudar a los usuarios durante la fase de generación de consultas al sistema. Para ello utiliza como base una ontología del dominio y va sugiriendo al usuario distintas alternativas conforme va completando la frase que constituirá la entrada al sistema. Las diferentes alternativas provienen de la combinación de la estructura de la ontología (relaciones entre conceptos) y una ontología ("*Question Ontology*") que define el formato que deben tomar las frases (gramática permisible).

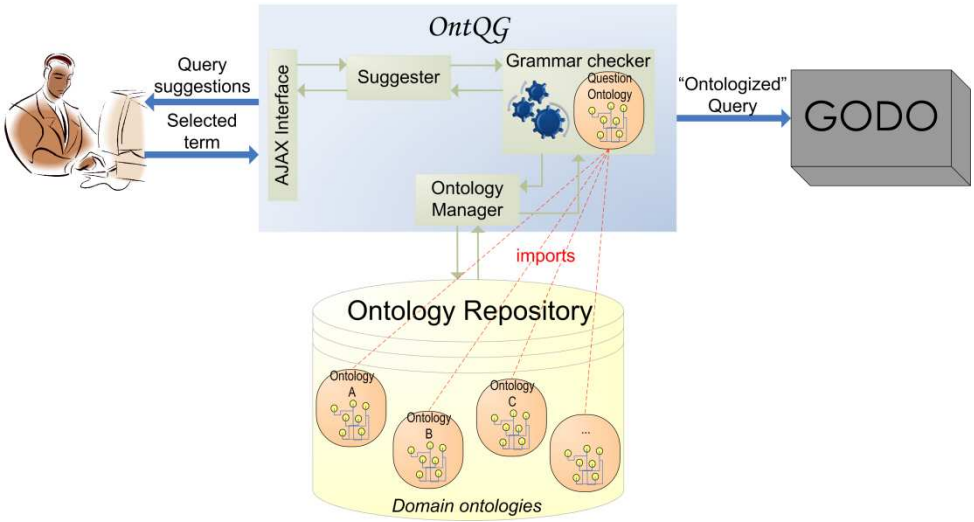

Figura 20. Arquitectura de OWLPath (Chirlaque-Medrano et al., 2008).

En la figura 20 se presenta la arquitectura de OWLPath. A continuación, se describe cada uno de los componentes de los que se compone.

Ontology-based Query Generator. Este componente engloba a todos los demás componentes del sistema, a excepción del repositorio de ontologías que es independiente. Su misión fundamental es la de sugerir al usuario las distintas alternativas mientras va escribiendo la frase que constituirá la consulta de entrada al sistema (Query). Una vez introducida la consulta, el sistema generará la "Ontologized query" (consulta con contenido semántico), que se enviará a GODO.

Ajax interface. Este componente Ajax proporciona una interfaz de entrada de texto, mostrando al usuario las palabras más apropiadas en cada momento. Dichas palabras son suministradas por el componente *Suggester* a partir de la ontología cargada y la información previamente introducida. Se utilizó Ajax para el desarrollo de este componente ya que se necesitan realizar muchas consultas al servidor mientras que el usuario va seleccionando su consulta.

Suggester. El componente *Suggester* tiene por objetivo listar los conceptos, propiedades o relaciones que el usuario tiene como alternativas para generar la consulta, de acuerdo con las relaciones semánticas que aparecen en la ontología del dominio y la ontología de la aplicación. Las dependencias existentes entre estas ontologías se comprobarán con el componente *Grammar Checker*.

Una vez cargadas la lista de palabras, el componente *Ajax Interface* las mostrará al usuario para que éste seleccione una de ellas. Una vez seleccionada la almacenará junto con toda la información semántica necesaria para la generación de la consulta, una vez finalizado el proceso de introducción de información.

Grammar Checker. Este componente proporciona al *Suggester* todos los posibles términos alternativos que el usuario podrá utilizar, gracias a la importación de los elementos de la ontología de la aplicación y haciendo uso de la base de conocimiento (constituida por las ontologías disponibles). Una vez ha obtenido el listado de palabras, el *Suggester* las comunicará al componente Ajax Interface. De este modo evitamos entradas gramaticalmente incorrectas, que provocarían que no fuesen comprendidas por la máquina.

Ontology Manager. Este componente tiene por objetivo el proveer todo el contenido de las ontologías que necesite el componente *Grammar Checker*. Para llevar a cabo su objetivo, el *Ontology Manager* proporciona una interfaz de acceso al repositorio de ontologías. Esta interfaz puede ser implementada mediante el uso de la API de Jena, Protégé-OWL API, o cualquier otra alternativa. El componente cargará las ontologías seleccionadas, tanto la ontología del dominio como la ontología de la aplicación (*Question Ontology*) y realizará las consultas sobre las mismas.

Ontology repository. El componente *Ontology repository* almacena las ontologías del dominio que utilizará la aplicación. La ontología de la aplicación importará una o varias ontologías del dominio que serán almacenadas en este repositorio constituyendo la base de conocimiento del sistema. Por lo tanto, nos encontramos con dos tipos de ontologías:

- *Question Ontology*: representa la gramática del sistema y modela las consultas gramaticalmente correctas. Importa las ontologías del dominio necesarias para constituir la base de conocimiento de la aplicación.

- *Domain Ontologie:* son las ontologías específicas del dominio a representar. Serán importadas por la *Question Ontology* para obtener una ontología global de consulta. El objetivo de una ontología del dominio es representar, organizar, formalizar y estandarizar el conocimiento de un dominio, para que pueda ser compartido y reutilizado por distintos grupos de personas y aplicaciones software.

En la siguiente figura (ver figura 21) se puede apreciar una representación de ambas ontologías utilizadas en el repositorio de ontologías.

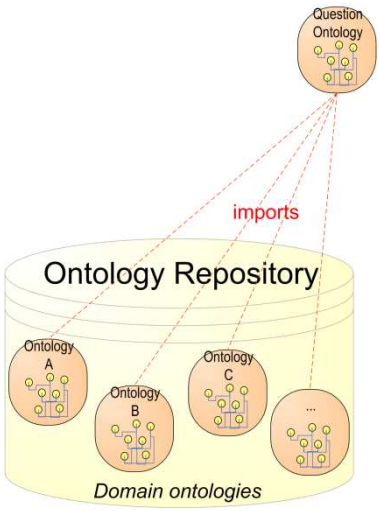

Figura 21. Repositorio de ontologías (Chirlaque-Medrano et al., 2008).

3.2. ANÁLISIS DE REQUISITOS

En esta sección se describen de forma detallada los requisitos recogidos durante la fase de análisis del proyecto. Estos requisitos se incorporarán al documento de requisitos completo del proyecto GODO. Los requisitos se encuentran clasificados en cinco categorías: requisitos de interfaz, requisitos de calidad, requisitos funcionales, requisitos de entorno y requisitos de seguridad.

3.2.1. REQUISITOS DE INTERFAZ (RI)

RI.1- La interfaz del sistema permitirá al usuario seleccionar el tipo de introducción de texto, mediante lenguaje natural o mediante la ayuda en línea guiada por la ontología del dominio.

RI.2- Una vez el usuario ha introducido el texto y su petición ha sido procesada, la interfaz del sistema mostrará una lista de objetivos que satisfagan las necesidades del usuario.

RI.3- El usuario podrá comprobar que los objetivos seleccionados por la herramienta son correctos, ejecutando los mismos o volviendo atrás para afinar más la búsqueda.

RI.4- Una vez ejecutados los objetivos el sistema mostrará una ventana con los resultados de los mismos.

RI.5- Todas las interfaces de la herramienta se definirán haciendo uso de estándares validados por el W3C como HTML, CSS y XHTML.

RI.6- Las interfaces cumplirán los requerimientos de accesibilidad requeridos por las entidades nacionales.

3.2.2. REQUISITOS DE CALIDAD (RC)

RC.1- El sistema tendrá un control sobre los errores y excepciones ocurridas, indicando el error o excepción en el momento en el que se produzca.

RC.2- El tiempo de carga de las páginas será inferior a 10 segundos de media para una conexión de 256Kbps. Las vistas que dependan de algún proceso de búsqueda tendrán como tiempo máximo de carga 60 segundos.

RC.3- El sistema controlará las peticiones duplicadas de los usuarios, mostrando un error cuando se produzca este caso.

3.2.3. REQUISITOS FUNCIONALES (RF)

RF.1- El sistema permitirá al usuario seleccionar entre dos tipos de introducción de texto para expresar sus necesidades.

RF.2- El sistema permitirá la introducción de texto en lenguaje natural por parte del usuario.

RF.3- El sistema permitirá la introducción de texto guiado por la ontología del dominio.

RF.4- El sistema procesará la petición del usuario, ya sea en lenguaje natural o guiado por la ontología, una vez haya finalizado la introducción de texto y haya presionado el botón *Send*.

RF.5- Una vez procesada la petición el sistema mostrará al usuario una lista de objetivos WSML, a ejecutar por WSMX.

RF.6- Una vez el usuario ha indicado que desea ejecutar los objetivos a través de la opción *Execute*, el sistema reaccionará invocando los objetivos a través de WSMX.

RF.7- El sistema permitirá al usuario cambiar el entorno WSMX de ejecución en todo momento.

3.2.4. REQUISITOS DE ENTORNO (RE)

RE.1- La aplicación Web estará alojada en un servidor Apache Tomcat versión 6.0.18.

RE.2- La ejecución del sistema necesita de una máquina virtual de Java (JVM) 1.5 o superior.

RE.3- El sistema permitirá la ejecución tanto en sistemas Windows como Linux, gracias a su desarrollo en Java.

RE.4- El procesador que ejecute el sistema tendrá una potencia de al menos 1GHz y una memoria de al menos 512MBytes (recomendado).

3.2.5. REQUISITOS DE SEGURIDAD (RS)

RS.1- El servidor que contenga la aplicación se alojará en un entorno libre de amenazas, tales como fuego, agua, radiaciones, etc.

RS.2- El lugar donde se encuentre alojado el sistema, poseerá un mecanismo de seguridad que evite el acceso no autorizado a las instalaciones.

RS.3- Existirán protocolos de actuación en casos de emergencia.

RS.4- Existirá un protocolo para la elaboración de copias de seguridad y dichas copias se almacenarán en un lugar diferente al que se encuentra la aplicación.

3.3. DESCRIPCIÓN GENERAL

En esta sección se describe de forma detallada la arquitectura e implementación del sistema GODO. En primer lugar, se muestra la arquitectura de la plataforma, haciendo referencia al nuevo diseño modular, y las herramientas utilizadas para llevar a cabo este propósito. En segundo lugar, se explica detalladamente la arquitectura de la aplicación. En el último apartado se comenta el funcionamiento de la aplicación.

3.3.1. ARQUITECTURA DE LA PLATAFORMA

La arquitectura de GODO está formada por cinco módulos principales encargados de realizar las tareas de procesado, búsqueda y comparación de los objetivos del usuario. Esta arquitectura inicial presenta problemas de extensibilidad ya que, por ejemplo, si se desea incorporar una nueva técnica de procesado de la entrada de texto del usuario, es necesario modificar el código del *ControlManager* para poder utilizar el nuevo módulo. Otros de los inconvenientes que presentaba GODO es que no almacena la información semántica de forma estandarizada, sino que utiliza su propio mecanismo de persistencia para almacenar la información ontológica, no pudiendo así utilizar muchas de las herramientas de consulta y razonamiento existentes.

Para solucionar estos problemas se han realizado las siguientes actuaciones:

- Se ha rediseñado la herramienta, de forma que ahora presenta una nueva capa, que abstrae al *ControlManager* del resto de módulos. De esta forma, si se desea incorporar un nueva tecnología sólo se tiene que desarrollar un nuevo adaptador, que extienda de la nueva capa de abstracción, evitando así modificar el *ControlManager*.

- Se modificó el mecanismo de persistencia de información semántica, utilizando la API de Jena para realizar la estandarización a OWL.

En la figura 22, se presenta la nueva arquitectura del sistema.

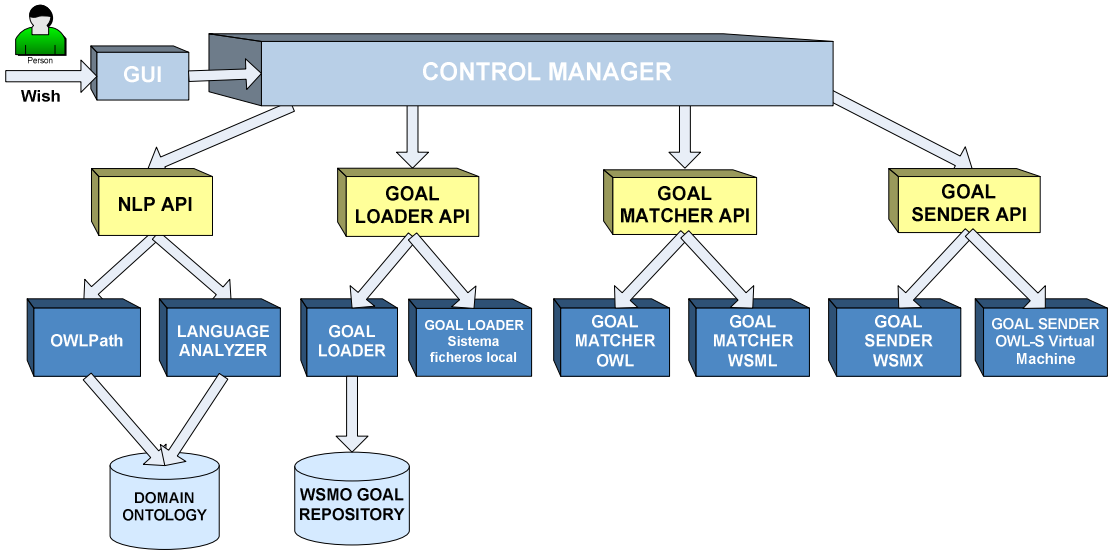

Figura 22. Arquitectura de GODO con un nuevo nivel de abstracción.

3.3.1.1. Rediseño del Sistema

Para la creación de la nueva capa de abstracción se han utilizado los patrones Singleton y Método Factoría. En el siguiente diagrama de clases (ver figura 23) se puede observar cómo se han adaptado estos patrones, descritos en el apartado 2.3.3 del capítulo II, a nuestro proyecto GODO.

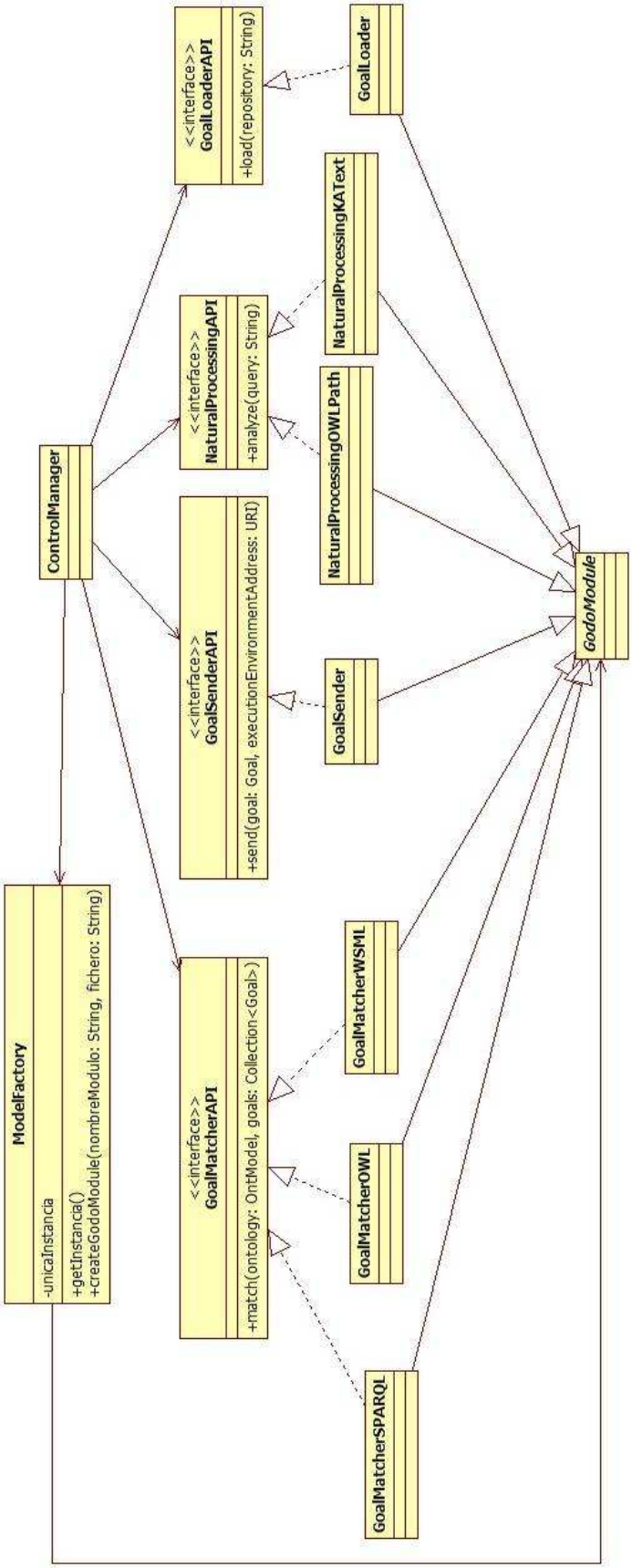

Figura 23. Diagrama de clases del sistema GODO con la nueva capa de abstracción

Para que el módulo central de GODO, el *ControlManager*, pueda configurarse para que emplee diferentes tecnologías para cada uno de los componentes del sistema, se ha implementado el patrón Método Factoría. En este caso se usan metaclases para la instanciación de cada uno de los módulos de GODO. La clase *ModuleFactory* representa la factoría, y *ControlManager* es el cliente de la misma. El Controlador (*ControlManager*) crea cada módulo a través del método *createGodoModule* de *ModuleFactory*. Este método recibe como parámetros el nombre de la clase a instanciar y el nombre del fichero de propiedades donde se encuentra el mapeo entre el nombre de la clase y la ruta del paquete donde se encuentra (en nuestro caso la API). Cada uno de los módulos de GODO extiende de *GodoModule* e implementa su correspondiente API. De este modo, el método *createGodoModule* devuelve como objeto un *GodoModule* instanciado con la tecnología concreta que se vaya a emplear.

Como podemos observar, la clase *ModuleFactory* está implementada bajo el patrón Singleton, por lo que presenta un método estático *getInstance* y un atributo *unicaInstancia* de tipo *ModuleFactory* también estático. El constructor de la clase tiene visibilidad privada, garantizando la instanciación de la clase a través del método público *getInstance*. Este método comprueba si ya se instanció la clase *ModuleFactory* para no crearla de nuevo, en caso contrario se instanciará por primera vez y única, durante el ciclo de vida de la aplicación. Gracias al patrón Singleton evitamos crear más de una instancia de la clase Factoría.

3.3.1.2. Estandarización del Sistema

Como se mencionó anteriormente, la estandarización en el uso de OWL se consiguió haciendo uso de la API de Jena. La clase *NaturalProcessingJena* contiene un método denominado *processConclusion*, que recibe la *ConclusionMcRDR* (una clase de la librería KAText utilizada para realizar el análisis de sentencias en lenguaje natural) y devuelve un *OntModel* (la interfaz de la API de Jena que representa una ontología en OWL).

Al inicio del método, se instancia la ontología OWL a través del método *createOntologyModel*.

```
newModel=ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
```

Una vez creada la ontología, se realiza el mapeo entre los conceptos de la *ConclusionMcRDR* y OWL.

Una clase OWL en Jena se instancia a través del método *createClass*, que recibe una cadena de texto con el nombre de la clase. En nuestro caso, la obtendremos mediante el método *getConcepto* de *ConceptoMcRDR*. Como resultado, este método devuelve una *OntClass* estándar de OWL.

Un atributo OWL en Jena se crea a partir del método *createDatatypeProperty*, que recibe como parámetro el nombre del atributo, obtenido del *ConceptoMcRDR* a través del método *getAtributos*. Este método devuelve un *DatatypeProperty*, que presenta opciones para añadir el rango y dominio de la propiedad.

Una relación OWL en Jena se instancia a través del método *createObjectProperty*, recibiendo el nombre de la relación como parámetro. En este caso, una relación establece un vínculo entre dos clases OWL, por lo tanto debemos establecer como rango y dominio dos clases. Éstas se obtienen de los métodos *getConcepto1* (dominio) y *getConcepto2* (rango) de una *RelacionMcRDR*.

En la siguiente figura (ver figura 24) se puede observar el diagrama de colaboración que refleja la interacción entre las clases para transformar una conclusión procesada por la librería KAText en una ontología OWL.

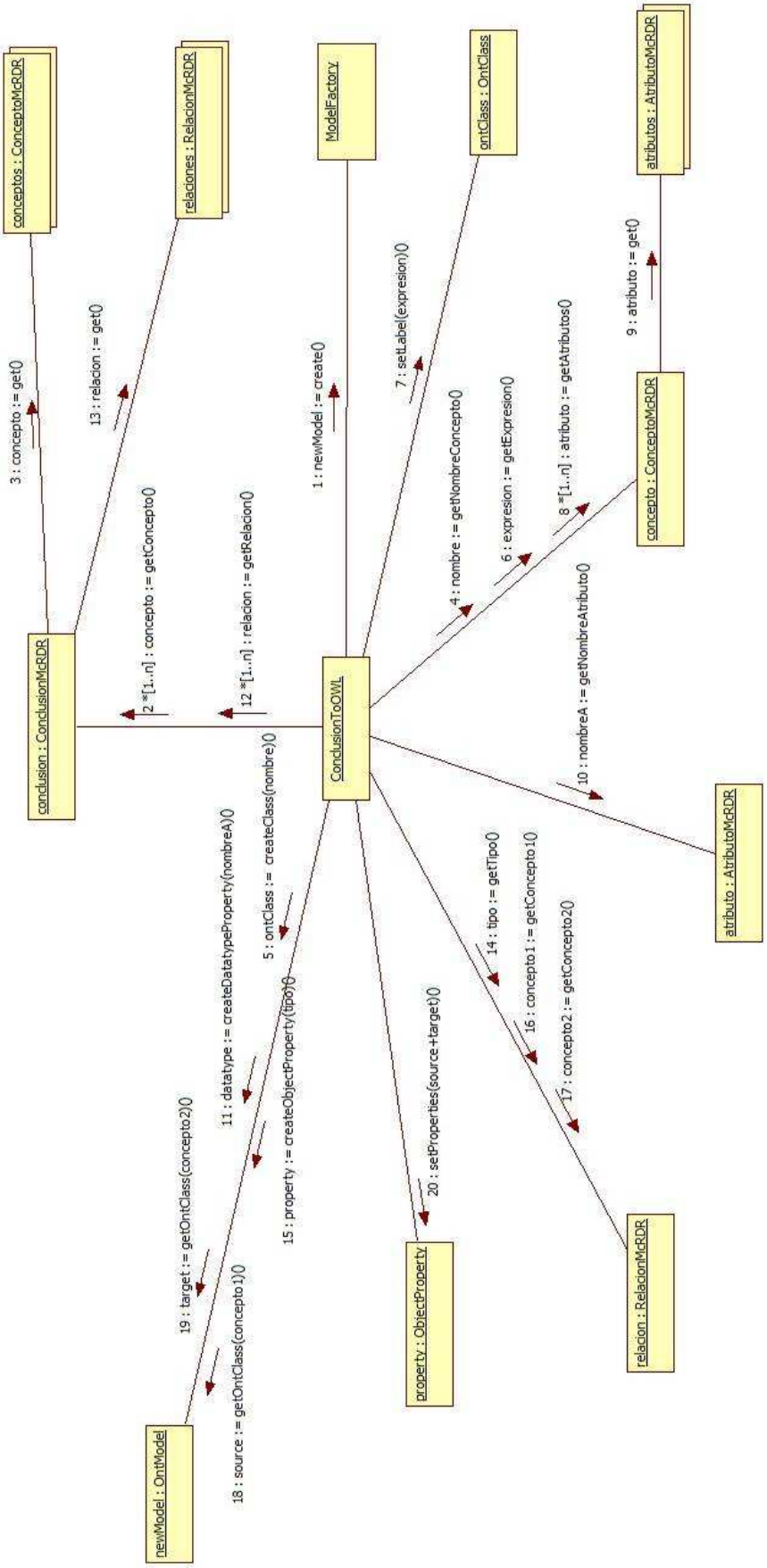

Figura 24. Diagrama de colaboración de la estandarización a OWL.

En la tabla 2 se muestra el mapeo entre una *ConclusionMcRDR*, Jena y OWL.

ConclusionMcRDR	Jena	OWL
ConceptoMcRDR	OntClass	Class
AtributoMcRDR	DatatypeProperty	Datatype Property (El rango de la propiedad contiene "*anyURI*")*
La Expresión de un ConceptoMcRDR	OntResource	rdfs:label
RelacionMcRDR	ObjectProperty	Object Property

Tabla 2. Correspondencia KAText, Jena y OWL.

(*) Debido a que no se especifica el rango en un *ConceptoMcRDR*, todas las *DatatypeProperty* se definen con rango "*anyURI*".

3.3.2. ARQUITECTURA DE LA APLICACIÓN

Una vez estudiado a fondo el estado anterior de GODO, a continuación se describe cómo se encuentra después del proyecto. En la siguiente figura (ver figura 25) podemos ver que GODO presenta un nuevo módulo denominado *Ontology Guided Input*. Este módulo se encarga de ayudar al usuario a expresar sus deseos siguiendo una aproximación guiada por la ontología. También y entrando más en detalle, se puede observar en la figura 26 un diagrama de clases de GODO, en el que se incorpora este componente.

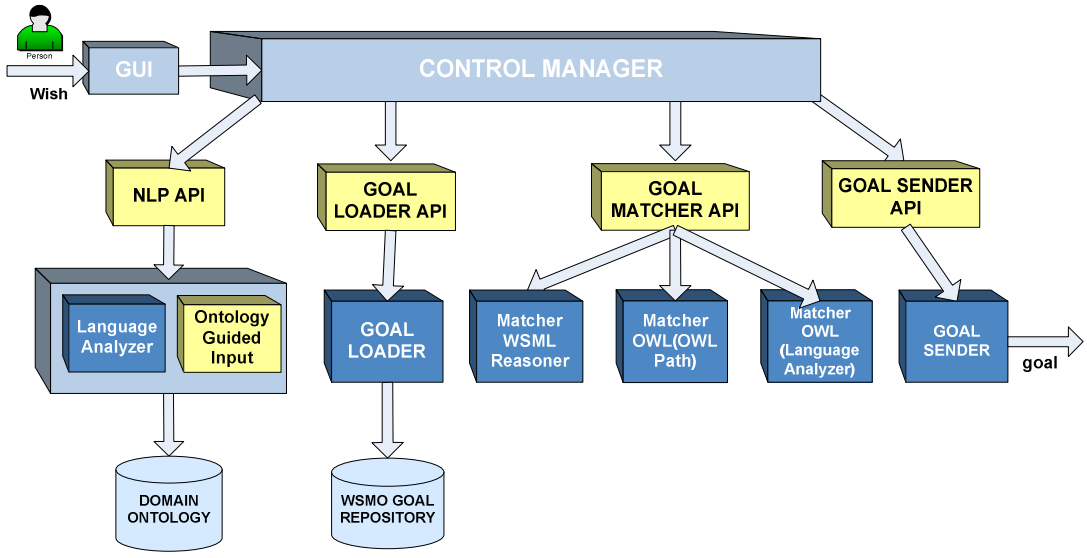

Figura 25. Arquitectura de GODO ampliada.

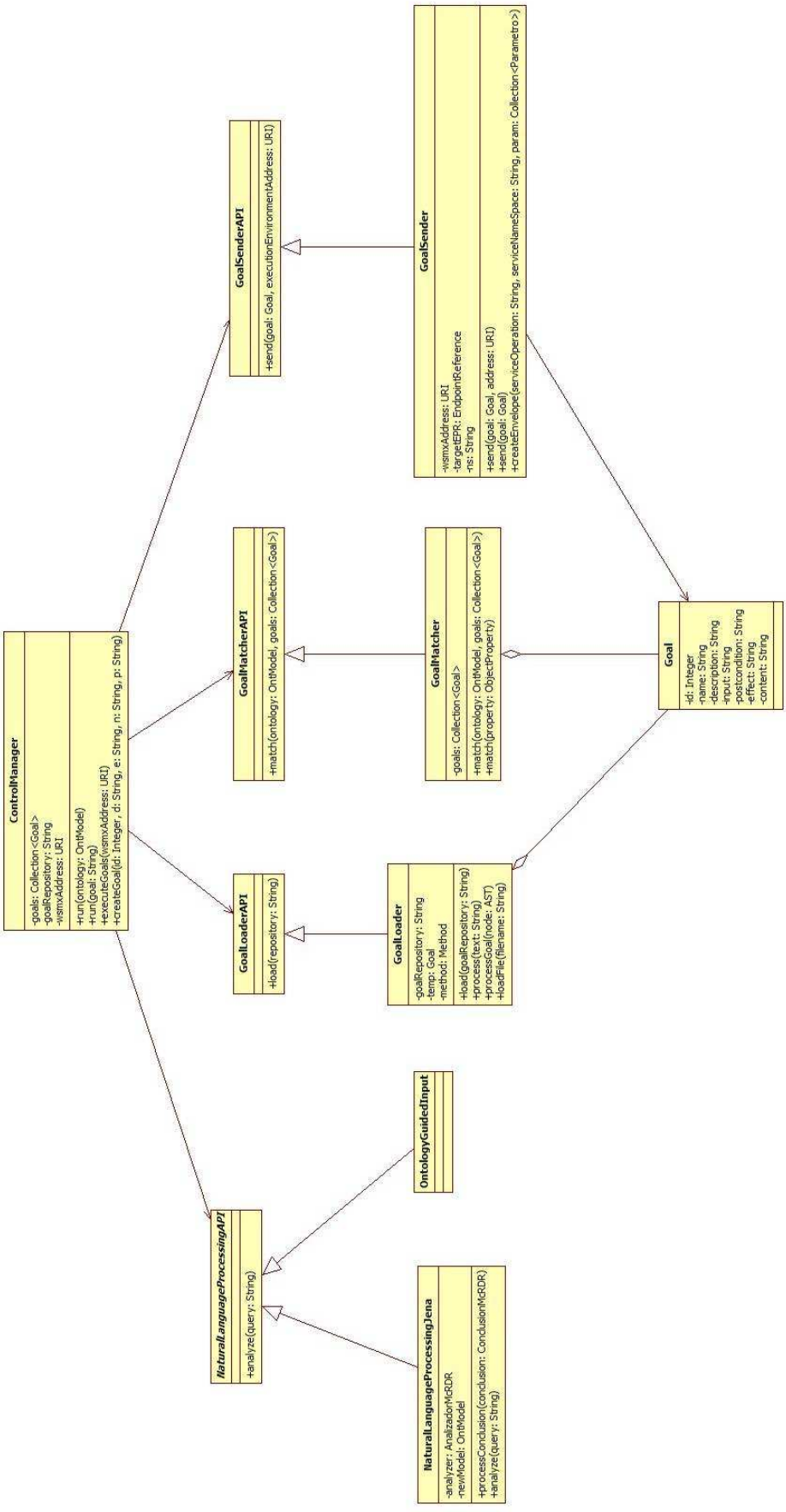

Figura 26. Diagrama de clases del sistema GODO.

La aplicación Web GODO utiliza el servlet *FrontController* (ver figura 27), que es el responsable de procesar todas las peticiones que se realizan, determinando qué acción realizar para satisfacer cada una de éstas. Para llevar a cabo esta tarea, se hace uso de la clase *PeticionHelper*, encargada de realizar las asociaciones entre peticiones y acciones a través de un fichero de propiedades donde se hayan especificadas de la forma <petición>=<clase de acción>. Estas acciones realizan toda la lógica de negocio de la aplicación y devuelven como resultado las vistas que se mostrarán tras la petición.

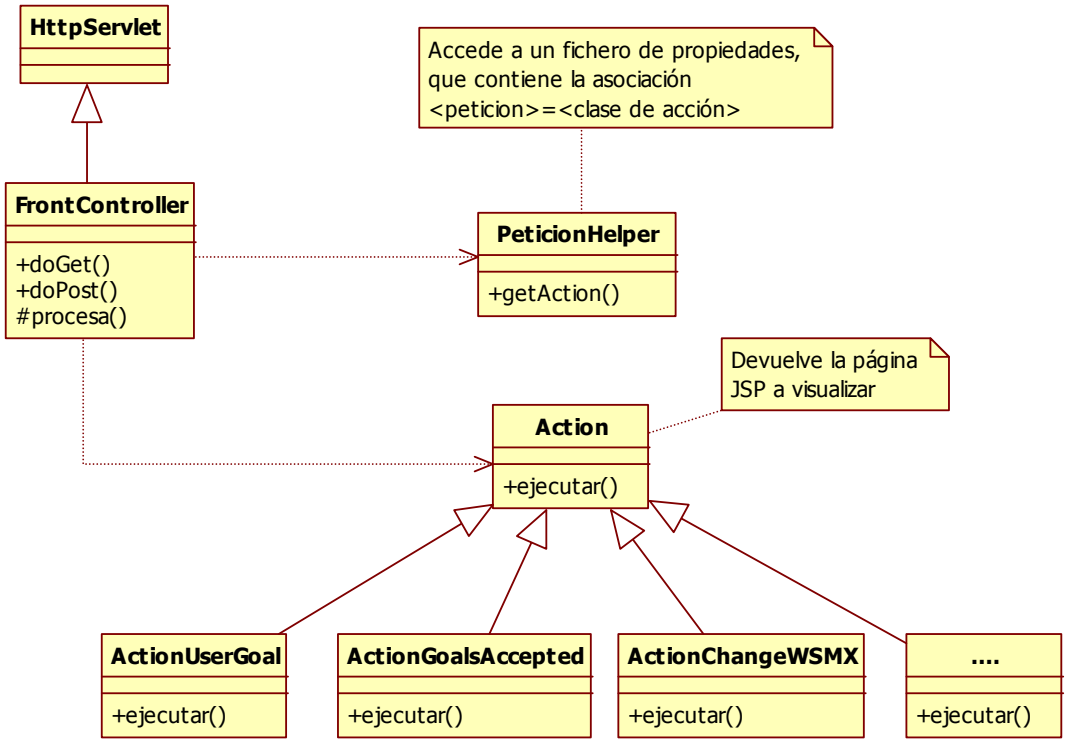

Figura 27. Diagrama de clases de FrontController y PeticionHelper (adaptado de Bermúdez, 2009b).

Una vez la clase *PeticionHelper* (ver figura 27) obtiene la acción, el servlet *FrontController* la invoca mediante el método ejecutar, que poseen cada una de las acciones de la aplicación. A través de este método, las acciones realizan las operaciones correspondientes a la lógica de negocio y devuelven una página JSP con la vista que

3.3.2.1. ControlManager

Como se ha comentado, el *ControlManager* se encarga de mediar entre los distintos módulos del sistema. Primero realiza una llamada al método *analyze* recibiendo como

parámetro la entrada de texto introducida por el usuario. El método *analyze* se encuentra en la clase *NaturalLanguageProcessing* y devuelve una ontología de Jena (*OntModel*).

A continuación, se cargan todos los objetivos a través del *GoalLoader*, mediante el método *load*, que recibe como parámetro el directorio donde se encuentran los goals. Una vez obtenida la colección de goals y la ontología mencionada anteriormente (*OntModel*), el módulo *GoalMatcher* se encarga de realizar la comparación entre cada goals y la ontología, a través del método *match*. Por último, *GoalSender* envía el goal WSML a WSMX para ser ejecutado.

3.3.2.2. NaturalLanguageProcessing

Este módulo se encarga de realizar el procesado en lenguaje natural de la sentencia introducida por el usuario. Para llevar a cabo su propósito presenta el método *analyze*, que recibe como entrada una cadena de texto con la sentencia del usuario. Este método procesa la cadena haciendo uso de KAText, una librería que realiza el análisis de sentencias en lenguaje natural. El método encargado de esto, denominado *extraeConocimiento*, devuelve una *ConclusionMcRDR*. Esta clase de KAText presenta métodos para obtener los conceptos, relaciones, y atributos de la frase introducida por el usuario.

3.3.2.3. Ontology Guided Input

Como se puede apreciar en la figura anterior (ver figura 25), la agregación de este nuevo módulo implica la incorporación de nuevos comparadores en el *Goal Matcher*.

El *Ontology Guided Input* tras guiar al usuario en el proceso de creación de la consulta, devuelve como resultado una ontología. El problema que surge es que los objetivos (goals) se encuentran descritos en WSML y por lo tanto no es posible realizar la comparación en el proceso de "*Matching*". Para solucionar este problema existen dos posibles vías:

- La primera consiste en realizar la traducción de la ontología devuelta por el *Ontology Guided Input* a WSML.
- La segunda consiste en el proceso inverso, es decir, en traducir los goals descritos en WSML a OWL.

Tras estudiar ambas posibilidades se llegó a la conclusión de que la transformación de OWL a WSML no es recomendable, ya que WSML es un lenguaje mucho menos expresivo y por lo tanto menos potente, perdiendo las ventajas que nos aporta OWLPath.

En la segunda aproximación se produce la traducción de los objetivos descritos en WSML a un lenguaje de codificación de ontologías estándar como es OWL. Esta opción es más recomendable puesto que se trabaja en todo momento con OWL, a través de la API de Jena.

Para realizar esta traducción se utilizó la librería WSMO4j. En la tabla 3 podemos ver la correspondencia que existe entre WSML y OWL.

WSML	OWL
Concept	Class
(Atributo) nombre ofType _primitivo (*)	Datatype Property
instance nombre mememOf Concept	rdf:Description
(Relación) nombre impliesType Rango (**)	Object Property

Tabla 3. Correspondencia WSML- OWL.

(*) Para especificar un atributo en WSML se establece dentro de un concepto el nombre del atributo seguido de la palabra reservada ofType y posteriormente se indica su tipo primitivo, es decir, _string, _integer, etcétera.

(**) Para definir una relación en WSML se establece dentro del concepto, que pasará a formar parte del dominio de la relación, el nombre de la misma seguido de la palabra reservada impliesType y a continuación su rango (otro concepto definido en WSML).

3.3.2.4. GoalLoader

Como ya se ha comentado, este módulo se encarga de buscar los objetivos (goals) en el repositorio de WSML, o en otro caso, en el sistema de ficheros local donde se encuentren almacenados. Para alcanzar este propósito este módulo presenta un método *load*, que recibe como entrada una cadena de texto con la ruta en la que se encuentran los objetivos. Este método se encarga de cargar un conjunto de goals que serán ejecutados por WSMX. Cada goal almacenado poseerá un nombre, una descripción, una entrada, una o varias postcondiciones y uno o varios efectos.

3.3.2.5. GoalMatcher

El *Goal Matcher* realiza la comparación entre los elementos de la ontología procedente de la fase de *NaturalLanguageProcessing* y los objetivos extraídos del repositorio de goals.

Para realizar esta comparación, este módulo presenta un método denominado *match*, que recibe como entrada la ontología *OntModel* (interfaz de la API de Jena que representa una ontología) y la colección de objetivos obtenidos durante la fase de *GoalLoader*. Este método extrae cada una de las relaciones de la ontología y delega a otro método también denominado *match* el proceso de comparación. Este método recibe como parámetro un *ObjectProperty* (interfaz de la API de Jena que representa una relación) e itera sobre la colección de goals comparando sus post-condiciones y efectos con los de la relación. Si el goal coincide lo añade a la solución.

3.3.2.6. GoalSender

Por último el módulo *GoalSender* se encarga de enviar los diferentes objetivos a WSMX para ser ejecutados. El *GoalSender* cuenta con un método denominado *send*, que recibe como parámetro el goal que será enviado a WSMX y la URI (dirección a la que se enviarán los objetivos). Este método simplemente se encarga de invocar al entorno de ejecución encargado de procesar el objetivo.

3.3.3. FUNCIONAMIENTO

En esta sección, se detalla el funcionamiento del proyecto GODO a través de las capturas de pantalla de la aplicación Web.

Inicialmente, se le presenta al usuario una interfaz sencilla e intuitiva, tomando como referencia, a la hora de realizar el diseño, Google, el buscador Web más utilizado en el mundo (ver figura 28).

Figura 28. Interfaz de GODO.

En la imagen (ver figura 28), se puede observar cómo la aplicación muestra al usuario dos opciones para la introducción de sus objetivos.

Si el usuario selecciona la opción denominada OWLPath, el sistema mostrará automáticamente, el primer elemento guiado por la ontología que el usuario puede seleccionar para realizar su consulta (ver figura 29). Conforme vaya seleccionando un elemento irá variando el siguiente a seleccionar, según la ontología del dominio. En la siguiente imagen podemos ver un ejemplo de entrada guiada.

Figura 29. Opción guiada por la ontología.

Si el usuario selecciona la opción NLP (*Natural Language Processing*), aparecerá una interfaz similar a la de comienzo, donde el usuario podrá escribir lo que desea buscar en el componente de texto que aparece (ver figura 30). Además se ofrece la posibilidad de elegir algunos ejemplos de prueba en lugar de introducir la entrada uno mismo.

Figura 30. Opción de introducción en lenguaje natural.

Imaginemos que el usuario selecciona una de las opciones de testeo que ofrece GODO, entonces la aplicación indicará la elección del usuario introduciendo la oración en el componente de texto, como podemos ver en la siguiente imagen (ver figura 31). Entonces el usuario debe indicar que desea iniciar la búsqueda de los Servicios Web, a través del botón *Send*.

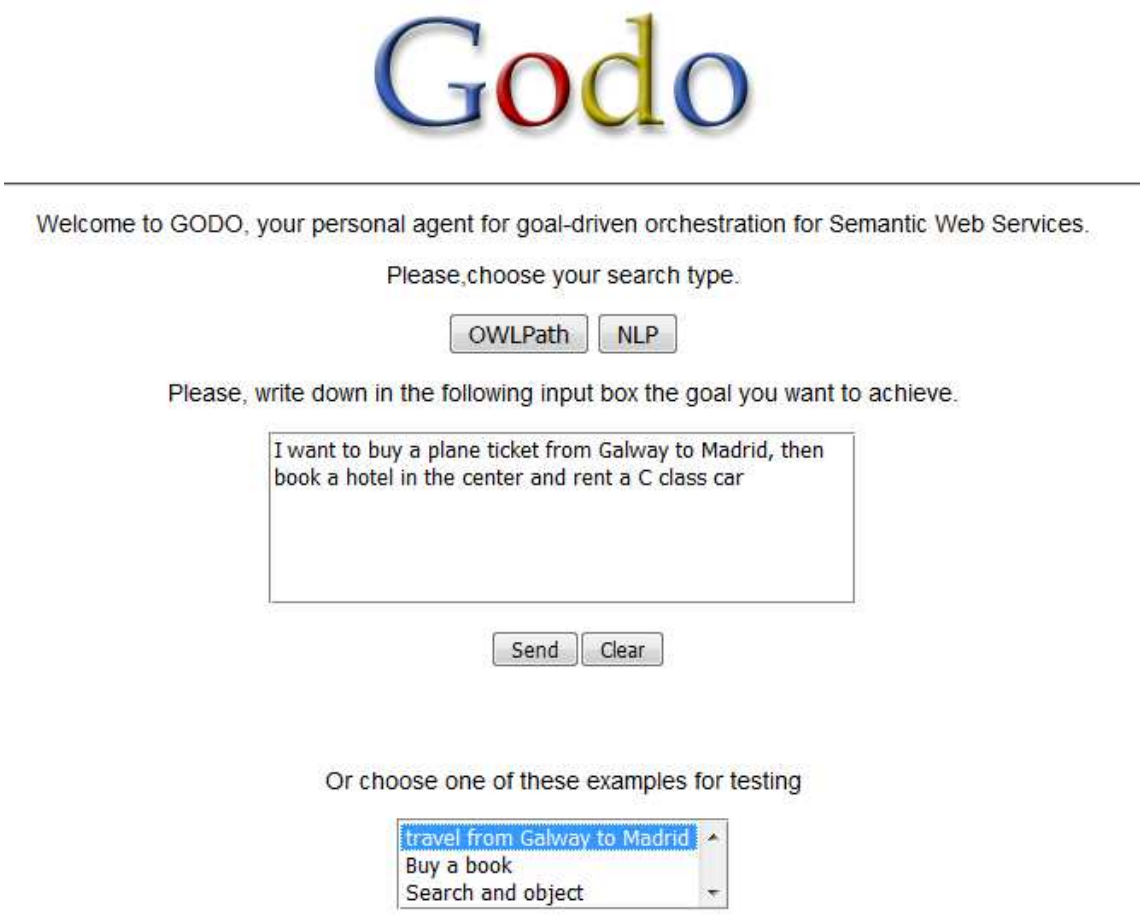

Figura 31. Ejemplo de búsqueda a través del lenguaje natural.

Una vez el usuario ha enviado la petición, aparecerá una pantalla, como la que mostramos a continuación (ver figura 32), con los objetivos (goals) que el sistema ha encontrado, junto con el orden de ejecución en WSMX.

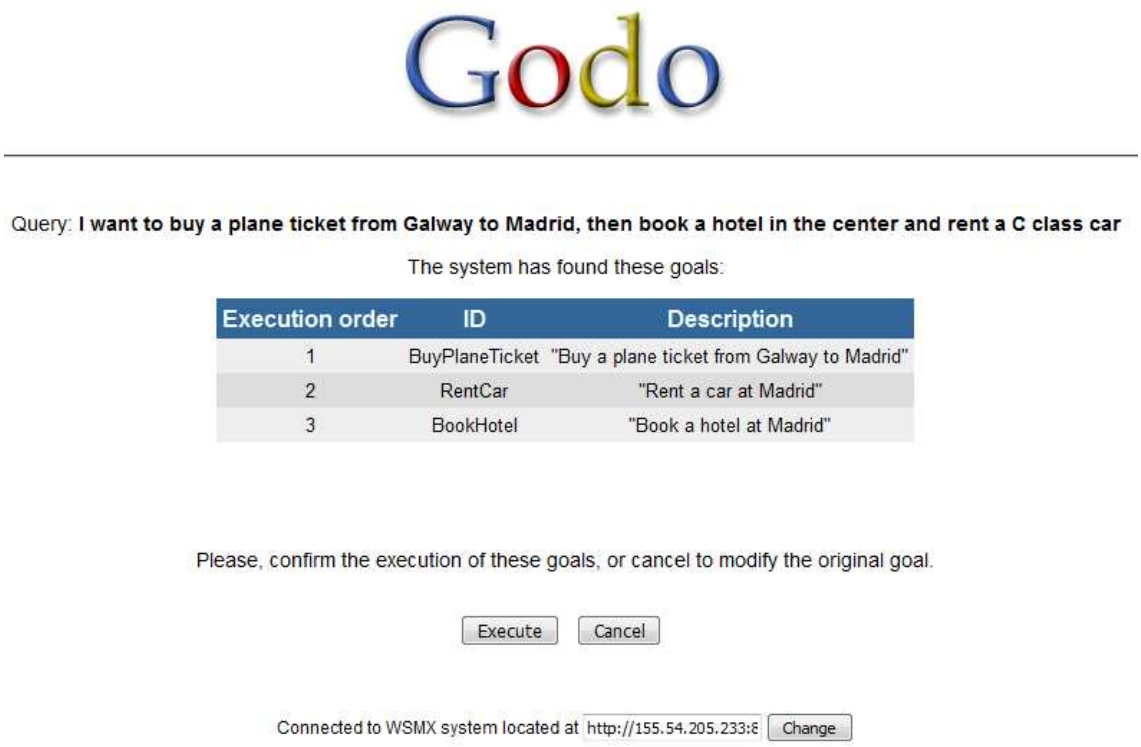

Figura 32. Resultado obtenido por GODO.

Cuando el usuario seleccione la opción *Execute*, los objetivos se ejecutarán en WSMX de forma secuencial y se mostrarán los resultados obtenidos.

CAPÍTULO IV. CONCLUSIONES Y VÍAS FUTURAS

4.1. CONCLUSIONES

La Web constituye hoy en día uno de los pilares fundamentales en el intercambio de conocimiento, pero aún sufre de ciertas carencias. Estas insuficiencias se deben a que la Web es un recurso en expansión que se ha multiplicado en los últimos años de manera considerable, y encontrar información precisa de forma eficiente y eficaz se ha convertido en una tarea prácticamente imposible para los usuarios. Este problema es el que dio lugar a lo que se conoce como Web Semántica. La Web Semántica es una Web extendida, dotada de mayor significado gracias a la incorporación de información semántica. A través de esta nueva Web enriquecida se puede poner fin a los problemas frecuentes en cuanto a la búsqueda de información en Internet.

Del mismo modo que sucedió con la Web, con la aparición de los Servicios Web, se produjo el mismo problema, es decir, un usuario no era capaz de encontrar los servicios que satisfacían sus necesidades de forma eficiente. De esta manera surgió lo que se denominan a día de hoy Servicios Web Semánticos. Los Servicios Web Semánticos consisten en la integración de la Web Semántica y los Servicios Web para obtener lo mejor de ambas tecnologías. De este modo conseguimos descubrir Servicios Web más fácilmente gracias al significado que incorporan.

El proyecto GODO nació con la idea de desarrollar una plataforma sencilla en la que el usuario pudiera buscar los Servicios Web disponibles en la red de forma eficaz y eficiente. Para llevar a cabo estos objetivos, se utilizaron técnicas de procesamiento de lenguaje natural y ontologías, para facilitar la búsqueda al usuario. Antes de comenzar con el desarrollo de la aplicación se realizaron tareas de investigación en el campo de los Servicios Web Semánticos, como el estudio de las diversas aproximaciones existentes para la descripción semántica de estos servicios, el lenguaje de representación de cada una de estas plataformas, modo de ejecución, razonadores existentes, etcétera.

Como resultado de todo este proceso, GODO dio lugar a una herramienta en la que las ontologías y los Servicios Web Semánticos se integran para automatizar las tareas de descubrimiento e invocación de Servicios Web.

Este proyecto se ha centrado, en concreto, en estandarizar el mecanismo de persistencia de información semántica a OWL utilizando la API de Jena, así como de integrar el nuevo módulo de entrada guiada por la ontología, para favorecer al usuario en su proceso de búsqueda. Además gracias a esta nueva incorporación a la arquitectura, podemos comprobar el correcto funcionamiento del nivel de abstracción y que realmente facilita la extensibilidad y se homogeniza el mecanismo a través del cual se almacena la información semántica, usando como estándar OWL. Como consecuencia de este diseño modular se consigue independizar la solución pretendida por GODO de la implementación concreta de

las diferentes tecnologías empleadas para definir Servicios Web Semánticos, facilitando además su evaluación.

Para concluir, añadir que los Servicios Web Semánticos se encuentran en continua investigación y que aún presentan debilidades como la interoperabilidad entre servicios encadenados. El fin perseguido es que algún día la incorporación de información semántica sea tan perfecta que simplemente introduciendo nuestras necesidades en la Web, obtengamos como respuesta un número de coincidencias tan reducido y tan preciso que los millones de resultados que obtenemos hoy en día sean motivo de parodia.

4.2. VÍAS FUTURAS

Gracias a la incorporación de un nuevo nivel de abstracción se consigue mejorar la extensibilidad de GODO sin repercutir en el resto del sistema. En base a esta idea se realizarían las posteriores tareas para perfeccionar la herramienta. A continuación se detallan las diferentes propuestas de mejora de GODO.

4.2.1. REPOSITORIO INTELIGENTE PARA RECUPERACIÓN DE GOALS

Actualmente la herramienta obtiene del sistema de ficheros local los diferentes objetivos que proporcionan la base de conocimiento para realizar la búsqueda de Servicios Web. Para mejorar la gestión de este almacén convendría crear un repositorio de goals. Además, sería conveniente que en dicho repositorio también existieran las ontologías del dominio que los goals utilizan para describir los Servicios Web.

Una de las tareas principales a realizar, una vez construido este repositorio, consistiría en la incorporación de un nuevo módulo para cargar en el sistema aquellos objetivos con los que la herramienta va a trabajar en una determinada consulta. La idea consiste en almacenar en el sistema sólo aquellos objetivos que cumplan con un primer proceso de selección, para evitar tener que cargar todos los goals en el sistema. Con esta nueva incorporación se lograría mejorar el rendimiento de la aplicación evitando comprobar todos los objetivos que contenga el sistema.

4.2.2. AÑADIR NUEVOS PROCESOS DE "MATCH"

El proceso de mapeo entre los goals y la ontología derivada de la sentencia introducida por el usuario ya sea usando procesamiento en lenguaje natural o guiado por la ontología, se basa exclusivamente en la comprobación entre términos semánticos equivalentes.

Se propone como trabajo futuro la incorporación de un nuevo módulo más sofisticado, utilizando para ello razonadores tanto basados en WSML como en OWL. Gracias a la

utilización de estos razonadores además de cotejar en base a los conceptos y relaciones de la ontología, se consigue inferir conocimiento que afine aún más el proceso de descubrimiento de Servicios Web.

4.2.3. INCORPORACIÓN Y EVALUACIÓN DE ENTORNOS DE EJECUCIÓN DE SERVICIOS WEB SEMÁNTICOS

Como se comenta en capítulos anteriores existen diferentes entornos de ejecución de Servicios Web Semánticos. Actualmente GODO utiliza WSMX como plataforma de ejecución. Gracias al rediseño de los módulos de GODO, será posible incorporar nuevas plataformas de ejecución de Servicios, como OWL-S Virtual Machine, IRS-III o METEOR-S, facilitando la evaluación de las mismas.

BIBLIOGRAFÍA

Akkiraju, R., Farrell, J., Miller, J., and others. (2005). Web Service Semantics – WSDL-S. Disponible en: http://www.w3.org/Submission/WSDL-S/, consultado en Enero 2009.

Atanasova, T., Agre,G., Nern, H.J. (2005). INFRAWEBS Semantic Web Unit for Design and Composition of Semantic Web Services. EUROMEDIA 2005, Toulouse, France.

Battle, S., Bernstein, A., Boley, H., and others. (2005). Semantic Web Services Framework (SWSF) Overview. 2005. Disponible en: http://www.w3.org/Submission/SWSF/, consultado en Enero 2009.

Bermúdez, F. J. (2009a). Tema 2 de los apuntes de la Asignatura de Arquitectura del Software. Disponible en: http://dis.um.es/~jbermudez/as/Tema2.pdf, consultado en Enero 2009

Bermúdez, F. J. (2009b). Tema 7 de los apuntes de la Asignatura de Desarrollo de Aplicaciones Distribuidas. Disponible en: http://dis.um.es/~jbermudez/dad/, consultado en Enero 2009.

Berners-Lee, T. (2003). Standards, Semantics and Survival. SIIA Upgrade 2003.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web: A new form of Web content thar is meaningful to computers will unleash a revolution of new possibilities. The Scientific American.

Brujin, J., Bussler, C. (2005a). Web Service Modeling Ontology (WSMO). Disponible en: http://www.w3.org/Submission/WSMO/, consultado en Enero 2009.

Brujin, J., Feier, C., Keller, U., Lara, R., Polleres, A., Predoiu, L. (2005b). WSML Reasoner Survey.

Bussler, C., Cimpian, E., Fensel, D., and others. (2005)-WSMX. Web Service Execution Environment (WSMX). Disponible en: http://www.w3.org/Submission/WSMX/, consultado en Marzo 2009.

Chirlaque-Medrano, J. L., Valencia García, R., García Sánchez, F., Castellanos Nieves, D., Fernández Breis, J.T. (2008).OWLPath: an OWL ontology-guided query editor

Davies, J., Fensel, D., van Harmelen, F. (2003). Introduction, Towards the Semantic Web. 1-10. John Wiley and Sons, Ltd.

Davies, J., Studer, R., Warren, P. (2006). Semantic Web Technologies Trends and Research in Ontology-based Systems.

Dimitrov, M., Momtchev, V., Ognyanoff, D., Konstantinov, M.(2006). Wsmo4j Programmers Guide.

Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., and Pedrinaci, C. (2008). IRS-III: A Broker-based Approach to Semantic Web Services, Journal of Web Semantics, 6, 2, pp. 109-132, Elsevier.

Dumitru, R. (2004). WSMO – new structure, main intermediate deliverables -. 2nd F2F meeting SDK cluster working group on Semantic Web Services. Lausanne, Suiza.

Dumitru, R., Holger, L., and Others. (2004). D2v1.0. Web Service Modeling Ontology (WSMO). Disponible en :http://www.wsmo.org/2004/d2/v1.0/, consultado en Enero 2009.

ECLIPSE . Eclipse. Disponible en: http://www.eclipse.org/, consultado en Diciembre 2008.

Fact++. Disponible en: http://owl.man.ac.uk/factplusplus/, consultado en Junio 2009.

Falkner, J., Jones, K. (2004) Servlets and JavaServer Pages: The J2EE Technologies Web Tier. Addison-Wesley.

Farrell, J., Lausen, H. (2007). Semantic Annotarions for WSDL and XML Schema. Disponible en: http://www.w3.org/TR/sawsdl/, consultado en Enero 2009.

Fensel, D & Bussler, C. (2002).The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications

Fernández Breis, J. T., Prendes Espinosa, M.P., Castellanos Nieves, D., Martínez Sánchez, F. Valencia García, R. Ruiz Martínez, J.M. (2007). Evaluación en e-learning basada en Tecnologías de la Web Semántica y de procesamiento del lenguaje natural.

Flanagan, D. (2006). Javascript: The Definitive Guide, Fifth Edition. O'REILLY.

García Sánchez, F., Gómez, J.M., Rico, M., Valencia García, R., Fernández Breis, J.T. (2008). Enabling Intelligent Service Discovery with GGODO.

García Sánchez, F. (2007). Sistema Basado en tecnologías del conocimiento para entornos de Servicios Web Semánticos.

Gómez, J.M., Rico, M., García Sánchez, F. Acuña, C. (2005). El tetraedro de Servicios Web Semánticos: Integración basada en Servicios Web Semánticos.

Gómez, J.M., Rico, M., García Sánchez, F., Martínez Béjar, R., Bussler, C. (2007). GODO: Goal driven orchestration for Semantic Web Services. Disponible en: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-113/paper1.pdf, consultado en Noviembre 2008.

Gruber, T. (2008). Ontology. Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds.), Springer-Verlag.

Herold, M. (2008). WSMX Documentation. Disponible en: http://www.wsmx.org/papers/documentation/WSMXDocumentation.pdf, consultado en Febrero 2009.

Horstmann, C.S., Cornell, G. (2006). Java 2. Vol I. Fundamentos. Pearson/Prentice Hall.

HT (2009). Hoy Tecnología. Disponible en: http://www.hoytecnologia.com/noticias/Internet-crece-diariamente-diez/66142, consultado en Abril 2009.

JENA . Jena – A Semantic Web Framework for Java. Disponible en: http://jena.sourceforge.net/, consultado en Febrero 2009.

Kang, B. (1996). Multiple classification ripple down rules. PhD Thesis, University of New South Wales.

Larman Craig. (2004). UML y Patrones. Una introducción al Análisis y el Diseño Orientado a Objetos y al Proceso Unificado

Lausen, H., Brujin, J. (2005). D16.1v0.3 The Web Service Modeling Language WSML Disponible en: http://www.wsmo.org/TR/d16/d16.1/v0.3/20050320/, consultado en Febrero 2009.

Martin, D., Burstein, M., Hobbs, J., Lassila, O. (2004). OWL-S: Semantic Markup for Web Services. Disponible en: http://www.w3.org/Submission/OWL-S/, consultado en Enero 2009.

McBride, B. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Disponible en: http://www.w3.org/TR/rdf-schema/, consultado en Enero 2009.

Meyer, B. (1999). Construcción de software orientado a objetos. 2nd edición, Prentice-Hall.

Meyer, E.A. (2000). Cascading Style Sheets: The Definitive Guide. O'Reilly & Associates.

Olaya, A., Pérez, N. (2006). Los Servicios Web Semánticos, una solución a los problemas de interoperabilidad en Modelos Digitales de Terreno. GSDI-9 Conference Proceedings, 6-10, Santiago, Chile.

OWL Ontology Web Language. Disponible en: http://www.w3.org/TR/owl-features/ , consultado en Enero 2009.

Paolucci, M., Ankolekar, A., Srinivasan, N. and Sycara, K.P. (2003). The DAML-S virtual machine. International Semantic Web Conference, volume 2870 of Lecture Notes in Computer Science, pages 290–305. Springer.

Patil, A., Oundhakar, S., Sheth, A., Verma, K. (2004). METEOR-S Web service Annotation Framework, Proceeding of the World Wide Web Conference (Proceeding of the World Wide Web Conference).

Pellet. Disponible en: http://pellet.owldl.com/, consultado en Junio 2009.

Pressman, R. (2005). Software Enginnering: A Practitioner's Approach. McGraw-Hill series in computer science.

PROTEGE . Protègè. Disponible en: http://protege.stanford.edu/, consultado en Enero 2009.

RDF Resource Description Framework. Disponible en: http://www.w3.org/RDF/, consultado en Enero 2009.

Roman, D., Lausen, H., Keller, U., and others. (2004). D2v1.0 Web Service Modeling Ontology (WSMO). Disponible en: http://www.wsmo.org/2004/d2/v1.0/, consultado en Febrero 2009.

Ruiz Sáncher, J.M., Valencia García, R., Fernández Breis, J.T., Martínez Béjar, R & Compton, P. (2003). And approach for incremental knowledge acquisition from text. Expert Systemss With Applications.

STARUML . StarUML. The Open source UML/MDA Platform. Disponible en: http://staruml.sourceforge.net/en/, consultado en Marzo 2009.

Studer, R., Benjamins, R., and Fensel, D. (1998). Knowledge Engineering: Principles and Methods.

TOMCAT . Tomcat. Disponible en: http://tomcat.apache.org/, consultado en Diciembre 2008.

UML . Unified Modeling Language. Disponible en: http://www.uml.org/, consultado en Enero 2009.

Valencia Castillo, E. (2007). Recuperación y organización de la información a través de RDF usando SPARQL.

Valencia García, R. (2005). Un Entorno para la Extracción Incremental de Conocimiento desde Texto en Lenguaje Natural. Tesis Doctoral, Universidad de Murcia. Disponible en: http://www.tdr.cesca.es/TESIS_UM/AVAILABLE/TDR-1123105-140512//rvalencia.pdf, consultado en Abril 2009.

Valencia García, R., Ruíz Sáncher, J.M., Vivancos Vicente, P.J., Fernández Breis, J.T & Martínez Béjar, R. (2004). An incremental approach for discovering medical knowledge from text. Expert Systems With Application.

Valledor, P. (2006). Servicios Web Semánticos. Disponible en: http://www.di.uniovi.es/~cueva/asignaturas/doctorado/2006/trabajos/sws.pdf, consultado en Junio 2009.

Vicente Torres, M. C. (2007). Entorno de trabajo basado en uso intensivo de ontologías para ejecución de Servicios Web Semánticos.

W3CSPARQL (2008). SPARQL Query Language for RDF. Disponible en: http://www.w3.org/TR/rdf-sparql-query/, consultado en Febrero 2009.

W3CWS (2008). Guía breve de Web Semántica. Disponible en: http://www.w3c.es/Divulgacion/Guiasbreves/WebSemantica, consultado en Enero 2009

WWWS (2009). World Wide Web Size. Disponible en: http://www.worldwidewebsize.com/ , consultado en Abril 2009.

XML Extensible Markup Language. Disponible en: http://www.w3.org/XML/, consultado en Enero 2009.

Zakas, N., Mcpeak, J., Fawcett, J. (2006). Ajax Profesional. ANAYA.