Universidad Carlos III de Madrid

Escuela Politécnica Superior

SEMSE search: Sistema de recuperación conceptual de esquemas de metadatos

Proyecto Fin de Carrera

Ingeniería Técnica en Informática de Gestión

Julio 2011

AUTOR:	Miguel Zazo Torrubias
TUTOR:	Vicente Palacios Madrid

Resumen

La Web actual ofrece una gran cantidad de información pero que es poco o nada procesable de forma automática, esto es, entendible y capaz de ser tratada por un ordenador, más allá de las búsquedas sintácticas que son capaces de realizar los buscadores actuales.

Esta carencia, limita en gran medida las posibilidades que podría ofrecer una Web preparada para el procesamiento automático de la información. Una Web en la que los buscadores supieran exactamente qué quiere buscar el usuario y con qué significado, poder mostrar la información sin obligar al usuario a visitar página por página o incluso mostrar la información mediante resúmenes o análisis automáticos.

Como solución se propone la Web Semántica, una Web en la que cada recurso publicado en la Web contenga una capa semántica desarrollada mediante esquemas de metadatos, capaces de ser leídos y procesados de forma automática por un ordenador. Sin embargo, esto implica que los desarrolladores utilicen unos esquemas de metadatos apropiados que les permita especificar qué tipo de información es la que se muestra en cada recurso. En muchos casos, esto suele provocar que cada desarrollador cree su propio esquema adaptado a sus necesidades, pasando por alto el hecho de que puedan existir ya esquemas que le puedan resultar de utilidad.

En este punto es en el que entra este proyecto, de forma que propone solucionar este problema mediante la creación de un buscador específico, que permita al desarrollador buscar un concepto y que, mediante una base semántica propia, se ofrezca al usuario un listado de todos los esquemas de metadatos que contengan dicho concepto o alguno semánticamente equivalente. De esta forma se facilita la reutilización de esquemas y se allana el camino hacia la Web Semántica.

Índice general

Capítulo I. Introducción 1

1.1 Motivación 3

1.2 Objetivos 4

1.3 Estructura del Proyecto 6

Capítulo II. Estado del Arte 7

2.1 La Web 8

2.1.1 Funcionamiento de la Web 9

2.1.2 Problemática de la Web actual 10

2.2 La Web Semántica 12

2.2.1 Descripción de la Web Semántica 13

2.2.2 Arquitectura 15

2.2.3 Lenguajes 19

2.2.4 Ontologías 29

2.3 Diseño de aplicaciones Web 36

2.3.1 Elementos clave de sitios Web centrados en usuario 37

2.3.2 UML (Unified Modeling Language) 39

2.3.3 Estilos y patrones de diseño 45

2.4 Semantic Metadata Search (SEMSE) 54

2.4.1 Esquemas semánticamente cualificados 55

2.4.2 Ontologías específicas 58

2.4.3 Búsqueda y selección de esquemas y su representación semántica 63

2.4.4 Selección de la ontología de referencia 68

2.4.5 Correspondencia entre los esquemas y la ontología de referencia 73

Capítulo III. Herramientas para la elaboración del proyecto 77

3.1 Infraestructura del servidor 78

3.1.1 Sistema Operativo 78

3.1.2 Servidor Web 80
3.1.3 Servidor de aplicaciones 82
3.1.4 Sistema gestor de bases de datos 83
3.2 Entorno de trabajo 87
3.2.1 Java 87
3.2.2 Netbeans 98
3.3 Frameworks empleados 101
3.3.1 JSP 101
3.3.2 JPA 103
3.3.3 Spring 110
3.3.4 Jena 121
3.4 Otras herramientas 125
3.4.1 PgAdminIII 125
3.4.2 Protégé 126
Capítulo IV. Desarrollo del proyecto 129
4.1 Fase inicial 130
4.1.1 Proceso de desarrollo 132
4.1.2 Especificación de requisitos de Usuario. 134
4.2 Fase de análisis 138
4.2.1 Diagrama de casos de uso de la fase de análisis 138
4.3 Diseño Arquitectónico 153
4.3.1 Infraestructura Hardware 154
4.3.2 Infraestructura Software 155
4.4 Diseño Detallado 157
4.4.1 Diagrama de clases 159
4.4.2 Presentación 161
4.4.3 Negocio 162
4.4.4 Infraestructura 191
4.4.5 Persistencia 194
4.5 Manual de Usuario 195
4.5.1 Búsqueda 195

4.5.2 Autenticación de usuario 198
4.5.3 Nuevo esquema 199
4.5.4 Eliminar esquema 200
4.5.5 Consulta y modificación de la URI de la Ontología de Mapeo 203
4.5.6 Consulta y modificación de la URI de la Ontología de Referencia..... 204
4.6 Planificación final y Presupuesto 205
4.6.1 Planificación final 205
4.6.2 Presupuesto 207
Capítulo V. Conclusiones 209
5.1 Desarrollos futuros 211
Bibliografía 212

Lista de Figuras

Figura II-1. Arquitectura de la Web Semántica, obtenido de (Obikto, 2007) 15

Figura II-2. Representación gráfica de una sentencia RDF. 23

Figura II-3. Triángulo del Significado, obtenido de(Ogdeny Richards, 1923). 30

Figura II-4. Ejemplo de uso de ontologías, obtenido de (Krsulovic Morales, 2003) 33

Figura II-5. Ejemplo de diagrama de casos de uso 42

Figura II-6. Figura de un paquete 42

Figura II-7. Figura de una clase 43

Figura II-8. Figura de una interfaz. 43

Figura II-9. Ejemplo de diagrama de clases 44

Figura II-10. Ejemplo diagrama de secuencia 45

Figura II-11. Modelo Vista Controlador, obtenido de (Casanovas, 2004) 47

Figura II-12. Arquitectura MVC modelo1 49

Figura II-13. Arquitectura MVC modelo2. 49

Figura II-14. Estructura del patrón capas 50

Figura III-1. Funcionamiento de Java, obtenido de (Patel, 2008) 93

Figura III-2. Ejemplos de clases, obtenido de (Wikipedia, 2010a) 96

Figura III-3. Captura de NetBeans 6.9.1 101

Figura III-4. Interpretación de una página JSP 102

Figura III-5. Uso de etiquetas de mapeo de JPA 104

Figura III-6. Arquitectura de JPA, obtenido de (IBM, 2011) 107

Figura III-7. Arquitectura de Spring 112

Figura III-8. Diagrama de clases que participan en la autenticación de Spring Security 119

Figura III-9. Funcionamiento del DaoAuthenticationManager 120

Figura III-10. Filtros Web de Spring Security 120

Figura III-11. Tripleta RDF 122

Figura III-12. Almacenamiento persistente de modelos 124

Figura III-13. Inferencia en Jena 124

Figura III-14. PgAdminIII 126

Figura III-15. Protégé 128

Figura IV-1. Diagrama de Casos de Uso inicial 137

Figura IV-2. Diagrama general de Casos de Uso 139

Figura IV-3. Caso de Uso Búsqueda 141

Figura IV-4. Caso de Uso Gestión de Esquemas 145

Figura IV-5. Casos de uso Gestión de Ontologías 147

Figura IV-6. Arquitectura del sistema 154

Figura IV-7. Arquitectura Cliente-Servidor 154

Figura IV-8. Diagrama de Componentes 155

Figura IV-9. Diagrama de clases 159

Figura IV-10. Vista buscar.jsp 161

Figura IV-11. Vista Logon.jsp 161

Figura IV-12. Vista nuevo.jsp 162

Figura IV-13. Vista eliminar.jsp 162

Figura IV-14. Esquema relacional de la base de datos 194

Figura IV-15. Captura de las partes de la entrada de una búsqueda 196

Figura IV-16. Captura de los resultado de una búsqueda 197

Figura IV-17.Captura de la introducción de usuario y contraseña 198

Figura IV-18. Captura de la pestañas de las funciones especiales 198

Figura IV-19. Captura de la adición de un nuevo esquema..... 199

Figura IV-20. Captura del mensaje de información sobre el resultado de una adición de esquemas..... 200

Figura IV-21. Captura del listado para eliminar un esquema..... 201

Figura IV-22. Captura de la confirmación de la eliminación de un esquema..... 202

Figura IV-23. Captura del mensaje de información sobre la eliminación de un esquema 202

Figura IV-24. Captura de la modificación de la URI de la ontología de mapeo..... 203

Figura IV-25. Captura del mensaje informativo sobre la modificación de la URI de la ontología de mapeo..... 203

Figura IV-26. Captura de la modificación de la URI de la ontología de referencia 204

Figura IV-27. Captura del mensaje informativo sobre la modificación de la URI de la ontología de referencia 205

Figura IV-28. Planificación final del proyecto..... 206

Lista de Tablas

Tabla II-1. Resumen de los esquemas cualificados generados 66

Tabla II-2. Resumen de las ontologías específicas generadas..... 68

Lista de Ejemplos

Ejemplo II-1. XML para representar la ficha de una persona 19

Ejemplo II-2. Representación textual de una sentencia RDF. 23

Ejemplo II-4. Codificación de la clase SubjectProperty incluida en la ontología SEMSE 60

Ejemplo II-3. Codificación de la clase SubjecConcept incluida en la ontología SEMSE .. 60

Ejemplo II-5. Codificación del concepto Semse incluida en la ontología SEMSE 61

Ejemplo II-6. Codificación de la propiedad HasSemset incluida en la ontología SEMSE 62

Ejemplo II-7. Codificación de la propiedad HasPropertySemset incluida en la ontología SEMSE 62

Ejemplo III-1. persistance.xml 109

Capítulo I. Introducción

Hoy en día, Internet y el uso de páginas Web se ha convertido en el recurso por excelencia de quienes buscan cualquier tipo de información o quienes desean publicarla.

El diseño original de la Web ha estado orientado al uso que las personas hacen de ella y en cómo la información se presenta al usuario. Sin embargo, en los últimos años, se ha ido notando una pérdida de potencial o limitaciones de este planteamiento en el proceso de automatización del uso de la información por parte de los sistemas informáticos, tales como la recuperación de dicha información, utilización de sistemas de tipo pregunta-respuesta o el razonamiento mediante inferencias. Dichas limitaciones vienen marcadas por la tecnología utilizada, el número y calidad de los metadatos y el tamaño de la propia Web.

La solución a estas limitaciones, consiste en transformar la información a un formato más comprensible para el software, con una mayor carga de semántica codificada, es decir, crear una "capa semántica" que pueda ser "comprendida" por el

software. Este es el objetivo de la Web Semántica, y para conseguirlo utiliza como elementos clave los metadatos semánticos y ontológicos.

Las ontologías son una parte importante de la Web semántica. En la Web semántica, las ontologías pueden ser usadas para incluir significado dentro de un recurso Web. De este modo las aplicaciones entienden sobre qué trata el recurso, el significado de los conceptos que incluye, y además proporciona a los humanos servicios cooperativos de mayor utilidad.

Las ontologías proporcionan la vía para que el conocimiento de la Web esté representado de forma que sea comprensible por los ordenadores. Una ontología define un vocabulario común para el personal que necesita compartir la información en un dominio. Incluye definiciones de conceptos básicos en el dominio y sus relaciones de forma que puedan ser interpretados por una máquina.

El proyecto SEMSE (SEMSE, 2008) tiene como objetivo el desarrollo de un sistema de gestión de conocimiento basado en ontologías. Concretamente, proporciona representaciones ontológicas de esquemas de metadatos consistentes en dotar a aquellos esquemas que no lo incluyen, de un plano semántico, permitiendo así la asignación de semántica y desambiguación de los términos, mediante una estructura conceptual flexible. En SEMSE, a través de una ontología de referencia, se consigue un mapa conceptual y relacional que incluye y permite equivalencias léxicas entre los términos de las etiquetas de los documentos y la estructura ontológica general. De este modo, se interrelacionan los elementos incluidos en los esquemas, con los conceptos incluidos en una ontología de referencia. La ontología de referencia realiza las funciones de diccionario general mientras que los esquemas de metadatos hacen las funciones de diccionarios especializados, superando las limitaciones de los aspectos fraseológicos y las irregularidades en cuanto a la información en las definiciones. Además de servir como vocabulario controlado y proporcionar significado a los elementos, a través de la ontología de referencia se consigue un mapa conceptual y relacional que incluye y permite equivalencias léxicas entre los términos de las etiquetas de los documentos y la estructura ontológica general.

1.1 Motivación

Como parte del proyecto SEMSE y basándose en el trabajo ya realizado, el presente Proyecto Fin de Carrera (en adelante PFC) surge de la necesidad de compartir dicho trabajo con el resto de usuarios que lo necesiten. Con el objetivo de facilitar el camino al desarrollo y extensión de las Webs basadas en la idea propuesta por la Web semántica, este proyecto presenta una forma de ofrecer a los programadores un medio para poder localizar y reutilizar, de una manera sencilla y rápida, esquemas de metadatos ya existentes que puedan adecuarse a sus necesidades sin tener que generar esquemas nuevos.

Esto se materializa a través de una aplicación web cuyo principal objetivo será proporcionar un buscador semántico con el que, a partir de un concepto introducido por el usuario, y aprovechando el conocimiento ofrecido por la base ontológica existente en el proyecto SEMSE, muestre al usuario un listado completo de los esquemas que ya incluyen dicho concepto. A partir de los conceptos encontrados, el usuario podrá visualizar sus esquemas de origen con el fin de poder localizar uno que se adapte a sus necesidades.

La combinación de la aplicación propuesta y la base ontológica de SEMSE permitirá la recuperación semántica de conceptos, eliminando las barreras producidas por la ambigüedad semántica inherente a la definición de los esquemas actuales: el plurilingüismo, la sinonimia y la homonimia.

De forma adicional, parte de los requisitos y necesidades de este proyecto han sido heredadas del propio proyecto SEMSE, las cuales se resumen en:

- **Necesidades tecnológicas**: El sistema será accesible vía Web, tendrá en cuenta aspectos de seguridad y estará basado en herramientas de software libre, tanto para minimizar el coste en software, como para ofrecer una solución independiente de la plataforma.
- **Necesidades de diseño:** El sistema tendrá un diseño amigable, sencillo y fácil de utilizar. Deberá proporcionar flexibilidad, ser ampliable y adaptable

fácilmente, además de basarse en una arquitectura que proporcione un bajo acoplamiento y una alta cohesión.

1.2 Objetivos

El objetivo principal de este proyecto, como ya se ha adelantado en el punto anterior, es la creación de un buscador alojado en una página Web, que permita realizar consultas sobre una base ontológica contra la que se mapean los esquemas de metadatos incluidos en el proyecto SEMSE, así como los esquemas que se deseen adicionar en un futuro.

Aunque será descrito con mayor profundidad en el apartado *2.4 Semantic Metadata Search (SEMSE)*, a partir de cada esquema original, se dispone de dos esquemas más que completan la información semántica de cada uno de ellos, además de hacerla entendible para el resto de la aplicación. Dichos esquemas son: el esquema cualificado semánticamente y la ontología específica de cada uno. Dichas ontologías estarán almacenadas en la base de datos de la que partirá la aplicación. El conjunto de éstas, además de las ontologías de mapeo y de referencia, completan el conocimiento sobre el que operará el buscador creado, de forma que, a partir del lugar exacto sobre el que se aplicará la búsqueda y de la forma de buscar, se podrán realizar tres tipos de búsquedas diferentes.

- Búsquedas sintácticas
- Búsquedas conceptuales
- Búsquedas contextuales

Estos tres tipos de búsqueda no son excluyentes unos a otros, sino que son incrementales. Es decir, partiendo de la búsqueda sintáctica, las más sencilla de las tres, se va añadiendo complejidad al algoritmo de búsqueda de forma que se obtienen conceptos relacionados semánticamente con el concepto buscado de una forma más compleja aprovechando la semántica aportada por las ontologías.

Además, tanto para las búsquedas conceptuales como para las contextuales, el buscador permitirá, de forma adicional, desglosar la búsqueda en dos pasos. De esta forma, se le da al usuario la posibilidad de elegir qué concepto, de los mostrados en el primer paso, es sobre el que desea saber más.

Por último, con el objetivo de ampliar los esquemas originales y sus correspondientes ontologías, la aplicación deberá de ser capaz de añadir más esquemas a la base de datos sobre la que trabaja, así como poder eliminarlos si ya no fueran útiles o relevantes. Además, dado que las ontologías de mapeo y referencia no estarán almacenadas en la base de datos, sino que lo que se almacenará será su URI[1] pública en Internet para su consulta en tiempo real, la aplicación deberá permitir cambiar dichas URIs en cualquier momento. Tanto la adición o eliminación de esquemas como la modificación de las URIs de las ontologías de mapeo y referencia, estarán limitadas a un único usuario administrador.

Para lograr estos objetivos, se aplicará un ciclo de vida en cascada, con las siguientes fases, tal y como se detallará en el *Capítulo IV Desarrollo del proyecto*.

1. Obtención de los requisitos.
2. Definición del modelo arquitectónico de la aplicación y evaluación de las tecnologías a utilizar.
3. Desarrollo de la aplicación.
4. Despliegue y puesta en producción.

[1] Uniform Resource Identifier, identificador uniforme de recurso, definido en RFC 3986. Es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico…) normalmente accesible desde una red o sistema.

1.3 Estructura del Proyecto

En el presente Capítulo se ha ofrecido una introducción al proyecto, presentándose las motivaciones, así como los objetivos a alcanzar.

En el *Capítulo II Estado del Arte* se aborda el estado del arte en materias relacionadas con este trabajo, tales como: conceptos sobre la Web semántica, ontologías, aplicaciones Web, el desarrollo de software y el análisis de tecnologías.

En el *Capítulo III Herramientas para la elaboración del proyecto* se describe el conjunto de herramientas que se han utilizado para la elaboración del proyecto, tanto en su desarrollo, como en el despliegue de la aplicación.

El *Capítulo IV Desarrollo del proyecto* es la síntesis del esfuerzo de elaboración de este trabajo. Se recoge el proceso de desarrollo del buscador implementado, así como una planificación y un presupuesto final.

En el *Capítulo V Conclusiones* se abordan las conclusiones del proyecto y se presentan las líneas futuras de ampliación y mejora.

Finalmente se incluye la *Bibliografía* con las distintas referencias bibliográficas utilizadas en la documentación de este proyecto.

Capítulo II. Estado del Arte

Tal y como se ha planteado en la introducción de este trabajo, este PFC consiste en el desarrollo de un buscador alojado en una página Web, que permita la localización de esquemas que incluyan un concepto concreto a través de su semántica. Se potencia de esta manera, la reutilización de esquemas de metadatos a usuarios y desarrolladores, evitando así la definición de esquemas ad-hoc en dominios ya modelados.

En este capítulo veremos cuál es el funcionamiento de la Web hoy en día y cuáles son los problemas que se han ido planteando a lo largo de los años, así como las limitaciones que generan a lo que, según los expertos, podría ofrecernos la Web en un futuro mediante la Web Semántica.

Se definirá qué es la Web Semántica describiendo en qué consiste este concepto y cómo propone el uso de las ontologías como parte fundamental para la gestión de la semántica en la Web. Se tratará también su arquitectura y los lenguajes necesarios para su implementación.

A continuación, dado que el fin inmediato de este proyecto es la creación del buscador Web, se tratará la forma de crear aplicaciones Web y los requisitos que necesitan cumplir para poder ser usables, objetivo heredado del proyecto SEMSE y que se ha aplicado también en este buscador. Adicionalmente, se presentan los conceptos relacionados con el desarrollo del software, el Lenguaje de Modelado Unificado (UML) y los distintos patrones que se implementarán en el diseño del buscador.

Finalmente, se incluye un apartado en el que se realiza un resumen del trabajo realizado ya en el proyecto SEMSE y en cómo se ha obtenido el conocimiento semántico con el que cuenta esta aplicación.

2.1 La Web

La Web apareció en 1989 de la mano de sus creadores, el inglés Tim Berners-Lee y el belga Robert Cailliau(Wikipedia, 2011f), mientras trabajaban en el CERN (Organización Europea de Investigación Nuclear) en Ginebra, Suiza. Fue concebida como un sistema de distribución de información basado en hipertexto[2] o hipermedios enlazados y accesibles a través de Internet. De esta forma, cualquier usuario que disponga de un navegador Web[3] y esté conectado a un servidor Web[4], ya sea local o en Internet, puede consultar los sitios Web alojados en él. Dichos sitios, a su vez, están compuestos por páginas Web capaces de mostrar texto, imágenes, videos y cualquier otro contenido multimedia, además de permitir la navegabilidad a otras páginas mediante los hiperenlaces.

[2] Hipertexto es el nombre que recibe el texto que en la pantalla de un dispositivo electrónico conduce a otro texto relacionado.
[3] Un navegador Web es un programa informático capaz de interpretar el código en el que está escrita una página Web para poder mostrársela al usuario.
[4] Un servidor Web es un sistema informático que ejecuta programas capaces de atender las peticiones de otros equipos para mostrarles las páginas Web en él contenidas.

2.1.1 Funcionamiento de la Web

Existen tres conceptos fundamentales para comprender el funcionamiento de la Web:

- HTML(World Wide Web Consortium, 1999a): HyperText Markup Language (Lenguaje de Marcado de Hipertexto). Es el lenguaje utilizado para escribir las páginas Web. Con él se define su contenido, su estructura y su aspecto, así como los enlaces a otras páginas de Hipertexto. Es interpretado por los navegadores Web de los equipos clientes o solicitantes.
- HTTP(World Wide Web Consortium, 2011): Es el protocolo TCP/IP por el que son transferidas las páginas Web. Define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web para comunicarse. Es un protocolo orientado a transacciones y que sigue el esquema petición-respuesta entre un cliente y un servidor, además no guarda información entre una petición y otra.
- URL(World Wide Web Consortium, 2006b): Es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, videos, presentaciones digitales, etc.

Por lo tanto, el proceso a grandes rasgos para poder visualizar una página web sería el siguiente:

1. Conocer la URL de la página o como se le conoce comúnmente, su dirección Web e introducirla en un navegador Web.
2. Mediante el protocolo HTTP, el navegador realiza una petición al servidor Web que devuelve el código HTML en el que está escrita la página o el recurso solicitado.
3. El navegador interpreta el código recibido y renderiza la página mediante los estándares del código HTML para, finalmente, poder mostrársela al usuario.

Este es el funcionamiento básico de la Web, permitiendo que desde cualquier ordenador conectado a Internet se pueda acceder a todas las páginas Web públicas de

todos los servidores Web del mundo. De esta manera tan sencilla se puede acceder a una enorme cantidad de información y un enorme número de servicios en continuo crecimiento. Sin embargo, desde hace unos años, se ha ido echando en falta determinada funcionalidad no disponible aún en la Web actual.

2.1.2 Problemática de la Web actual

Desde el nacimiento de la web y su publicación en 1992 por el CERN (Wikipedia, 2011e), ha experimentado un crecimiento abrumador a lo largo de estos años. Tanto es así, que según datos de *worldwidewebsize.com* (Kunder, 2010) basados en el número de páginas indexadas de los principales exploradores, se estima que a finales del año 2010, el tamaño de la web era de 13,79 billones de páginas. Dicho de otra manera, unas 2300 páginas por cada habitante del planeta.

Por esta razón se ha comenzado a hablar de sobreinformación en la Web. Existe una gran cantidad de información pero poco aprovechada, debido a que la Web actual no incorpora mecanismos que permitan el procesado automático de dicha información en un plano semántico. Es decir, no permite que las máquinas procesen información, más allá de acciones muy básicas como la búsqueda de palabras clave en el texto, esto es, en un plano sintáctico sin atender al significado de los conceptos. Esto hace que no se puedan hacer búsquedas en la Web a partir de la semántica de los términos a consultar.

Otro de los problemas a los que se enfrenta la web actual es la poca interoperabilidad existente entre los distintos sistemas informáticos. Ya sea por cuestiones técnicas o por falta de entendimiento, la Web actual no facilita la comprensión común y compartida de un dominio, dificultándose de esta forma que la información no sea utilizada por personas, organizaciones y máquinas.

2.1.2.1 Dificultad para recuperar información

Los buscadores actuales basan sus búsquedas en palabras clave, en la correlación que tienen las palabras de la búsqueda con una base de datos que tienen

los buscadores de las páginas de la Web y sus términos principales. La relevancia que le dan a los resultados, dependiendo del buscador, puede ser por el número de visitas que tiene cada página, el número de páginas que referencian a la primera, etc. Todos ellos son factores sobre los que el usuario no puede actuar, siendo muchas veces las búsquedas infructuosas, por la escasa precisión de los resultados.

Además las búsquedas son altamente sensibles al vocabulario empleado en la las mismas, si los documentos a recuperar no usan el mismo vocabulario que el empleado en la búsqueda, nunca aparecerán como resultado.

Otra carencia de la situación actual es que, con los estándares Web del momento, no se puede diferenciar entre información personal, académica, comercial, etc. Es decir, cuando un buscador Web realiza una consulta con algunas palabras clave, normalmente aparece información que no es útil porque no corresponde al ámbito de lo que se estaba buscando. Además no todas las páginas proporcionan la misma cantidad ni la misma calidad de información debido, precisamente, a que no existe un formato o convenio que regule el contenido que deben tener las páginas Web.

Por otro lado, los agentes de búsqueda actuales no se diseñan para "*comprender*" la información que reside en la Web, precisamente porque es prácticamente imposible conocer la representación de los datos ubicados en las diferentes páginas. Por lo que los resultados devueltos por un buscador hoy en día no van más allá de la coincidencia sintáctica de los conceptos solicitados sin tener en cuenta su significado.

2.1.2.2 Falta de interoperabilidad

Se pueden distinguir tres tipos diferentes de interoperabilidad (Paredes, 2010):

- **Interoperabilidad técnica**: Se refiere la capacidad de los Sistemas de Información (en adelante, SI) para intercambiar señales, lo que implica estar conectados físicamente, aun teniendo cada uno tecnologías distintas.

- **Interoperabilidad sintáctica**: Es la capacidad de un SI de leer datos procedentes de otro SI. Significa que todos ellos utilizan un sistema compatible de datos, lo que no implica que comprendan su significado.
- **Interoperabilidad semántica**: Es la capacidad de los SI para intercambiar información basándose en un significado común de los términos y expresiones que usan, es decir, consistente en el significado que tiene la información.

La falta de interoperabilidad en la Web actual provoca una dificultad de comprensión común entre los distintos SI. Desde el punto de vista de las empresas esto es crítico, ya que la comunicación con los clientes, proveedores, fabricantes, etc. es tediosa y con un coste económico muy elevado.

La forma de integrar los procesos de negocio y los SI de las organizaciones que forman parte de una misma cadena de aprovisionamiento se basa en usar EDI (Electronic Data Interchange). Consiste en un sistema de intercambio de documentos comerciales entre empresas, basado en la utilización de estándares aceptados por todas las partes y que permite realizar transacciones informáticas automáticas sin apenas intervención humana. Sin embargo, el enfoque EDI convencional presenta dos problemas fundamentales: por un lado, el elevado coste que supone crear aplicaciones traductoras hechas a medida y por otro lado, el enfoque adolece de flexibilidad.

2.2 La Web Semántica

Según (Lozano Tello, 2001), la Web es un espacio preparado para el intercambio de información diseñado para el consumo humano. Las páginas Web son creadas por personas para ser entendidas por personas. No existe un formato común para mostrar la información, por lo cual, los desarrolladores de páginas Web crean sus páginas dependiendo de los potenciales usuarios que van a visitarlas.

Ya hemos visto en el apartado anterior las carencias que presentan los actuales buscadores Web, y como, mediante la búsqueda únicamente de palabras

clave, no consiguen mostrar toda la información que se podría encontrar o mostrar sólo aquella que coincida realmente con nuestra búsqueda.

En esta sección se mostrará como la Web Semántica puede solucionar los problemas indicados y se verá qué ventajas puede aportar a la situación actual. A continuación se explicará el concepto de ontología, pieza fundamental para soportar la representación del conocimiento que necesita la Web Semántica.

2.2.1 Descripción de la Web Semántica

La Web Semántica es una extensión de la Web actual en la que se proporciona la información con un significado bien definido, mejorando la forma en la que las máquinas y las personas trabajan en cooperación (Berners-Lee, et al., 2001). La Web Semántica propone superar las limitaciones de la Web actual mediante la introducción de descripciones explícitas del significado, la estructura interna y la estructura global de los contenidos y servicios disponibles en la WWW (World Wide Web).

Tiene como misión la búsqueda de una Web más inteligente en la que se pueda conseguir una comunicación efectiva entre ordenadores y centra sus esfuerzos principalmente en la búsqueda de descripciones enriquecidas semánticamente para los datos en la Web. En este sentido, se promueven descripciones que incluyan no solo las estructuras de datos, sino también las relaciones existentes con otros conceptos, las restricciones, reglas que permitan realizar inferencia, etc.

Así mismo, se promueve la definición y reutilización de vocabularios u ontologías de conceptos que faciliten el procesamiento por parte de las máquinas. Tal y como aparece reflejado en (Malik, 2003), está previsto que la Web Semántica sea un lugar donde los datos puedan ser compartidos y procesados tanto por herramientas de manera automatizada como por personas. La clave subyace en la automatización y la integración de los procesos a través de lenguajes legibles por máquinas.

Para poder usar y enlazar la gran cantidad de información disponible en la Web, los agentes software deben de ser capaces de comprender la información, es decir, los datos deben de estar escritos haciendo uso de una semántica legible y

entendible por las máquinas. Por tanto, en las páginas Web actuales debe añadirse una capa semántica adicional, para que los programas software puedan establecer el significado de las etiquetas de dichos documentos.

Tim Berners Lee, en el artículo (Berners-Lee, et al., 2001) menciona cuatro componentes o características básicas necesarias para la evolución de la Web Semántica. Estos componentes son:

- **Expresar significado:** La Web Semántica debe brindar una estructura y añadir semántica al contenido de las páginas Web, creando un entorno donde agentes software puedan viajar de una página a otra llevando a cabo sofisticadas tareas para los usuarios.
- **Acceso a representaciones del conocimiento:** La Web Semántica debe encargarse de resolver las limitaciones de los sistemas de representación de conocimiento tradicionales creando lenguajes de reglas suficientemente expresivos como para permitir a la Web razonar tan ampliamente como se desee.
- **Ontologías:** Para conseguir que los computadores sean mucho más útiles, la Web Semántica extiende la Web actual con conocimiento formalizado y datos que son procesados por ordenadores. Para ser capaz de buscar y procesar información relativa a alguna materia de interés, los programas necesitan información que haya sido modelada de una forma coherente. Una ontología modela todas las entidades y relaciones en un dominio. La ontología es necesaria para la representación del conocimiento. La clave de las ontologías es que pueden ser compartidas y por lo tanto incrementan en eficiencia e interoperabilidad. Sin embargo, se puede dar el caso en el que dos organizaciones distintas usen dos nombres diferentes para identificar el mismo concepto, es decir, las ontologías sean distintas. En tales casos, la habilidad para asociar los términos de una y otra (mapping o mapeado) es crucial para mantener las ventajas de la Web Semántica.
- **Agentes** La potencia real de la Web Semántica se conoce cuando agentes capaces de manejar contenido semántico son usados para recoger y procesar

información Web e intercambiar los resultados con otros agentes. Herramientas como el intercambio de pruebas o la firma digital asegurarán que los resultados intercambiados entre agentes sean válidos y se pueda confiar en ellos.

2.2.2 Arquitectura

Para lograr una Web Semántica basada en metadatos procesables por máquinas, Berners-Lee propone un conjunto de capas que definen distintos niveles de representación semántica. De este modo, se progresa en la representación desde el nivel inferior, meramente sintáctico, hasta el nivel superior, donde es posible discernir si la información es fiable. La estructura que muestra la *Figura II-1. Arquitectura de la Web Semántica*, obtenido de es la versión actual de la que fue presentada por Tim Berners-Lee en una conferencia sobre el *Extensible Markup Language* (XML) en el año 2000.

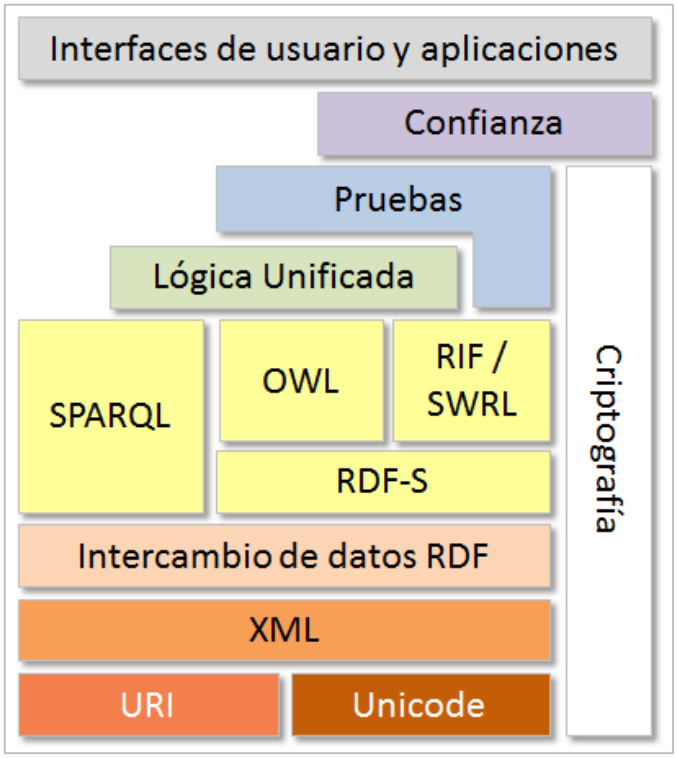

Figura II-1. Arquitectura de la Web Semántica, obtenido de (Obikto, 2007)

La estructura propuesta establece el siguiente conjunto de capas, en función de su capacidad de representación semántica, donde cada una de ellas presta un servicio a los niveles superiores.

2.2.2.1 Capa sintáctica

Se encuentra situada en el nivel inferior de la pirámide y tiene como objetivo facilitar el intercambio de datos a través de una sintaxis de serialización concreta. En el nivel superior de la capa, suele situarse XML por ser el lenguaje de intercambio de datos más utilizado por las capas superiores, no obstante, no es obligatorio y podrían utilizarse otros lenguajes. XML y XML Schema proporcionan la sintaxis necesaria para el intercambio y persistencia de la información. En el apartado *2.2.3.1 XML (Extensible Markup Languge) y XML Schema* se describen las principales características de este lenguaje.

El nivel XML hace uso del nivel inferior, donde se sitúan *Unicode* y los *URIs*. Unicode (The Unicode Consortium, 2010) es una codificación de caracteres internacionales donde cada carácter recibe un identificador unívoco, de este modo, las páginas XML pueden representarse sin importar el idioma, plataforma o programa que las gestione. Dentro de este nivel también se incluyen los Identificadores Uniformes de Recursos (*Universal Resource Identifier: URI*). Los URIs (Berners-Lee, 1994b) permiten la identificación unívoca de recursos dentro de la Web Semántica. En el caso de una página Web, el URI puede ser el URL (*Uniform Resource Locator*) de la página. En general, un URI no tiene por qué permitir el acceso a un recurso a través de Web, sino que su propósito es identificar de forma no ambigua el recurso.

2.2.2.2 Capa RDF(S)

La siguiente capa dentro de la pirámide tiene como objetivo proporcionar la suficiente flexibilidad para representar conceptos y reglas lógicas relacionadas, generando así valor añadido. El Marco de Descripción de Recursos (*Resource Description Framework: RDF*) (World Wide Web Consortium, 1999b)es una recomendación del W3C que define un lenguaje simple para expresar relaciones en

forma de terna[5] Recurso-Propiedad-Valor. La especificación de *RDF-Schema* (World Wide Web Consortium, 2004a) es también una recomendación del W3C cuyo objetivo es facilitar la interpretación semántica de RDF. De este modo, permite la representación de modelos semánticos, orientados a objetos, en forma de redes de conceptos. Entre las primitivas[6] que incluye, se encuentran aquellas que permiten la definición de: Clases, propiedades, subpropiedades, restricciones de dominio y rango, etiquetas y comentarios. En el apartado *2.2.3.2 RDF (Resource Definition Framework) y RDF Schema* se describen las principales características de ambas recomendaciones.

2.2.2.3 Capa ontológica

La siguiente capa dentro de la estructura tiene como objetivo proporcionar mayor expresividad que la expresada por la capa inferior. Concretamente, es necesario poder representar primitivas complejas tales como: propiedades transitivas, restricciones de cardinalidad en las relaciones, etc. Existen distintos lenguajes que proporcionan la expresividad semántica necesaria en esta capa, entre otros: OIL(Lamarca Lapuente, 2002), DAML+OIL (Defense Advance Research Projects Agency, 2002) y OWL(Bechhofer, et al., 2004). Este último lenguaje OWL Web Ontology Language es una recomendación del W3C desarrollada para facilitar el procesamiento automático de contenido Web. Posee tres versiones distintas que van de la menos expresiva, OWL Lite, pasando por una intermedia y decidible, OWL DL (*DL: Description Logic*), hasta llegar a la versión de máxima expresividad semántica OWL Full. En el apartado *2.2.3.3 OWL (Web Ontology Language)* se incluyen las principales características de este lenguaje.

[5] Se ha utilizado el término terna como traducción del término inglés triple. En la literatura se pueden encontrar traducciones como tripleta o tripla, también utilizadas con frecuencia. Según el diccionario de la RAE, tanto terna como tripleta tienen un significado similar a trío, como conjunto de tres cosas. Tripla, por el contrario, tiene un significado próximo a triple, como conjunto de tres cosas iguales. Se ha seleccionado el término terna, aunque podría haber sido también tripleta, por aproximarse mejor a la semántica del concepto. Del mismo modo, no se aconseja el uso del término tripla.

[6] Palabra que representa una función interpretable por computadoras, definida en un lenguaje de programación o representación de conocimiento.

2.2.2.4 Capa lógica

La capa lógica tiene como objetivo permitir la inferencia necesaria para proporcionar explicaciones a las respuestas obtenidas. Para ello deben definirse las reglas lógicas que permitan inferir nuevo conocimiento mediante un proceso de razonamiento. Para poder comprobar que los resultados son correctos es preciso traducir el razonamiento interno a un lenguaje de representación de pruebas (*proof*).

2.2.2.5 Capa de prueba y confianza

La capa más alta de la pirámide tiene como objetivo proporcionar los mecanismos para comprobar que la información recibida es coherente desde un punto de vista lógico. Apoyándose en el conjunto de reglas de la capa inferior, será capaz de mostrar dicho razonamiento de forma inteligible para los humanos de forma que pueda ser validable y, por tanto, confiable. Junto con la necesidad de validar la información desde el punto de vista lógico, existe la necesidad de validarla desde el punto de vista de la fiabilidad. De este modo, para confiar en la información debemos poder comprobar que es válida lógicamente y que es fiable, en cuanto a que: proviene de la fuente correcta, no ha sido alterada, no puede ser repudiada por su emisor, etc. La Firma Electrónica (*Digital Signature*) incluida en la estructura de la pirámide, incorpora los mecanismos de seguridad que garantizan la fiabilidad de la información. De este modo, el objetivo final es lograr una Web de Confianza (*Web of Trust*) donde la información intercambiada sea confiable a través de su certificación.

El W3C es el organismo encargado de dirigir los procesos de estandarización de los lenguajes para la Web Semántica. A través de los distintos grupos de trabajo, especializados en distintas áreas de investigación, desarrolla recomendaciones en conjunción con entidades académicas, empresas y particulares. Actualmente existen recomendaciones desde la capa sintáctica a la ontológica inclusive. Actualmente se está trabajando en las capas más altas de la pirámide: lógica, prueba y confianza.

2.2.3 Lenguajes

2.2.3.1 XML (Extensible Markup Languge) y XML Schema

XML es el estándar del W3C (World Wide Web Consortium, 2008) predominante para codificación e intercambio de datos entre aplicaciones. En XML los datos están agrupados en elementos delimitados por etiquetas. Estos elementos pueden estar anidados y contener atributos. De este modo, XML permite incluir *metadatos* (datos que describen datos) dentro de los documentos, de modo que la información incluida pueda ser procesada de forma automática.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ficha>
<nombre> Miguel </nombre>
<apellido> Zazo </apellido>
<direccion> Génova Nº38 </direccion>
</ficha>
```

Ejemplo II-1. XML para representar la ficha de una persona

En este ejemplo de XML básico podemos ver como se ha declarado una ficha que define a una persona. A su vez, la ficha contiene tres atributos: nombre, apellido y dirección, a los cuales se le ha dado el valor “Miguel”, “Zazo” y “Génova Nº38” respectivamente.

La estructura de elementos, subelementos y atributos, así como la cardinalidad viene frecuentemente expresada en un documento denominado DTD (*Document Type Definition*) mediante una gramática formalizada. Las DTD, aunque fáciles de interpretar y muy utilizadas, tienen una serie de desventajas como son el no estar escritas con sintaxis XML o la escasa definición de tipos de datos, por este motivo se definió el Esquema XML (*XML Schema*)(World Wide Web Consortium, 2004d).

Los esquemas son documentos escritos en XML haciendo uso de espacios de nombres (*namespaces*) asociados mediante su URI y, que como los DTDs, expresan la estructura de los documentos XML. De forma similar a los DTD pero con mayor capacidad de representación sintáctica, permiten definir qué elementos puede contener un documento XML, cómo están estructurados, qué atributos y de qué tipo

pueden tener esos elementos. Por lo anterior podemos decir que XMLS es un *metalenguaje* dado que permite establecer la estructura y los elementos para generar documentos XML, es decir, permite definir nuevos lenguajes XML para su uso en dominios concretos, de forma flexible y extensible. Esquemas con elementos definidos en un alto nivel de abstracción, como OWL, expresan un marco común para la incorporación de ontologías(Mihalceay Mihalcea, 2001).

Según el W3C(World Wide Web Consortium, 2009a) los espacios de nombres, mediante la inclusión de un prefijo asociado al URI, constituyen un método sencillo para cualificar elementos y atributos en XML, permitiendo así su posterior reutilización. Los espacios de nombres se crearon para resolver conflictos en las denominaciones de elementos procedentes de distintas fuentes. En los documentos XML, para hacer referencia a un término del espacio de nombres se utiliza el Nombre Cualificado (*Qualified Name: QName*). El QNname está formado por un prefijo seguido de dos puntos y el nombre del término. En la cabecera del documento, el prefijo estará declarado como un espacio de nombres junto con su URL. El conjunto de espacios de nombres no está acotado, ventaja que suele provocar la utilización en los documentos de forma no homogénea.

Las principales ventajas que proporciona XML pueden resumirse en:

- Proporciona la estructura sintáctica necesaria para que la información contenida en documentos Web pueda ser procesada tanto por humanos como por máquinas.
- Permite separar la información de un documento de la presentación de la misma, a diferencia de otros lenguajes de marcado, por ejemplo HTML (*Hipertext Markup Language*).
- Establece una representación estándar para información proveniente de distintas fuentes, lo que facilita la intercomunicación entre aplicaciones.

Por otro lado, la principal limitación que se encuentra en el lenguaje XML es la imposibilidad de añadir información semántica a su contenido, más allá de la definición de su estructura y de sus elementos permitidos. Aunque permite la

definición de elementos como *Nombre* o *Dirección* esto es sólo semánticamente comprensible para el usuario humano, pero desde el punto de vista de una máquina, si se hubieran nombrado por ejemplo como *Elemento1* y *Elemento2*, el resultado hubiera sido el mismo. Para el usuario humano ambas etiquetas tienen significado, aportado por la interpretación personal de la etiqueta.

Sin embargo, desde el punto de vista de la Web semántica esto no supone un problema, ya que si observamos en qué nivel se encuentra XML en su estructura, veremos que de él sólo se espera la aportación de sintáctica a la información. La especificación semántica está delegada en capas superiores, por lo que, como veremos más adelante, son otros lenguajes los encargados de definir aspectos semánticos a los documentos.

Para finalizar esta breve introducción a XML es preciso comentar otras tecnologías relacionadas con XML, a saber:

- XLink 1.0(World Wide Web Consortium, 2001): Especificación que define elementos insertables en documentos XML para crear o describir enlaces entre recursos.
- XHTML 1.0(World Wide Web Consortium, 2002): Especificación en la que se reformula HTML 4 como aplicación XML 1.0. Proporciona compatibilidad con HTML y las bases para futuras ampliaciones de XHTML.
- DOM 1.0(World Wide Web Consortium, 1998): Especificación del modelo de objetos de un documento (*Document Object Model*) consistente en una plataforma y un interfaz independiente de lenguajes que permite el acceso y manipulación del contenido, estructura y estilo de los documentos XML.
- CSS 2(World Wide Web Consortium, 2010a): Primera revisión de la especificación de hojas de estilo en cascada (*Cascading Style Sheets*), nivel 2. Compatible con CSS nivel 1, permite la especificación de estilos a documentos estructurados (HTML, XML, etc.).
- RFC 1630(Berners-Lee, 1994a): Especificación de los Identificadores Universales de Recursos (*Universal Resource Identifier: URI*).

- XML Namespaces(World Wide Web Consortium, 2009a): Especificación que proporciona un método de cualificación de los nombres utilizados en documentos XML, mediante la asociación de cada espacio de nombres con un URI.
- XML Infoset(World Wide Web Consortium, 2004c): Especificación que proporciona un conjunto abstracto de datos para referenciar información dentro de un documento XML.
- XSL 1.1(World Wide Web Consortium, 2006a): Lenguaje para la especificación de hojas de estilo para documentos XML.
- XSLT 1.0(World Wide Web Consortium, 1999d): Especificación de la sintaxis y semántica del lenguaje de transformación de documentos XML en otros documentos XML. Forma parte de XSL e incluye etiquetas para especificar el formato de documentos XML.
- XPointer(World Wide Web Consortium, 2003): Especificación que define el lenguaje a utilizar como base para localizar un fragmento de información identificado por un URI.
- XPath 1.0(World Wide Web Consortium, 1999c): Lenguaje para acceder a partes de un documento XML. Utilizado por XSLT y XPointer.

2.2.3.2 RDF (Resource Definition Framework) y RDF Schema

El Marco de Descripción de Recursos (*Resource Description Framework: RDF*) (World Wide Web Consortium, 1999b)es el modelo de datos para la representación de metadatos en la Web Semántica. El modelo de datos se soporta sobre tres conceptos fundamentales:

- **Recurso (*Resource*):** Un recurso es *algo* sobre lo que podemos hacer una *sentencia*. Un recurso puede ser cualquier entidad: una persona, un libro, un lugar, etc. Todo recurso está identificado por un URI que lo identifica de forma unívoca aunque que no tiene por qué proporcionar acceso vía Web al recurso. En el caso de una página Web el URI puede ser el URL de dicha página.

- **Propiedad (*Property*):** Una propiedad define un aspecto, característica, atributo o relación de un recurso. Las propiedades poseen un significado y pueden tener relaciones con otras propiedades. La definición de la propiedad incluye los valores que puede tomar así como los tipos de recursos a los que puede asociarse.
- **Sentencia (*Statement*):** Una sentencia es una terna formada por un sujeto, predicado y objeto. El sujeto es el recurso a describir, el predicado es la propiedad que define una característica del objeto y el objeto es el valor que toma la propiedad para el recurso que define. Además, una propiedad es un recurso y el valor de la sentencia puede ser tanto un valor simple, como otro recurso, por ejemplo especificado mediante un URI.

Las sentencias son la estructura fundamental en la que se basa el modelo de datos RDF. Éstas pueden ser expresadas de forma textual, típicamente mediante ternas recurso-propiedad-valor (ver *Ejemplo II-2*), o de forma gráfica, mediante un grafo dirigido. En la representación gráfica los recursos y los valores se representan mediante nodos, mientras que las propiedades se representan mediante arcos dirigidos desde el recurso al valor correspondiente (ver *Figura II-2*).

```
<?xml version="1.0"?>
<rdf:RDFxmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:exterms="http://www.example.org/terms/">
<rdf:Description rdf:about="http://www.example.org/index.html">
<exterms:creation-date>August 16, 1999</exterms:creation-date>
</rdf:Description>
</rdf:RDF>
```

Ejemplo II-2. Representación textual de una sentencia RDF.

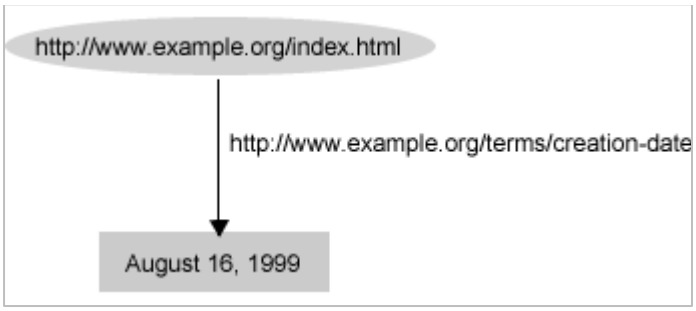

Figura II-2. Representación gráfica de una sentencia RDF.

Tanto en el ejemplo como en la figura, se muestra la representación de la sentencia que especifica, para el recurso "http://www.example.org/index.html", la propiedad fecha de creación (*creation-date*) y le asigna el valor "August 16, 1999". En el caso de la representación textual se ha incluido un ejemplo de RDF serializado con XML. Como se puede observar, y con el fin de facilitar su interpretación, en la representación gráfica suele representarse con un óvalo los recursos y con un cuadrado los valores.

La serialización de RDF con XML, consiste en la posibilidad de expresar documentos RDF mediante XML. En gran medida, esto ha contribuido al éxito en cuanto a uso y difusión de RDF en la Web.

No obstante, a diferencia de los documentos XML, los documentos RDF no son serializados por un esquema XML, es decir, no son restringidos por un esquema, sino que conforman un grafo donde los elementos que pueden representarse, el vocabulario, son definidos en un esquema representado mediante el lenguaje Esquema RDF (*RDF Schema: RDFS*)(World Wide Web Consortium, 2004b).

RDFS proporciona un conjunto de tipos básicos y un método para definir nuevos tipos, asociados a un espacio de nombres, que permiten a los usuarios definir y compartir sus propios esquemas.

Un esquema RDF define los términos que puede utilizar un documento RDF. Además, para cada término, define un significado específico. Para facilitar la reutilización de términos, RDFS hace uso de los Espacios de Nombres, pudiendo utilizarse términos de distintos esquemas de forma no ambigua. En general, una descripción de un recurso puede incluir sentencias con propiedades de distintos esquemas. El esquema RDFS puede contener también restricciones sobre los tipos de Recursos y Valores dentro de las sentencias.

Las principales primitivas de RDFS pueden agruparse en Clases, Propiedades y Restricciones, a continuación se incluye una breve descripción de cada una de ellas.

Clases

RDFS permite la definición Recursos, Clases y Propiedades. El Recurso es una clase genérica de la cual heredan todos los objetos descritos en un documento RDF. Las Clases y Propiedades son similares a las definidas en la orientación a objetos aunque, principalmente las últimas, no son definidas como integrantes de la clase sino que se definen en función del Recurso y el Valor que asocian. De este modo, algo impensable en orientación a objetos, como es el que un atributo pueda no pertenecer a una clase (o pertenecer a más de una), en RDF(S) sí puede darse. Para este propósito, las principales primitivas que ofrece RDFS son:

- ***rdfs:Resource*:** Representa la clase genérica en el modelo RDF Schema. Todo objeto descrito por expresiones RDF es un recurso.
- ***rdfs:Class*:** Es subclase de *rdfs:Resource* y representa el concepto genérico de tipo o categoría, similar a la noción de clase en orientación a objetos.
- ***rdf:Property*:** Es subclase de *rdfs:Resource* y representa un aspecto del recurso que se está describiendo, similar a la noción de atributo en orientación a objetos.

Propiedades

Las propiedades posibilitan expresar relaciones entre instancias y sus clases, o clases y sus superclases. Las relaciones entre propiedades también son permitidas, obteniéndose así una jerarquía de propiedades. Destacar que además es posible la implementación de herencia múltiple, lo que proporciona gran flexibilidad al modelo RDF, al permitirse que una clase pueda heredar propiedades de varias clases *padre.*

La herencia facilita, en gran medida, la reutilización y especialización de los términos definidos en otros esquemas. Las principales propiedades disponibles en RDFS son:

- ***rdf:type*:** Es una subclase de *rdf:Prorperty* y denota que un recurso es instancia de una clase, poseyendo todas sus características. Un recurso puede ser instancia de más de una clase.

- ***rdfs:subClassOf*:** Es una subclase de *rdf:Property* y denota la relación de subclase/superclase entre dos clases. Esta propiedad es la base para la especificación de la herencia múltiple, y tiene característica de transitividad.
- ***rdfs:subPropertyOf*:** Es una subclase de *rdf:Property* y denota la relación de especialización entre dos propiedades, posibilitando la definición de una jerarquía de propiedades.

Restricciones

Mediante estas primitivas es posible restringir las propiedades de un recurso, concretamente, estableciendo cuál es el recurso al que se puede aplicar la propiedad y sobre qué rango de datos puede tomar valores dicha propiedad. Las principales primitivas de restricciones son:

- ***rdfs:domain*:** Es una instancia de la clase *rdfs:ConstraintProperty* y específica a qué recurso se aplica una propiedad.
- ***rdfs:range*:** Es una instancia de la clase *rdfs:ConstraintProperty* y restringe los valores (u objetos) que una propiedad puede asumir.

Tal y como se ha descrito, RDF(S) tiene capacidad para representar, en resumen, predicados binarios, jerarquías de clases y propiedades, así como restricciones de dominio y de rango para propiedades. No obstante, la expresividad de RDF(S) está limitada a la hora de modelar ontologías, dado que carece de capacidad para representar axiomas, propiedades de propiedades (por ejemplo, transitividad) o condiciones de pertenencia a una clase. Para poder representar los elementos necesarios para la representación de ontologías es necesario utilizar lenguajes con mayor expresividad. Entre ellos el recomendado por el W3C es OWL, descrito en el siguiente apartado.

2.2.3.3 OWL (Web Ontology Language)

El Lenguaje de Ontologías Web OWL (*OWL Web Ontology Language*) es una especificación del W3C (Bechhofer, et al., 2004) desarrollada para la definición de

ontologías en la Web. OWL está especificado sobre RDF(S) y, por tanto, puede ser representado como un documento RDF(S)

Para poder representar ontologías, OWL incorpora un conjunto de primitivas que proporcionan mayor expresividad que RDF(S). Las principales mejoras de OWL frente a RDF pueden resumirse en:

- Ámbito local de propiedades: Permite definir una propiedad con un ámbito local a una clase, a diferencia de RDF(S) donde el ámbito es global.
- Clases relacionadas: Permite definir clases donde la pertenencia, *instancia*, a una de ellas implica una relación con respecto a otra/s, por ejemplo: *igualdad de clases, clases disjuntas* o *clases complemento*.
- Combinación booleana de clases: Permite definir clases como la unión, intersección o complemento de otras clases. El conjunto de instancias de la clase agregada será el resultado de la aplicación de la regla al conjunto de instancias de cada una de las clases combinadas.
- Restricciones de cardinalidad: Posee mayor capacidad para expresar restricciones de cardinalidad en los valores de una propiedad, por ejemplo: *al menos uno* o *exactamente dos*.
- Características de propiedades: Proporciona un mayor conjunto de tipos de propiedades, así como características especiales, por ejemplo: *transitividad*, *unicidad* o *inversibilidad*.
- Clases enumeradas: Permite la definición de clases como la enumeración de los individuos que la componen.

Un lenguaje para expresar ontologías debe atender a dos objetivos fundamentales: máxima expresividad y soporte a razonamiento. El primero proporciona la capacidad para representar axiomas, conceptos y relaciones complejas; el segundo la capacidad de razonamiento eficiente, es decir, poder desarrollar un

razonamiento completo y decidible[7]. Como puede intuirse, la conjunción de ambos objetivos es difícil lograr, por este motivo, el grupo de trabajo del W3C decidió definir tres sublenguajes[8] de OWL con distintas capacidades de representación.

- **OWL Lite:** Es el sublenguaje con menor capacidad expresiva. Básicamente está pensado parar usuarios que necesiten jerarquías de clasificación y restricciones simples. Es una buena opción para la representación de tesauros y taxonomías.
- **OWL DL:** Es el sublenguaje con capacidad expresiva intermedia. Proporciona la mayor capacidad expresiva garantizando que el razonamiento es computacionalmente completo y decidible. Básicamente, OWL DL posee todas las primitivas que OWL Full aunque con algunas restricciones en la combinación con RDF(S), básicamente, los constructores de OWL y RDF(S) no pueden aplicarse entre ellos, por ejemplo: no se permite que una clase sea instancia de otra. El acrónimo *DL* de OWL DL, proviene de su correspondencia con la Lógica Descriptiva (*Description Logic*), lógica en la que se basan los fundamentos de OWL.
- **OWL Full:** Es el sublenguaje con mayor capacidad expresiva. Está diseñado para usuarios que precisen la máxima expresividad y la libertad sintáctica de RDF, a cambio, no se garantiza que el razonamiento sea completo ni decidible. Básicamente, elimina las restricciones de OWL DL, permitiendo cualquier combinación de las primitivas OWL con RDF(S).

[7] El razonamiento completo garantiza que todas las conclusiones serán computables. El razonamiento decidible garantiza que el razonamiento finalizará en tiempo finito. No obstante, no se garantiza que el razonamiento sea decidible a tiempo. En función del dominio del problema, es posible que el tiempo necesario para llevar a cabo un razonamiento exceda del tiempo en el que los resultados son válidos. Por ejemplo, si llevar a cabo una predicción del tiempo para la próxima semana implica un tiempo de procesamiento de un mes, evidentemente el razonamiento no es válido.

[8] También llamados *especies*, del término anglosajón *species*.

Dentro de la estructura de la Web Semántica, el lenguaje OWL es el estándar propuesto para implementar ontologías en la Web y especificar axiomas de la capa lógica, puesto que permite la descripción semántica del conocimiento y proporciona soporte a razonamiento. Cabe destacar que, desde octubre de 2009 existe una nueva versión de OWL, denominada OWL 2(World Wide Web Consortium, 2009b). En esta nueva versión se revisan algunas carencias del lenguaje como limitaciones en la expresividad, problemas sintácticos o problemas semánticos(Bernardo Cuenca, et al., 2008), detectadas en la aplicación del lenguaje al modelado de distintos dominios del conocimiento. En la presente propuesta se ha preferido el uso de OWL 1, dado que el soporte a OWL 2 aún no está lo suficientemente extendido, debido a su relativa novedad.

2.2.4 Ontologías

El término ontología proviene de la filosofía, una ontología es una teoría que trata de la naturaleza y organización de la realidad, es decir, de lo que "*existe*". Conocimiento del ser (del griego onto: ser y logos: conocimiento).

Platón trató con la cuestión de dar un apropiado nombre a las cosas, en su opinión esto era de suma importancia para que cualquiera pudiera unívocamente identificarlas. Aristóteles más allá de la cuestión de los nombres, se interesó por las definiciones. Una definición significaba explicar claramente lo que una cosa era mediante la existencia de una declaración esencial de la entidad. Por lo tanto, creyó que para decir lo que algo era, siempre requería decir por qué era ese algo. Aristóteles ignoró las limitaciones inevitables de comunicar el significado vía un lenguaje y las ambigüedades creadas por el cambio implícito de los sentidos diferentes de significado(Maedche, 2002).

Gottlob Frege introdujo una distinción entre dos tipos de significado: el concepto y el referente. La interpretación gráfica de esta distinción es comúnmente referida como el "*triángulo del significado*" (meaning triangle) y fue introducida por Ogden y Richards en 1923(Ogdeny Richards, 1923). "*El triángulo del significado*", que

se muestra en la Figura II-3, define la interacción entre símbolos o palabras, pensamientos y cosas del mundo real.

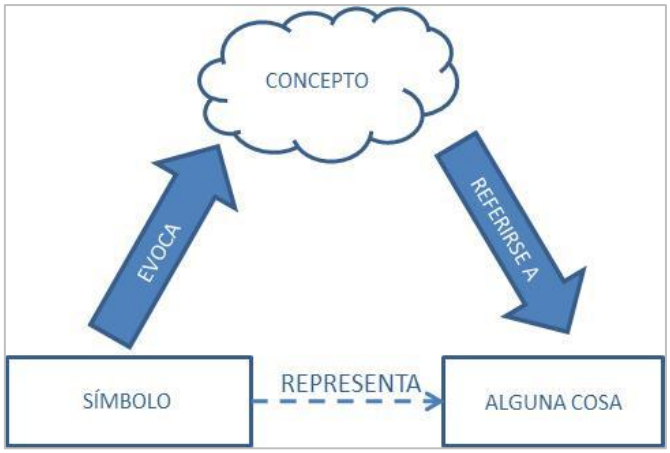

Figura II-3. Triángulo del Significado, obtenido de(Ogdeny Richards, 1923).

El diagrama ilustra que aunque los símbolos no pueden completamente capturar la esencia de una referencia (o concepto) o de un referente (o cosa), hay una correspondencia entre ellas. La relación entre una palabra y una cosa es indirecta. El enlace puede solamente ser completado, cuando un intérprete previamente procesa la palabra, la cual invoca un correspondiente concepto, para posteriormente enlazar este concepto a una cosa en el mundo. Hay una cierta correspondencia entre este triángulo de significado y el aspecto que cubren cada uno de sus elementos. Mientras que la representación de los símbolos puede ser llevada a cabo mediante diferentes modelos, la construcción de las ontologías se convierte en esencial para establecer los otros dos vértices del triángulo y sus relaciones, gracias a su fuerte semántica lógico-conceptual.

Posteriormente los investigadores de Inteligencia Artificial incorporaron el término de ontología a su jerga. En el campo de la Inteligencia Artificial "*lo que existe es aquello que puede ser representado*".

Una de las primeras definiciones en el área de la ciencia de la información la hizo Neches (Neches, et al., 1991) y su equipo de trabajo: *"una ontología define los términos básicos y relaciones incluyendo el vocabulario de un área, así como las reglas para la combinación de términos y relaciones para definir ampliaciones de un*

vocabulario" Se puede decir que esta definición aporta unas líneas a seguir para crear una ontología: identificar términos básicos y relaciones entre los términos, identificar reglas para combinar las relaciones, aportar definiciones de los términos y las relaciones. Así que según esta definición, una ontología no incluye solo los términos que son explícitamente definidos en ella, sino que también los términos que pueden ser deducidos usando reglas.

En 1993, Thomas R. Gruber (Gruber, 1993) dio una de las definiciones más empleadas: "las ontologías se definen como una especificación explícita de una conceptualización"

En 1998, Rudi Studer y sus compañeros (Studer, et al., 1998) modificaron ligeramente la definición de Gruber diciendo que: "las ontologías se definen como una especificación explícita y formal de una conceptualización compartida" Donde:

- **Conceptualización** se refiere a una representación abstracta o modelo, de algún fenómeno en el mundo, perteneciente al Universo del Discurso. En dicho modelo estarán representados los conceptos y relaciones relevantes de dicho fenómeno.
- **Explícita** se refiere a definición explícita que, para su uso, es necesario hacer de los conceptos, relaciones y restricciones.
- **Formal** se refiere al hecho de emplear un formalismo de representación, que permita a la ontología ser legible o interpretable por una computadora.
- **Compartida** expresa la noción de conocimiento consensuado, es decir, el conocimiento compartido no es privado de un individuo, sino que ha sido consensuado por un grupo o comunidad.

Nicola Guarino (Guarino, 1998) complementó la propuesta de Studer diciendo que: "*una ontología es una teoría lógica que da cuenta del significado intencional de un vocabulario formal, es decir, de su compromiso ontológico hacia una conceptualización particular del mundo"*

Por tanto, Guarino define ontología como teoría lógica que aporta la semántica a un conjunto de términos del Universo del Discurso, según la

interpretación que su creador haga de la realidad. De este modo, es posible tener distintas representaciones de una misma realidad, dado que éstas dependen de la visión que el desarrollador tenga de esa realidad.

Para solucionar esto Robert Jasper y Mike Uschold (Uscholdy Jasper, 1999) proponen que: *"una ontología puede tomar una gran variedad de formas, pero necesariamente incluirá un vocabulario de términos, y alguna especificación de su significado. Esto incluye definiciones y una indicación de cómo los conceptos se interrelacionan lo que colectivamente impone una estructura sobre el dominio y restringe las posibles interpretaciones de los términos".*

Con las definiciones dadas, se puede ver que una ontología puede ser, una teoría lógica, una descripción formal de semántica, el vocabulario de una teoría lógica y una especificación de una conceptualización.

En informática, una ontología es un vocabulario controlado que describe de manera formal objetos y las relaciones entre ellos, y que tiene una gramática para usar los términos del vocabulario con el fin de expresar algo con significado dentro de un dominio de interés específico, dicho vocabulario se utiliza para hacer búsquedas y aserciones.

Una ontología formal es un vocabulario controlado que se expresa en un lenguaje de representación de ontologías. En lo que se refiere al ámbito del proyecto únicamente se consideran las ontologías formales, las cuáles poseen semánticas formales, es decir, que describen el significado del conocimiento de una forma precisa. Es indispensable disponer de una semántica formal para implementar sistemas de inferencia o de razonamiento automático. Las utilidades del razonamiento automático son varias:

- Comprobar automáticamente si una ontología es consistente con el conocimiento del dominio de interés del cual está asociada.
- Comprobar automáticamente si las relaciones entre las clases corresponden a los propósitos de la ontología, pudiendo detectar relaciones espúreas.
- Clasificar automáticamente las instancias en clases.

Las ontologías son una herramienta para compartir información y conocimiento, es decir, para conseguir la interoperabilidad y la Web Semántica. En el ámbito de la Web, las ontologías suelen ser representadas en formatos XML, los cuales carecen de una semántica que permita el razonamiento automático a diferencia de las ontologías.

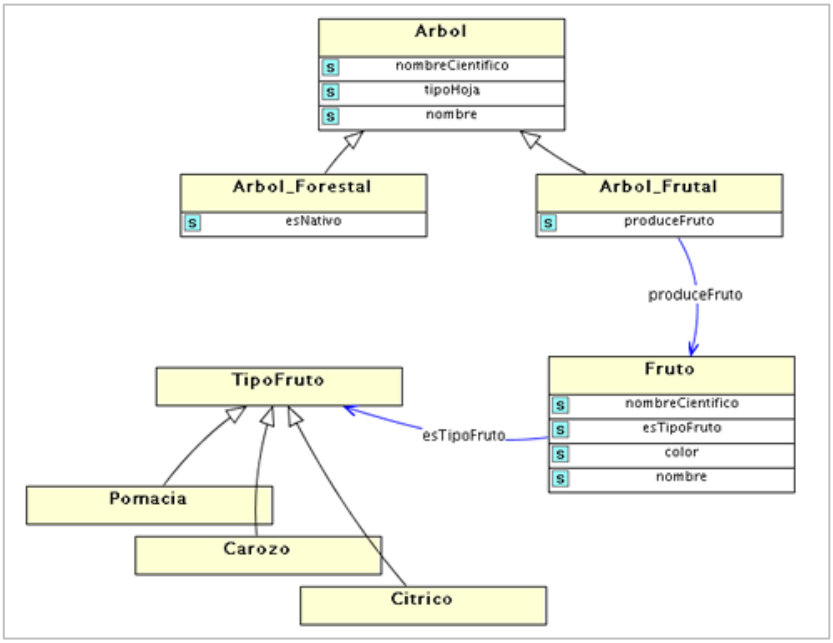

Figura II-4. Ejemplo de uso de ontologías, obtenido de (Krsulovic Morales, 2003)

2.2.4.1 Componentes de una ontología

Una ontología está formada por un conjunto de conceptos, relaciones y restricciones, expresados mediante definiciones aceptadas por los miembros de una comunidad para expresar su área de conocimiento y en un formato procesable por la máquina. Concretamente, los elementos que forman una ontología se resumen en:

- **Conceptos:** Un concepto puede ser cualquier cosa acerca de la cual se pueda aseverar dentro del Universo del Discurso. De este modo, un concepto puede ser un objeto tangible (físico) o un objeto intangible, por ejemplo la descripción de una tarea, función, acción, estrategia, etc. Cada concepto tiene un nombre que lo identifica y puede poseer un conjunto de atributos.
- **Relaciones:** Las relaciones representan el tipo de interacción entre los conceptos de un dominio, son formalmente definidas como subconjuntos del

producto cartesiano de *n* conjuntos, esto es *R: C1 x C2 x … x Cn*, donde cada conjunto está formado por el agregado de instancias de las clases relacionadas. Existen, además, relaciones con un significado especial, concretamente: la relación de especialización expresa una asociación *es_un* entre dos conceptos y la relación de composición expresa una asociación *es_parte_de* entre dos conceptos.

- **Funciones:** Son un caso especial de relaciones donde el enésimo elemento de la relación es único para los n-1 anteriores. Formalmente las funciones se definen como *F: C1 x C2 x….x Cn-1 x Cn*, donde la relación entre los valores de los *n-1* conceptos, determina el valor del concepto *enésimo*.
- **Axiomas:** Los axiomas se usan para definir afirmaciones siempre ciertas en la realidad modelada. Los axiomas definidos en una ontología pueden ser estructurales o no estructurales. Un axioma estructural establece condiciones relacionadas a las jerarquías de la ontología, conceptos y atributos definidos. Un axioma no estructural se establece entre atributos de un concepto.
- **Instancias:** Representan elementos específicos del dominio de la ontología y son instancias concretas de los conceptos modelados.

2.2.4.2 Utilidad de las ontologías

La utilidad del uso de ontologías aporta numerosas ventajas de las que carece la Web actual. A continuación se procede a numerar las más importantes:

- Las ontologías se usan para favorecer la comunicación entre personas, organizaciones y aplicaciones, lograr la interoperabilidad entre sistemas informáticos, razonar automáticamente y para la ingeniería de software(Abián, 2005).
- Las ontologías favorecen la comunicación entre personas, organizaciones y aplicaciones porque proporcionan una comprensión común de un dominio, de modo que se eliminan confusiones conceptuales y terminológicas. Los problemas derivados de la falta de comprensión común entre personas revisten una gran importancia en la ciencia y en la tecnología. Las ontologías

favorecen también la comunicación entre aplicaciones y la comprensión común de la información entre ellas. Las ontologías serán imprescindibles en la Web Semántica y en los futuros sistemas de gestión empresarial porque permitirán que las aplicaciones estén de acuerdo en los términos que usan cuando se comunican. Mediante ellas, será mucho más fácil recuperar información relacionada temáticamente, aun cuando no existan enlaces directos entre las páginas Web.

- Las ontologías también sirven para conseguir que los sistemas interoperen. Dos sistemas son interoperables si pueden trabajar conjuntamente de una forma automática, sin esfuerzo por parte del usuario.
- Las ontologías resultan muy útiles para facilitar el razonamiento automático, es decir, sin intervención humana. Partiendo de unas reglas de inferencia, un motor de razonamiento puede usar los datos de las ontologías para inferir conclusiones de ellos.
- En la ingeniería del software, las ontologías ayudan a la especificación de los sistemas de software. Como la falta de un entendimiento común conduce a dificultades en identificar los requisitos y especificaciones del sistema que se busca desarrollar, las ontologías facilitan el acuerdo entre desarrolladores y usuarios.
- Las ontologías pueden utilizarse para mejorar tanto las búsquedas de información como la navegación por la Web. Los contenidos Web podrían definirse en función de los términos ontológicos si cada dominio definiera una o más ontologías. De este modo se pueden evitar las ambigüedades en las búsquedas al poder acceder a categorías más específicas de la ontología.
- Favorecen la interoperabilidad, ya que se pueden especificar cómo los términos se relacionan con otros términos, facilitando la interoperabilidad semántica.
- Posibilitan la creación de agentes inteligentes que entenderán e integrarán las informaciones de distintas fuentes.

- Facilitan la organización de colecciones de recursos multimedia, gracias a la posibilidad de incluir anotaciones semánticas a los recursos no textuales.
- Facilitan el comercio electrónico, al suplir la falta de integración de las heterogéneas descripciones de los productos, así como de un sistema de catalogar deficiente. Además dotan a los datos de semántica comprensible por las máquinas, lo que facilita la automatización de procesos.

2.3 Diseño de aplicaciones Web

En la actualidad, el uso de la red se ha convertido en un elemento indispensable, transformando las necesidades de información de las empresas y organizaciones. No existe prácticamente ninguna empresa u organismo que no necesite que la información esté accesible tanto dentro como fuera de su edificio o que ésta sea compartida entre todas las partes interesadas en ella, ya sea en su totalidad o en aquella parte que corresponda según su función.

Estas necesidades han sido las causantes de un movimiento creciente de cambio desde las clásicas aplicaciones de escritorio a aplicaciones Web, que las satisfacen ampliamente. Así, las páginas Web, que antes tan sólo se dedicaban a mostrar información, se han convertido en aplicaciones con las cuales el usuario interactúa. Hoy el foco se encuentra en la construcción de sitios Web potentes que proporcionen valor real y ofrezcan una experiencia positiva al usuario o cliente. Cuando los visitantes coinciden en evaluar con alta puntuación el contenido, facilidad de uso, rendimiento, fiabilidad y satisfacción general, se denomina sitio Web centrado en el usuario.

Frente a las imposiciones de diseñadores, tecnología o compañía, el diseño de un sitio Web debe centrarse en el cliente o usuario. Construir sitios Web centrados en el usuario se basa en proporcionar una experiencia positiva para todos los visitantes, ya sea en la búsqueda de información, formar parte de una comunidad, comprar artículos o entretenerse. Este enfoque enfatiza la importancia de comprender la necesidad del usuario, las herramientas y tecnologías que utiliza, y su contexto social

y organizativo. Además, concierne al modo en que ese conocimiento del usuario se plasma en los diseños, que deberán ser probados para asegurar que las necesidades detectadas son cubiertas.

Incluso los desarrollos de sitios en los que no hay competidores directos, como es el caso de instituciones educativas e intranets corporativas, pueden beneficiarse de la orientación a usuario. Sitios Web sencillos, claros y bien diseñados pueden reducir drásticamente el tiempo de trabajo de usuarios, reducir los costes de mantenimiento y mejorar la satisfacción general.

En este apartado se tratarán los aspectos fundamentales que se deben de tener en cuenta a la hora de crear una aplicación Web orientada al usuario, así como la principal herramienta para el análisis y diseño de la aplicación: el lenguaje UML.

Finalmente se tratarán los patrones de estilo y diseño que se han utilizado para la implementación de este proyecto.

2.3.1 Elementos clave de sitios Web centrados en usuario

Dado que la aplicación que se ha desarrollado está claramente orientada al usuario se ha procurado seguir las siguientes pautas:

Facilidad de uso

La experiencia en el uso de las tecnologías de la información varía enormemente entre los usuarios. Es necesario que el sitio sea tan sencillo de usar como sea posible. El diseño de la interfaz de usuario ocupa muchos libros, pero se ofrecen unas líneas básicas:

- Mantener el sitio tan sencillo como sea posible. Cuantas más opciones, publicidad y distracciones ocupen la pantalla, más fácilmente se confundirá el usuario.
- Mantener el texto claro. Utilizar fuentes sencillas y claras. No utilizar tamaños de fuente muy pequeños y mantener en mente que el tamaño puede variar entre diferentes tipos de máquinas.

- Simplificar los procesos de interacción con el usuario. Tanto la intuición como la evidencia soportan la idea de que cuanto mayor es el número de clicks para completar un proceso (por ejemplo una hoja de pedido), menor probabilidad hay de que el usuario lo complete. Se debe mantener el número de pasos al mínimo.
- No permitir que el usuario se pierda. Deben proporcionarse señales y pistas de navegación que indiquen al usuario dónde se encuentra. Por ejemplo, si se utiliza la metáfora de la cesta de compra, que permite al usuario acumular compras en un contenedor virtual antes de finalizar el pedido, se debe mantener un enlace a la cesta visible en todo momento.

Rendimiento

No sólo referido al tiempo de descarga de las páginas, sino a la implementación de los procesos en los que el usuario tenga que interaccionar con el sistema. La navegación debe ser fluida y debe minimizarse el retardo introducido por el procesamiento en el sitio Web. Este parámetro puede ser especialmente crítico en intranets donde el usuario espera la misma respuesta que ofrecen las aplicaciones locales.

Satisfacción

La medida en que el sitio Web se adecua a sus objetivos es proporcional a la satisfacción de los usuarios en su uso.

Contenido

La información publicada en el sitio Web debe orientarse al usuario. El contenido, tanto en su estructura, como redacción y mensaje, será fiel al valor de marca. El sitio debe ofrecer información de utilidad al cliente sin sobrecargarle con datos innecesarios.

Incompatibilidad entre navegadores

Se debe verificar el funcionamiento del sitio en diferentes navegadores y sistemas operativos. Si el portal no funciona para un navegador popular, el sitio

parecerá poco profesional y se perderá una sección de usuarios potenciales. Como regla general, para ser visible a la gran mayoría de usuarios, esta aplicación se ha desarrollado para que sea compatible con Mozilla Firefox 3 y 4, Microsoft Internet Explorer 8 y 9 y Google Chrome 10-12. El cliente se ejecutará en dicho navegador, y el servidor será una aplicación a desplegar en un entorno Java, por lo que será multiplataforma.

2.3.2 UML (Unified Modeling Language)

El Lenguaje Unificado de Modelado (UML) (Booch, 2006)es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende ser apoyo a la mayoría de los procesos de desarrollo orientados a objetos.

(Geraldo, 2005)UML es un lenguaje que nos ayuda a interpretar grandes sistemas mediante gráficos o mediante texto obteniendo modelos explícitos que ayudan a la comunicación durante el desarrollo ya que al ser estándar, los modelos podrán ser interpretados por personas que no participaron en su diseño (e incluso por herramientas) sin ninguna ambigüedad. En este contexto, UML sirve para *especificar*, modelos concretos, no ambiguos y completos.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las

relaciones entre objetos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. El modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos.

En la definición de UML, se establecieron como objetivos principales la consecución de un método que aunara los mejores aspectos de sus predecesores. Para ello, se plantearon las siguientes características:

- El método debe ser capaz de modelar no sólo sistemas de software, sino otro tipo de sistemas reales de la empresa, siempre utilizando los conceptos de la orientación a objetos (OO).
- El lenguaje de modelado que se pretendía definir, debía poder ser utilizado, a la vez, por máquinas y por personas.
- Establece un acoplamiento explícito de los conceptos y los artefactos ejecutables.
- Maneja los problemas típicos de los sistemas complejos de tiempo real.

Lo que se pretende con esto, es lograr que los lenguajes que se aplican siguiendo los métodos más utilizados sigan evolucionando en conjunto y no por separado. Y además, unificar las perspectivas entre diferentes tipos de sistemas, no sólo software, al facilitar la comprensión de las fases de desarrollo, los requerimientos de análisis, el diseño, la implementación y los conceptos propios de la orientación a objetos.

Para el desarrollo del proyecto se ha empleado la versión 1.5 de UML que se adapta a los requerimientos y necesidades del proyecto, no siendo necesario emplear versiones posteriores.

Diagramas UML

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones. En concreto, un diagrama ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de

diagramas para visualizar el sistema desde varias perspectivas. UML incluye los siguientes diagramas:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de objetos.
- Diagrama de secuencia
- Diagrama de colaboración.
- Diagrama de estados.
- Diagrama de actividades.
- Diagrama de componentes.
- Diagrama de despliegue.

Los diagramas más utilizados y empleados para el desarrollo del proyecto son, los de casos de uso, clases y secuencia, por lo que nos centraremos en éstos.

2.3.2.1 Diagrama de caso de uso

Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema, mediante su interacción con los usuarios, u otros sistemas. Esto quiere decir que, estos diagramas muestran la relación entre los actores y los casos de uso. En este tipo de diagrama intervienen algunos conceptos nuevos. Un caso de uso es una secuencia de transacciones que son desarrolladas por un sistema, en respuesta a un evento que inicia un actor sobre el propio sistema. Una relación es una conexión entre los elementos del modelo. Un actor es una entidad externa al sistema, que se modela y que puede interactuar con él. Un ejemplo de actor, podría ser un usuario o cualquier otro sistema.

Las relaciones entre casos de uso y actores pueden ser las siguientes:

- Un actor se comunica con un caso de uso
- Un caso de uso extiende otro caso de uso (extend)
- Un caso de uso usa otro caso de uso (include)

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema, al mostrar la reacción producida como respuesta a los eventos que se producen en el mismo, en la Figura II-5 se muestra un ejemplo.

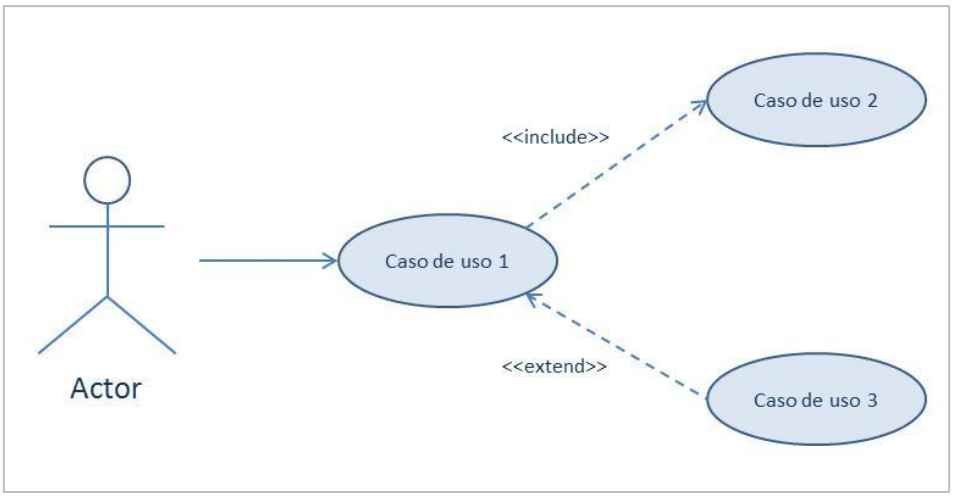

Figura II-5. Ejemplo de diagrama de casos de uso

2.3.2.2 Diagramas de clases

Los diagramas de clases representan un conjunto de elementos del modelo que son estáticos, como las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos. Algunos de los elementos que se pueden clasificar como estáticos en UML, son los siguientes:

- **Paquete:** Es el mecanismo de que dispone UML para organizar sus elementos en grupos.

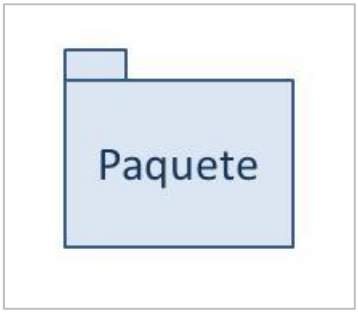

Figura II-6. Figura de un paquete

- **Clase:** Describe un conjunto de objetos que comparte los mismos atributos, operaciones, métodos, relaciones y significado. Los componentes de una clase son:
    - Atributo: Se corresponde con las propiedades de una clase.
    - Operación: También conocido como método, es un servicio proporcionado por la clase que puede ser solicitado por otras clases y que produce un comportamiento en ellas cuando se realiza.

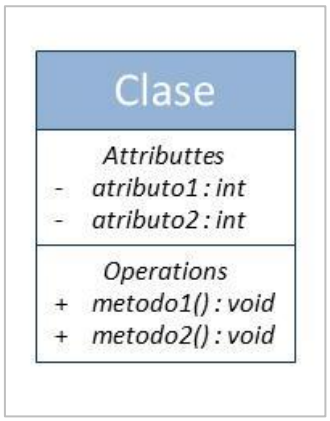

Figura II-7. Figura de una clase

- **Interfaz:** Representa a la funcionalidad que proporciona uno o varios elementos del modelo al resto de elementos. Esta interfaz no tiene asociado un comportamiento, por lo que tendrá que ser implementado por otro elemento.

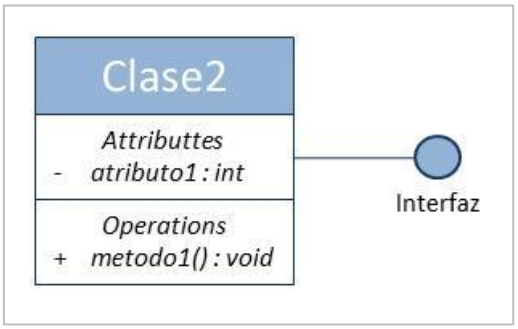

Figura II-8. Figura de una interfaz.

En estos diagramas se muestra, además, las relaciones estructurales existentes entre los elementos descritos anteriormente. Un ejemplo de un diagrama de clases completo, puede ser el de la Figura II-9 .

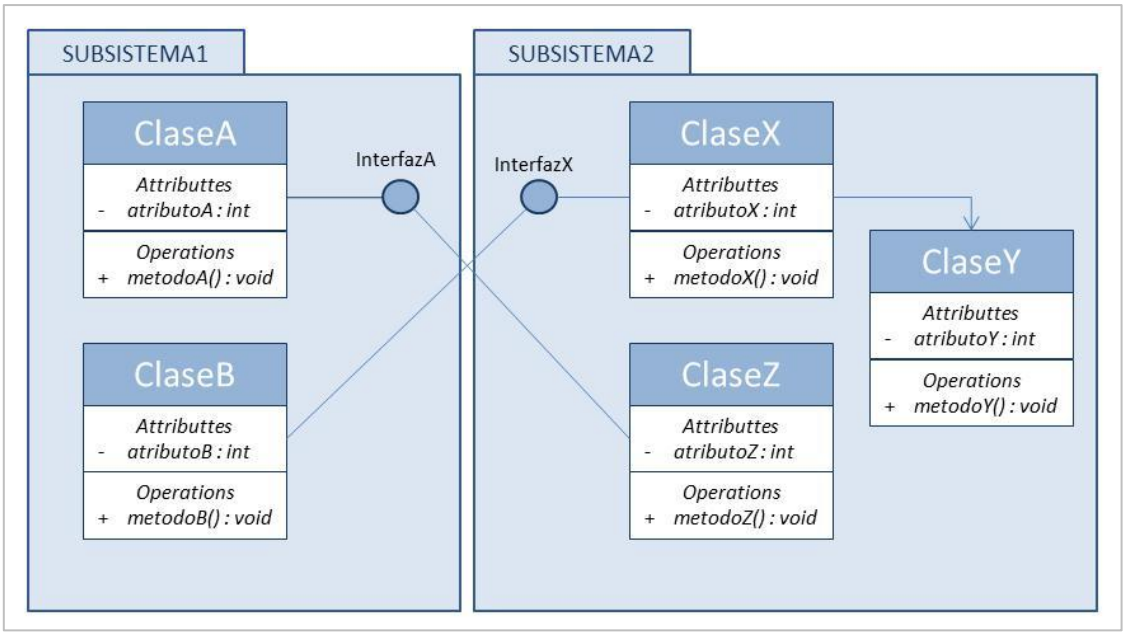

Figura II-9. Ejemplo de diagrama de clases

2.3.2.3 Diagrama de secuencia

Muestran las interacciones entre un conjunto de objetos, ordenadas según el momento del tiempo en que tienen lugar. El objeto puede existir sólo durante la ejecución de la interacción, momento en el que puede ser creado o destruido. Un diagrama de secuencia representa una forma de indicar el periodo durante el que un objeto está desarrollando una acción directamente o a través de un procedimiento.

En este tipo de diagramas también intervienen los mensajes, que son la forma en que se comunican los objetos. Los mensajes consisten en la solicitud de una operación de un objeto origen a un objeto destino. Existen distintos tipos de mensajes según cómo se producen en el tiempo: simples, síncronos, y asíncronos.

Los diagramas de secuencia permiten indicar cuál es el momento en el que se envía o se completa un mensaje mediante el tiempo de transición, especificado en el diagrama.

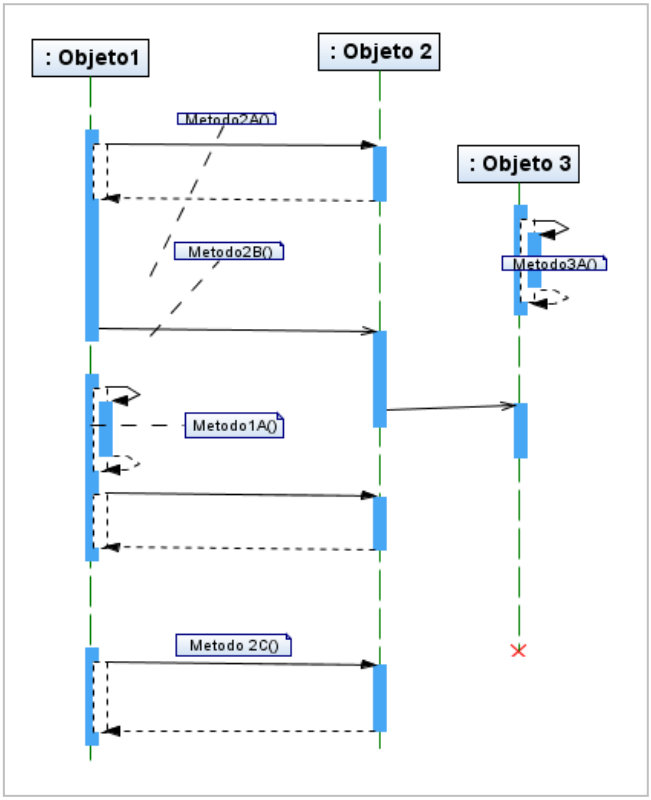

Figura II-10. Ejemplo diagrama de secuencia

2.3.3 Estilos y patrones de diseño

Un patrón describe un problema que ocurre en repetidas ocasiones en nuestro entorno, y describe a su vez el núcleo de la solución al problema, de un modo que es posible utilizar dicha solución en muchas ocasiones sin repetir la forma dos veces. Los diseñadores de software disponen de patrones que capturan la experiencia existente y probada en resolución de problemas repetitivos. Los patrones, además, ayudan a promover buenas prácticas de diseño y facilitan la construcción de arquitecturas software complejas.

Los patrones existentes cubren diferentes rangos de escala y abstracción. Algunos ayudan a estructurar el software en subsistemas, otros soportan el refinamiento de subsistemas, componentes y sus relaciones, o bien desacoplan componentes relacionados, otros ayudan a implementar ciertos aspectos de diseño en un lenguaje de programación específico.

A continuación se describen los estilos arquitectónicos y los patrones de diseño que se han aplicado en el proyecto.

2.3.3.1 Estilos o patrones arquitectónicos

Los patrones arquitectónicos expresan esquemas para la organización estructural de los sistemas software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. A continuación se describe el patrón MVC utilizado en este proyecto.

Modelo Vista Controlador

MVC (Model View Contoller) es un patrón que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos de forma que las modificaciones al componente de la vista, o a cualquier parte del sistema, puedan ser hechas con un mínimo impacto en el componente del modelo de datos o en los otros componentes del sistema. Este patrón cumple perfectamente el cometido de modularizar un sistema. Los tres principales componentes del patrón MVC son:

- **Modelo**: es la representación específica de los datos con los cuales el sistema opera.
- **Vista**: usualmente la interfaz de usuario. Es la responsable de transformar el modelo para que sea visualizado por el usuario, es decir, convierte los datos para que al usuario le sean significativos y los pueda interpretar fácilmente. Se encarga de la interacción con el usuario.
- **Controlador**: Es el intermediario entre los otros dos componentes, se trata de la parte lógica responsable del procesamiento y comportamiento de acuerdo a las peticiones del usuario, construyendo un modelo apropiado y pasándolo a la vista para su correcta visualización.

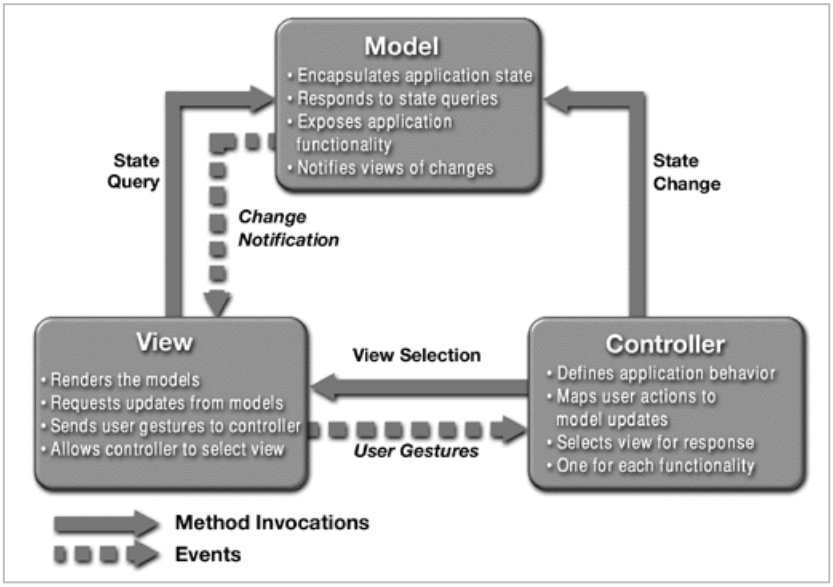

Figura II-11. Modelo Vista Controlador, obtenido de (Casanovas, 2004)

Este patrón es muy popular y ha sido portado a una gran cantidad de entornos y frameworks[9] como pueden ser Spring, Struts, JSF, etc. Algunos de sus principales beneficios son:

- Menor acoplamiento: Desacopla las vistas de los modelos y los modelos de la forma en que se muestran e ingresan los datos.
- Mayor cohesión: Cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio)
- Las vistas proveen mayor flexibilidad y agilidad.
    - Se puede crear múltiples vistas de un modelo.
    - Se puede crear, añadir, modificar y eliminar nuevas vistas dinámicamente.

[9] Un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

  - Las vistas pueden anidarse.
  - Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
  - Se puede sincronizar las vistas.
  - Las vistas pueden concentrarse en diferentes aspectos del modelo.

- Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad.

MVC Modelo 1

En el MVC Modelo 1 las páginas JSP[10] están en el centro de la aplicación y contienen tanto la lógica de control como la de presentación. Este tipo de arquitectura funciona de la siguiente manera: el cliente hace una petición a una página JSP, se construye la lógica de la página, generalmente en objetos Java (Plain Old Java Object - POJO[11]), y se transforma el modelo para ser desplegado una vez más.

[10] Las páginas JSP funcionan como aplicaciones del lado del servidor. Son conjuntos de etiquetas y scriptlets en lenguaje Java que se integran dentro del código HTML y que el servidor de aplicaciones compila para dar como resultado un servlet y de la ejecución de éste.

[11] Un POJO es una clase JAVA, en su forma más pura, y básicamente consta de un conjunto de atributos y métodos de obtención y establecimiento (métodos getters y setters) para cada uno de los atributos definidos para la clase.

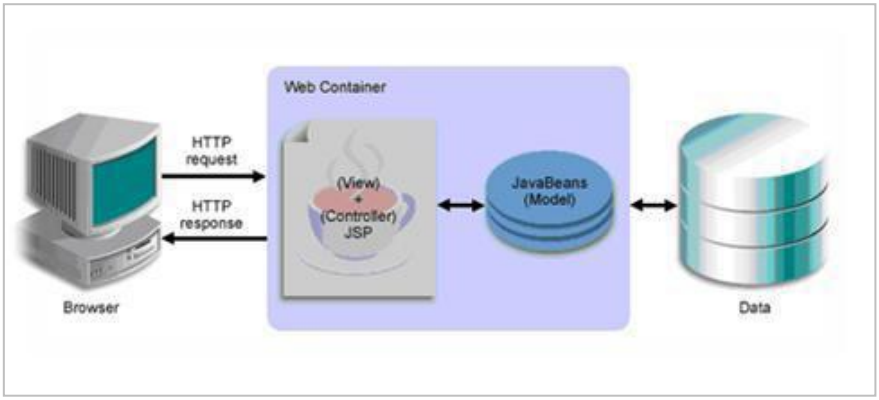

Figura II-12. Arquitectura MVC modelo1

MVC Modelo 2

En el MVC Modelo 2 existe una clara separación entre el controlador y la vista, ahora es directamente el controlador quien recibe la petición, prepara el modelo y lo transforma para que sea desplegado en la vista.

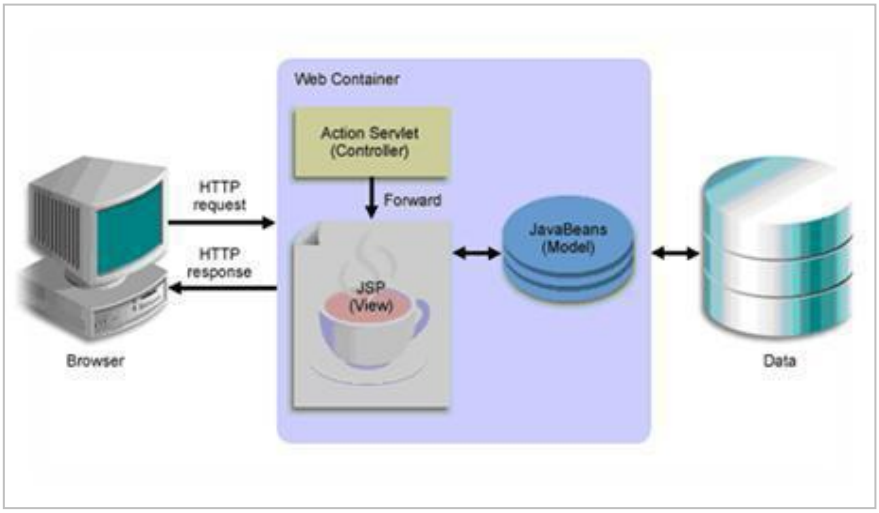

Figura II-13. Arquitectura MVC modelo2.

Capas

El patrón layers ayuda a estructurar las aplicaciones que pueden ser descompuestas en grupos de subtareas en las que cada grupo está a un nivel particular de abstracción. Se suele aplicar en el diseño de sistemas cuya característica dominante es una mezcla de operaciones a alto y bajo nivel, donde las de nivel alto se apoyan en las de nivel bajo. La comunicación típica en estos sistemas consiste en un flujo de

peticiones moviéndose de alto a bajo nivel, y respuestas a las peticiones, datos entrantes o notificaciones sobre eventos viajando en dirección contraria. Este patrón estructura el sistema en un número apropiado de capas, cada una de las cuales se compone de componentes que trabajan al mismo nivel de abstracción, y las sitúa una sobre otra. Los servicios de cada capa implementan una estrategia para combinar las operaciones de la capa inferior y proporcionar servicio a su capa superior

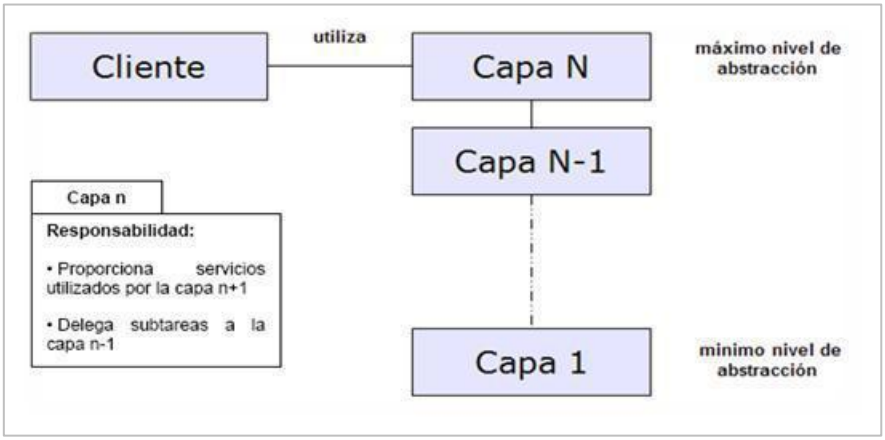

Figura II-14. Estructura del patrón capas

A continuación se ofrecen los pasos que, en función de la aplicación en desarrollo, deberán considerarse en la implementación de este patrón:

- Definir el criterio de abstracción para agrupar las tareas en capas. Por ejemplo, la distancia desde el hardware puede dirigir las capas bajas y la complejidad conceptual las altas.
- Determinar el número de niveles de abstracción de acuerdo al criterio anterior. Cada nivel corresponderá a una capa del patrón. Debe alcanzarse un punto de equilibrio entre la sobrecarga que impone un número elevado de capas y la complejidad que puede provocar utilizar pocas.
- Identificar las capas y asignar tareas a cada una de ellas. La tarea de más alto nivel comprenderá la funcionalidad del sistema tal como la percibe el cliente. Las demás capas servirán de apoyo a capas superiores.

- Especificar los servicios. El principio más importante en la implementación es que las capas se encuentren estrictamente separadas, y por tanto, ningún componente deberá extenderse a más de una capa.
- Refinar la estructura iterando sobre los puntos anteriores hasta evolucionar a una arquitectura natural y estable.
- Especificar una interfaz para cada capa. Si la capa n debe ser una caja negra para la capa n+1, deberá diseñarse una interfaz que ofrezca todos los servicios de la capa n.
- Estructurar cada capa individualmente. En función de su complejidad puede ser necesario separar en componentes la capa.
- Especificar la comunicación entre capas adyacentes. El mecanismo más utilizado para la comunicación entre capas es el modelo solicitud respuesta.
- Desacoplar las capas adyacentes mediante el uso de interfaces.
- Diseñar una estrategia para el manejo de errores. Como regla general, los errores se deben tratar en la capa más baja posible. Hacia capas superiores sólo deben propagarse errores generales que tengan sentido en el nivel de abstracción.

2.3.3.2 Patrones de diseño

Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software. Estos patrones resuelven un problema de diseño general en un contexto particular. A continuación se describen los patrones de diseño que se han utilizado en este proyecto.

IoC (Inversión de control)

El patrón de diseño IoC (*Inversion of Control*) describe la forma en que un objeto resuelve sus dependencias con otros objetos. La idea se basa en que dado un objeto, el contenedor de inversión de control resuelva sus dependencias,

inyectándoselas ya sea a través de métodos set o de uno de los constructores del objeto. Para que esto sea posible, ese contenedor de inversión[12] debe manejar el ciclo de vida del objeto. Este patrón puede parecer confuso, incluso parecer que va en contra de paradigmas de la programación orientada a objetos como la "*encapsulación*", sin embargo, constituye una poderosa filosofía de trabajo. En cierto modo es una implementación del principio de Hollywood: "*No me llames, yo te llamo*". Este principio tiene varios beneficios entre los que se puede destacar:

- Elimina la responsabilidad de buscar o crear objetos dependientes, dejando estos configurables. De esta forma búsquedas de componentes complejas como es el uso de JNDI pueden ser delegadas al contenedor.
- Reduce a cero las dependencias entre implementaciones, favoreciendo el uso de diseño de modelos de objetos basados en interfaces.
- Permite a nuestra(s) aplicación(es) ser reconfigurada(s) sin tocar el código.
- Estimula la escritura de componentes testables, pues la Inversión del Control facilita el uso de pruebas unitarias.

Factory

Factory Method define una interfaz para crear objetos, pero deja que sean las subclases quienes decidan qué clases instanciar. Permite que una clase delegue en sus subclases la creación de objetos, de esta forma la clase derivada toma la decisión sobre qué clase instanciar y cómo instanciarla. El patrón Factory Method permite escribir aplicaciones que son más flexibles respecto de los tipos a utilizar difiriendo la creación de las instancias en el sistema a subclases que pueden ser extendidas a medida que evoluciona el sistema. Permite también encapsular el conocimiento referente a la creación de objetos. Factory Method hace también que el diseño sea más adaptable a cambio de sólo un poco más de complejidad.

[12] Spring es un ejemplo de contenedor de inversión.

La estructura del patrón Factory Method, está formada por los siguientes elementos:

- Producto. Define la interfaz de los objetos que crea el método de fabricación.
- ProductoConcreto. Implementa la interfaz Producto.
- Creador. Declara el método de fabricación, el cual devuelve un objeto del tipo Producto. También puede definir una implementación predeterminada del método de fabricación que devuelve un objeto ProductoConcreto. Puede llamar al método de fabricación para crear un objeto Producto.
- CreadorConcreto. Redefine el método de fabricación para devolver una instancia de ProductoConcreto.

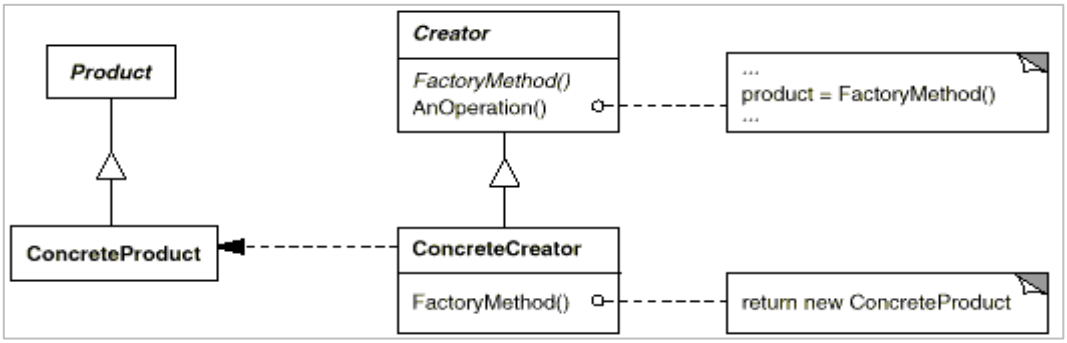

Figura 2.1. Estructura del Patrón Factory

Mediante esta estructura el Creador se basa en sus subclases para definir el Factory Method de modo que devuelve una instancia del ProductoConcreto adecuado. El patrón Factory Method proporciona los siguientes beneficios:

- Proporciona enganches para las subclases: Crear objetos dentro de una clase con un Factory Method es siempre más flexible que hacerlo directamente.
- Conecta jerarquías de clases paralelas.

2.4 Semantic Metadata Search (SEMSE)

El proyecto SEMSE(Palacios Madrid, 2010) surge con el objetivo de solventar los problemas a los que se enfrenta la Web Semántica a la hora de recuperar y tratar los esquemas de metadatos que conforman la capa semántica necesaria para el funcionamiento de dicha Web. Algunos de los problemas derivados de la utilización de dichos esquemas son:

- Dificultades en la recuperación semántica de sus elementos componentes, especialmente relevantes para la interoperabilidad entre ellos.
- Problemas de recuperación mediante motores de búsqueda Web.
- Ambigüedad, ya que la referencia a vocabularios de metadatos es opcional.
- Dificultades en la recuperación, al no existir un repositorio donde se almacenen de forma uniforme y conceptualmente relacionados, al menos aquellos esquemas que gocen de mayor popularidad.

Los problemas mencionados hacen imposible realizar una recuperación conceptual. Lamentablemente este tipo de recuperación no es factible sin:

- Consenso de las unidades léxicas y terminológicas representativas para los conceptos. Sin éste se produce ambigüedad conceptual.
- Aspectos sintagmáticos y paradigmáticos de las unidades léxicas y las unidades terminológicas representativas de los conceptos, esto es la estructura y significado de las mismas.
- Mapa conceptual y relacional, que deberá incluir las equivalencias léxicas.

Como solución para la ambigüedad conceptual SEMSE propone interrelacionar los elementos, incluidos en los esquemas, con los conceptos incluidos en una ontología de referencia. La ontología de referencia realiza las funciones de diccionario general mientras que los esquemas de metadatos hacen las funciones de diccionarios especializados, superando las limitaciones de los aspectos fraseológicos y las irregularidades en cuanto a la información en las definiciones.

Además de servir como vocabulario controlado y proporcionar significado a los elementos, a través de la ontología de referencia se consigue un mapa conceptual y relacional que incluye y permite equivalencias léxicas entre los términos de las etiquetas de los documentos y la estructura ontológica general.

Sin embargo, para conseguir esta relación entre los esquemas de metadatos y la ontología de referencia, además de paliar la ambigüedad sintáctica y semántica propia de los vocabularios en su definición clásica, es necesario dotar a dichos esquemas de una capa semántica extendida, es decir, una nueva representación de dichos esquemas que permita la representación semántica de sus elementos, independiente de su estructura, compatible con el esquema original y aplicable a cualquier esquema. SEMSE propone dos nuevas representaciones adicionales para cada esquema basadas en el uso de ontologías:

- La primera de ellas, la versión semánticamente cualificada del esquema, tiene como objetivo sustituir en su uso al esquema original. De este modo, esta representación maximiza la compatibilidad con el esquema de origen, al tiempo que permite incorporar la semántica de los elementos incluidos en el mismo.
- La segunda representación del esquema, denominada representación como ontología específica, es una ontología de dominio dirigida a capturar la semántica concreta de cada elemento, tal y como se define en el esquema original, al tiempo que aborda aspectos como sinonimia y plurilingüismo. Centrada en la semántica de los elementos, incluye la definición de cada elemento como clase o propiedad, según corresponda, en base a la ontología de referencia desarrollada con este fin y denominada SEMSE(*SEMSE: SEmantic Metadata SEarch*, 2008).

2.4.1 Esquemas semánticamente cualificados

Como primer paso para la representación semántica de esquemas, fue necesario establecer la forma en que ésta puede ser incluida partiendo de cualquier

esquema de metadatos. Para ello, el proyecto SEMSE se basó en el DCMI (Dublin Core Metadata Iniciative, 2011) al proponer múltiples recomendaciones destinadas, no sólo al establecimiento de un esquema de metadatos, sino también a la ampliación de dicho esquema y, lo que es más importante, la estandarización en la definición de esquemas de metadatos.

En relación con el modo de extender los términos para incluir la semántica, SEMSE se basó en la generalización de los términos en subclase y subpropiedades, debido a que con esta generalización de consigue que:

- Un término pueda tener cualquier número de subtérminos, lo que facilita la extensibilidad.
- El término genérico en la jerarquía, es independiente de los términos específicos o subtipos. De nuevo se facilita la extensibilidad al no ser necesaria la modificación del término original.
- Los términos específicos son, desde un punto de vista genérico, del tipo del término padre. De este modo, un subtérmino conserva las líneas generales definidas por el término del cual proviene y se dice que es un subtipo del padre. Se trata de un aspecto de gran relevancia en lo concerniente a la compatibilidad, al mantener las características del término padre, se proporciona la posibilidad de utilizar dichas características en caso de no poderse procesar el término hijo. También tiene relación con la reutilización al heredarse la definición del padre por los hijos.
- En relación con el punto anterior, si bien un término hijo comparte por defecto las propiedades del término padre del cual proviene, el hijo tiene la capacidad de redefinir esas propiedades. Este aspecto tiene relevancia en relación con la definición de la semántica de los elementos, permitiendo al término hijo redefinir el significado heredado de su padre.

Una vez encontrada la forma de ampliar los elementos, mediante la relación de generalización, se analizó la forma en la que se definían los subtérminos en el esquema del DCMI[13]. Esto llevó al análisis de los cualificadores incluidos en el Conjunto de Elementos de Metadatos de DC (*DCMES*). Sin embargo, de los elementos y cualificadores evaluados, únicamente el término *relation* se aproximó a la representación semántica buscada. Por lo que se decidió crear un nuevo conjunto de *cualificadores semánticos* de forma que, el nuevo esquema semántico incluiría un término derivado de *relation*, que expresaría la relación semántica entre el término original y su significado.

Es por ello que se creó un esquema de metadatos denominado *Esquema de Cualificadores Semánticos* cuyo objetivo es albergar este conjunto de *Cualificadores Semánticos,* que permiten definir la semántica de los elementos de cualquier esquema mediante la cualificación semántica de sus elementos. Dicho esquema es público y accesible vía web e identificado con una URL persistente proporcionada por el servicio Persistent Uniform Resource Locator (PURL), *http://purl.org/semse/sq/*. Adicionalmete se creó el *cualificador semántico* denominado *hasSemantics* (tiene semántica) basado en el término *relation* y con él es posible relacionar términos de esquemas con sus significados, relacionándolos mediante un sistema formal de identificación. Al igual que el esquema que lo contiene, hasSemantics es accesible vía web e identificado por una url persistente, *http://purl.org/semse/sq/hasSemantics*.

Finalmente, una vez disponible este nuevo concepto y el esquema que lo contiene, se llegó a la conclusión de que la mejor manera para extender los esquemas originales y aportarles la semántica de la ontología de referencia, fue la creación de un nuevo esquema para albergar los nuevos elementos semánticamente cualificados. Dichos elementos están definidos como subpropiedades del conjunto original de

[13] Dublin Core Metadata Iniciative. Es una organización dedicada a fomentar la adopción extensa de los estándares interoperables de los metadatos y a promover el desarrollo de los vocabularios especializados de metadatos para describir recursos para permitir sistemas más inteligentes del descubrimiento del recurso.

elementos, además de incluir el cualificador semántico *hasSemantics* que permitirá albegar una referencia al significado de elemento. De esta forma se facilita la compatibilidad y se reutilizan los términos del esquema original, manteniéndose su simplicidad. El nombre de este nuevo esquema provendrá del nombre del esquema original, precedido por el acrónimo 'sq' *Semantic Qualified* (*Cualificado Semánticamente*).

Cabe destacar que, para maximizar la compatibilidad de las aplicaciones con las nuevas versiones del esquema, se han mantenido los tipos originales de los elementos que se cualifican, así como el lenguaje original en el que estuvieran escritos.

2.4.2 Ontologías específicas

Una vez desarrollada la representación cualificada de esquema que parte del esquema original, proporcionando compatibilidad, homogeneidad en su definición y que permite la relación de cada elemento con su semántica, el siguiente paso fue abordar la definición de la semántica de los elementos.

El porqué de definir una nueva representación para albergar la semántica de los elementos obedece a varios motivos. Primeramente, separar la semántica de los elementos haciendo uso de un formalismo específico, en este caso una ontología, permite especializar la representación de los significados, favoreciéndose la modificabilidad y mantenibilidad de éstos. Otra de las ventajas que aporta una representación especializada de la semántica es la posibilidad de utilizar lenguajes que aporten mayor capacidad de representación semántica, el objetivo fundamental de esta versión del esquema. Al tratarse de una representación distinta, no es preciso mantener el lenguaje de definición del esquema cualificado que, en la mayoría de los casos, no es el más indicado para representar o albergar semántica. Además de la ventaja en la especialización de los significados que aporta el uso de una ontología, este formalismo permite abordar otros aspectos como la sinonimia de términos o la posibilidad de definirlos en múltiples idiomas, todo ello evitando la necesidad de complicar el esquema original o su versión cualificada.

Al igual que el esquema de cualificadores semánticos define el conjunto de cualificadores que permiten generar los distintos esquemas cualificados, fue preciso definir la ontología a partir de la cual generar las distintas ontologías específicas. Esta ontología padre, denominada *SEMSE*, es una ontología de representación que tiene como objetivo definir la estructura, conceptos y propiedades, destinados a albergar la semántica de los elementos, tal y como se define en los esquemas originales. Asimismo, además de la semántica de los elementos, la ontología SEMSE también proporciona soporte para albergar sinónimos y múltiples idiomas en la definición de los conceptos.

Para la definición de la estructura de la ontología SEMSE se inició un proceso de evaluación de distintas ontologías, el cual concluyó con la ontología UMBEL(Umbel.org, 2008), como la mejor candidata para la representación de la ontología de referencia. Aunque sólo permite la representación de clases, su estructura es clara y está preparada para dar soporte a sinonimia y plurilingüismo.

Tomando como base la estructura de la ontología UMBEL se procedió a definir la estructura de la ontología SEMSE, adaptando y añadiendo los elementos necesarios para dar cobertura a los requisitos planteados.

El lenguaje elegido para su representación fue el OWL Full, dadas las posibilidades que ofrece para la representación de semántica y por estar diseñado para la publicación en el entorno Web, aunque para algunos aspectos en concreto se ha complementado con OWL, SKOS y RDF. Adicionalmente, y al igual que con el resto de esquemas creados con anterioridad, la ontología de referencia se encuentra publicada en la web mediante una URL persistente.

Una vez definida la ontología, fue necesario incluir en ella los conceptos y relaciones destinados a servir de base para la generación de ontologías específicas. Comenzando por la representación de las clases de los esquemas, se incluyó una primera clase denominada *SubjectConcept* o concepto temático. Este concepto expresa la semántica de un elemento de tipo clase, contextualizada por el esquema

donde se define. Se muestra a continuación su definición extraída de la ontología SEMSE:

```
<owl:Class rdf:about="http://purl.org/semse/SubjectConcept">
        <isDefinedBy rdf:resource="http://purl.org/semse/"/>
        <subClassOf rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <subClassOf rdf:parseType="Resource">
                <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
        <owl:minCardinality>1</owl:minCardinality>
        <owl:onProperty rdf:resource="http://purl.org/semse/hasSemset"/>
  </subClassOf>
  <owl:disjointWith rdf:resource="http://purl.org/semse/Semset"/>
  <owl:disjointWith rdf:resource="http://purl.org/semse/SubjectProperty"/>
  <vs:term_status>testing</vs:term_status>
<skos:definition xml:lang="en-EN">Subject concept es un concepto concreto que representa la semántica de un
concepto o clase definido en un esquema de metadatos.</skos:definition>
  <skos:prefLabel xml:lang="en-EN">subject concept</skos:prefLabel>
</owl:Class>
```

Ejemplo II-4. Codificación de la clase SubjecConcept incluida en la ontología SEMSE

Definido el concepto para albergar las clases de los esquemas, fue preciso incluir el equivalente para las propiedades. El objetivo es representar la semántica de las propiedades, definida en el esquema del cual provienen. De este modo, se definió una clase denominada *SubjectProperty* o propiedad temática, que representa la semántica de una propiedad, contextualizada por el esquema en el que se incluye. La codificación del concepto se muestra a continuación:

```
<owl:Class rdf:about="http://purl.org/semse/SubjectProperty">
        <isDefinedBy rdf:resource="http://purl.org/semse/"/>
        <subClassOf rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <subClassOf rdf:parseType="Resource">
                <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
        <owl:minCardinality>1</owl:minCardinality>
<owl:onProperty rdf:resource="http://purl.org/semse/hasPropertySemset"/>
  </subClassOf>
  <owl:disjointWith rdf:resource="http://purl.org/semse/Semset"/>
  <owl:disjointWith rdf:resource="http://purl.org/semse/SubjectConcept"/>
  <vs:term_status>testing</vs:term_status>
<skos:definition xml:lang="en-EN"> Subject property es un concepto concreto que representa la semántica de
una propiedad definida en un esquema de metadatos.</skos:definition>
  <skos:prefLabel xml:lang="en-EN">subject property</skos:prefLabel>
</owl:Class>
```

Ejemplo II-3. Codificación de la clase SubjectProperty incluida en la ontología SEMSE

Una vez incluidos los conceptos para albergar la semántica de los elementos, el siguiente paso fue definir la estructura necesaria para albergar las distintas etiquetas o nombres que puede tener el elemento.

Fue por ello que se utilizó otro concepto, denominado *Semset*, para albergar al menos una etiqueta preferente y cero o varias etiquetas alternativas. La primera indica el nombre con el que se identifica el elemento en el esquema y las otras, en su caso, otras formas alternativas de referirse al elemento. La codificación del concepto Semset se muestra a continuación:

```
<owl:Class rdf:about="http://purl.org/semse/Semset">
        <isDefinedBy rdf:resource="http://purl.org/semse/"/>
<subClassOf rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
        <subClassOf rdf:parseType="Resource">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Restriction"/>
        <owl:minCardinality>1</owl:minCardinality>
<owl:onProperty rdf:resource="http://www.w3.org/2004/02/skos/core#prefLabel"/>
  </subClassOf>
        <owl:disjointWith rdf:resource="http://purl.org/semse/SubjectConcept"/>
        <owl:disjointWith rdf:resource="http://purl.org/semse/SubjectProperty"/>
  <vs:term_status>testing</vs:term_status>
<skos:definition xml:lang="en-EN">Semset es un término semánticamente próximo o sinónimo a un Concepto
o Propiedad. Todo concepto y propiedad tendrán un semset por cada idioma representado.</skos:definition>
  <skos:prefLabel xml:lang="en-EN">semset</skos:prefLabel>
</owl:Class>
```

Ejemplo II-5. Codificación del concepto Semse incluida en la ontología SEMSE

Una vez se incluyó el concepto que permite albergar las distintas denominaciones de un concepto, sólo quedaba definir las propiedades que permitieran relacionar los conceptos y propiedades temáticas con sus respectivos conjuntos de etiquetas. Con este fin se definieron dos propiedades, *hasSemset y hasPropertySemset*. La primera de ellas permite relacionar un Semset con un SubjectConcept, la segunda hace lo mismo entre un Semset y un SubjectProperty. La codificación de ambas propiedades se incluye a continuación:

```
<owl:ObjectProperty rdf:about="http://purl.org/semse/hasSemset">
        <isDefinedBy rdf:resource="http://purl.org/semse/"/>
        <domain rdf:resource="http://purl.org/semse/SubjectConcept"/>
        <range rdf:resource="http://purl.org/semse/Semset"/>
<subPropertyOf rdf:resource="http://www.w3.org/2004/02/skos/core#related"/>
  <vs:term_status>testing</vs:term_status>
<skos:definition xml:lang="en-EN">Propiedad que permite enlazar un concepto con sus semsets.</skos:definition>
        <skos:prefLabel xml:lang="en-EN">has semset</skos:prefLabel>
</owl:ObjectProperty>
```

Ejemplo II-6. Codificación de la propiedad HasSemset incluida en la ontología SEMSE

```
<owl:ObjectProperty rdf:about="http://purl.org/semse/hasPropertySemset">
        <isDefinedBy rdf:resource="http://purl.org/semse/"/>
        <domain rdf:resource="http://purl.org/semse/SubjectProperty"/>
        <range rdf:resource="http://purl.org/semse/Semset"/>
<subPropertyOf rdf:resource="http://www.w3.org/2004/02/skos/core#related"/>
        <vs:term_status>testing</vs:term_status>
<skos:definition xml:lang="en-EN">Propiedad que permite enlazar una propiedad con sus semsets.</skos:definition>
        <skos:prefLabel xml:lang="en-EN">has property semset</skos:prefLabel>
</owl:ObjectProperty>
```

Ejemplo II-7. Codificación de la propiedad HasPropertySemset incluida en la ontología SEMSE

De forma adicional y con la intención de abordar el tema del plurilingüismo, al generar las ontologías específicas como descripciones basadas en la ontología SEMSE, se incluyen, tanto para las definiciones de los conceptos como para los Semsets relacionados con SubjectConcepts y SubjectProperties, una versión para cada idioma que se desee soportar. De este modo es posible contemplar tantos idiomas como se desee en la definición de la ontología específica, esto es, en la representación de la semántica de los elementos.

Una vez definida y codificada la ontología SEMSE, padre de las ontologías específicas, el siguiente paso es generar las ontologías específicas a partir de los esquemas originales. Al igual que en la ontología SEMSE, el lenguaje seleccionado para la definición de las ontologías específicas fue OWL, complementado con el uso del vocabulario SKOS en los casos en los que ha pretendió proporcionar mayor semántica a las descripciones de los conceptos. Como en los anteriores casos, cada ontología es

publicada con una URL persistente y en este caso, el nombre proviene del nombre del esquema original seguido del acrónimo 'so' (*specific ontology*).

2.4.3 Búsqueda y selección de esquemas y su representación semántica

El siguiente paso dentro del proceso de representación ontológica de esquemas consistió en la selección de un primer conjunto de esquemas de metadatos sobre el cual poner en práctica la representación semántica propuesta.

Con este objetivo, se inició un proceso de evaluación de esquemas de metadatos, basado en los siguientes criterios:

- Ámbito: El conjunto de esquemas debían de provenir de distintos dominios de conocimiento con el fin de demostrar cómo se logra la interoperatividad de esquemas definidos para distintos dominios.
- Estandarización: Era necesario que los esquemas tuvieran distinto grado de estandarización para mostrar, por un lado, como la propuesta no depende de este aspecto y, por otro, como permitía completar y formalizar la definición de los esquemas, tanto en su estructura como en su semántica.
- Estabilidad: Como en el caso de la estandarización, era necesario incluir esquemas con distinto nivel de estabilidad para mostrar cómo la propuesta no depende de este aspecto, así como para evaluar cómo respondía a problemas derivados de este criterio, principalmente, modificaciones en la definición del esquema original.
- Popularidad: Cuanto más popular es un esquema, mayor es su uso y mayor número de descripciones, documentos, habrán sido generados a partir de él. Dentro de cada ámbito concreto, se ha seleccionado el esquema más popular, teniendo en cuenta los criterios anteriores.

A partir de los criterios expuestos, el conjunto de esquemas seleccionados fue:

- *DCMI Dublin Core Elements Set* (dcelements)
- *W3C vCard Ontology* (vcard-rdf) (World Wide Web Consortium, 2010b)
- *Friend of a Friend* (foaf) (Brickleyy Miller, 2011)
- *Description of a Career* (doac) (Parada, 2008)
- *Description of a Project* (doap) (Dumbill, 2008)
- *Personal Information Markup* (pim) (Berners-Lee, 2007)

2.4.3.1 Representación semántica de los esquemas seleccionados

En este apartado se detallan los pasos seguidos para la representación semántica de los esquemas que se seleccionaron como muestra en el proyecto SEMSE. Como ya se ha detallado en apartados anteriores, esta representación consta de dos partes: la creación de un nuevo esquema semánticamente cualificado y de una ontología específica para cada uno de ellos.

Cualificación semántica

Comenzando el proceso de cualificación por la generación de la versión interpretable por personas, tal y como se planteaba anteriormente, el objetivo era generar un nuevo esquema cualificado semánticamente, en formato HTML y publicado en Web, de forma que esté accesible para su localización y uso. Para la generación de dicho documento aplicable a cualquier esquema de metadatos son necesarios los siguientes pasos:

1. Incluir un registro de estado del documento. El objeto de este registro es proporcionar información sobre el documento y su estado. Entre esta información se encuentra el título, autor, identificador del documento, etc.
2. Incluir información relevante para la interpretación del esquema.
    - Incluir el objetivo del esquema: Incluir un breve resumen del objetivo del esquema. En el caso del esquema seleccionado como ejemplo, el objetivo es la definición de los elementos que desambiguan semánticamente los elementos del esquema de metadatos de Dublin Core.

- Incluir el ámbito del esquema: Incluir una breve descripción del ámbito de publicación y uso del esquema. En el esquema seleccionado, en el ejemplo, el ámbito es público para su uso por cualquier persona u organización.
- Incluir definiciones, acrónimos y abreviaturas que faciliten la interpretación del esquema. En el esquema seleccionado, podrían incluirse distintas definiciones como el término *espacio de nombres* o acrónimos como DCMI.
- Incluir referencias útiles para la interpretación del esquema y a otros esquemas utilizados en su definición. En el esquema seleccionado, podrían incluirse referencias a documentos del DCMI o a esquemas como el de Dublin Core.
- Describir los atributos necesarios para la definición de cada elemento del esquema. Se recomienda la definición de un conjunto mínimo y un adicional recomendado.

3. Incluir un elemento cualificado semánticamente para cada elemento del esquema original. Siguiendo el conjunto de atributos definido en el punto anterior, definir cada término proporcionando la información correspondiente según cada atributo.
4. Incluir el extracto del documento de gramática para aquellos tipos de términos definidos para el esquema y no incluidos en el esquema original. En el caso de no definirse nuevos tipos, el apartado no será de aplicación y deberá etiquetarse como *No Aplicable*. Las referencias a los documentos de gramática donde se definen los tipos utilizados en el esquema original, deberán estar incluidas en el apartado de referencias.
5. Incluir la versión RDF del esquema cualificado semánticamente (opcional). En caso de contarse con la versión RDF del esquema, o una vez generada (como se verá en el siguiente apartado), se recomienda la inclusión de la misma en este documento.

Una vez definida la versión del esquema interpretable por personas, se procederá a definir la versión interpretable por computadoras. Dicha versión, como ya se ha descrito en apartados anteriores consiste en la codificación de los elementos, definidos en la versión ya desarrollada, haciendo uso del lenguaje RDF. Para ello, además de incluir todos los espacios de nombres y descripciones como el título, el autor, el publicador, etc. se incluye un elemento cualificado por cada elemento del esquema original.

En la siguiente tabla se puede observar los esquemas cualificados generados en el proyecto SEMSE:

Esquema original	Esquema cualificado	Espacio de nombres
Dcelements	sqdcelements	http://purl.org/semse/sqdcelements
Vcard-rdf	sqvcard	http://purl.org/semse/sqdcelements
FOAF	sqfoaf	http://purl.org/semse/sqfoaf
DOAC	sqdoac	http://purl.org/semse/sqdoac
DOAP	sqdoap	http://purl.org/semse/sqdoap
PIM	sqpim	http://purl.org/semse/sqpim

Tabla II-1. Resumen de los esquemas cualificados generados

2.4.3.2 Representación ontológica de los esquemas seleccionados

Una vez creadas las versiones cualificadas de los esquemas originales, el siguiente paso es generar las ontologías específicas correspondientes a dichos esquemas.

Cabe destacar que, a diferencia de los esquemas cualificados semánticamente, las ontologías específicas no están dirigidas al usuario final y no precisan, en principio, de una versión amigable para el usuario final.

Siguiendo el planteamiento propuesto en el apartado *2.4.2 Ontologías específicas* se han generado las ontologías correspondientes a los esquemas seleccionados en este apartado . Concretamente, los pasos se pueden resumir en:

1. Incluir el conjunto de metadatos que define la ontología específica, incluyendo: nombre, título, descripción, localización, publicador, idioma, fecha de publicación y versión.
2. Incluir, para cada elemento definido en el esquema original, la representación semántica según corresponda: *SubjectConcept* si el elemento original es una clase o *SubjectProperty* si el elemento original es una propiedad. Destacar que, en este punto, es fundamental incluir la definición precisa del término para cada uno de los lenguajes que se deseen cubrir. El grado de estandarización del esquema original es un aspecto clave dado que, como se ha podido comprobar en durante el proceso, aquellos esquemas con mayor grado de estandarización incluyen definiciones precisas para todos sus elementos. Esto no sucede para aquellos esquemas poco estandarizados, como doac, doap o pim. En estos casos fue necesario completar, precisar, e incluso definir, la semántica de varios de sus elementos, siempre dentro del contexto semántico establecido por el esquema. Por contexto semántico del esquema debe entenderse el significado que, dentro del esquema, se le da al término. De este modo, un elemento puede tener diferente semántica, en función del esquema que lo define. Las definiciones modificadas durante este proceso, y que difieren de las incluidas en el esquema original, han sido etiquetadas como *[SEMSE]* para su posterior identificación.
3. Incluir, para cada elemento representado, el *Semset* asociado, uno por cada idioma que se desee cubrir, donde se incluye la etiqueta preferente del elemento y sus etiquetas alternativas o sinónimos. En este punto destacar que todos los conceptos y propiedades poseen una etiqueta preferente y, sólo algunos, etiquetas alternativas. Es importante tener en cuenta que las etiquetas alternativas se obtuvieron haciendo uso de diccionarios de definiciones y sinónimos, siempre dentro del contexto semántico definido por el esquema. Así, el conjunto de sinónimos de un mismo elemento puede diferir en función del esquema en el que se define.

Aplicando el método propuesto para la generación de ontologías específicas, se ha procedido a generar una ontología específica para cada esquema seleccionado:

Esquema original	Ontología específica	Espacio de nombres
Dcelements	dcelementsso	http://purl.org/semse/dcelementsso
Vcard-rdf	vcardso	http://purl.org/semse/vcardso
FOAF	foafso	http://purl.org/semse/foafso
DOAC	doacso	http://purl.org/semse/doacso
DOAP	doapso	http://purl.org/semse/doapso
PIM	pimso	http://purl.org/semse/pimso

Tabla II-2. Resumen de las ontologías específicas generadas

2.4.4 Selección de la ontología de referencia

Una vez publicada la representación semántica de los esquemas, el siguiente paso consistió en seleccionar una ontología de referencia que permitiera la desambiguación semántica de los conceptos incluidos en dichos esquemas.

Para realizar una valoración de las ontologías fue necesario establecer un conjunto de requisitos a satisfacer, sobre el cual realizar la selección. Dichos requisitos fueron los siguientes:

- Extensible/modificable: Es importante el hecho de la ontología permitiera su modificación y extensión. El porqué es la necesidad futura de añadir nuevos conceptos, relaciones, axiomas, etc., para la representación de esquemas. Este requisito se basa en la suposición de que no va a ser posible encontrar una ontología de referencia que cubra la semántica de los elementos para cualquier esquema.

- Consultable/navegable: Es necesario que la ontología permitiera su acceso para realizar consultas sobre conceptos, así como la navegación a partir de las relaciones entre ellos. Valor fundamental para la realización de este PFC.
- Lenguaje: Se valoró positivamente el hecho de que la ontología se encuentra en alguno de los lenguajes recomendados para la Web Semántica, principalmente OWL. No obstante, la mayoría de los ejemplos encontrados no cumplían esta característica (aunque varios de ellos se encuentraban en proceso de conversión a OWL).
- Referenciable vía identificación formal: Relacionado con el siguiente punto, era preciso que la ontología incluyera algún modo de identificación unívoca de los conceptos. Teniendo en cuenta que el contexto de publicación de la propuesta será la Web, se valoraron positivamente aquellas ontologías que utilizaban URIs como sistema de identificación formal.
- Descargable: En algunos casos, las ontologías encontradas permitían su descarga para su uso. De este modo se facilita su modificabilidad, aunque poseen el inconveniente de precisar publicación y mantenimiento locales.
- Popularidad: Se valoró la popularidad de las ontologías en base a los proyectos e iniciativas en los que se han utilizado.
- Mantenimiento/Actualidad: Se valoró positivamente aquellas ontologías que poseyeran alguna entidad que actualizase y mantuviese sus conceptos y relaciones, garantizando, además, la disponibilidad del recurso.
- Eficiencia: Se ha valoró positivamente la eficiencia del recurso, referida a tiempo de consulta e inferencia. En este caso, como es lógico, el número de conceptos y relaciones incluidos en la ontología influyen en el tiempo de respuesta del recurso.
- Completitud: La completitud del recurso se valoró positivamente aunque con menor prioridad que la extensibilidad y la eficiencia. El porqué es la necesidad de extensibilidad del recurso, mencionada anteriormente, así como la de realizar consultas en tiempos aceptables para el usuario.

- Estructura: Se valoró positivamente aquellas ontologías que presentan una estructura clara y sencilla en cuanto a conceptos y relaciones básicas. Una estructura de este tipo facilita la adición de nuevos conceptos, así como los procesos de consulta e inferencia.

Tras un completo estudio sobre las ontologías disponibles, entre las que se encontraban ontologías como WordNet(Miller, 1995), WebKB-2(webKB.org, 2003) o UMBEL (Umbel.org, 2008) se concluyó que la ontología PROTON (Terziev, et al., 2005) era la más adecuada debido a que:

- Posee una estructura jerárquica, modular y sencilla, con amplia cobertura semántica, tomada parcialmente de Dolce.
- Está diseñada para ser extendida de modo que, aunque no cubre dominios específicos, sí proporciona la estructura que facilita su inclusión.
- Se define utilizando OWL, lo que facilita su publicación y la integración con ontologías publicadas un entorno Web.
- Otras de sus características son: inclusión de definiciones ligeras de conceptos y relaciones, alineamiento con estándares y documentación completa.

Adicionalmente, se realizó una valoración de la cobertura semántica del recurso, con el fin de comprobar la cobertura de los significados de los conceptos incluidos en los esquemas seleccionados. Se pudo comprobar cómo el esquema de muestra, los elementos de metadatos de Dublin Core, estaba prácticamente alineado con PROTON. Del mismo modo, una segunda valoración con FOAF mostró también una cobertura parcial. En ambos casos, la inclusión de la semántica necesaria era viable, gracias a las posibilidades de extensión que brinda la ontología.

Una vez seleccionada la ontología de referencia, y habiendo comprobado que cubría las necesidades de la propuesta, el objetivo siguiente fue el desarrollo de la representación de dicha ontología de forma que sea compatible e integrable con el resto de artefactos desarrollados en el proyecto SEMSE.

Tal y como se desarrollaba al comienzo de este apartado, la representación de la ontología debía de cubrir los siguientes requisitos:

- Debe permitir la extensión y la modificación de la ontología de referencia, ya que será preciso incluir nuevos términos y precisar algunos de los existentes en la ontología. Con este fin, la solución más recomendable pasaría por mantener la ontología de referencia original como componente independiente en el proceso de extensión, de modo que las actualizaciones no precisen la modificación de la ontología de referencia original.
- Debe permitir la consulta de los conceptos incluidos en la ontología así como de los añadidos durante el proceso de especialización/extensión. Asimismo debe mantener las relaciones entre conceptos que permitan la navegación entre ellos, tanto para los pertenecientes a la ontología de referencia original como para los añadidos/especializados con posterioridad.
- Debe estar expresada en un lenguaje que permita la publicación: compartición, acceso y relación, en un entorno Web. Dicho lenguaje, además, deberá soportar la representación de semántica de conceptos y relaciones.
- Debe hacer uso de un sistema de identificación formal para la identificación unívoca de los conceptos incluidos. En relación con el punto anterior, el sistema de identificación formal deberá estar, preferiblemente, basado en URIs.
- Debe permitir su descarga. También en relación con la publicación, debe permitirse el acceso y descarga de la ontología de referencia así como de sus extensiones.
- Debe facilitar el mantenimiento y la actualización de la ontología de referencia original. Previendo futuras actualizaciones en la ontología de referencia, por parte de la entidad responsable de su desarrollo, la representación debe permitir dicho proceso y minimizar el impacto que dichos cambios puedan producir. La independencia de la ontología original de los componentes de extensión permitiría, ante una actualización del recurso original, realizar el análisis y posibles modificaciones únicamente en dichos componentes de extensión, siendo la ontología de referencia original independiente en dicho proceso.

- Debe garantizar la eficiencia del recurso. Logrado en parte por el tipo de recurso seleccionado, la representación debe permitir la inclusión incremental de aquellos términos que sean pertinentes en el proceso de extensión, evitando la necesidad de incluir términos espúreos o semánticamente no relacionados.
- Debe facilitar la completitud del recurso. Aunque esta característica ha sido tratada con menor prioridad en pro de la extensibilidad y la eficiencia, la representación debe facilitar la completitud a través de la facilidad de extensión y el ámbito de la ontología seleccionada.
- Debe proporcionar una estructura clara y compatible con la de la ontología de referencia original. La representación debe mantener la sencillez de la estructura de la ontología de referencia y ser compatible con ella, teniendo en cuenta que podrían utilizarse múltiples ontologías de referencia, además de la seleccionada.

A partir de estos requisitos, se propone como representación la definición de una ontología de aplicación que, a través de mecanismos de importación, relacione las extensiones con la ontología de referencia original. De este modo, la ontología de aplicación dependería de la ontología de referencia pero mantendría como elemento independiente a la ontología original.

Más concretamente, la representación implica la definición de una ontología de aplicación para cada ontología de referencia que se desee utilizar. Dentro de la ontología de aplicación se incluirán los conceptos y propiedades, no incluidos en la ontología de referencia original, así como las modificaciones sobre los elementos originales, es decir, las extensiones de la ontología original. El lenguaje de definición de la ontología de aplicación será OWL, habilitándose la inclusión de relaciones con los conceptos y propiedades de la ontología original, así como la publicación Web del recurso. La definición de conceptos y propiedades se realizará haciendo uso de URIs como sistema de identificación formal, lo que facilitará su consulta y acceso, una vez publicado el recurso.

La denominación de las ontologías de aplicación consistirá en el nombre de la ontología de referencia seguido del acrónimo 'a' (*application*), y deberá de estar identificada con una URI persistente. Para el caso de la ontología de referencia PROTON, la URI que la identifica es *http://purl.org/semse/proton/protona#*, siendo el carácter '#' un identificador de que se trata de un subdominio de la ontología de aplicación, en este caso, *protona*.

2.4.5 Correspondencia entre los esquemas y la ontología de referencia

Una vez desarrolladas las representaciones semánticas de los esquemas y seleccionada la ontología de referencia, se llevó a cabo la interrelación de los conceptos incluidos en los esquemas semánticos con los incluidos en la ontología de referencia.

En primer lugar es necesario establecer el modo de interrelacionar las representaciones semánticas de los esquemas con la ontología de referencia. Para ello deberán analizarse los métodos existentes y seleccionar aquel que mejor se adecúe a la propuesta.

Una vez evaluados distintos métodos se definió un método de correspondencia que tiene las siguientes características:

- La información de entrada a procesar será la relativa a la definición del esquema. La información referente a las instancias será recuperada posteriormente, a través de un proceso de rastreo web.
- El alineamiento se realizará de forma manual. Se pretende, de este modo, obtener la máxima calidad y fiabilidad en las correspondencias generadas.
- No se devolverán correspondencias incorrectas. Relacionado con el punto anterior, se asume que todas las correspondencias de los alineamientos son correctas.
- El conjunto de correspondencias devuelto como resultado de un proceso de consulta será completo, esto es, se devolverán todas las correspondencias.

- La correspondencia entre esquemas y la ontología de referencia se realizará en tiempo de diseño, debido al carácter del sistema y a la posibilidad de obtener el alineamiento previo a la ejecución del sistema.
- La función de alineamiento hará uso de un algoritmo híbrido que incluirá técnicas terminológicas, estructurales y semánticas. La técnica terminológica se basará en la comparación de las etiquetas de los elementos del esquema y la ontología de referencia, lo que proporcionará un conjunto de correspondencias tentativas. En paralelo se hará uso de la estructura proporcionada por la ontología de referencia con el fin de obtener nuevas correspondencias y refinar las anteriores. Finalmente, las correspondencias serán validadas o descartadas mediante la semántica de los conceptos, definida en el esquema y en la ontología de referencia.

Una vez seleccionado el método de correspondencia e inspirándose en el recurso de correspondencia incluido en la ontología UMBEL, se propuso la creación de una ontología de alineamiento que consistiera en un conjunto de reglas en las que se establecieran las relaciones entre conceptos de las ontologías específicas y de referencia. Partiendo de la posibilidad de utilizar múltiples ontologías de referencia, se definirá una ontología de alineamiento por cada ontología de referencia. Más concretamente, para cada ontología de referencia se definirá una ontología para albergar el alineamiento “interno” (con conceptos de las ontologías específicas) y otra para albergar el alineamiento “externo” (con conceptos de ontologías externas al sistema). Aunque en el proyecto SEMSE no se definieron relaciones con ontologías externas, se definió el recurso de cara a cubrir la necesidad de ampliar la ontología de referencia, incluyendo relaciones entre ésta y recursos externos, ampliándose de este modo su contexto.

Respecto a las relaciones definidas es preciso destacar varios aspectos importantes. En primer lugar, las reglas siempre van a tener dos partes, en primer lugar incluirán una referencia a un concepto de una ontología específica, en segundo lugar incluirán una referencia a un concepto de la ontología de referencia contra la cual relacionan. En segundo lugar, las reglas establecidas son bidireccionales, de modo

que establecen relaciones de equivalencia entre ambos elementos. En tercer lugar la semántica de las relaciones, expresada como equivalencia (*equivalent*), debe interpretarse como proximidad semántica y no como igual-a (*same-as*). De este modo es posible interrelacionar elementos con distinta semántica específica, definida en contexto del esquema que los incluye, con un mismo concepto "próximo", definido en la ontología de referencia. Finalmente, destacar que se hizo uso de las relaciones de equivalencia definidas, de forma nativa, por el lenguaje OWL (*equivalentClass* y *equivalentProperty*). No obstante, debería ser posible definir relaciones propias, por ejemplo con semánticas específicas.

Respecto a la política de nombrado, las ontologías de alineamiento recibirán el nombre de la ontología de referencia que extienden, seguido de la etiqueta '*specific_ontologies_linkage*', si es alineamiento interno, o '*external_ontologies_linkage*'. Respecto a la estructura, partiendo de la URL base, se han definido subdominios para cada una de las ontologías de referencia, dentro de los cuales, se ubican la ontología de aplicación correspondiente. De este modo, cada representación tiene un subdominio asignado, establecido como *internalmapping* y *externalmapping*, que permite el acceso unívoco a su contenido. En el caso de la ontología de alineamiento interno generada para la ontología de referencia PROTON, se generó una URI persistente para denominar su subdominio asignado: *http://purl.org/semse/proton/internalmapping/*.

Capítulo III. Herramientas para la elaboración del proyecto

En el presente capítulo se presenta el análisis y la elección de la infraestructura del servidor utilizado, así como las herramientas y tecnologías empleadas en la elaboración de este proyecto.

Para ello, el capítulo de ha dividido en cuatro puntos bien diferenciados, en los que se detallan por un lado, la infraestructura del servidor, detallando el sistema operativo utilizado, el servidor Web, el servidor de aplicaciones y el sistema gestor de base de datos; por otro, el entorno de trabajo, en el que se describe el lenguaje de programación e IDE utilizados; a continuación se describen los frameworks que se han empleado, desde la capa de presentación hasta la de negocio; y finalmente se detalla el uso de otras herramientas utilizadas para tareas auxiliares y que quedan fuera de las agrupaciones anteriores.

3.1 Infraestructura del servidor

A la hora de la elaboración de un proyecto o del despliegue de un determinado servicio una de las tareas que se deben hacer tras la captación de requisitos, es el diseño de la arquitectura hardware que se va a necesitar. En este proyecto la máquina disponible para el desarrollo se trata de un PC con esta configuración:

- Procesador Intel Core 2 Quad a 2.4 Ghz.
- 4 GB de memoria RAM.
- 500 GB de HDD (hard disk drive)

3.1.1 Sistema Operativo

La elección del sistema operativo es fundamental y generalmente debe hacerse acorde a la especificación de requisitos del proyecto en cuestión, obviando motivaciones económicas, sociales o filosóficas. Se eligió GNU/Linux como sistema operativo del servidor por las siguientes razones(Escartín Vigo, 2005):

- Es más seguro: Ya que la gran mayoría de los ataques de hackers son dirigidos a equipos con el sistema operativo Microsoft Windows, debido a que prácticamente el 90% (Brodkin, 2011) de los equipos en el mundo usan este sistema operativo, por lo que los virus y ataques de hackers resultan más lucrativos si son dirigidos a dicho sistema. La plataforma Linux es más robusta lo cual hace más difícil que algún intruso pueda violar la seguridad del sistema.
- Es más rápido: La eficiencia de su código fuente hace que la velocidad de las mismas aplicaciones corriendo en Linux sean superiores a las que corren sobre otros sistemas operativos. Además los requisitos mínimos para el correcto funcionamiento de Linux resultan ser muy inferiores a los de otros sistemas.
- Es más económico: Debido precisamente a las dos razones anteriores, las tareas de supervisión y las frecuentes tareas de monitoreo contra ataques de

virus, hackers y errores de código que requieren otras plataformas, hacen que Linux sea un sistema menos costoso de mantener. El software Linux, así como también un sin fin de aplicaciones, son de código abierto y están protegidas por la licencia GPL1[14], motivo por el que son distribuidas gratuitamente.

Dentro del sistema operativo GNU/Linux, hay una amplia variedad de distribuciones, que no son más que colecciones de software y maneras de empaquetado del mismo, pero que todas ellas corren encima de un Kernel Linux. En lo que se refiere a servidores tenemos las siguientes distribuciones más usuales:

- **Ubuntu y Mepis** (basadas en Debian): se pueden utilizar sin tener grandes conocimientos de Linux puesto que están orientadas a la utilización de un potente interfaz gráfico, además de incluir numerosos asistentes de usuario para la correcta configuración y utilización del equipo.
- **Gentoo, Debian o Slackware**: Estas distribuciones tienen una curva de aprendizaje muy pronunciada antes de poder ser utilizadas de manera correcta, pero que una vez superado ese estadio, ofrecen un sistema estable, robusto y prácticamente sin mantenimiento.
- **LUC3M, Knoppix** (también basadas en Debian): son un caso especial de distribuciones, que pueden arrancar directamente desde un CD o un DVD y que permiten probar su funcionamiento sin tener que instalar nada.

La distribución elegida ha sido Ubuntu en su versión estable a fecha de instalación de este sistema operativo. Ubuntu (doc.Ubuntu-es, 2011) está basado en Debian, pero el planteamiento está inspirado en los principios de la corriente ubuntu, un movimiento humanista encabezado por el obispo Desmond Tutu, premio Nobel de la Paz en 1984. Económicamente el proyecto se sostiene con aportaciones de la empresa Canonical Ltd. del millonario sudafricano Mark Shuttleworth y está mantenido por una incipiente comunidad que no para de crecer.

[14] La Licencia Pública General de GNU (GNU GPL) posibilita la modificación y redistribución del software, pero únicamente bajo esa misma licencia.

La distribución Ubuntu, está basada en los principios del desarrollo de software libre, ofrece un sistema operativo predominantemente enfocado a ordenadores de escritorio aunque también proporciona soporte para servidores. Ubuntu concentra su objetivo en la facilidad de uso, la libertad de uso, los lanzamientos regulares y la facilidad en la instalación. Ubuntu incluye una cuidadosa selección de los paquetes de Debian, y mantiene su poderoso sistema de gestión de paquetes que nos permite instalar y desinstalar programas de una forma fácil y limpia. Con la mirada puesta en la calidad, Ubuntu proporciona un entorno robusto y funcional, adecuado tanto para uso doméstico como profesional. Se publica una nueva versión cada seis meses y se mantienen actualizadas en materia de seguridad hasta 18 meses después de su lanzamiento, excepto para las versiones LTS (*Long term support*) que ofrece 3 años para la versión escritorio y 5 años para la versión servidor, a partir de la fecha del lanzamiento.

3.1.2 Servidor Web

Un servidor web (Wikipedia, 2011d) es un programa que implementa el protocolo HTTP (hypertext transfer protocol). Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas web o páginas HTML (hypertext markup language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de sonidos.

Sin embargo, el hecho de que HTTP y HTML estén íntimamente ligados no debe dar lugar a confundir ambos términos. HTML es un formato de archivo y HTTP es un protocolo.

Cabe destacar el hecho de que la palabra servidor identifica tanto al programa como a la máquina en la que dicho programa se ejecuta. Existe, por tanto, cierta ambigüedad en el término, aunque no será difícil diferenciar a cuál de los dos nos referimos en cada caso. En este apartado nos referiremos siempre a la aplicación.

Un servidor web se encarga de mantenerse a la espera de peticiones HTTP llevada a cabo por un cliente HTTP que solemos conocer como navegador. El navegador realiza una petición al servidor y éste le responde con el contenido que el cliente solicita. A modo de ejemplo, al teclear *www.wikipedia.org* en un navegador, éste realiza una petición HTTP al servidor de dicha dirección. El servidor responde al cliente enviando el código HTML de la página; el cliente, una vez recibido el código, lo interpreta y lo muestra en pantalla. Como vemos con este ejemplo, el cliente es el encargado de interpretar el código HTML, es decir, de mostrar las fuentes, los colores y la disposición de los textos y objetos de la página; el servidor tan sólo se limita a transferir el código de la página sin llevar a cabo ninguna interpretación de la misma. Sobre el servicio web clásico podemos disponer de aplicaciones web. Éstas son fragmentos de código que se ejecutan cuando se realizan ciertas peticiones o respuestas HTTP. Hay que distinguir entre:

- **Aplicaciones en el lado del cliente.** El cliente web es el encargado de ejecutarlas en la máquina del usuario. Son las aplicaciones tipo Java o Javascript: el servidor proporciona el código de las aplicaciones al cliente y éste, mediante el navegador, las ejecuta. Es necesario, por tanto, que el cliente disponga de un navegador con capacidad para ejecutar aplicaciones (también llamadas scripts). Normalmente, los navegadores permiten ejecutar aplicaciones escritas en lenguaje Javascript y Java, aunque pueden añadirse más lenguajes mediante el uso de plugins.
- **Aplicaciones en el lado del servidor.** El servidor web ejecuta la aplicación; ésta, una vez ejecutada, genera cierto código HTML; el servidor toma este código recién creado y lo envía al cliente por medio del protocolo HTTP.

Las aplicaciones de servidor suelen ser la opción por la que se opta en la mayoría de las ocasiones para realizar aplicaciones web. La razón es que, al ejecutarse ésta en el servidor y no en la máquina del cliente, éste no necesita ninguna capacidad adicional, como sí ocurre en el caso de querer ejecutar aplicaciones Javascript o Java. Así pues, cualquier cliente dotado de un navegador web puede utilizar la aplicación sin requerir ningún otro medio o instalación.

Para la realización de este proyecto, siguiendo la línea de mínimo coste y dados los requerimientos de la aplicación, se ha elegido el servidor web Glassfish incluido en NetBeans y que se detalla en el siguiente apartado.

3.1.3 Servidor de aplicaciones

Se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente (Wikipedia, 2011c). Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Si bien el término es aplicable a todas las plataformas de software, hoy en día el término *servidor de aplicaciones* se ha convertido en sinónimo de la plataforma Java EE de Sun Microsystems. Java EE provee estándares que permiten a un servidor de aplicaciones servir como "contenedor" de los componentes que conforman dichas aplicaciones. Estos componentes, escritos en lenguaje Java, usualmente se conocen como Servlets, Java Server Pages (JSPs) y Enterprise JavaBeans(EJBs) y permiten implementar diferentes capas de la aplicación, como la interfaz de usuario, la lógica de negocio, la gestión de sesiones de usuario o el acceso a bases de datos remotas.

Para la elaboración del proyecto se ha utilizado GlassFish v2 como servidor de aplicaciones.

3.1.3.1 Glassfish v2

GlassFish (Wikipedia, 2010b) es un servidor de aplicaciones de código abierto desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa Java EE 5. La plataforma Java EE 5 incluye las versiones de tecnologías más recientes, como por ejemplo JavaServer Pages (JSP) 2.1, JavaServer Faces (JSF) 1.2, Servlet 2.5, Enterprise JavaBeans 3.0, Java API for Web Services(JAX-WS) 2.0, Java Architecture for XML Binding(JAXB) 2.0, Web Services Metadata for the Java Platform 1.0, y muchas otras nuevas tecnologías.

Glassfish (Vargas Romo, 2009) además de ser un servidor de aplicaciones, es una comunidad de usuarios, que descargan y utilizan libremente Glassfish, existen partners que contribuyen agregándole más características importantes a Glassfish.

Ingenieros y beta testers desarrollan código y prueban las versiones liberadas para eliminar cualquier fallo que se encuentre. La comunidad fue lanzada en el año 2005 en java.net. Glassfish es transparente en cuanto a términos de entrega de código fuente, discusiones de ingeniería, agendas, datos de descarga, etc.

3.1.4 Sistema gestor de bases de datos

Una Base de Datos[15] es una colección estructurada de datos que puede ser, desde una simple lista de artículos, a las inmensas cantidades de información en una red corporativa.

Los sistemas de gestión de base de datos (SGBD) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. El propósito general de los sistemas de gestión de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, para un buen manejo de datos.

Existen distintos objetivos que deben cumplir los SGBD:

- **Abstracción de la información**. Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario. Así, se definen varios *niveles de abstracción*.

[15] Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

- **Independencia**. La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- **Consistencia**. En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. Por otra parte, la base de datos representa una realidad determinada que tiene determinadas condiciones, por ejemplo que los menores de edad no pueden tener licencia de conducir. El sistema no debería aceptar datos de un conductor menor de edad. En los SGBD existen herramientas que facilitan la programación de este tipo de condiciones.
- **Seguridad**. La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos**.**
- **Manejo de Transacciones**. Una Transacción es un programa que se ejecuta como una sola operación. Esto quiere decir, que después de una ejecución en la que se produce un fallo el resultado que obtenemos es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.
- **Tiempo de respuesta**. Lógicamente, es deseable minimizar el tiempo que el SGBD tarda en darnos la información solicitada y en almacenar los cambios realizados.

3.1.4.1 PostgreSQL

Como sistema gestor de base de datos se ha utilizado PostgreSQL versión 8.4, se trata de un servidor de bases de datos relacional orientado a objetos de software libre, publicado bajo la licencia BSD[16]. Que presenta las siguientes características:

Alta concurrencia

Mediante un sistema denominado MVCC (Multiversion Concurrency Control) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla para leer sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo *commit*. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Amplia variedad de tipos nativos

PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.
- Texto de largo ilimitado (*text*).
- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e IPv6).
- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- Arrays.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indizables gracias a la infraestructura GiS de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS[17].

[16] Berkeley Software Distribution. Esta licencia asegura "verdadero" software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir incluso redistribuirlo como no libre.

Claves ajenas

También denominadas Llaves ajenas o Claves Foráneas (foreign keys). Es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (puede ser la misma). Dicho campo hará la función de clave primaria en la tabla referenciada. Las claves ajenas representan relaciones entre datos. Se dice que un valor de clave ajena representa una referencia a la tupla que contiene el mismo valor en su clave primaria (tupla referenciada).

Disparadores (trigger)

Un *disparador* o *trigger* se define en una acción específica basada en algo ocurrente dentro de la base de datos. En PostgreSQL esto significa la ejecución de un procedimiento almacenado basado en una determinada acción sobre una tabla específica. Ahora todos los disparadores se definen por seis características:

- El nombre del trigger o disparador.
- El momento en que el disparador debe arrancar.
- El evento sobre el que se deberá activar el disparador.
- La tabla donde el disparador se activara.
- La frecuencia de la ejecución.
- La función que podría ser llamada.

Vistas

Se puede considerar una *vista* como una *tabla virtual*, es decir, una tabla que no existe físicamente en la base de datos, pero aparece al usuario como si existiese. Se pueden crear *vistas* sobre las consultas (querys), asignándolas un nombre para así poder referenciarlas como si se tratara de una tabla ordinaria.

[17] PostGIS es un módulo que añade soporte de objetos geográficos a la base de datos objeto-relacional PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en Sistema de Información Geográfica. Se publica bajo la Licencia pública general de GNU.

Integridad transaccional

Las transacciones son *atómicas*, es decir, o se realizan todas las operaciones que la componen o no se realiza ninguna. Gracias a ello, si ocurre un error en una operación no se producirá ningún cambio en la base de datos.

Herencia de tablas

Si se crea una tabla que hereda de otra, la nueva tabla contendrá los atributos de la clase padre. En Postgre, una clase puede heredar de ninguna o varias otras clases, y una consulta puede hacer referencia tanto a todas las instancias de una clase como a todas las instancias de una clase y sus descendientes.

3.2 Entorno de trabajo

3.2.1 Java

Java, por su estabilidad y características, será el lenguaje de programación seleccionado para la implementación del sistema. A fecha de la realización del proyecto se contaba con la versión 6 de Java.

A continuación se describen las principales características de este lenguaje (Álvarez Marañón, 1999; Webtaller, 2011):

Lenguaje simple

Java posee una curva de aprendizaje muy rápida. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características para mantener reducidas las especificaciones del lenguaje y añadir otras muy útiles como el *garbage collector* (reciclador de memoria dinámica).

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de sus características, entre las que destacan:

- La aritmética de punteros

- No existen referencias
- Registros (struct)
- Definición de tipos (typedef)
- Macros (#define)
- Necesidad de liberar memoria (free)

Orientado a objetos

Java está totalmente orientado a objetos, proporcionando los mecanismos para que el programador haga utilización de todas las técnicas de diseños y programación orientadas a objetos como la herencia, el polimorfismo, la abstracción, etc. que se detallan más adelante en el *3.2.1.1 Programación orientada a objetos*. El lenguaje va acompañado de numerosas librerías de objetos que cubren todas las áreas, desde los tipos básicos de datos a los interfaces de Entrada/Salida y de red, pasando por el soporte para la construcción de interfaces gráficos de usuario.

Todas estas librerías pueden ser extendidas (aunque con ciertas limitaciones) utilizando mecanismos de herencia para modificar su comportamiento y adaptarlo a las necesidades del programador.

Aunque la orientación a objetos ya está incluida en numerosos lenguajes de programación, Java incorpora nuevas funcionalidades como por ejemplo, la resolución dinámica de métodos. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (*RunTime Type Identification*) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación durante el tiempo de ejecución de un programa que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la consulta.

Distribuido

Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas. Desde el principio, Sun diseñó Java para adaptarse a Internet, esto puede observarse en la posibilidad de desarrollar una serie de aplicaciones especiales denominadas Applets que pueden ser incrustados en páginas HTML mediante la etiqueta <APPLET> y ser ejecutadas en navegadores que soportan esta tecnología. Además, mediante Java RMI (Remote Method Interface) un cliente puede ejecutar acciones en objetos que se encuentran en un servidor independientemente de dónde esté situado.

Interpretado y compilado a la vez

Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes. Los bytecodes generados por el compilador son ejecutados por una parte de la máquina virtual conocida como intérprete. Una vez que el sistema de soporte de ejecución para la máquina virtual y el intérprete han sido portados a una plataforma hardware, todos los programas se pueden ejecutar sin necesidad de recompilación. Al ser interpretado, no existe una fase separada de enlace (link), el enlace es ahora el proceso de cargar nuevas clases a través de la red por el Class Loader.

Su naturaleza interpretada también le permite una mayor rapidez en el ciclo de desarrollo, ya que no es necesario tener el programa totalmente libre de errores para poder ejecutarlo. Hay una mayor facilidad de depuración y los errores pueden ser detectados en fases más tempranas del ciclo de desarrollo.

Robusto

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java es muy estricta, ya que Java no permite realizar declaraciones implícitas, esto ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. El modelo de memoria utilizado en Java elimina muchas de las preocupaciones del programador referentes al uso de la

memoria (reserva, liberación,...) ya que en Java desaparecen los punteros, y se sustituyen las estructuras dinámicas de datos por objetos (Vector,List,...) o por arrays que permiten realizar la comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria resultado de punteros que señalan a zonas equivocadas.

Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java. Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los bytecodes, que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes. Java proporciona:

- Comprobación de punteros.
- Comprobación de límites de arrays.
- Excepciones.
- Verificación de bytecodes.

Seguro

El código Java pasa muchas comprobaciones antes de ejecutarse en una máquina. El código se pasa a través de un verificador de bytecodes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto).

Si los bytecodes pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

- El código no produce desbordamiento de operandos en la pila.
- El tipo de los parámetros de todos los códigos de operación son conocidos y correctos.
- No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros.

- El acceso a los campos de un objeto se sabe que es legal: public, private, protected.
- No hay ningún intento de violar las reglas de acceso y seguridad establecidas.

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo *Caballo de Troya*[18], ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

Las clases importadas de la red se almacenan en un espacio de nombres privado, asociado con el origen. Cuando una clase del espacio de nombres privado accede a otra clase, primero se busca en las clases predefinidas (del sistema local) y luego en el espacio de nombres de la clase que hace la referencia. Esto imposibilita que una clase suplante a una predefinida.

En resumen, las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel[19] del sistema operativo. Además, para evitar modificaciones por parte de usuarios mal intencionados de la red, implementa un método ultra seguro de autentificación por clave pública. El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria, sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

[18] Software malicioso que se presenta al usuario como un software inofensivo pero que al ejecutarse produce daños en el equipo.

[19] Es un software que actúa de sistema operativo, siendo el principal responsable de facilitar a los distintos programas acceso seguro al hardware del equipo o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

Independiente a la arquitectura

Java ha sido desarrollado para crear aplicaciones que funcionen en entornos de red, operando en una amplia variedad de arquitecturas y sistemas operativos. Para conseguir esta independencia, el código fuente Java se compila a un código de bytes de alto nivel independiente de la máquina, este código (bytecode) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema runtime[20], que sí es dependiente de la máquina (Máquina virtual de Java, JVM).

Por esta razón, y debido a la naturaleza interpretada del lenguaje, el mismo bytecode puede ejecutarse sobre cualquier plataforma sin necesidad de recompilar. Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (runtime) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.

[20] Intervalo de tiempo en el que un programa se ejecuta en un sistema operativo. Este tiempo se inicia con la puesta en memoria principal del programa y finaliza en el momento en que éste envía al sistema operativo la señal de terminación.

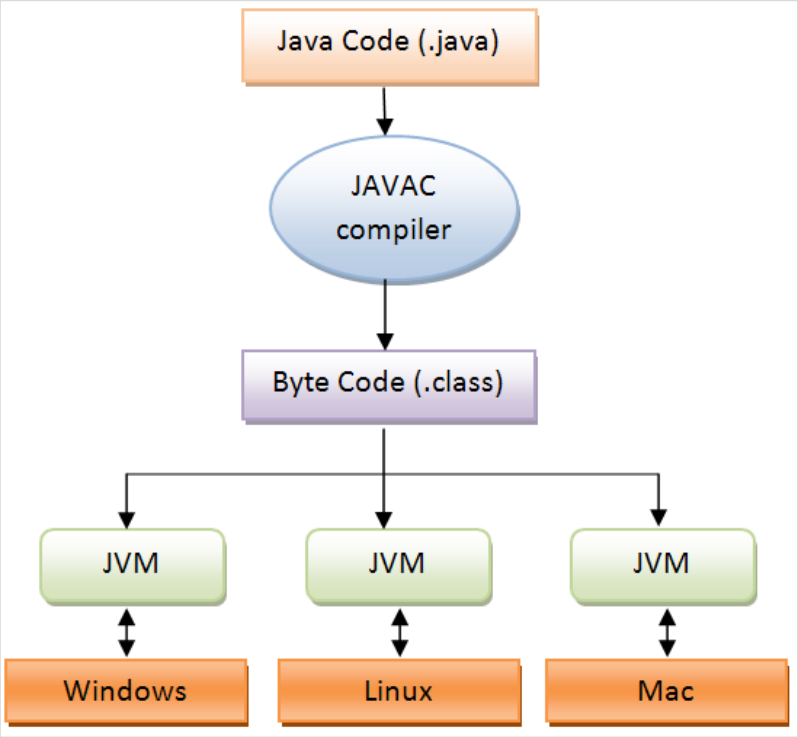

Figura III-1. Funcionamiento de Java, obtenido de (Patel, 2008)

Portable

La neutralidad respecto a la arquitectura es sólo una parte de un sistema verdaderamente portable. El lenguaje Java va más allá, definiendo estrictamente las especificaciones del lenguaje. En Java están definidos, por ejemplo, el tamaño de los tipos de datos básicos o el comportamiento estricto de todos los operadores aritméticos. Los programas se ejecutan sobre la Máquina Virtual Java, que especifica todas las instrucciones permitidas y su significado.

Para que los bytecodes se puedan ejecutar sobre un nuevo sistema hardware, sólo es necesario portar la máquina virtual a ese nuevo sistema, y todos los programas existentes pasarán a ejecutarse sin problemas

Dinámico

El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

Soporte para Multithread

Al ser multihilo (multithread), Java permite muchas actividades simultáneas en un programa. Los threads (hilos), son básicamente pequeños procesos o piezas independientes de un gran proceso, para la utilización de varios hilos de ejecución, Java proporciona la clase Thread, a partir de la cual pueden derivarse nuevos objetos, y que incluye métodos para crear un nuevo hilo de ejecución, detenerlo, dormirlo, destruirlo o conocer su estado.

También se incluyen mecanismos para manejar la concurrencia basados en monitores y variables de condición. El beneficio de ser multithreaded consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

3.2.1.1 Programación orientada a objetos

La programación orientada a objetos es una herramienta para reducir la complejidad de los sistemas de software. Usando la orientación a objetos, un sistema que ocupa miles o millones de líneas de código puede organizarse como una colección de pequeñas unidades (objetos), cada una con cierta independencia y ciertas responsabilidades. Un programa OO consiste en un conjunto de objetos que intercambian mensajes; cada objeto decide por sí mismo si debe o no aceptar los mensajes que recibe, así como la interpretación de cada mensaje. En un programa OO medianamente complicado, los objetos no son totalmente independientes: unos heredan propiedades y métodos de otros; algunos necesitan consultar a otros para desempeñar sus tareas.

Alan Kay (Kay, 2007) resumió cinco características básicas de Smalltalk, el primer lenguaje orientado a objetos y uno de los lenguajes en que se basa Java. Estas características representan un acercamiento puro a la programación orientada a objetos:

- Todo es un objeto. Todo se puede representar como objetos en el programa.
- Un programa es un conjunto de objetos que se indican entre sí lo que tienen que hacer enviándose mensajes.
- Cada objeto tiene su propia memoria formada por otros objetos.
- Todo objeto tiene un tipo asociado.
- Todos los objetos de un tipo particular pueden recibir los mismos mensajes.

Concepto de Clase

Una clase es una agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina variables y métodos. La programación orientada a objetos se basa en la programación de clases y un programa se construye a partir de un conjunto de clases.

Una vez definida e implementada una clase, es posible declarar elementos de esta clase de modo similar a como se declaran las variables del lenguaje (int, double…). Los elementos declarados de una clase se denominan objetos de la clase, de una única clase se pueden declarar numerosos objetos. La clase es lo genérico, el patrón para generar objetos.

Uno de los aspectos más importantes en la programación orientada a objetos es la forma en la cual son creados y eliminados los objetos. En Java la forma de crear nuevos objetos es utilizando el operador *new*. Cuando se utiliza el operador *new*, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. Para cada objeto se lleva cuenta de por cuántas variables de tipo referencia es apuntado. La eliminación de los objetos la realiza el programa denominado *garbage collector* (recolector de basura), el cual libera o borra de forma automática la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto

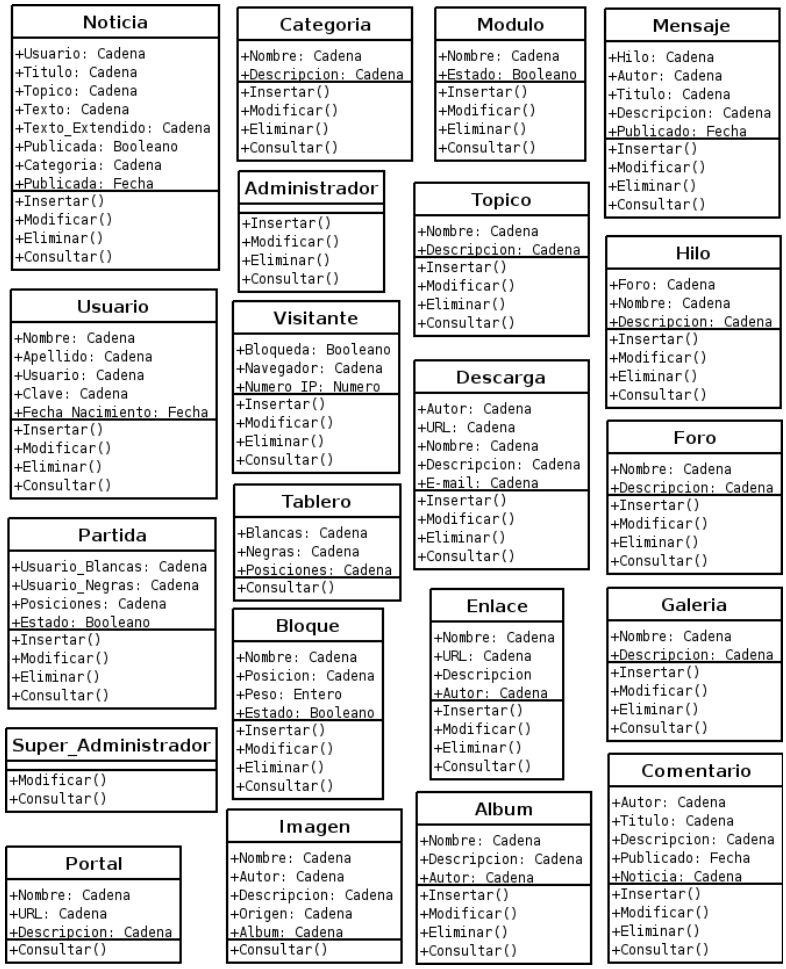

Figura III-2. Ejemplos de clases, obtenido de (Wikipedia, 2010a)

La herencia

La herencia permite que se pueden definir nuevas clases basadas en clases existentes, lo cual facilita re-utilizar código previamente desarrollado. Si una clase deriva de otra (extends) hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados.

En Java, a diferencia de otros lenguajes orientados a objetos, una clase sólo puede derivar de una única clase, con lo cual no es posible realizar herencia múltiple en base a clases. Sin embargo es posible "*simular*" la herencia múltiple en base a las interfaces.

Polimorfismo

Proporciona otra dimensión de separación entre la interfaz y la implementación, con el fin de desacoplar el qué con respecto al cómo. El polimorfismo nos permite mejorar la organización y legibilidad del código, así como la creación de programas extensibles que pueden "*crecer*" no sólo durante la creación del proyecto, sino también cuando se deseen añadir nuevas características. La idea básica es que una referencia a un objeto de una determinada clase es capaz de servir de referencia o de nombre a objetos de cualquiera de sus clases derivadas.

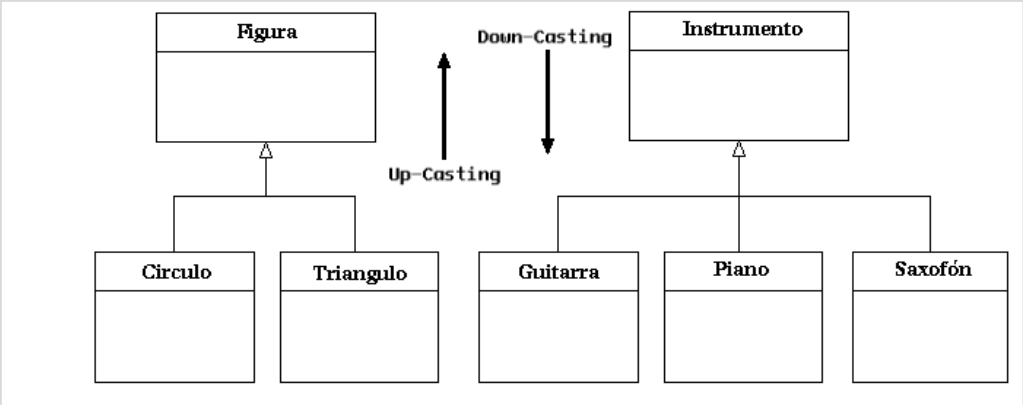

Figura 3.1. Herencia y polimorfismo en java

El polimorfismo tiene que ver con la relación que se establece entre la llamada a un método y el código que efectivamente se asocia con dicha llamada. A esta relación se llama vinculación (binding). La vinculación puede ser temprana (en tiempo de compilación) o tardía (en tiempo de ejecución). Con funciones normales o sobrecargadas se utiliza vinculación temprana (es posible y es lo más eficiente). Con funciones redefinidas en Java se utiliza siempre vinculación tardía, excepto si el método es final.

Existen tres tipos de polimorfismo(Kioskea, 2008):

- El polimorfismo de sobrecarga: Ocurre cuando las funciones del mismo nombre existen, con funcionalidad similar, en clases que son completamente independientes una de otra.
- El polimorfismo paramétrico: Es la capacidad para definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o

tipo). El polimorfismo paramétrico selecciona automáticamente el método correcto a aplicar en función del tipo de datos pasados en el parámetro.

- El polimorfismo de subtipado: Se trata de la redefinición de un método en una clase que hereda de otra de forma que permite no tomar en cuenta detalles de las clases especializadas de una familia de objetos, enmascarándolos con una interfaz común.

3.2.2 Netbeans

Para la realización de este proyecto se ha utilizado NetBeans IDE[21] 6.9.1(Netbeans, 2011). Se trata de un reconocido entorno de programación integrado que funciona en las principales arquitecturas y sistemas operativos (Windows, Linux, Solaris, MacOS), es sencillo de instalar, y proporciona a los desarrolladores todas las herramientas necesarias para crear aplicaciones orientadas a escritorios, empresariales, Web y aplicaciones móviles usando Java, JavaFX[22], PHP[23], JavaScript y Ajax, Ruby, Groovy y Grails, y C/C++. Netbeans es un programa de código abierto y sin restricciones de uso creado por Sun MicroSystems y que cuenta con una gran comunidad de usuarios en constante crecimiento.

De forma global, las tecnologías soportadas por Netbeans son las siguientes:

- Java EE 5, Java EE 6 y J2EE 1.4
- JavaFX SDK 1.3.1
- Java Card 3.0.2
- Struts 1.3.8

[21] Entorno de desarrollo integrado (Integrated development environment)

[22] Java FX es una familia de productos y tecnologías para la creación de Rich Internet Applications (RIA), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia dinámicas.

[23] PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

- Spring 3.0, 2.5
- Hibernate 3.2.5
- API Java para Servicios Web REST (JAX-RS) 1.1
- Seguimiento de errores:
- Bugzilla 3.4 y anteriores
- ira 3.4 y anteriores
- PHP 5.3, 5.2, 5.1
- Ruby 1.9, 1.8
- JRuby 1.5.0
- Rails 2.3.4, 3.0 Beta
- Groovy 1.6.4
- Grails 1.1
- Control de versiones:
- CVS: 1.11.x, 1.12.x
- Subversion: 1.5.x, 1.6.x
- Mercurial: 1.0.x o posterior
- ClearCase V7.0

Los servidores de aplicaciones soportados por Netbeans son:

- GlassFish Server Open Source Edition 3.0.1
- GlassFish Enterprise Server v2.1.1
- Sun Java System Application Server PE 8.2
- WebLogic 11g (10.3.3.0)
- Tomcat 6.0.26
- Tomcat 5.5
- JBoss 5.0
- JBoss 4.2.3 (J2EE 1.4)

Entre las mejoras de las últimas versiones se encuentra el soporte completo a PHP ofreciendo dispositivos como compleción de código, codificación coloreada por

semántica e integración a bases de datos. También hay elementos mejorados Ruby[24] en el editor, debugger y Rake (variante de codificación Ruby).

El soporte a JavaFX también se ha visto mejorado, ya que además de soportar la nueva versión del JDK, también se ha mejorado el depurador y se ha incluido la herramienta JavaFX Composer para simplificar el desarrollo de una manera visual.

El rendimiento del código se ha optimizado para Java con un dispositivo *compile-and-save* donde la compilación se realiza en un segundo plano cada vez que el programador salva su tarea. También se puede testear lo salvado en forma inmediata.

Otra novedad ha sido un editor para desarrollo JavaScript con compleción de código CSS/HTML, gestor de bibliotecas JavaScript incluyendo Yahoo UI, Woodstock, jQuery, Dojo, Scriptaculous, Prototype y capacidad de debugging del código del lado cliente en los navegadores Firefox y Explorer. También trae mejoras en los asistentes para conexión y exploración de bases de datos, mejoras en el debugger, en especial cuando se trabaja con aplicaciones con varios hilos, y mejoras en el soporte de UML.

Además se proporciona soporte ampliado para Spring, Hibernate[25], Java Server Pages y la API Java Persistence. De manera opcional las últimas versiones incluyen el servidor de aplicaciones GlassFish 3.0. GlassFish 3.0 apunta a la capa Web para servicio de aplicaciones Web y tiene un diseño muy modular. Ocupa muy pocos recursos, tanto como unos 100 KB de base, para incorporar nueva funcionalidad a medida que la requiere.

[25] Es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

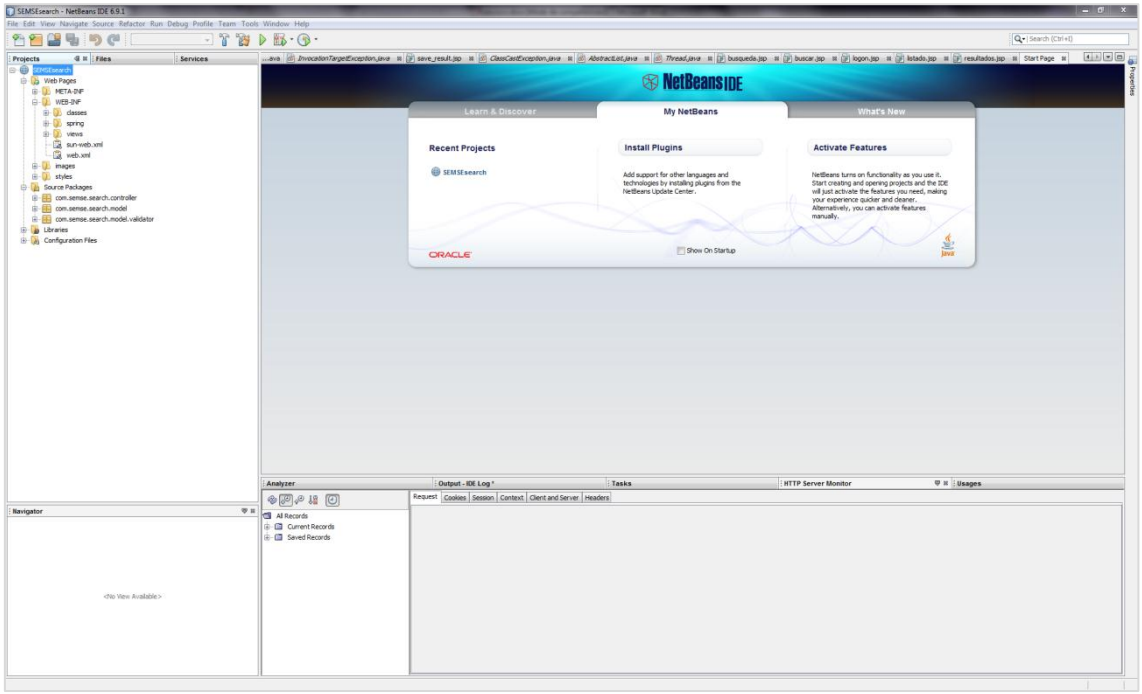

Figura III-3. Captura de NetBeans 6.9.1

3.3 Frameworks empleados

3.3.1 JSP

JSP (Java Server Pages) es una tecnología orientada a la creación de páginas web dinámicas mediante el uso de Java(JGuru, 2000). Una de sus grandes ventajas es que al estar basado en esta tecnología hereda todas las ventajas de este lenguaje, incluyendo la capacidad de ser multiplataforma.

El funcionamiento de JSP está basado en los Servlets de Java ejecutados en el lado del servidor aunque simplificando enormemente su desarrollo. Las páginas Web de JSP son archivos con extensión .jsp que incluyen, combinado junto con etiquetas HTML o XML, las sentencias Java que serán ejecutadas en el servidor.

La primera vez que el servidor recibe una petición o salvo que la página JSP haya sido modificada, el motor JSP realiza un proceso de traducción por el que se interpreta la página en un archivo .class, propio de Java, que será ejecutado como un servlet en el servidor web.

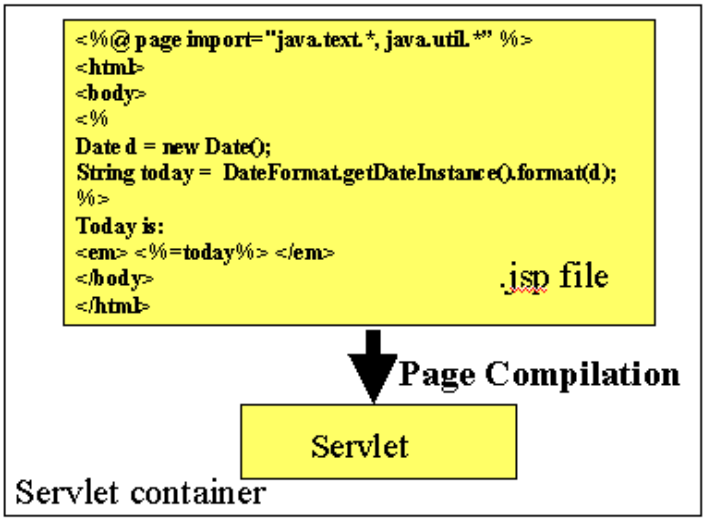

Figura III-4. Interpretación de una página JSP

JSP surgió como respuesta a la tecnología ASP de Microsoft, que al igual que JSP permite la creación de contenido dinámico en las páginas Web, sin embargo, son muchas las ventajas que aporta JSP sobre ASP:

- ASP sólo es compatible con la tecnología propia de Microsoft, como Microsoft ISS o Personal Web Server, mientras que JSP sigue la filosofía de *Write Once Run Anywhere,* lo que traducido al castellano significaría algo así como *Escribe una vez, ejecuta en cualquier parte.* Y es que las aplicaciones JSP pueden ser transportadas fácilmente entre los sistemas operativos y servidores Web más populares como Apache, Netscape o Microsoft ISS sin sufrir ningún cambio.
- La tecnología JSP usa Java como lenguaje de Script, que es más potente y escalable que otros lenguajes de script como pueden ser VBScript o Jscript utilizados por ASP.
- JSP permite a los desarrolladores crear etiquetas personalizadas, mientras que ASP no.
- JSP, al ser Open Source, se beneficia de la amplia comunidad Java, mientras que ASP depende de una empresa privada como es Microsoft.

3.3.2 JPA

JPA (Java Persistance API) es un framework del lenguaje de programación Java destinado a la gestión de la persistencia de los objetos o POJOs[26] utilizados en las aplicaciones con el objetivo de no perder las ventajas de la programación orientada a objetos cuando se trabaja con una base de datos. Esto se consigue mediante la utilización del mapeo objeto-relacional que describe la forma en la que un objeto es almacenado y convertido a un formato y tipo que una base de datos admita, evitando así a los desarrolladores la implementación de un método de almacenamiento específico para cada una de las clases utilizadas en una aplicación.

La forma en que JPA permite realizar este mapeo es mediante el uso de etiquetas o anotaciones especiales en la declaración de la clase. En ellas se especifican aspectos como la tabla en la que se almacenan los objetos, qué clave primaria utilizan o qué atributos de dicha clase se mapean con qué columnas en la tabla, entre otras muchas cosas.

[26] POJO (acrónimo de Plain Old Java Object) es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

Algunas de las etiquetas más importantes utilizadas en el mapeo objeto relacional pueden observarse en la siguiente figura identificadas por comenzar con el carácter '@'.

```
@Entity
@Table(name = "esquema")
@NamedQueries({
   @NamedQuery(name = "Esquema.findAll", query = "SELECT e FROM Esquema e"),
   @NamedQuery(name = "Esquema.findById", query = "SELECT e FROM Esquema e WHERE e.id = :id"),
   @NamedQuery(name = "Esquema.findByName", query =
            "SELECT e FROM Esquema e WHERE e.name = :name"),
   @NamedQuery(name = "Esquema.findByUri", query = "SELECT e FROM Esquema e WHERE e.uri = :uri"),
   @NamedQuery(name = "Esquema.findByDescription", query =
            "SELECT e FROM Esquema e WHERE e.description = :description"),
   @NamedQuery(name = "Esquema.findByUri_DifId", query =
             "SELECT e FROM Esquema e WHERE e.id <> :id AND e.uri = :uri")})

public class Esquema implements IEsquema, Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   @Basic(optional = false)
   @Column(name = "id")
   private Long id;
   @Basic(optional = false)
   @Column(name = "name")
   private String name;
   @Basic(optional = false)
   @Column(name = "uri")
   private String uri;
   @Column(name = "description")
   private String description
   public Esquema() {...}
   public Esquema(Long id) {...}
   public Esquema(Long id, String name, String uri) {...}
   public Long getId(){...}
   public void setId(Long id) {...}
   public String getName(){...}
   public void setName(String name) {...}
   public String getUri(){...}
   public void setUri(String uri) {...}
   public String getDescription(){...}
   public void setDescription(String desc) {...}
   @Override
   public int hashCode(){...}
   @Override
   public boolean equals(Object object) {...}
   @Override
}
```

Figura III-5. Uso de etiquetas de mapeo de JPA

- @Entity: Define una clase entidad, esto significa que JPA identifica a la clase como una clase persistente. Por lo general, una clase entidad suele coincidir con una tabla de la base de datos. Para que una clase pueda ser considerada

como una clase entidad debe de implementar la interfaz *Serializable* y disponer de un método constructor sin parámetros.

- @Table: Indica el nombre de la tabla en la que JPA guarda los datos de los objetos de la clase. Si esta etiqueta no se indica, JPA asume que el nombre de la tabla es el mismo que el de la clase.
- @Id: Define la propiedad de la clase que actúa como clave primaria en la tabla donde se almacena en la base de datos.
- @GeneratedValue: Indica que el valor será generado se forma automática. Admite cuatro tipos de generación automática: *sequence, identity, table* o *auto.*
- @Column: Esta etiqueta permite la definición de multiples propiedades sobre la forma en la que se almacena el atributo del objeto en la tabla de la base de datos. La propiedad *name* especifica el nombre de la columna donde va a ser persistido el campo, si esta propiedad no se indica, JPA usará el nombre de la variable como nombre de la columna. La propiedad *nullable* indica si la columna acepta valores null o no, si no se incluye el valor por defecto es true. Además esta anotación soporta otras muchas propiedades como pueden ser *columnDefinition*, *length*, *precision*, *scale*, *insertable*, *updatable* y *table*.
- @Basic: Permite definir otras propiedades como *optional*, cuyo significado es parecido a la propiedad *nullable* de la etiqueta @Column; o *fetch,* que indicará si la carga del campo se realizará de forma perezosa (*FetchType.LAZY*) o en el momento en el que se cargue el resto del objeto (*FetchType.EAGER*).
- @Transient: Esta anotación permite indicar los atributos de la clase que no participan en la persistencia, de forma que podrán ser utilizados en la aplicación como parte de la clase, pero sus valores no serán guardados en la base de datos cuando JPA persista el objeto en los que están contenidos.
- @NamedQuery: Permite declarar consultas SQL otorgándolas un nombre identificativo por el que hacerlas referencia desde cualquier punto de la aplicación y con la posibilidad de utilizar parámetros de forma dinámica.

Adicionalmente, si una clase entidad está relacionada con alguna otra y por tanto, almacenada en varias tablas de la base de datos, JPA permite definir estas relaciones con unas anotaciones específicas aplicadas sobre los atributos de la clase que actúan como clave ajena de otra tabla(Madeja, 2011):

- OneToOne: Multiplicidad 1:1. Cada entidad se relaciona con una única instancia de otra entidad.
- ManyToOne: Multiplicidad n:1. Múltiples instancias de una entidad pueden estar relacionadas con una única instancia de otra entidad.
- OneToMany: Multiplicidad 1:n. Una entidad puede estar relacionada con múltiples instancias de otra entidad.
- ManyToMany: Multiplicidad n:n. En este caso varias instancias de una entidad pueden estar relacionadas con varias instancias de otra entidad.

Para el caso de las relaciones OneToOne y OneToMany puede especificarse que las operaciones efectuadas sobre una entidad en concreto puedan propagarse a las entidades con las que se relaciona en forma de cascada. Existen cuatro tipos de definiciones(Plunchete, 2009):

- *CascadeType.PERSIST*: Cuando se persiste la entidad todas las entidades con las que está relacionada serán persistidas también.
- *CascadeType.REMOVE*: Cuando se borra una entidad, todas aquellas entidades con las que se relacione serán borradas también.
- *CascadeType.REFRESH*: Cuando se actualice una entidad, también lo harán todas aquellas entidades con las que se relacione.
- *CascadeType.MERGE*: Cuando se realiza una operación *merge* sobre una entidad, también se realizará sobre todas las entidades con las que se relacione. Una operación *merge* consiste en realizar una copia del objeto sobre el que se realiza la persistencia y sólo guardar los cambios realizados sobre esta copia en el momento de persistir el objeto.
- *CascadeType.ALL*: Todas las operaciones citadas anteriormente.

3.3.2.1 Arquitectura de JPA

En la figura siguiente se pueden observar los componentes principales de la tecnología JPA:

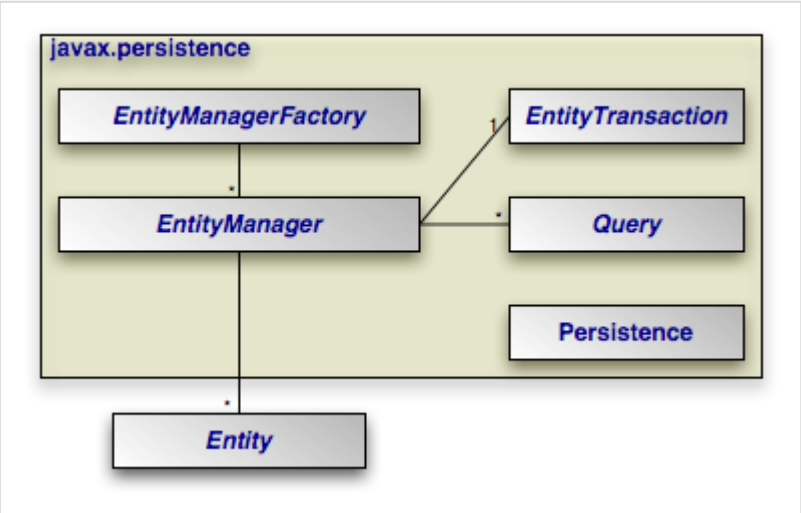

Figura III-6. Arquitectura de JPA, obtenido de (IBM, 2011)

Como se puede observar en la figura anterior, existen varios componentes dentro de la librería que JPA ofrece, la javax.persistance.

- Persistence: La clase *javax.persistence.Persistence* contiene métodos estáticos de ayuda para obtener una instancia de *EntityManagerFactory* de una forma independiente al proveedor de la implementación de JPA.
- EntityManagerFactory: La clase *javax.persistence.EntityManagerFactory* es al encargada de generar objetos de la clase *EntityManager* utilizando el patrón de diseño *Factory* (ver apartado 0 Factory).
- EntityManager: La clase *javax.persistence.EntityManager* es la interfaz principal de JPA utilizada para la persistencia de las aplicaciones. Cada *EntityManager* puede realizar operaciones CRUD (Create, Read, Update, Delete) sobre un conjunto de objetos persistentes.
- Entity: La clase *javax.persistence.Entity* es una anotación Java que se coloca a nivel de clases Java serializables y que cada objeto de una de estas clases anotadas representa un registro de una base de datos.

- EntityTransaction: Cada instancia de *EntityManager* tiene una relación de uno a uno con una instancia de *javax.persistence.EntityTransaction*, permite operaciones sobre datos persistentes de manera que agrupados formen una unidad de trabajo transaccional, en el que todo el grupo sincroniza su estado de persistencia en la base de datos o todos fallan en el intento, en caso de fallo, la base de datos quedará con su estado original. Maneja el concepto de todos o ninguno para mantener la integridad de los datos.
- Query: La interface *javax.persistence.Query* está implementada por cada proveedor de JPA para encontrar objetos persistentes manejando cierto criterio de búsqueda. JPA estandariza el soporte para consultas utilizando Java Persistence Query Language (JPQL) y Structured Query Language (SQL). Podemos obtener una instancia de *Query* desde una instancia de un *EntityManager*.

3.3.2.2 Persistance.xml

La configuración de JPA en las aplicaciones viene definida por un archivo en formato XML llamado persistance.xml(CIST, 2009). Este archivo es de vital importancia en JPA, ya que en él se detallan las unidades de persistencia que se van a utilizar. Este archivo debe estar presente en el directorio raíz de la aplicaciones para aplicaciones de escritorio o en el WEB-INF/clases/ si se trata de una aplicación Web.

Por cada unidad de persistencia se define un nombre identificativo y un tipo de transacción, que dependerá del tipo de proveedor a utilizar. Existen dos tipos de transacciones:

- RESOURCE_LOCAL: Es el proveedor de persistencia el encargado de gestionar las transacciones por las que se realizan las operaciones en JPA. Este tipo de transacciones son propias de las plataformas JSDK y J2EE.

- JTA: Es el contenedor Web el encargado de la gestión de las transacciones, pero requiere que la conexión a la base de datos se realice mediante JNDI[27] y un pool de conexiones configurado en el servidor. Este tipo de transacciones se utiliza en plataformas JEE.

Algo que también hay que indicar en el archivo persistance.xml es el proveedor de persistencia que se desea utilizar. Existen múltiples proveedores basados en las especificaciones de JPA. Los más comunes son Hibernate, TopLink, EclipseLink y OpenJPA, aunque existen muchos otros.

Dependiendo del proveedor y del tipo de transacción que se utilice se deben de indicar de forma adicional determinados parámetros que indiquen el origen de los datos o la conexión a la base de datos. Por lo general, si se utiliza JTA se debe de indicar el origen JNDI; por el contrario, si se utiliza *RESOURCE_LOCAL* se debe de indicar al proveedor de persistencia los parámetros de conexión a la base de datos, que por lo general son URL de conexión, usuario, contraseña y driver.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns=http://java.sun.com/xml/ns/persistence
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="AlmacenPU" transaction-type="RESOURCE_LOCAL">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>almacen.modelo.Articulo</class>
    <class>almacen.modelo.Proveedor</class>
    <properties>
      <property name="toplink.jdbc.user" value="almacen"/>
      <property name="toplink.jdbc.password" value="almacen"/>
      <property name="toplink.jdbc.url" value="jdbc:postgresql://localhost:5432/almacen"/>
      <property name="toplink.jdbc.driver" value="org.postgresql.Driver"/>
    </properties>
  </persistence-unit>
<persistence>
```

Ejemplo III-1. persistance.xml

[27] JNDI (Java Naming and Directory Interface) es un API de java para servicios de directorio. Esto permite a los clientes descubrir y buscar objetos y nombres a través de un único nombre identificativo.

3.3.3 Spring

Spring (Spring Source, 2010) es un framework de código abierto para el desarrollo de aplicaciones Java, aunque también existe la versión para .NET, Spring .NET. La primera versión fue escrita por Rod Johnson con el objetivo de simplificar el desarrollo de aplicaciones empresariales. Spring pretende integrar las diferentes tecnologías existentes en un solo framework para el desarrollo más sencillo y eficaz de aplicaciones J2EE (ahora JEE 5) portables entre servidores de aplicación. Spring facilita el desarrollo de aplicaciones J2EE al intentar evitar el uso de EJB ofreciendo los mismos servicios pero simplificando el modelo de programación.

Si se desmontase Spring en sus partes básicas obtendríamos una descripción de sus principales funcionalidades(Walls, 2008):

- **Contenedor:** Spring es un contenedor en el sentido de que contiene y gestiona el ciclo de vida y configuración de objetos de aplicación. Se basa en el uso de JavaBeans y puede definirse cómo deben ser creados, cómo configurarse o cómo pueden relacionarse entre sí. Esta configuración es lo que se llama contexto de la aplicación.
- **Marco de trabajo:** Spring permite configurar y escribir aplicaciones complejas a partir de componentes sencillos. En Spring los objetos se escriben de forma declarativa, normalmente en un archivo XML. Spring también proporciona gran funcionalidad de infraestructura (gestión de transacciones, persistencia…) permitiendo a los desarrolladores centrarse en la aplicación.
- **Ligero:** Spring es ligero tanto en términos de tamaño como en tiempo de procesamiento. El volumen del marco de trabajo Spring puede distribuirse en un único archivo JAR que ocupa poco más de 2,5 MB. Y el tiempo de procesamiento requerido por Spring es prácticamente insignificante, gracias entre otras cosas a que Spring no es intrusivo. Es decir, los objetos de una aplicación habilitada para Spring a menudo no tienen dependencias de clases específicas de Spring.

- **Inyección de dependencia:** Spring fomenta el acoplamiento débil mediante una tecnología conocida como inyección de dependencia (DI). Cuando se aplica la DI, se otorga a los objetos de forma pasiva sus dependencias, en lugar de crear o buscar objetos dependientes por sí mismos.
- **Orientado a aspectos:** Spring tiene un amplio soporte para la programación orientada a aspectos (AOP) que permite el desarrollo cohesivo separando la lógica empresarial de la aplicación de los servicios de sistema, como auditoría y gestión de transacciones.

Las premisas que los desarrolladores de Spring aseguran seguir son que(Spring Source, 2010):

- J2EE debería ser fácil de usar.
- Es mejor programar con interfaces que con clases, por eso Spring reduce la complejidad del tratamiento de las interfaces a 0.
- JavaBeans es muy útil para la configuración de las aplicaciones.
- El diseño orientado a objetos es más importante que cualquier tecnología, como puede ser J2EE.
- El chequeo de excepciones en Java es abusivo. Una plataforma no debería de obligar a los desarrolladores a capturar excepciones de las que es poco probable recuperarse.
- Las pruebas son esenciales y Spring ayuda a que el código sea fácil de probar.

A día de hoy la última versión de Spring es la 3.0. Algunas de las características que incluye esta nueva versión son:

- El Framework está basado en Java 5 y se añade soporte completo para Java 6. Al mismo tiempo se ha añadido soporte para J2EE 1.4 y Java EE 5, mientras que se comienza a ofrecer soporte para Java EE 6.
- Se incluye el nuevo Spring Expression Language, similar a Unified EL, que puede ser usado en las definiciones en XML, definiciones de Beans basadas en anotaciones y en su uso en el portafolio de Spring.

- Mejoras en el contenedor de inversión de control (IoC) al incorporar algunas de las mejoras del proyecto JavaConfig.
- Se ha añadido OXM, que se detallará en el apartado siguiente.
- Nuevas anotaciones para el módulo MVC, como pueden ser @CookieValue o @RequestHeader.

3.3.3.1 Arquitectura de Spring.

El framework de Spring está divido en 20 módulos tal y como se puede observar en la siguiente figura:

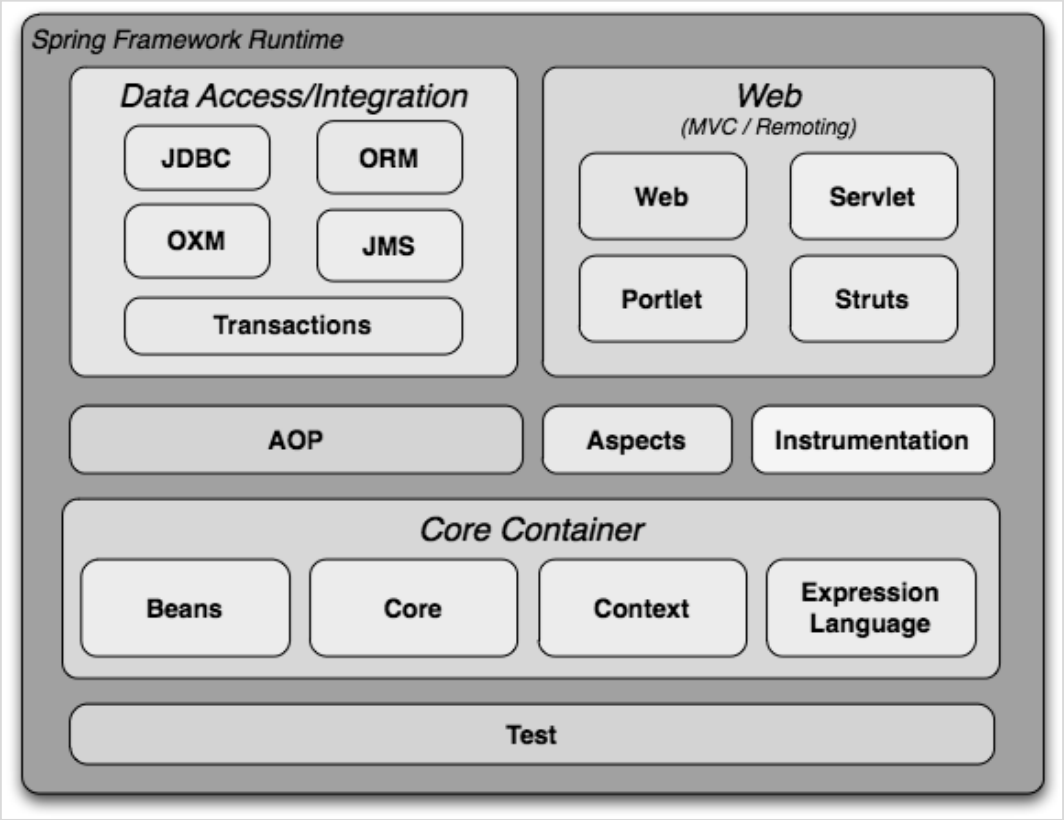

Figura III-7. Arquitectura de Spring

Core Container

El contenedor del núcleo o Core Container está dividido en cuatro módulos:

- **Core y Beans:** Forman la parte fundamental del framework ya que provee a éste las características de Inversión de Control (IoC) e Inyección de dependencias (ID). El concepto básico dentro del Core es el BeanFactory[28], el cual provee una sofisticada implementación del Patrón Factory, que elimina la necesidad generalizada de Singletons y nos permite desacoplar la configuración y especificación de las dependencias de nuestra lógica de programación.
- **Context:** Está construido sobre una sólida base provista por el paquete Core, el cual proporciona un medio de acceso a los objetos contenidos en el framework de tal forma que recuerda en cierta medida a la manera en cómo trabaja el registro JNDI. Este paquete hereda algunas características del paquete Beans y añade soporte para internacionalización (I18N), propagación de eventos, carga de recursos y creación transparente de contextos, como por ejemplo, un Servlet Container.
- **Expression Language:** Esta nueva funcionalidad de Spring 3.0 has sido añadida dentro del paquete Core y como se ha comentado con anterioridad, se trata de una extensión de la especificación Unified UL incluída en la especificación JSP 2.1. Este lenguaje permite la consulta y modificación de objetos en tiempo de ejecución. Soporta métodos getter y setter, asignación de propiedades, invocación a métodos, acceso a arrays de contexto, colecciones e índices, operadores lógicos y matemáticos, variables nombradas y recuperación de objetos del IoC container por nombre**.**

Data Access/Integration

El módulo de acceso e integración de la información está dividido a su vez en cuatro módulos:

[28] Implementación del patrón Factory, permite instanciar objetos de forma muy rápida y fácil. Puede crear muchos tipos diferentes de beans.

- **JDBC**: El módulo JDBC proporciona una capa de abstracción a JDBC, eliminando la tediosa codificación propia de JDBC y el tratamiento de los códigos de errores específicos de cada proveedor de base de datos.
- **ORM**: Provee una capa de integración con las APIs más populares de Mapeo Objeto-Relacional, tales como JPA, JDO, Hibernate e iBatis. Usando este paquete se puede utilizar cualquiera de estos ORM en combinación con todas las otras características que ofrece Spring.
- **OXM:** Este módulo proporciona una capa de abstracción que soporta el mapeo objeto/XML de implementaciones para JAXB, Castor, XMLBeans, JiBX y XStream.
- **JMS:** El servicio de mensajes de Java (Java Messaging Service) ofrece soporte para la creación y utilización de mensajes.
- **Transaction:** El módulo de transacciones soporta la gestión de transacciones programáticas y declarativas de las clases que implementen una interfaces determinadas y de todos los POJOs que hayan creado los desarrolladores.

Web

El módulo Web está compuesto de los siguientes cuatro módulos:

- **Web:** Provee características de integración orientadas a la Web, tales como funcionalidades para la carga de archivos, la inicialización del contenedor IoC usando *Servlet Listeners* y un contexto de aplicación orientado a la Web.
- **Web-Servlet:** Provee de una implementación del patrón MVC (*Moldeo-Vista-Controlador*) para las aplicaciones Web. El framework MVC de Spring proporciona una clara separación entre la codificación del modelo y los formularios Web, además de estar integrado con el resto de funcionalidades de Spring.
- **Web-Struts:** Ofrece clases que dan soporte al uso de Struts clásico dentro de una aplicación Spring. Sin embargo, en la versión 3.0 de Spring este módulo se ha quedado obsoleto, por lo que es recomendable utilizar Struts 2.0.

- **Web-Portlet**: proporciona la implementación MVC para ser utilizado en un entorno de portlet y refleja la funcionalidad del módulo Web Servlet.

Aop e Instrumentation

Provee una implementación para la programación Orientada a Aspectos compatible con el AOP Alliance que nos permite definir, por ejemplo, interceptores a un método (method-interceptors) y puntos de corte (pointcuts) para desacoplar limpiamente algunas funcionalidades implementadas en el código que lógicamente deberían conversar por separado.

Test

Soporta la realización de pruebas de los componentes de Spring mediante el uso de JUnit o TestNG.

3.3.3.2 Inversión de Control (IoC).

IoC (Wikipedia, 2011b) es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa (en Spring se trata del IoC Container) lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.

IoC se puede explicar mediante el principio Hollywood: "*No me llames, yo te llamare a ti*", es decir, en lugar de que el código de la aplicación llame a una clase de una librería, un framework que utiliza IoC llama al código.

Inyección de dependencias.

Existe cierta controversia sobre lo que es la inversión de control o si debería de ser llamado directamente inyección de dependencias, por ser un término menos genérico. A partir de un artículo publicado por Martin Fowler en 2004 (Fowler, 2004),

en el que el autor de preguntaba qué era lo que realmente se invertía, llegó a la conclusión de que era la adquisición de dependencias y no el control y que por tanto, el término “inyección de dependencia” describía mejor este proceso.

Independientemente de este debate, el patrón de la inyección de dependencias consiste en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor. De esta forma, estos objetos son instanciados una única vez y guardados en una factoría (Factory), por lo que pueden ser compartidos por todos los usuarios evitando de esta manera la extensión de clases y reduciendo la saturación de los servidores en aplicaciones web.

Spring soporta inyección de dependencias a través del constructor y a través de métodos set. El principio fundamental de la Inyección de Dependencias (Dependency Injection: DI) es que los objetos definan sus propias dependencias con otros objetos del dominio. Es decir, a una clase determinada se le inyectan objetos de otras en lugar de ser la propia clase la que cree el objeto. Es trabajo del contenedor inyectar estas dependencias cuando se crea un *Bean*.

Estas dependencias son declaradas en el archivo de configuración denominado applicationContext.xml.

El proceso de creación de dependencias es el siguiente:

1. El *BeanFactory* es creado e inicializado con la configuración que describe a todos los *Beans*.
2. Cada *Bean* tiene dependencias expresadas en forma de *properties*, *constructor arguments* o *static arguments* en sustitución de un constructor normal. Estas dependencias serán proporcionadas al *Bean*, cuando este sea creado.
3. Cada *property* o *constructor argument* en una definición del valor que tomará el atributo al inicio o una referencia a otro *Bean* del contenedor.

Spring soporta varios tipos de inyección de dependencia pero estos son los dos más utilizados:

- **Setter injection:** en este tipo la Inyección de Dependencia es aplicada por medio de métodos JavaBeans *setters* que a la vez tienen un *getter* respectivo.
- **Constructor injection:** esta Inyección es a través de los argumentos del constructor.

3.3.3.3 Programación Orientada a Aspectos (AOP)

AOP (Aspect-Oriented Programming) es un paradigma de la programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de las incumbencias(Wikipedia, 2011a). De esta forma se consigue razonar mejor los conceptos, eliminar las dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables, gracias por una lado a la separación de las funcionalidades más comunes utilizadas a lo largo de las aplicaciones; y por otro, a las funcionalidades propias de cada módulo.

El núcleo de construcción es el aspecto (aspect) que encapsula comportamiento que afecta a diferentes clases en módulos que pueden ser reutilizados. En otras palabras es una manera de eliminar código duplicado.

Spring AOP es portable entre servidores de aplicación, funciona tanto en servidores Web como en contenedores EJB. Spring AOP soporta las siguientes funcionalidades:

- **Intercepción:** se puede insertar comportamiento personalizado antes o después de invocar a un método en cualquier clase o interfaz.
- **Introducción:** declaraciones de métodos adicionales o campos sobre un tipo.
- ***Pointcuts* dinámicos y estáticos:** para especificar los puntos en la ejecución del programa donde debe haber intercepción.

3.3.3.4 Spring Security

Spring Security (Spring Source, 2011) es un potente y personalizable Framework de autenticación y control de acceso creado en 2003 y mantenido por SpringSource desde entonces. Se trata de uno de los más maduros y usados de todos los proyectos de Spring. Se usa en numerosos entornos seguros como pueden ser agencias gubernamentales, aplicaciones militares y bancos. Entre sus grandes ventajas destacan:

- Código y diseño excelentes.
- Gran uso en proyectos de referencia dentro del sector de Java.
- Solución madura.
- Facilidad de configuración y parametrización.
- Integración con los sistemas legacy más importantes como BBDD, LDAP, CAS, gestión de certificados, etc.
- Uso sencillo y muy extensible como toda la familia de productos Spring.
- Gran cantidad de documentación y ejemplos para su aprendizaje.

A continuación se detallan las principales características que ofrece este Framework y las que se han usado en la elaboración de este proyecto.

Autenticación

El proceso de autenticación se define como aquel por el que se garantiza que una entidad que solicita un servicio es quien dice ser(Caro, 2010). En Spring Security esta tarea es delegada al *AuthenticationManager*, el cual, a partir de los datos básicos de la entidad procederá a obtener toda la información de credenciales asociados a ella para comprobar que son correctos. Si no fuera así, se devuelve una excepción de tipo *AuthenticationException* pero si, por el contrario, sí que lo fueran, el *AuthenticationManger* devolverá un objeto *Authentication* con toda la información asociada a la entidad y el conjunto de roles o autoridades que posee. Este objeto, además, es añadido al contexto de seguridad de la aplicación manejado por la clase *SecurityContextHolder.*

Sin embargo, el proceso de autenticación que proporciona el *AuthenticationManager* es a su vez delegado en otro componente de la arquitectura de Spring Security. Este componente es el *AuthenticationProvider*, que dependiendo del tipo de autenticación que se precise, se nutre de otros componentes específicos para cada tipo de autenticación.

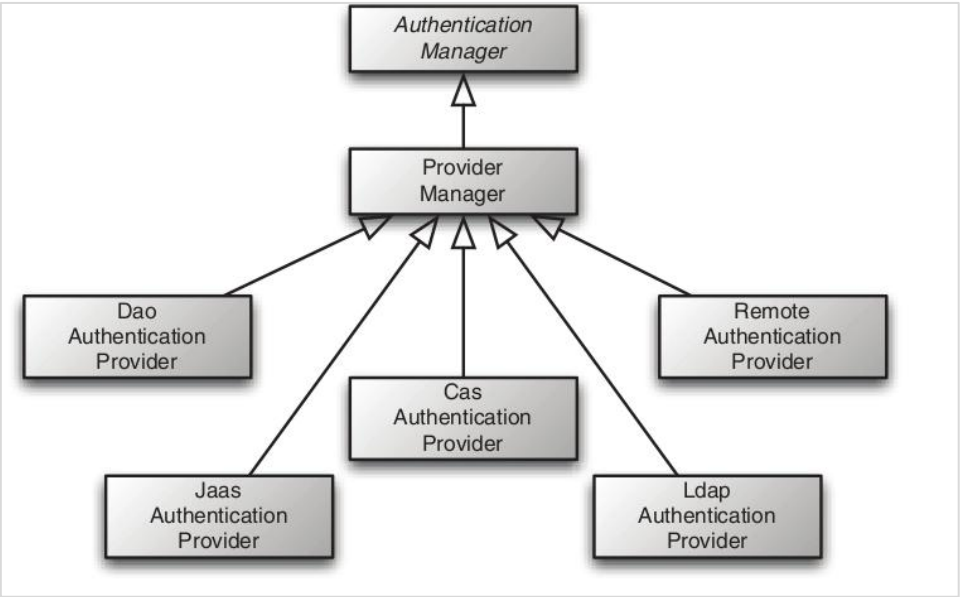

Figura III-8. Diagrama de clases que participan en la autenticación de Spring Security

En este proyecto, dadas sus características, el proveedor utilizado ha sido el *DaoAuthenticationProvider*, encargado de la autenticación de usuarios cuyas credenciales están almacenadas en una base de datos. Para su funcionamiento requiere de la clase *UserDetailsService* que es la encargada de ejecutar las consultas oportunas a partir del esquema de BBDD correspondiente. Su funcionamiento se refleja en la siguiente figura:

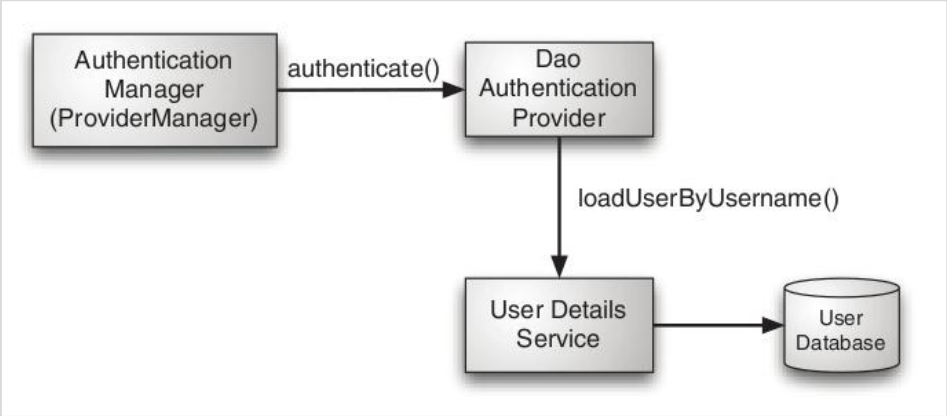

Figura III-9. Funcionamiento del DaoAuthenticationManager

Seguridad en aplicaciones Web

La seguridad que Spring Security ofrece a las aplicaciones Web están basadas en filtros Sevlets configurables desde el propio contexto de Spring. Sin embargo, para poder aprovechar esta funcionalidad es necesario declarar un filtro en el contexto web de la aplicación. Este filtro es el *DelegatingFilterProxy,* encargado de filtrar las peticiones que llegan a la aplicación web y redirigirlas a los filtros de Spring Security, configurados en el archivo XML donde se configuran el resto de Beans que Spring utiliza. Esta relación puede ser de tipo uno a uno, es decir, puede declararse un *DelegatingFilterProxy* por cada filtro declarado en el contexto de Spring o redirigir este filtro a un tipo especial llamado *FilterChainProxy* en el contexto de Spring, en el que se declaran el resto de filtros participantes en la aplicación.

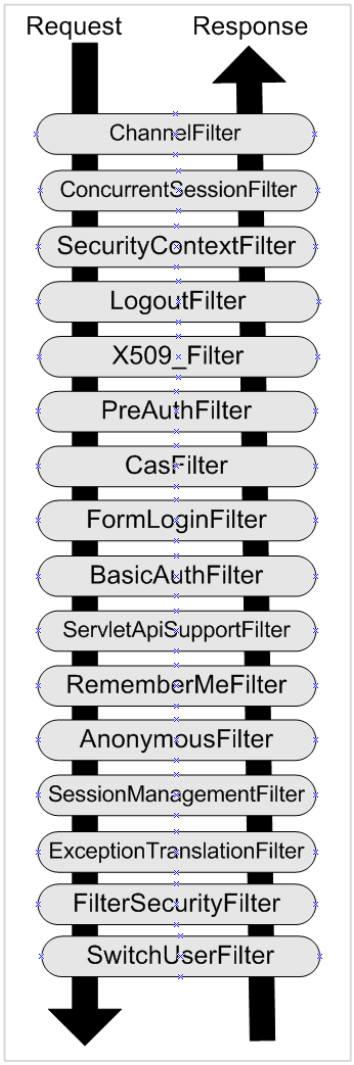

Figura III-10. Filtros Web de Spring Security

Tal y como se puede observar en la *Figura III-10. Filtros Web de Spring Security* existen multitud de filtros en Spring Security, cada uno con una funcionalidad

concreta dentro de los procesos de autenticación y autorización en cada petición y respuesta.

3.3.4 Jena

Jena es un Framework de Java que permite crear aplicaciones Web semánticas. Ofrece un entorno de programación para manejar esquemas de metadatos escritos en varios lenguajes, como RDF, RDFS y OWL, además de permitir la ejecución de consultas sobre dichos esquemas y utilizar un motor de inferencia. Jena es un producto de software libre fruto del trabajo de HP Labs Semantic Web Programme y se trata además de una recomendación del W3C para el tratamiento de esquemas de metadatos y descripciones de recursos.

Actualmente está disponible la versión número 2.6.4 de Jena(Jena, 2011), que incluye los siguientes componentes:

- API para RDF (Resource Description Framework).
- API de ontologías con soporte para OWL, DAML y RDF Schema.
- Lectura y escritura RDF en formato RDF/XML, N3 y N-Triples
- Motor de consultas SPARQL (ARQ).
- Almacenamiento en memoria y almacenamiento persistente.
- Subsistema de razonamiento.

A continuación se detallan las funcionalidades más importantes.

3.3.4.1 API para RDF.

Permite crear y manipular modelos RDF desde una aplicación Java, de forma que ofrece un amplio conjunto de clases para poder manejar todos los elementos necesarios para el tratamiento de dichos modelos. Dichos elementos son:

- **Recursos:** Todo aquello que se puede describir por una expresión RDF
- **Propiedades:** Características, atributos o relaciones usadas para describir un recurso
- **Literales:** Tipo de datos simple (String, Integer, etc.)

- **Sentencias (Statements):** recurso junto con una propiedad y con un valor asociado. Un modelo RDF es un conjunto de sentencias. Cada sentencia tiene tres partes:

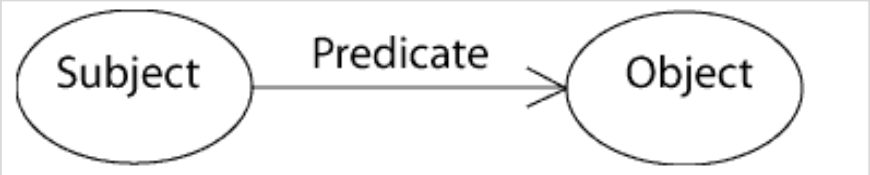

Figura III-11. Tripleta RDF

  - Sujeto: es el recurso a describir
  - Predicado: propiedad que define una característica del objeto
  - Objeto: es el valor que toma la propiedad para el recurso que define

- **Modelos**: son conjuntos de sentencias. Jena permite realizar las siguientes acciones sobre los modelos:
  - Creación de modelos
  - Escritura y lectura de modelos desde una fuente local, una base de datos o directamente desde un repositorio Web a partir de su URI.
  - Navegar un modelo a partir de la URI de un recurso
  - Consultar un modelo: se podrá buscar información del modelo y realizar consultas avanzadas.
  - Operaciones sobre modelos: unión, intersección y diferencia.

3.3.4.2 API para OWL.

El funcionamiento de la API de Jena para el tratamiento de lenguajes ontológicos como OWL está basado en la utilización de un lenguaje neutro como base para el tratamiento de los distintos lenguajes soportados por dicha API(Dickinson, 2009). Básicamente, se trata de representar el lenguaje sin hacer mención al lenguaje

subyacente que se está representando (OWL, RDF, etc.), por lo tanto es independiente del lenguaje. Tan sólo existe un perfil determinado para cada tipo de lenguaje, de forma que por cada uno de ellos se conocen los constructores permitidos así como los nombres de las clases y propiedades que utilizan.

Este perfil está unido al Ontology Model (OntModel) que es una versión extendida de la clase Model de Jena. La clase Model permite acceso a las sentencias en formato de recursos RDF. Cabe destacar que toda la información del modelo subyacente, ya sea mediante la utilización de la clase OntModel como con la clase Model, esta almacenada en forma de tripletas RDF (sujeto, predicado y objeto).

3.3.4.3 Almacenamiento en memoria y almacenamiento persistente.

El almacenamiento en memoria es la forma clásica con la que suelen trabajar habitualmente las aplicaciones. En el caso de aplicaciones que involucran el manejo de ontologías, equivale a tener el modelo OWL (ontología) almacenado en memoria principal, como si se tratase de una variable de programa. Una forma de trabajar con ontologías en Jena es declarando una variable *OntModel* e ir construyendo el modelo en la propia aplicación (creación de clases, subclases, propiedades, restricciones, etc). Con Jena también es posible cargar un modelo a partir de una fuente local, por ejemplo, un fichero que contenga una ontología definida en el lenguaje de marcado OWL.

El almacenamiento persistente surge ante la necesidad de guardar datos de programa de forma duradera para recuperarlos en otro momento. El término "*persistencia*" es sinónimo de "*durabilidad*" y "*permanencia*". Obviamente, el almacenamiento persistente en bases de datos supone grandes ventajas sobre el almacenamiento en memoria y el almacenamiento tradicional en el sistema de ficheros. Para ejecutar operaciones sobre bases de datos en una aplicación Java, es necesario JDBC (Java Database Connectivity). La API JDBC es una colección de interfaces Java y métodos de gestión de manejadores de conexión para cada modelo específico de base de datos. En este ámbito, Jena es capaz de trabajar con ontologías

almacenadas de forma persistente en una base de datos utilizando el driver JDBC(Blanco, 2008).

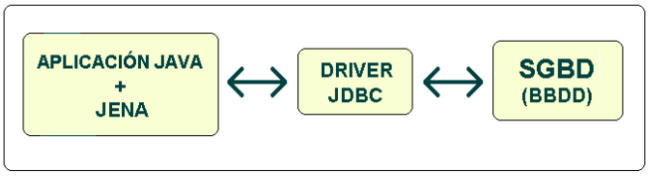

Figura III-12. Almacenamiento persistente de modelos

3.3.4.4 Inferencia en JENA.

Inferir consiste en deducir información adicional a partir de la existente. El código que realiza la tarea de inferir se le llama razonador (Reasoner). Jena incluye un conjunto básico de razonadores aunque nos permiter incluir cualquier otro:

- OWL Reasoner
- DAML Reasoner
- RDF Rule Reasoner
- Generic Rule Reasoner

La forma en la que Jena trata la inferencia es mediante la generación de setencias nuevas derivadas de todo aquello que el razonador ha podido deducir. De esta forma, se pueden lanzar consultas sobre los modelos tal y como se haría con un RDF plano.

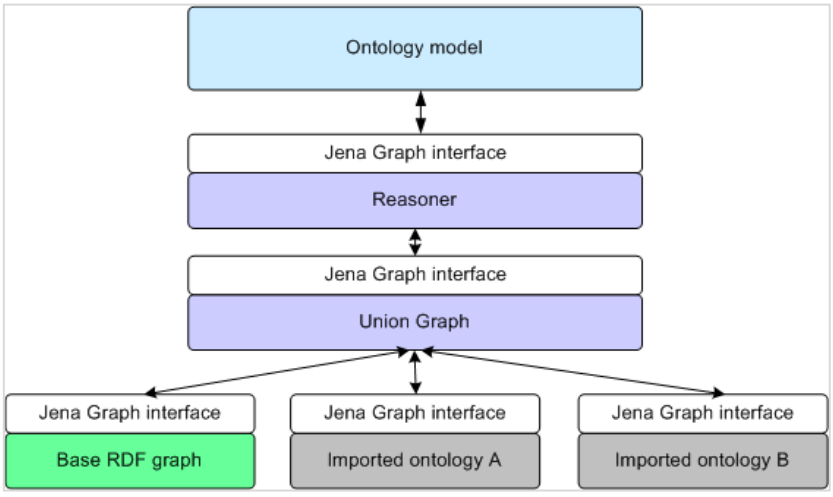

Figura III-13. Inferencia en Jena

La interfaz de Jena Graph es una interfaz interna que soporta la declaración de conjuntos de tripletas RDF. Por lo que todas las sentencias contenidas en un modelo, además de las que añade el motor de inferencia o razonador, están disponibles para la generación, de una forma muy sencilla, de los modelos ontológicos.

3.4 Otras herramientas

Además de las herramientas detalladas en los apartados anteriores, para la realización de este proyecto se han utilizado otras herramientas de apoyo para funciones concretas.

3.4.1 PgAdminIII

PgAdmin III es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, la más completa y popular con licencia Open Source(Guía Ubuntu, 2008). Está escrita en C++ usando la librería gráfica multiplataforma wxWidgets, que permite que se pueda usan en Linux, FreeBSD, Solaris, Mac OS X y Windows. Es capaz de gestionar versiones a partir de la PostgreSQL 7.3 ejecutándose en cualquier plataforma, así como versiones comerciales de PostgreSQL como Pervasive Postgres, EnterpriseDB, Mammoth Replicator y SRA PowerGres.

PgAdmin III está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. El interfaz gráfico soporta todas las características de PostgreSQL y facilita enormemente la administración. La aplicación también incluye un editor SQL con resaltado de sintaxis, un editor de código de la parte del servidor, un agente para lanzar scripts programados, soporte para el motor de replicación Slony-I y mucho más. La conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets (en plataformas *nix), y puede encriptarse mediante SSL para mayor seguridad.

PgAdmin es desarrollado por una gran comunidad de expertos en PostgreSQL alrededor del mundo y está disponible en una docena de lenguajes.

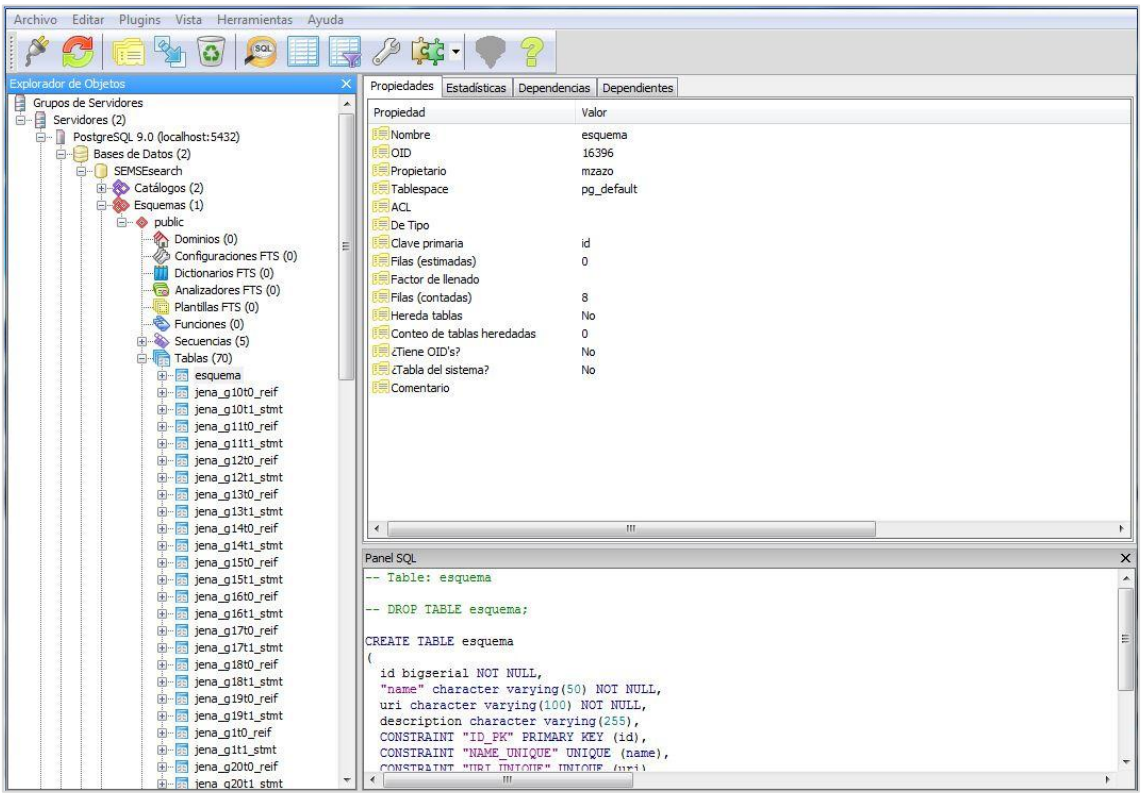

Figura III-14. PgAdminIII

3.4.2 Protégé

Protégé es una plataforma de código abierto que proporciona a una creciente comunidad de usuarios de un conjunto de herramientas para la construcción de modelos de dominio y aplicaciones basadas en el conocimiento mediante la utilización de ontologías(Protégé, 2011). En su núcleo, Protégé implementa un conjunto se estructuras de modelado del conocimiento y soporta acciones como la creación, visualización y manipulación de ontologías en varios formatos de representación.

Protégé se puede personalizar para ofrecer un entorno amigable y sencillo para la creación de modelos de conocimiento y la introducción de datos. Además, Protégé puede ser ampliado mediante una completa arquitectura de plugins y APIs basadas en Java para la creación de herramientas y aplicaciones.

3.4.2.1 Tipos principales de modelado de ontologías

Protégé soporta dos formas de modelado de ontologías: *Protégé-Frames* y *Protégé-OWL*.

Protégé-Frames

Ofrece una completa interfaz de usuario y un servidor de conocimiento para apoyar a los usuarios en la construcción y el almacenamiento de ontologías, personalizando los formularios de entrada de datos, y los propios datos en sí mismos. Protégé-Frames implementa un modelo de conocimiento que es compatible con el protocolo Open Knowledge Base Conectividad protocolo (OKBC).

En este modelo, una ontología consta de un conjunto de clases organizadas en una jerarquía de subcomponentes para representar los conceptos principales de un dominio, una serie de componentes asociados a las clases para describir sus propiedades y relaciones, y un conjunto de instancias de dichas clases.

Por tanto, las principales características de Protégé-Frames son:

- Un amplio conjunto de elementos de interfaz de usuario que pueden ser personalizados para permitir a los usuarios la creación de modelos de conocimiento y la introducción de datos en formularios sencillos y amigables.
- Una arquitectura de plug-ins que puede ser extendida con elementos de diseño personalizado, como puede ser gráficos, elementos multimedia, distintos formatos de almacenamiento, visualización de ontologías, inferencia y razonamiento, etc.
- Una API basada en Java que permite a los plug-ins y demás aplicaciones el acceso, uso y visualización de las ontologías creadas mediante Protégé-Frames.

Protégé-OWL

El editor de Protégé-OWL es una extensión de Protégé que soporta OWL. OWL es el desarrollo más reciente dentro de los estándares de idiomas de ontologías y se

trata de un lenguaje aprobado por el World Wide Web Consortium (W3C) con el objetivo de promover la Web Semántica.

El editor Protégé-OWL permite a los usuarios:

- Cargar y salvar ontologías OWL y RDF.
- Editar y visualizar clases, propiedades y reglas SWRL.
- Definir la lógica de las clases mediante expresiones OWL.
- Ejecutar razonadores como clasificadores lógicos descriptivos.
- Permitirá editar ontologías OWL individuales para marcado de la Web Semántica.

La flexibilidad de la arquitectura de Protégé-OWL hace más sencillo configurar y extender la herramienta. Protégé-OWL es compatible con la interfaz de Jena y tiene una API Java de código abierto para el desarrollo de componentes personalizados de la interfaz de usuario o de servicios para su uso en la Web Semántica.

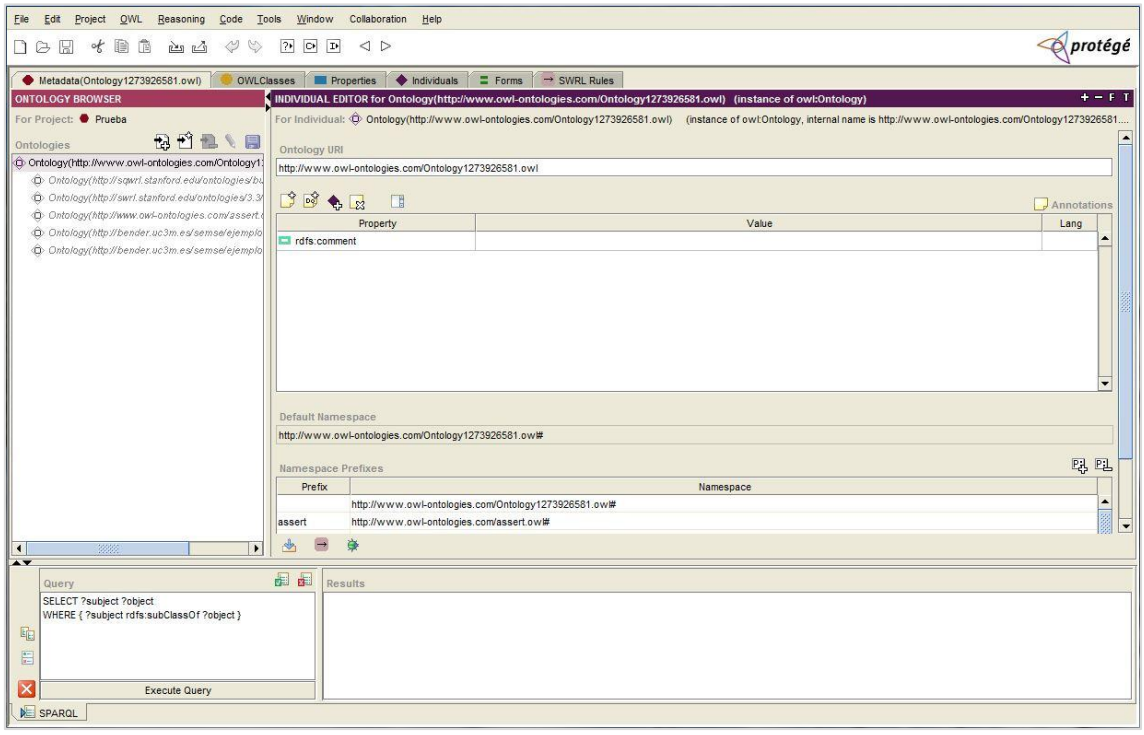

Figura III-15. Protégé

Capítulo IV. Desarrollo del proyecto

En este apartado se explica cómo ha sido desarrollado este proyecto y cómo, partiendo de la idea inicial del proyecto SEMSE y del trabajo ya realizado con anterioridad, surgió la necesidad de poder lanzar consultas sobre los esquemas de metadatos y ontologías existentes.

Es así como surgió la idea de este buscador. Un buscador público de conceptos sobre el material disponible del proyecto SEMSE y preparado para futuras adiciones de esquemas, con el objetivo de aumentar su operatividad y efectividad. De esta forma se ofrece al usuario interesado una forma rápida y eficaz de encontrar los esquemas de metadatos más útiles y populares de los publicados en la Web, que incluyan el concepto buscado o alguno relacionado semánticamente con él. De esta manera se evita que el usuario esté obligado a crearse un nuevo esquema o a tener que emplear más tiempo en realizar una búsqueda exhaustiva en la web o en repositorios que, a diferencia de SEMSE, no aplican el conocimiento semántico.

A continuación se detallan todas las fases de desarrollo llevadas a cabo en este proyecto, además de finalizar el capítulo con la planificación y presupuesto final.

4.1 Fase inicial

Esta primera fase es el punto de partida y arranque del proyecto. Comprende la recopilación de la información necesaria para iniciar el desarrollo de la aplicación, la especificación de requisitos y un primer análisis.

Desde un punto de vista general, la aplicación tiene como objetivo potenciar el uso de estructuras reusables de conocimiento (esquemas y ontologías) para su aplicación en el trabajo diario de los usuarios interesados en la materia. Se resuelven así los problemas planteados para la trascripción, registro, recuperación y explotación de documentos semánticos, mediante técnicas de ingeniería del conocimiento y, más específicamente, bajo el punto de vista de la ingeniería ontológica.

Para lograr este objetivo se partirá de la base ontológica generada en el proyecto SEMSE. Como ya se ha explicado anteriormente en el apartado *2.4 Semantic Metadata Search (SEMSE)*, por cada esquema de metadatos original, se cuenta con dos representaciones adicionales: el esquema cualificado semánticamente y la ontología específica de cada esquema. La primera representación tiene como objetivo el proporcionar al esquema de una estructura homogénea y compatible con el resto de recursos; mientras que la segunda, es la encargada de aportar la semántica de cada elemento en él contenido. Adicionalmente se cuenta con la ontología de referencia, que contiene todos los significados de los conceptos incluidos en todos los esquemas generados en el proyecto SEMSE y que además, son desambiguados mediante la ontología de alineamiento.

Estos recursos formarán la base sobre la que actuará esta aplicación. De este modo, a partir de un concepto buscado por el usuario, se le ofrecerá un listado de los esquemas de metadatos que contienen dicho concepto y conceptos semánticamente relacionados, facilitándose así la reutilización de conceptos y esquemas.

Como objetivo adicional, el sistema permitirá la gestión de la base ontológica, esto es: los esquemas originales, sus esquemas cualificados y ontologías específicas, la ontología de referencia y la ontología de alineamiento, permitiendo su adición o eliminación en cualquier momento.

Como requisitos generales de la aplicación se ha establecido las siguientes capacidades:

Seguridad

La seguridad debe ser un aspecto clave. Habrá que ser conscientes de los posibles ataques a los que se está expuesto en cuanto el servidor está conectado a la red.

Adecuación a Estándares

Las páginas estáticas que se generen, así como las generadas por la aplicación Web deben estar validadas contra un estándar apropiado, para facilitar la interpretación de la información a terminales de reducidas capacidades e incluso a navegadores especiales, favoreciendo la ruptura de la brecha digital y la accesibilidad de la información.

Facilidad de uso

La aplicación debe reducir al mínimo el esfuerzo del usuario a la hora de llevar a cabo sus tareas, presentando una interfaz clara y sencilla y cumpliendo en la medida de lo posible los puntos descritos en el apartado *2.3.1 Elementos clave de sitios Web centrados en usuario*, al respecto de la usabilidad de un sitio Web.

Fiabilidad

Las herramientas escogidas para el servidor en producción deberán ser estables y confiables para garantizar el buen funcionamiento en todo momento.

Escalabilidad

El diseño de la arquitectura del sistema contemplará la posibilidad de ser ampliada con el fin de proporcionar servicio a un mayor número de usuarios mediante una infraestructura de mayor capacidad. Además también se contempla el crecimiento

de la propia aplicación con el paso del tiempo, pudiendo ser ampliada con mayor contenido y nuevas funcionalidades.

Software Libre

La infraestructura y los programas utilizados para la elaboración del proyecto estarán basados en software libre, tanto para minimizar el coste en software, como para ofrecer una solución que no esté supeditada a un fabricante o distribuidor de software.

4.1.1 Proceso de desarrollo

Debido a la naturaleza del proyecto se ha optado por un modelo de Ciclo de Vida "*híbrido*", consistente en una especificación de requisitos y un análisis en cascada[29] para pasar a un diseño arquitectónico e implementación de modo iterativo-incremental[30].

A continuación se enumeran las motivaciones que han conducido a la elección de este ciclo de vida:

1. **Complejidad:** Debido a que este proyecto forma parte de un proyecto mayor, los requisitos estaban claramente definidos desde un primer momento, tanto el objetivo principal como la mejor forma de llegar a él. Por este motivo, en un primer momento se eligió un modelo de ciclo de vida en cascada. Sin embargo, dada la complejidad una vez comenzada las partes de diseño y desarrollo, se optó por acometer estas últimas fases basándose en un modelo iterativo-incremental, en función de los objetivos que se iban cumpliendo.

[29] La vida de los sistemas pasa por una serie de fases consecutivas y separables. Las fases se desarrollan en secuencia y sólo una vez, aunque puede haber iteraciones dentro de cada fase.

[30] Este ciclo de vida comienza determinando la especificación de requisitos y realizando un análisis del sistema. A partir de ese momento se realiza el resto del desarrollo como una secuencia de entregables en que cada entregable incorpora una parte de las capacidades planificadas.

2. **Riesgo tecnológico**: existían riesgos derivados del desconocimiento de tecnologías existentes para la gestión de ontologías y la forma de realizar consultas sobre ellas. Este ciclo de vida permitía asumir dichos riesgos.
3. **Dinamismo y Adaptabilidad**: Dado que se podían producir cambios y que estos cambios deberían estar en el producto final, este ciclo de vida permitía acercar el sistema a la solución de forma progresiva, permitiendo el desarrollo paralelo y la especialización del equipo de desarrollo en función del componente a desarrollar.

El inicio de cada etapa de este ciclo de vida debe esperar a la finalización de la etapa inmediatamente anterior. Estas etapas o fases son:

- **Análisis de requisitos:** Se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. Los requisitos son obtenidos a partir de las necesidades de los usuarios y refinados por los analistas. De esta fase surge un documento de requisitos, que contiene la especificación completa de lo que debe hacer el sistema sin entrar en aspectos tecnológicos. Es importante señalar que en esta etapa, los requisitos del sistema deben ser consensuados y aprobados por el usuario.
- **Diseño de la arquitectura del sistema:** Se descompone y organiza el sistema en elementos que puedan desarrollarse por separado. El resultado es la descripción de la estructura global del sistema y la especificación de lo que debe hacer cada uno de sus componentes, así como las relaciones entre ellos.
- **Diseño detallado del sistema:** Es la fase en donde se descomponen y detallan los componentes identificados en la arquitectura del sistema, incluso los algoritmos necesarios para el cumplimiento de los requerimientos del usuario.
- **Codificación:** Es la fase de programación o implementación propiamente dicha. Aquí se implementa el código fuente, haciendo uso de prototipos así como pruebas y ensayos para corregir errores. Dependiendo del lenguaje de programación y su versión se crean las librerías y componentes reutilizables dentro del mismo proyecto para agilizar y simplificar el proceso de programación.

- **Pruebas:** Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente antes de ser instalado en producción. Esta fase resulta fundamental para asegurar el correcto funcionamiento del sistema, en especial cuando un error o fallo del sistema pueda provocar pérdidas económicas o pueda asumir responsabilidad civil.
- **Implantación:** Esta fase es en la que se realiza el despliegue de la aplicación en el entorno productivo que utilicen los usuarios finales. Así mismo, puede incluir o no, dependiendo del proyecto y lo pactado con el usuario final, de una fase de mantenimiento, en la que se le da soporte al usuario y se corrigen los posibles errores no aparecidos en fases anteriores.

4.1.2 Especificación de requisitos de Usuario.

En este punto se pretende obtener una idea general de los requisitos funcionales, esto es, la funcionalidad, que debe proporcionar la aplicación. Para ello se elaboró un conjunto de especificaciones informales.

Especificaciones informales.

El objetivo principal de la aplicación será la creación de una página web que ofrezca al usuario la posibilidad de realizar búsquedas de conceptos que estén presentes en los esquemas de metadatos dados de alta en el sistema. Para ello el sistema deberá de ser capaz de, a partir de la URI de un esquema, obtenerlo, interpretarlo y guardarlo en una base de datos para su posterior recuperación y utilización.

Adicionalmente, el sistema deberá de ser capaz de poder ejecutar consultas sobre todos los esquemas con el fin de localizar el concepto introducido por el usuario, a partir de la búsqueda en los nombres y en todas las propiedades de los conceptos. Concretamente existirán tres tipos de búsqueda:

- Búsqueda sintáctica: Será aquella en la que se busque el concepto introducido por el usuario en el nombre y el resto de propiedades de cada uno de los

conceptos incluidos en todas las ontologías de específicas de los esquemas originales.

- Búsqueda conceptual: Adicionalmente a los resultados obtenidos en la búsqueda sintáctica, se añaden los resultados obtenidos de dos maneras adicionales:
    - También se buscarán aquellos conceptos que, mediante la ontología de alineamiento, sean equivalentes a algún concepto de la ontología de referencia coincidente con el concepto introducido por el usuario.
    - Basándose en la ontología de alineamiento, también se incluirán aquellos conceptos equivalentes entre sí a partir de los resultados de la búsqueda sintáctica. Esto es, todos aquellos conceptos que sean equivalentes con un mismo concepto de la ontología de referencia, en este caso sin que necesariamente el concepto de la ontología de referencia sea coincidente con el concepto introducido por el usuario.

  Debido a que muchos conceptos pueden aparecer como resultado en ambos métodos de búsqueda, el resultado final tendrá en cuenta esto y por lo tanto, se eliminarán los resultados duplicados.
- Búsqueda contextual: Adicionalmente a los resultados de la búsqueda conceptual y de nuevo, basándose en la ontología de alineamiento, se incluirán todos aquellos conceptos que resulten equivalentes a los conceptos padre del concepto resultado en la ontología de referencia.

De forma adicional, se le ofrecerá al usuario la posibilidad de ejecutar la búsqueda conceptual y contextual en dos pasos, de forma que, el usuario pueda intervenir en la mitad del proceso para enfocar la búsqueda hacia una dirección más adecuada a sus necesidades.

Finalmente y respecto a la usabilidad y el diseño de la aplicación, ésta será fácil de usar y mantendrá un estilo directo y minimalista.

4.1.2.1 Diagrama de casos de uso inicial.

Definidos los objetivos y especificaciones informales, se inicia el proceso de análisis del sistema correspondiente a la fase actual. En la primera iteración del proceso se estudia el sistema como un todo y se determinan sus límites y contenidos.

Existirán dos tipos de usuarios que puedan interactuar con la aplicación. Cada tipo de usuario representará un rol en el sistema. Se definen a continuación:

- **Usuario:** Usuario de la aplicación que puede realizar cualquiera de las búsquedas disponibles. Este será el usuario público y anónimo con el que se acceda al sistema.
- **Administrador:** Usuario encargado de la gestión de esquemas. Responsable del contenido que se almacenará en la base de datos y por lo tanto, de la base de conocimiento sobre la que se realizarán las búsquedas de conceptos. Además, como un usuario más del sistema heredará la funcionalidad de éstos, por lo que también podrá realizar búsquedas.

A continuación se presenta el diagrama de casos de uso inicial, en el cual se puede observar la funcionalidad básica con la que iba a contar el sistema en un primer momento:

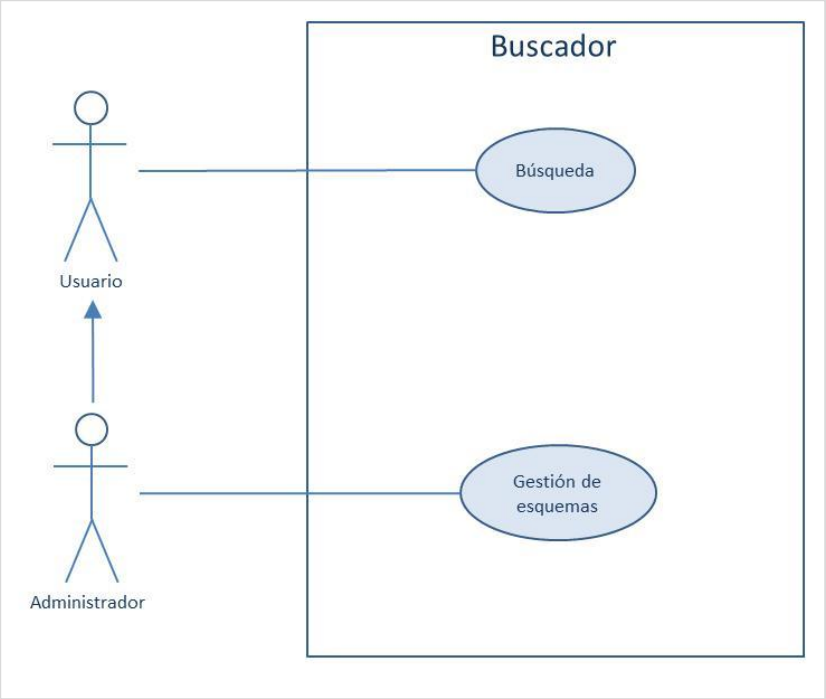

Figura IV-1. Diagrama de Casos de Uso inicial

Como se puede observar en el diagrama de la *Figura IV-1. Diagrama de Casos de Uso inicial*, el sistema cuenta con dos funcionalidades básicas: la realización de búsquedas y la gestión de esquemas de metadatos. A continuación se detallan cada una ellas.

Búsquedas

Esta es sin duda la funcionalidad principal del sistema cuya implementación debe de ser el objetivo fundamental del desarrollo. Dicha funcionalidad consiste en ofrecer al usuario la posibilidad de buscar un concepto localizado en los esquemas de metadatos contenidos en el sistema. De esta forma, el sistema deberá recoger dicho concepto, buscarlo en los esquemas y responder al usuario con un listado completo de cada coincidencia encontrada, independientemente del tipo de búsqueda realizada, como se detalla más adelante, cada resultado incluirá la URI del concepto de las ontologías específicas, su definición y la URI del concepto del esquema original en el que se encuentra. Adicionalmente, el sistema permitirá la navegabilidad al esquema original para que el usuario pueda visualizarlo al completo.

Gestión de esquemas

Esta es una funcionalidad exclusiva del usuario administrador. Debe de contemplar la posibilidad de agregar o eliminar esquemas, entendiendo como esquemas y tal como se comentaba en el apartado *2.4 Semantic Metadata Search (SEMSE)*, un esquema original, su esquema cualificado semánticamente y su ontología específica. Además, también permitirá la gestión de la ontología de referencia y la de alineamiento, manteniendo la referencia de cada una de ellas.

4.2 Fase de análisis

En este apartado se presenta una visión general del proceso de análisis, paso previo al diseño de la aplicación. Es por ello que se realiza un estudio más detallado y preciso de los requisitos de usuario definitivos que implementa la aplicación desde el punto de vista del desarrollador. Como se detalla a continuación, estos requisitos se han plasmado en los diagramas de casos de uso de la fase de análisis.

4.2.1 Diagrama de casos de uso de la fase de análisis

En esta fase se realiza un estudio más específico de los requisitos de usuario recopilados en la fase anterior. Desde un punto de vista más técnico, los requisitos de completan, refinan y precisan en mayor medida con el fin de disponer de un conjunto de requisitos lo más definitivo posible.

De este modo, se desarrolló una primera especificación de todo el sistema, con el fin de obtener una visión completa y más detallada de las necesidades que debía cubrir. Si bien la aparición de nuevos requisitos es inevitable, al menos se reduce su número y se maximiza la estabilidad de los obtenidos. Los requisitos, en forma de casos de uso, se muestran a continuación.

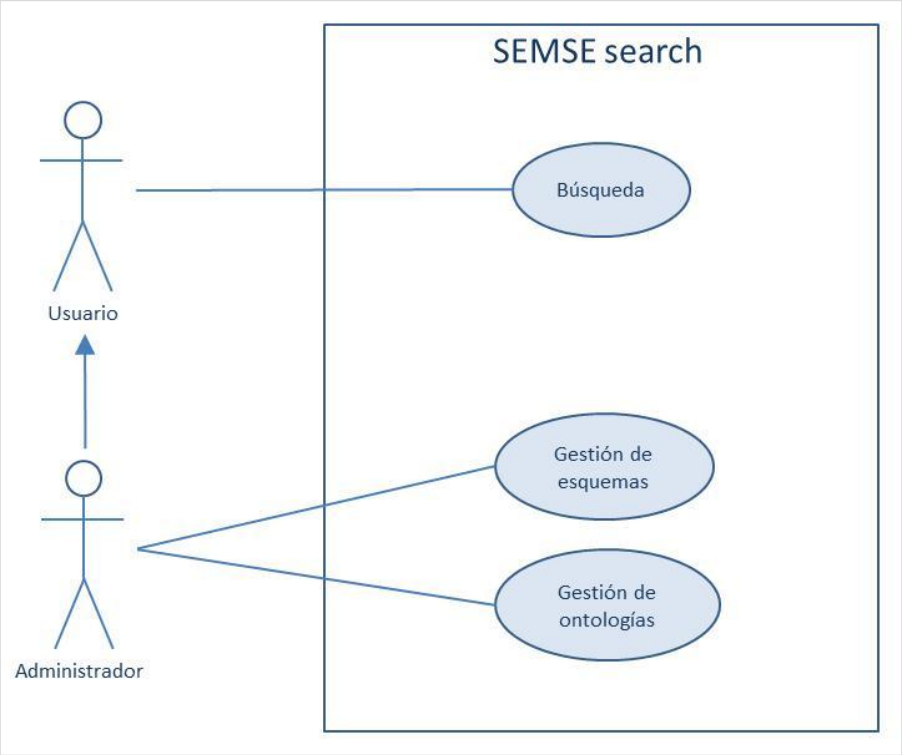

Figura IV-2. Diagrama general de Casos de Uso

Como se puede observar a simple vista en la *Figura IV-2. Diagrama general de Casos de Uso* los casos de uso a nivel global respecto al diseño de la fase inicial parece no diferenciarse demasiado, sin embargo, en esta fase se especificó más detalladamente qué tipos de búsquedas permitirá ejecutar la aplicación, así como la inclusión de una nueva funcionalidad que permitiese la consulta y modificación de las URIs de la ontología de referencia y de mapeo.

A continuación, veremos en detalle los casos de uso de toda la funcionalidad representada en el diagrama de casos de uso global representado en la *Figura IV-2. Diagrama general de Casos de Uso* se ofrece el objetivo del mismo, los actores que están involucrados en él, las precondiciones (estado del sistema que se tiene que cumplir para que el caso de uso se pueda instanciar), las postcondiciones (estado del sistema una vez instanciado el caso de uso) y el escenario básico (pasos principales del caso de uso ordenados).

Los casos de uso se han priorizado según la necesidad e importancia de los mismos, dada la extensión del proyecto. Los distintos grados de prioridad establecidos son: alta, media y baja. El objetivo de la asignación de prioridades a los casos de uso es establecer el orden en el cual se implementarán. De este modo el orden de implementación se establecerá de mayor a menor prioridad.

El orden de presentación de los casos de uso es tal y como aparecen en la *Figura IV-2. Diagrama general de Casos de Uso*, orden que coincide además con la prioridad asignada a cada caso.

4.2.1.1 Búsqueda

El caso de uso de Búsqueda, tal como veremos a continuación, se ha desglosado en tres casos de uso. Cada uno identifica un tipo de búsqueda diferente, en función de la forma en que se localizan los conceptos equivalentes al de búsqueda, y que, inicialmente se localizan mediante una búsqueda sintáctica sobre el contenido de los metadatos. Es por ello que podemos distinguir tres tipos de búsqueda: la búsqueda sintáctica; la búsqueda conceptual, que incluye a la anterior; y la búsqueda contextual, que incluye a las dos anteriores.

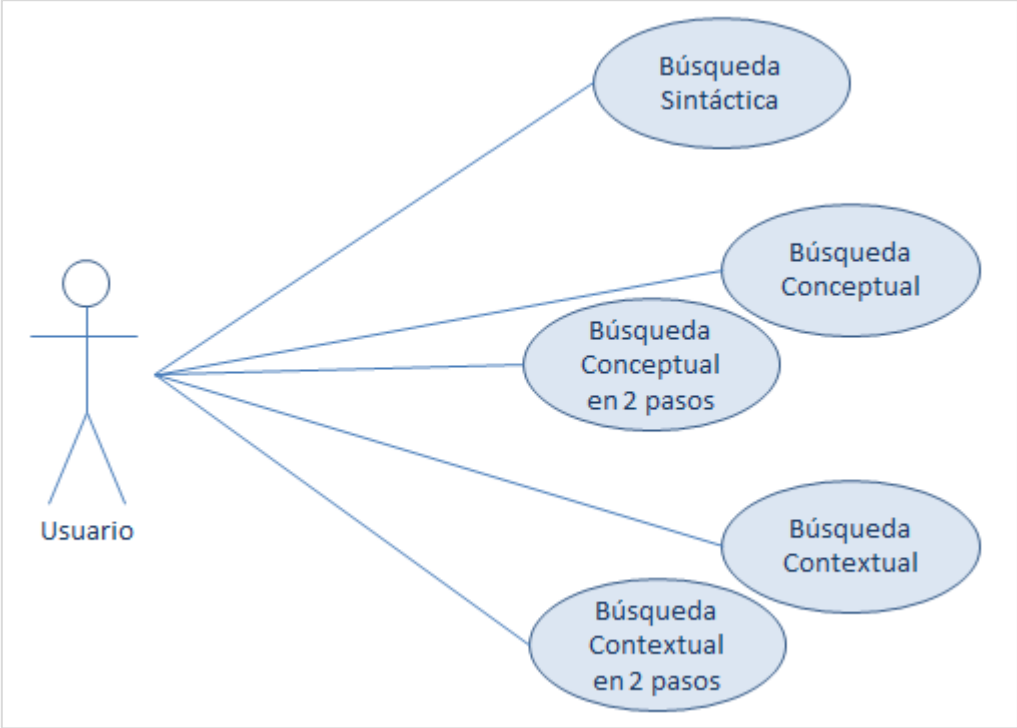

Figura IV-3. Caso de Uso Búsqueda

CU-01: Búsqueda Sintáctica			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Búsqueda sintáctica en las ontologías específicas de cada esquema de metadatos, de forma que se buscan coincidencias en el nombre del concepto y en sus propiedades.			
Escenario básico:	1. Introducir el concepto que se desea buscar. 2. Realizar la búsqueda sintáctica en esquemas. 3. Mostrar los resultados.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			
R-01: Listado de resultados. R-04: SuperModelo como base de conocimiento.			

CU-02: Búsqueda Conceptual			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se amplía la búsqueda sintáctica también a los conceptos de la ontología de referencia que aparecen en la ontología de alineamiento, por lo que se recuperan los conceptos de las ontologías específicas de cada esquema equivalentes a estos últimos. Adicionalmente, se incluyen también todos aquellos conceptos de las ontologías específicas que, mediante la ontología de mapeo, sean equivalentes a los que se encontraron en la búsqueda sintáctica inicial y no se hubieran obtenido ya.			
Escenario básico:	1. Introducir el concepto que se desea buscar. 2. Realizar búsqueda sintáctica en esquemas. 3. Obtener conceptos equivalentes mediante la ontología de alineamiento mediante la búsqueda sintáctica del nombre del concepto de la ontología de referencia del que son semánticamente equivalentes. 4. Obtener conceptos equivalentes a los obtenidos ya en los pasos anteriores mediante la ontología de alineamiento cuando los conceptos comparten una misma equivalencia con un mismo concepto de la ontología de referencia, sin que se aplique necesariamente la búsqueda sintáctica sobre este último. 5. Eliminar resultados duplicados. 6. Mostrar los resultados.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			
CU-01: Búsqueda Sintáctica. CU-03: Búsqueda Conceptual en 2 pasos. R-01: Listado de resultados. R-04: SuperModelo como base de conocimiento.			

CU-03: Búsqueda Conceptual en 2 pasos			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Media	Modificable	Usuario	1.0
Objetivo:			
En vez de realizar una búsqueda conceptual completa, se permite la interacción con el usuario en la mitad del proceso con el fin de poder refinar los resultados de la búsqueda a sus necesidades. De esta forma se le muestra un primer listado con los resultados de la búsqueda sintáctica y los conceptos equivalentes resultantes de aplicar la búsqueda sintáctica sobre el nombre de los conceptos de la ontología de referencia que aparecen en la ontología de alineamiento. Sobre este listado, el			

usuario podrá seleccionar uno de ellos, por lo que se le devolverán todos aquellos conceptos equivalentes al concepto de la ontología de referencia al que fuera equivalente el concepto seleccionado.	
Escenario básico:	1. Introducir el concepto que se desea buscar. 2. Realizar búsqueda sintáctica en esquemas. 3. Obtener conceptos equivalentes mediante la ontología de alineamiento mediante la búsqueda sintáctica del nombre del concepto de la ontología de referencia del que son semánticamente equivalentes. 4. Mostrar los resultados y esperar la selección de uno de ellos. 5. Obtener conceptos equivalentes al concepto seleccionado mediante la ontología de alineamiento cuando los conceptos comparten una misma equivalencia con un mismo concepto de la ontología de referencia. 6. Mostrar los resultados.
Precondiciones:	-----
Pos condiciones:	-----
Relacionado con:	
CU-01: Búsqueda Sintáctica. CU-02: Búsqueda Conceptual. R-01: Listado de resultados. R-04: SuperModelo como base de conocimiento.	

CU-04: Búsqueda Contextual			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Adicionalmente, por cada resultado obtenido en la búsqueda conceptual, se localizan los conceptos que, mediante la ontología de alineamiento, sean equivalentes a los conceptos padre del concepto resultado en la ontología de referencia.			
Escenario básico:	1. Introducir el concepto que se desea buscar. 2. Realizar la búsqueda conceptual. 3. Mostrar los resultados y esperar a que el usuario seleccione uno de ellos. 4. Obtener los conceptos equivalentes al concepto padre de la ontología de referencia al que fuera equivalente el concepto seleccionado por el usuario. 5. Mostrar los resultados.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			

CU-01: Búsqueda Sintáctica. CU-02: Búsqueda Conceptual. CU-05: Búsqueda Contextual en 2 pasos. R-01: Listado de resultados. R-04: SuperModelo como base de conocimiento.

CU-05: Búsqueda Contextual en 2 pasos			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Media	Modificable	Usuario	1.0
Objetivo:			
En vez de realizar una búsqueda contextual completa, se permite la interacción con el usuario en la mitad del proceso con el fin de poder refinar los resultados de la búsqueda a sus necesidades. De esta forma se le muestra un primer listado con los resultados de la búsqueda conceptual. Sobre este listado, el usuario podrá seleccionar uno de ellos, por lo que se le devolverán todos aquellos conceptos equivalentes al concepto padre de la ontología de referencia al que fuera equivalente el concepto seleccionado.			
Escenario básico:	1. Introducir el concepto que se desea buscar. 2. Realizar la búsqueda contextual. 3. Mostrar los resultados.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			
CU-01: Búsqueda Sintáctica. CU-02: Búsqueda Conceptual. CU-05: Búsqueda Contextual en 2 pasos. R-01: Listado de resultados. R-04: SuperModelo como base de conocimiento.			

4.2.1.2 Gestión de esquemas

La gestión de esquemas engloba el alta y la baja de esquemas de metadatos de la base de datos sobre la que trabaja la aplicación.

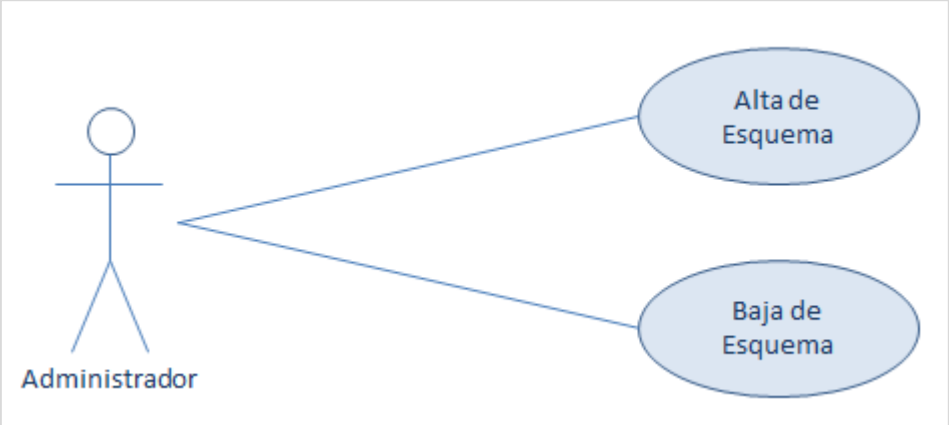

Figura IV-4. Caso de Uso Gestión de Esquemas

CU-06: Nuevo Esquema			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Media	Modificable	Administrador	1.0
Objetivo:			
Añadir un esquema original a la base de datos, así como su esquema cualificado y su ontología específica.			
Escenario básico:	1. Introducir los datos del nuevo esquema. 2. Comprobación de la disponibilidad del nuevo esquema, así como de su esquema semánticamente cualificado y su ontología específica. 3. Almacenamiento del esquema cualificado semánticamente y la ontología de referencia. 4. Creación del registro del esquema original como esquema dado de alta en la base de datos. 5. Actualización del SuperModelo.		
Precondiciones:	La URI del esquema debe de existir, así como las URIs de su esquema cualificado y de su ontología específica. El esquema no debe encontrarse dado de alta en la base de datos.		
Pos condiciones:	El esquema cualificado semánticamente y la ontología específica quedan añadidos a la base de datos. Se crea un registro con las propiedades y la URI del esquema original. El SuperModelo queda actualizado con la nueva información.		
Relacionado con:			
R-02: Información de un esquema. R-06: Actualización del SuperModelo. R-07: Convenciones del sistema de identificación de los esquemas cualificados semánticamente y de las ontologías específicas a partir de la URI del esquema original.			

R-08: Persistencia de la información. R-10: Persistencia de los esquemas originales. R-11: Persistencia de los esquemas semánticamente cualificados y las ontologías de referencia.

CU-07: Eliminar Esquema			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Media	Modificable	Administrador	1.0
Objetivo:			
Eliminar un esquema original a la base de datos, así como su esquema cualificado y su ontología específica.			
Escenario básico:	1. Eliminación del esquema cualificado semánticamente y la ontología de referencia del esquema seleccionado. 2. Eliminación del registro del esquema original en la base de datos. 3. Actualización del SuperModelo.		
Precondiciones:	El esquema debe de existir en la base de datos.		
Pos condiciones:	El esquema queda eliminado de la base de datos. El SuperModelo queda actualizado para no tener en cuenta el esquema eliminado.		
Relacionado con:			
R-02: Información de un esquema. R-06: Actualización del SuperModelo. R-07: Convenciones del sistema de identificación de los esquemas cualificados semánticamente y de las ontologías específicas a partir de la URI del esquema original. R-10: Persistencia de los esquemas originales. R-11: Persistencia de los esquemas semánticamente cualificados y las ontologías de referencia.			

4.2.1.3 Gestión de Ontologías

La gestión de ontologías permite consultar y modificar las referencias a la ontología de alineamiento y a la ontología de referencia. De esta forma es posible adaptar la aplicación en caso de que estas ontologías sean identificadas de otra manera, cambien de ubicación o incluso se quiera sustituir por otras diferentes que aporten otra semántica o amplíen la ya existente.

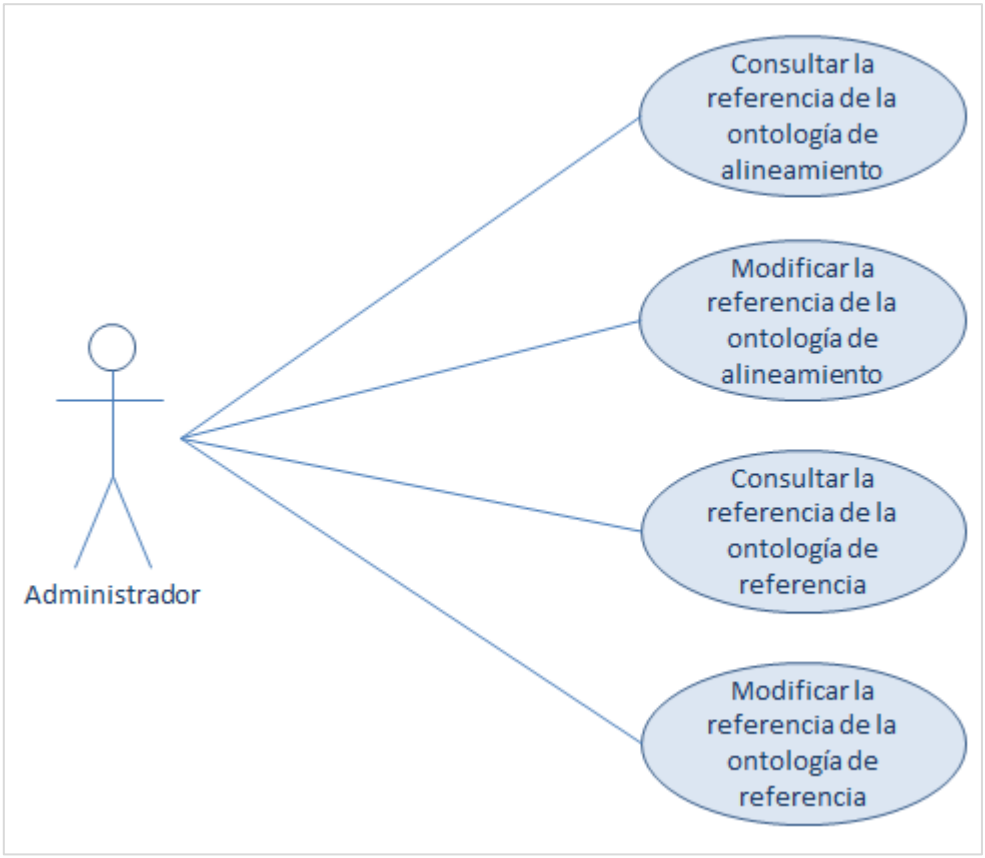

Figura IV-5. Casos de uso Gestión de Ontologías

CU-08: Consultar la referencia de la ontología de alineamiento			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Baja	Modificable	Administrador	1.0
Objetivo:			
Consultar la referencia que identifica la ontología de alineamiento.			
Escenario básico:	1. Localizar la referencia de la ontología de alineamiento. 2. Mostar la información al usuario.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			
R-09: Persistencia de la ontología de alineamiento y de referencia.			

CU-09: Modificar la referencia de la ontología de alineamiento			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Baja	Modificable	Administrador	1.0
Objetivo:			
Modificar la referencia que identifica la ontología de alineamiento.			
Escenario básico:	1. Localizar la referencia de la ontología de alineamiento. 2. Mostrar la información al usuario y esperar a que realice las modificaciones. 3. Guardar los cambios efectuados. 4. Actualización del SuperModelo.		
Precondiciones:	-----		
Pos condiciones:	La información de la ontología de alineamiento es almacenada. El SuperModelo queda actualizado con la nueva información.		
Relacionado con:			
R-06: Actualización del SuperModelo. R-09: Persistencia de la ontología de alineamiento y de referencia.			

CU-10: Consultar la referencia de la ontología de referencia			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Baja	Modificable	Administrador	1.0
Objetivo:			
Consultar la referencia que identifica la ontología de referencia			
Escenario básico:	1. Localizar la referencia de la ontología de referencia. 2. Mostar la información al usuario.		
Precondiciones:	-----		
Pos condiciones:	-----		
Relacionado con:			
R-09: Persistencia de la ontología de alineamiento y de referencia.			

CU-09: Modificar la referencia de la ontología de referencia			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Baja	Modificable	Administrador	1.0
Objetivo:			
Modificar la referencia que identifica la ontología de referencia			
Escenario básico:	1. Localizar la referencia de la ontología de referencia. 2. Mostrar la información al usuario y esperar a que realice las modificaciones. 3. Guardar los cambios efectuados. 4. Actualización del SuperModelo.		

Precondiciones:	-----
Poscondiciones:	Los datos de la ontología de referencia quedan almacenados en el sistema. El SuperModelo queda actualizado con la nueva información.
Relacionado con:	
R-06: Actualización del SuperModelo. R-09: Persistencia de la ontología de alineamiento y de referencia.	

Dado que los diagramas de casos de usos son una herramienta para la obtención de requisitos, se entienden todos los casos de uso anteriores como requisitos propios de la aplicación. Adicionalmente, y dado que son necesarios para la correcta aplicación de estos se define una lista de requisitos adicionales que completan la definición de los requisitos anteriores.

R-01: Listado de resultados		
Prioridad	**Estabilidad**	**Versión**
Media	Modificable	1.0
El resultado de todas las consultas deberá de ser un listado en el que cada resultado incluya: • URI del concepto • Definición • URI del esquema original. Deberá de ser navegable para que el usuario pueda visualizarlo al completo.		

R-02: Información de un esquema		
Prioridad	**Estabilidad**	**Versión**
Media	Modificable	1.0
La información necesaria para el alta de un nuevo esquema de matadatos deberá de ser la siguiente: • Nombre. • URI. • Descripción. Siendo el Nombre y la URI valores únicos. El nombre será utilizado como identificador del esquema en la base de datos.		

R-03: SuperModelo como información a gestionar por el sistema		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0

El sistema deberá de mantener un esquema único formado por todos los esquemas semánticamente cualificados, todas las ontologías específicas, la ontología de alineamiento y la ontología de referencia.

R-04: SuperModelo como base de conocimiento		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
Todas las consultas llevadas a cabo por la aplicación como requisito necesario para la realización de las búsquedas deberán de ejecutarse sobre el SuperModelo, siendo este una representación de todo el conocimiento contenido en el sistema.		

R-05: Generación del SuperModelo		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
El supermodelo deberá de generarse siempre que se ejecute la aplicación en el servidor.		

R-06: Actualización del SuperModelo		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
Siempre que se produzca un cambio en la gestión de esquemas o en las referencias de las ontologías de alineamiento o de referencia deberán de aplicarse en el SuperModelo, por lo que éste deberá de ser regenerado para asegurar, de esta manera, que cualquier búsqueda ejecutada a partir de ese momento por cualquiera de los usuarios responda con la información más actualizada.		

R-07: Convenciones del sistema de identificación de los esquemas cualificados semánticamente y de las ontologías específicas a partir de la URI del esquema original		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
De la URI del esquema original se obtendrá el nombre del esquema. La construcción de las URIs del esquema cualificado semánticamente y la ontología específica se realizará mediante la aplicación de un prefijo o un sufijo: • Esquema cualificado semánticamente: “sq” + nombre • Ontología específica: nombre + “so” El resto de la URI permanecerá igual.		

R-08: Persistencia de la información		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
Toda la información que por su importancia necesite ser persistente será almacenada en una base de datos, la cual será gestionada por un Sistema Gestor de Bases de Datos adecuado.		

R-09: Persistencia de la ontología de alineamiento y de referencia		
Prioridad	**Estabilidad**	**Versión**
Media	Modificable	1.0
Las URIs de la ontología de alineamiento y de la de referencia serán almacenadas en un fichero externo, siendo además modificado si dichas URIs son modificadas desde la aplicación.		

R-10: Persistencia de los esquemas originales		
Prioridad	**Estabilidad**	**Versión**
Media	Modificable	1.0
Debido a que se realizan dos nuevas representaciones de los esquemas originales que le aportan homogeneidad, consistencia y semántica, resulta innecesario la persistencia de estos esquemas más allá del almacenamiento de la información contenida en el requisito R-02.		

R-11: Persistencia de los esquemas semánticamente cualificados y las ontologías de referencia		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
Con el fin de mejorar el rendimiento de la aplicación y más concretamente de la generación y actualización del SuperModelo, tanto los esquemas cualificados y las ontologías de referencia de todos los esquemas serán almacenados en la base de datos utilizada por la aplicación.		

R-12: Interfaz de usuario		
Prioridad	**Estabilidad**	**Versión**
Media	Modificable	1.0

La interfaz de usuario deberá de ser sencilla, limpia y minimalista, con el fin de centrar la usabilidad de la aplicación en la correcta ejecución de las búsquedas y la visualización de los resultados.

R-13: Despliegue de la aplicación		
Prioridad	**Estabilidad**	**Versión**
Alta	Modificable	1.0
Debido a que el objetivo de esta aplicación es el ofrecer la información de la que dispone a todo aquel que esté interesado deberá de estar publicada en Internet como una aplicación accesible vía Web mediante un navegador Web estándar.		

4.3 Diseño Arquitectónico

En esta fase del proyecto se detalla la última configuración arquitectónica del sistema que da soporte a la aplicación. En este apartado se pretende dar una visión global de la descomposición en componentes del sistema, así como del despliegue de todos ellos, haciendo uso de los estilos arquitectónicos definidos en el Capítulo Capítulo II, Estado del Arte.

En el desarrollo del sistema se ha aplicado el patrón MVC descrito en el apartado *2.3.3.1 Estilos o patrones arquitectónicos.* De modo que se puede descomponer la aplicación en función de los componentes desarrollados, los cuales se han identificado en base a la función que proporcionan: vista, modelo y controlador. Asimismo se ha aplicado el estilo arquitectónico de capas, de modo que la distribución de los componentes se ha organizado en niveles bien definidos. Para la capa de la vista se ha utilizado la tecnología JSP (Java Server Pages) que ofrece una manera sencilla y clara de generar páginas Web, mientras que para la capa de infraestructura se ha empleado Jena y JPA (Java Persistance API).

Entre la vista y el modelo se sitúa la capa de control. Es precisamente aquí donde se integra el framework utilizado: Spring, descrito en el apartado *3.3.3 Spring*.

Para la capa de persistencia y almacenamiento en base de datos se ha utilizado la tecnología PostgreSQL la cual se ajusta al volumen de datos y tráfico que se prevé soporte esta aplicación. En la *Figura IV-6. Arquitectura del sistema,* se muestra el particionado en capas de la aplicación y las tecnologías empleadas en cada una de ellas.

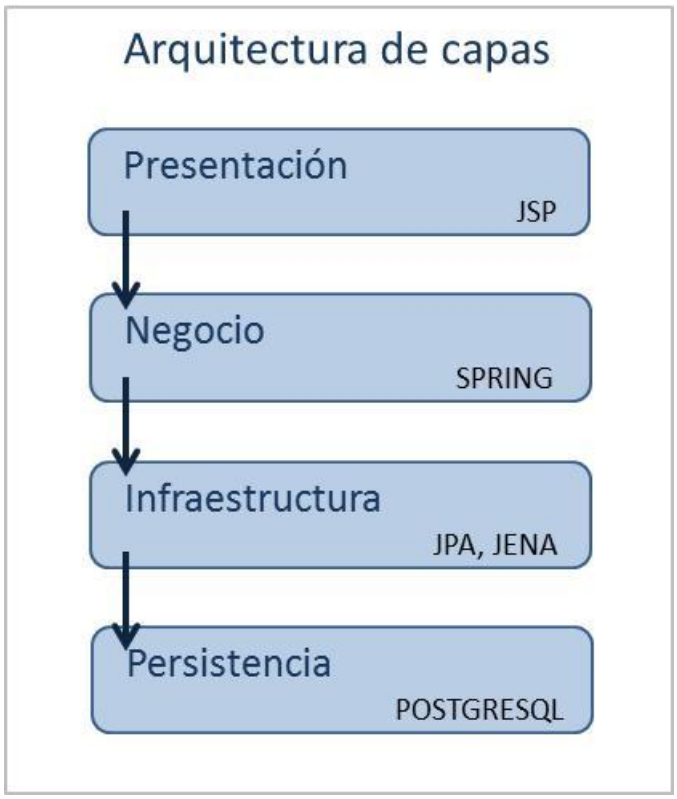

Figura IV-6. Arquitectura del sistema

4.3.1 Infraestructura Hardware

Debido a que se trata de una aplicación Web accesible a través de Internet, la infraestructura hardware responde a la clásica arquitectura de Cliente-Servidor.

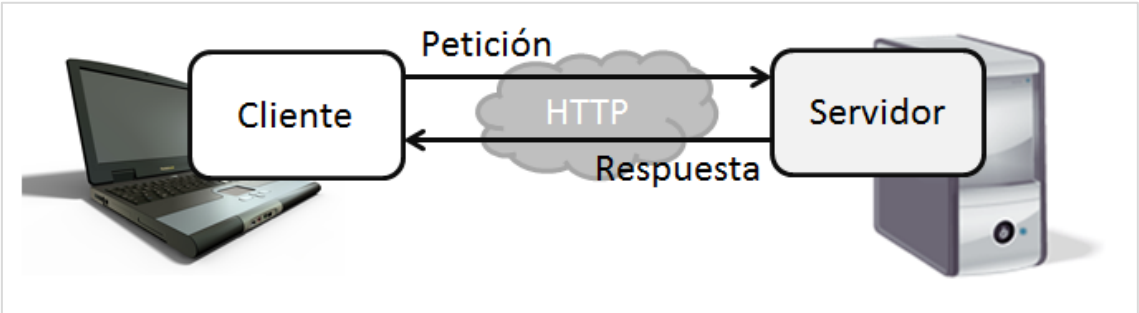

Figura IV-7. Arquitectura Cliente-Servidor

En esta arquitectura, el cliente, a través de un navegador Web estándar realiza peticiones mediante el protocolo HTTP al servidor Web instalado en el servidor. Éste, a su vez, procesa la petición y ejecuta los procesos necesarios para devolver al

cliente una respuesta mediante el mismo protocolo y visible desde el navegador del cliente.

4.3.2 Infraestructura Software

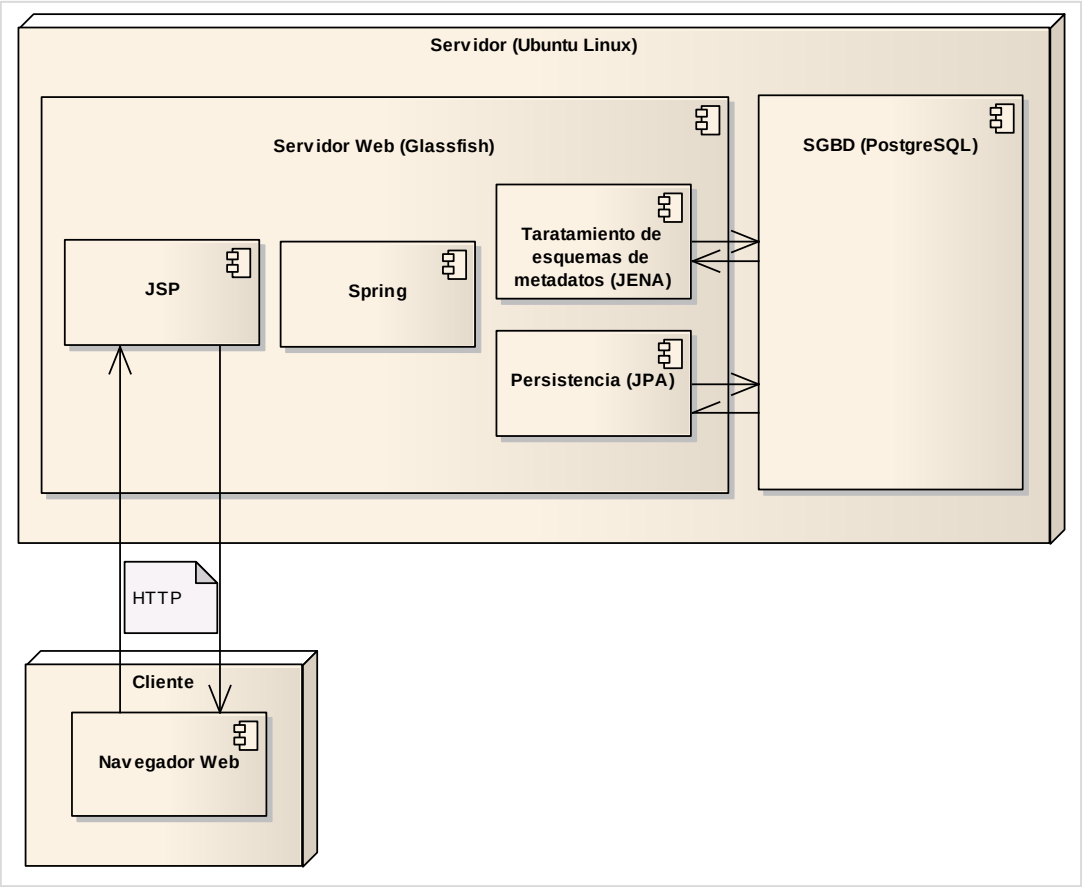

Figura IV-8. Diagrama de Componentes

En la *Figura IV-8. Diagrama de Componentes* se pueden apreciar todas las decisiones sobre qué software utilizar tomadas en el *Capítulo III Herramientas para la elaboración del proyecto.* A continuación se muestra un resumen de dichas elecciones:

- Se decidió la instalación, en el PC a modo de servidor, del sistema operativo Ubuntu/Linux en su versión estable debido a la fiabilidad, robustez y facilidad de gestión y administración una vez puesto en marcha el servicio, amén de otras muchas características nombradas con anterioridad.

- Sobre este sistema operativo se instaló Glassfish v2, como servidor Web y contenedor de los componentes ejecutables vía Web (Servlets), así como el gestor de Bases de Datos PostgreSQL.
- Para la gestión de la persistencia de los esquemas y ontologías se ha empleado el framework de Jena, con el que se independiza la gestión de los repositorios semánticos y su persistencia del resto de la aplicación. Su elección se ha basado principalmente en la compatibilidad con las últimas versiones de Java y en su facilidad de uso.
- Se ha utilizado JPA para la persistencia de la información asociada a los esquemas originales.
- JSP se ha utilizado como tecnología para la creación de las páginas webs integradas en la aplicación, y que conforman la parte visible de ésta. JSP permite la creación de páginas web dinámicas y cuenta con la ventaja de que la mayoría de los frameworks existentes permiten la integración con esta tecnología.
- Para la inyección de dependencias en la capa de negocio se eligió Spring debido a que ofrece los mismos servicios que si se utilizase EJB pero simplificando el modelo de programación. Además cuanta con una librería de etiquetas propias para las páginas web JSP que facilitan la validación y entrada de datos, así como su vinculación para la visualización de los mismos.
- Para la implementación de la aplicación se utilizará el lenguaje JAVA, debido a la facilidad de desarrollo de aplicaciones con este lenguaje, la orientación a objetos y la portabilidad de las aplicaciones hechas en este lenguaje.
- Como entorno integrado de desarrollo se ha seleccionado Netbeans 6.9.1 ya que funciona en las principales arquitecturas y sistemas operativos (Windows, Linux, Solaris, MacOS), es sencillo de instalar, y proporciona a los desarrolladores todas las herramientas necesarias para crear todo tipo de aplicaciones.

Teniendo en cuenta la elección de software descrita anteriormente cada componente queda distribuido en capas de la siguiente manera:

La aplicación queda desplegada en el servidor de aplicaciones Glasfish v2, relacionándose cada capa de la arquitectura según el patrón MVC implementado. Así las vistas de la capa de presentación se vinculan con la capa de negocio a través del Framework Spring, que controla las peticiones del usuario y está enlazada con todas las clases de acción. Esta capa, a su vez interactúa con la capa de infraestructura, donde se encuentran las clases que encapsulan la información que permite la gestión de los esquemas y ontologías a través del Framework de Jena. Desde estas clases, que conforman el modelo del dominio, se accede a la Base de Datos (PostgreSQL) para operaciones de inserción, borrado y consulta.

4.4 Diseño Detallado

Como se ha comentado anteriormente en el apartado *2.3.3.1 Estilos o patrones arquitectónicos*, en este proyecto se ha aplicado el patrón MVC (Modelo Vista Controlador) que engloba, por una parte, un modelo que representa toda la información que maneja la aplicación, por otra, las vistas o interfaz de usuario que muestran al usuario el contenido de dichas clases y finalmente, un controlador que maneja las peticiones enviadas desde las vistas e interactúa con el modelo según sea necesario. Dentro del patrón de capas descrito en la *Figura II-14 Estructura del patrón capas*, el MVC estaría situado dentro de las dos primeras capas, más concretamente las vistas corresponderían a la capa de presentación, y el modelo y el controlador se situarían en la capa de negocio. Es por ello que para describir la forma en la que se ha implementado este proyecto se seguirá el orden de dichas capas. De esta manera, en la capa de presentación se detallarán las principales vistas utilizadas; en la capa de negocio, se verán las principales clases que actúan como modelo de datos y las que lo hacen como controladores, así como la configuración del Framework de Spring; en la capa de infraestructura se describirá la forma en la que se usan JPA y Jena; y finalmente, en la capa de persistencia se detalla la base de datos utilizada por la aplicación.

Sin embargo, para una mejor comprensión de este capítulo, se ha incluido en primer lugar el diagrama de clases de la aplicación, en el que se pueden observar todas las clases utilizadas en el proyecto y las relaciones entre ellas. De esta forma pueden ser identificadas fácilmente cuando se detallen en apartados subsiguientes.

4.4.1 Diagrama de clases

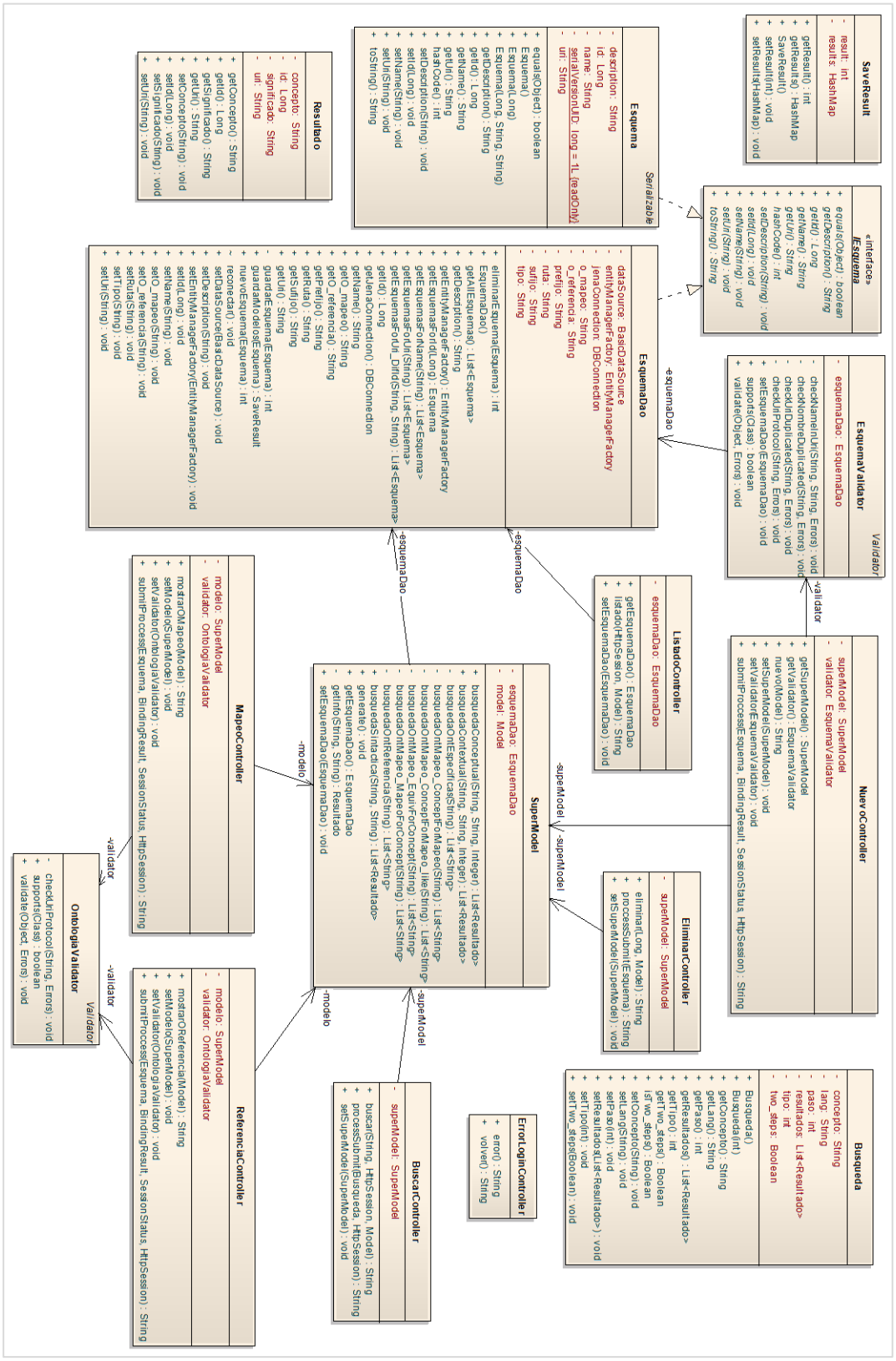

Figura IV-9. Diagrama de clases

Como se puede observar en el diagrama de clases de la figura anterior existen numerosas clases con distinta funcionalidad creadas durante el desarrollo de esta aplicación. A modo resumen, debido a que se verán con mayor profundidad más adelante, se muestra un listado de todas las clases creadas organizadas por paquetes:

- com.semse.search.model:
  - Búsqueda.java
  - Esquema.java
  - EsquemaDao.java
  - IEsquema.java
  - Resultado.java
  - SaveResult.java
  - SuperModel.java
- com.semse.search.model.validator:
  - EsquemaValidator.java
  - OntologiaValidator.java
- com.semse.search.controller:
  - BuscarController.java
  - EliminarController.java
  - ErrorLoginController.java
  - ListadoController.java
  - MapeoController.java
  - NuevoController.java
  - ReferenciaController.java

A continuación y tal como se adelantaba al comienzo de este capítulo, se detallará la funcionalidad de las principales clases utilizadas en esta aplicación ordenadas por el rol que asumen dentro de la jerarquía que establece el patrón de Capas.

4.4.2 Presentación

En este apartado se muestran los componentes de la vista principal más importantes generados como páginas JSP y que representan de las principales clases que actúan como modelos en la aplicación y que se detallan en el apartado siguiente *4.4.3.1 Modelos*.

Buscar.jsp

Figura IV-10. Vista buscar.jsp

Esta es la vista cuya función es la representación de la clase Busqueda.java utilizada para almacenar los parámetros de selección introducidos por el usuario y los resultados obtenidos en la búsqueda.

Logon.jsp

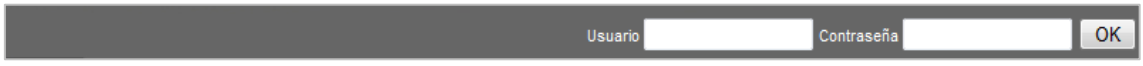

Figura IV-11. Vista Logon.jsp

Esta vista es la utilizada para que el usuario administrador pueda introducir sus credenciales necesarias para su autenticación.

Nuevo.jsp y Eliminar.jsp

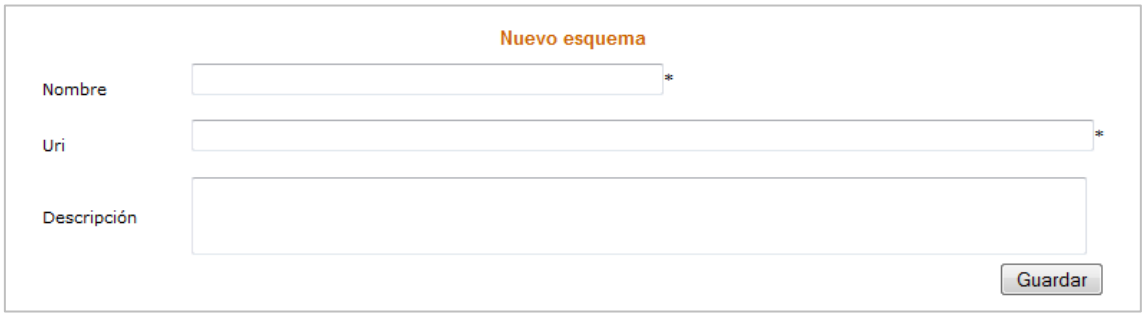

Figura IV-12. Vista nuevo.jsp

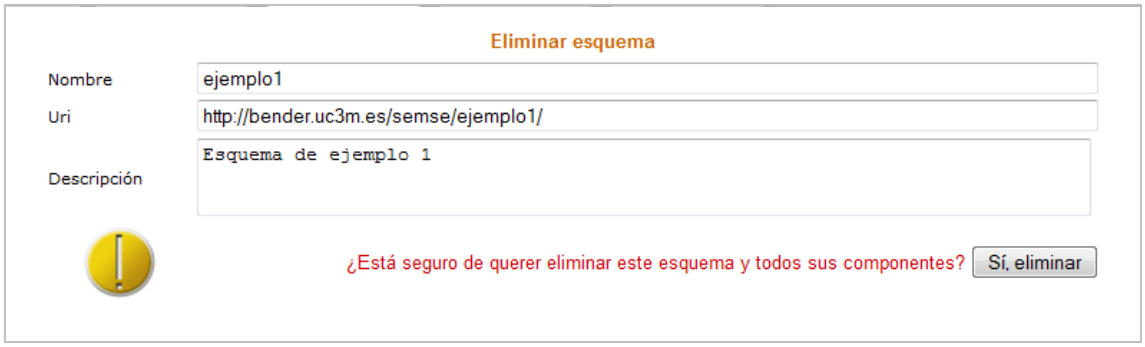

Figura IV-13. Vista eliminar.jsp

Ambas vistas representan la clase Esquema.java encargada de la información de los esquemas originales que se almacenan en la base de datos. Cada una representa una funcionalidad distinta, mientras que una se utiliza para que el usuario introduzca la información de un nuevo esquema, la otra pide confirmación al usuario para eliminar uno almacenado previamente.

4.4.3 Negocio

En este apartado se define en detalle las clases más importantes utilizadas como modelos y las que actúan como controladores. Finalmente se detalla la configuración completa que se ha definido para el framework de Spring.

4.4.3.1 Modelos

Busqueda.java

Esta clase es la encargada de manejar los parámetros de la búsqueda que ha seleccionado el usuario y que debe de ejecutar la aplicación, así como el listado de resultados devueltos por la ejecución de la búsqueda. Es por ello que tiene las siguientes propiedades:

- Concepto: El concepto que el usuario desea buscar.
- Lang: Lenguaje seleccionado por el usuario para visualizar las definiciones de los concepto que aparecen como resultado de la búsqueda.
- Tipo: Tipo de búsqueda seleccionada. Es un dominico que comprende los valores:
- 1: Búsqueda sintáctica
- 2: Búsqueda Conceptual
- 3: Búsqueda Contextual
- Two_steps: Se trata de un valor tipo True o False que indica si se ha marcado la opción de búsqueda en dos pasos.
- Paso: Indica el paso en el que se encuentra la búsqueda en caso de que el usuario utilice la búsqueda en dos pasos.

- Resultados: Es un listado con los resultados devueltos por la búsqueda. Cada elemento de este listado es una clase en si misma: Resultado.java descrita más adelante.

```
package com.semse.search.model;
import java.util.List;

public class Busqueda {
   private String concepto;
   private String lang;
   private int tipo;
   private Boolean two_steps;
   private int paso;
   private List<Resultado> resultados;

   public Busqueda(){}
   public Busqueda(int tipo){...}
   public String getConcepto(){...}
   public void setConcepto(String concepto) {...}
   public String getLang(){...}
   public void setLang(String lang) {...}
   public int getTipo(){...}
   public void setTipo(int tipo) {...}
   public void setTwo_steps(Boolean two_steps) {...}
   public Boolean isTwo_steps(){...}
   public Boolean getTwo_steps(){...}
   public int getPaso(){...}
   public void setPaso(int paso) {...}
   public List<Resultado> getResultados(){...}
   public void setResultados(List<Resultado> resultados) {...}
}
```

Implementación IV-1. Busqueda.java

Resultado.java

Esta clase representa un resultado obtenido en la ejecución de una búsqueda. Contiene la información necesaria que debe conocer el usuario y la propia de la aplicación para conocer de qué esquema se trata:

- ID: Identificación interna del esquema que contiene el concepto contenido en dicho resultado.
- Concepto: Elemento de un esquema ofrecido como resultado de la búsqueda.

- Significado: Definición del concepto obtenida de la ontología específica y en el idioma seleccionado por el usuario.
- URI: URI del esquema original.

```
package com.semse.search.model;
import java.util.ArrayList;
import java.util.List;

public class Resultado {
  private Long   id;
  private String concepto;
  private String significado;
  private String uri;

  public String getConcepto() {...}
  public void setConcepto(String concepto) {...}
  public String getSignificado() {...}
  public void setSignificado(String significado) {...}
  public String getUri() {...}
  public void setUri(String uri) {...}
  public Long getId() {...}
  public void setId(Long id) {...}
}
```

Implementación IV-2. Resultado.java

Esquema.java

Esta clase es la utilizada para representar la información almacenada para cada esquema original manejado por la aplicación. Sus propiedades son:

- ID: Identificador interno de un esquema.
- Name: Nombre que se le ha dado al esquema.
- URI: URI del esquema.
- Descripción: Descripción breve del contenido del esquema.

Dado que esta información es la única información que se guarda en la base de datos, debido a que no es necesario el almacenamiento de ninguna otra, se trata de la única clase identificada como una entidad de persistencia mediante el mapeo con JPA. Como se puede ver a continuación, este mapeo se realiza mediante la utilización de etiquetas propias de JPA que identifican, por cada elemento definido en la clase, la forma de mapeo respecto a la definición de la tabla en la base de datos. Adicionalmente JPA permite la definición de Named Queries (consultas nombradas) en la propia clase, por lo que simplemente con hacer referencia a su nombre se pueden ejecutar sin necesidad de construir una consulta específica para cada caso.

```
package com.semse.search.model;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table(name = "esquema")
@NamedQueries({
   @NamedQuery(name = "Esquema.findAll", query = "SELECT e FROM Esquema e"),
   @NamedQuery(name = "Esquema.findById", query = "SELECT e FROM Esquema e WHERE e.id = :id"),
   @NamedQuery(name = "Esquema.findByName", query =
           "SELECT e FROM Esquema e WHERE e.name = :name"),
   @NamedQuery(name = "Esquema.findByUri", query = "SELECT e FROM Esquema e WHERE e.uri = :uri"),
   @NamedQuery(name = "Esquema.findByDescription", query =
           "SELECT e FROM Esquema e WHERE e.description = :description"),
   @NamedQuery(name = "Esquema.findByUri_DifId", query =
            "SELECT e FROM Esquema e WHERE e.id <> :id AND e.uri = :uri")})

public class Esquema implements IEsquema, Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   @Basic(optional = false)
   @Column(name = "id")
   private Long id;
   @Basic(optional = false)
   @Column(name = "name")
   private String name;
   @Basic(optional = false)
   @Column(name = "uri")
   private String uri;
   @Column(name = "description")
   private String description
```

```
    public Esquema() {...}
    public Esquema(Long id) {...}
    public Esquema(Long id, String name, String uri) {...}
    public Long getId(){...}
    public void setId(Long id) {...}
    public String getName(){...}
    public void setName(String name) {...}
    public String getUri(){...}
    public void setUri(String uri) {...}
    public String getDescription(){...}
    public void setDescription(String desc) {...}
    @Override
    public int hashCode(){...}
    @Override
    public boolean equals(Object object) {...}
    @Override
}
```

Implementación IV-3. Esquema.java

EsquemaDao.java

Esta es una clase puramente de negocio sin ninguna vista asociada a ella, ya que es la clase que se utiliza como interconexión de la clase Esquema y la base de datos, ya sea mediante persistencia con JPA como por la forma de operar los esquemas mediante Jena. Es por ello que tiene asociadas todas las conexiones e información necesaria para la lectura o escritura de los datos de los esquemas, así como para la recuperación o almacenaje de los esquemas en sí mismos:

- O_referencia: URI de la ontología de referencia recuperada mediante un fichero de propiedades externo.
- O_mapeo: URI de la ontología de mapeo recuperada mediante un fichero de propiedades externo.
- Prefijo: Prefijo utilizado para la construcción de la URI de los esquemas semánticamente cualificados recuperado mediante un fichero de propiedades externo.
- Sufijo: Sufijo utilizado para la construcción de la URI de las ontologías específicas recuperado mediante un fichero de propiedades externo.
- Ruta: Ruta del fichero de propiedades utilizado para la recuperación de las propiedades anteriores.
- JenaConnection: Es una clase propia del Framework de Jena, DBConnection.java, y se trata de la conexión utilizada por dicho Framework

para la lectura, escritura y ejecución de consultas sobre los esquemas almacenadas en la base de datos.

- Tipo: Tipo de base de datos utilizada. Necesario para el establecimiento de la conexión de Jena.
- EntityManagerFactory: Es una clase propia de JPA denominada por el mismo nombre y de tipo Factory, utilizada para la construcción de objetos EntityManager, responsables de las operaciones sobre los esquemas que requieran de persistencia de datos. Es necesario asociarle una conexión a la base de datos e indicarle que clases son las que se deben mapear, en este caso sólo será necesario el mapeo de la clase Esquema.java, aunque esta configuración será descrita con más detalle posteriormente en el apartado *4.4.4.2 JPA*.
- Datasource: Se trata de una clase BasicDataSource del proveedor Apache, responsable de la gestión de conexiones a la base de datos. Se utiliza para proveer de conexiones a la conexión de Jena en caso de que ésta se cierre o se pierda.

```
package com.semse.search.model;

import com.hp.hpl.jena.db.DBConnection;
import com.hp.hpl.jena.db.RDFRDBException;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;
import com.hp.hpl.jena.shared.JenaException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.sql.SQLException;
import java.sql.Savepoint;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.List;
import java.util.Properties;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.EntityManagerFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
import org.springframework.core.io.support.PropertiesLoaderUtils;
import org.springframework.dao.DataAccessException;
import org.apache.commons.dbcp.BasicDataSource;
```

```
public class EsquemaDao implements IEsquema{

   private String o_referencia;
   private String o_mapeo;
   private String prefijo;
   private String sufijo;
   private DBConnection jenaConnection;
   private String ruta;
   private String tipo;
   private BasicDataSource dataSource;
   private EntityManagerFactory entityManagerFactory;

   public EsquemaDao(){...}
   public Esquema getEsquemasForId(Long id) {...}
   public List<Esquema> getEsquemasForUri(String uri) {...}
   public List<Esquema> getEsquemasForName(String name) {...}
   public List<Esquema> getEsquemasForUri_DifId(String uri, String id) {...}
   public List<Esquema> getAllEsquemas(){...}
   public int nuevoEsquema(Esquema esquema) {...}
   public SaveResult guardarModelos(Esquema esquema) {...}
   private int guardarEsquema(Esquema esquema) {...}
   public int eliminarEsquema(Esquema esquema) {...}
   public String getDescription(){...}
   public Long getId(){...}
   public String getName(){...}
   public String getUri(){...}
   public void setDescription(String desc) {...}
   public void setId(Long id) {...}
   public void setName(String name) {...}
   public void setUri(String uri) {...}
   public DBConnection getJenaConnection(){...}
   public String getO_referencia(){...}
   public String getO_mapeo(){...}
   public String getPrefijo(){...}
   public String getSufijo(){...}
   public void setO_referencia(String o_referencia) {...}
   public void setO_mapeo(String o_mapeo) {...}
   public String getRuta(){...}
   public void setRuta(String ruta){
   public EntityManagerFactory getEntityManagerFactory(){...}
   public void setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {...}
   void reconectar(){...}
   public void setDataSource(BasicDataSource dataSource) {...}
   public void setTipo(String tipo) {...}
}
```

Implementación IV-4. EsquemaDao.java

Como funcionalidad especial aportada por esta clase, destacan los métodos de guardarModelos y eliminarEsquema, responsables del guardado de los esquemas y de su eliminación de la base de datos.

Método guardarModelos

```
public SaveResult guardarModelos(Esquema esquema){
      SaveResult result = new SaveResult();
      String nombre_cualificado     = null;
      String nombre_ontologia       = null;
      String ruta_cualificado       = null;
      String ruta_ontologia         = null;
      if(result.getResult()!=-1){
      ModelMaker maker = ModelFactory.createModelRDBMaker(this.getJenaConnection());
      nombre_cualificado = this.getPrefijo() + esquema.getName();
      nombre_ontologia   = esquema.getName() + this.getSufijo();
      ruta_cualificado   = esquema.getUri().replaceFirst(esquema.getName(), nombre_cualificado);
      ruta_ontologia     = esquema.getUri().replaceFirst(esquema.getName(), nombre_ontologia);
      Model model_cualificado = ModelFactory.createDefaultModel();
      Model model_ontologia   = ModelFactory.createOntologyModel();
      if (!this.jenaConnection.containsModel(nombre_cualificado) ){
            try{
               model_cualificado = maker.createModel(nombre_cualificado);
            }
            catch(RDFRDBException rdfe){
               result.getResults().put(ruta_cualificado, 0);
            }
         }
      if (!this.jenaConnection.containsModel(nombre_ontologia) ){
            try{
               model_ontologia = maker.createModel(nombre_ontologia);
            }
            catch(RDFRDBException rdfe){
               result.getResults().put(ruta_ontologia, 0);
            }
         }
      if (!result.getResults().containsValue(0)){
      try{
         model_cualificado.read(ruta_cualificado);
         result.getResults().put(ruta_cualificado, 1);
      }catch(JenaException je){
         result.getResults().put(ruta_cualificado, 0);
      }
      try{
         model_ontologia.read(ruta_ontologia);
         result.getResults().put(ruta_ontologia, 1);
      }catch(JenaException je){
         result.getResults().put(ruta_ontologia, 0);
      }
       if (!result.getResults().containsValue(0)){
            result.setResult(this.guardarEsquema(esquema));
         } else{
            try{
            this.getJenaConnection().getConnection().rollback();
            }
            catch(java.sql.SQLException sqle){
            }
            result.setResult(0);
         }
      } else{
         result.setResult(0);
      }
      }
     return result;
   }
```

Implementación IV-5. Método guardarModelos de la clase EsquemaDao

Este método es el encargado de guardar los esquemas. Por un lado los datos del esquema original utilizando JPA y por otro, el esquema cualificado semánticamente y la ontología de referencia asociados a dicho esquema. Es por ello, que a partir del esquema que se recibe desde la vista, el método construye las URIs de los esquemas asociados y los busca. Si ambos esquemas están disponibles los recupera y graba en la base de datos con ayuda de las clases de Jena. Finalmente, si todo el proceso ha concluido correctamente, se guardarán los datos del esquema original utilizando JPA.

Método eliminarEsquema

```
public int eliminarEsquema(Esquema esquema) {
    EntityManager em = entityManagerFactory.createEntityManager();
    String nombre_cualificado     = null;
    String nombre_ontologia       = null;
    nombre_cualificado = this.getPrefijo() + esquema.getName();
    nombre_ontologia   = esquema.getName() + this.getSufijo();

    EntityTransaction et = em.getTransaction();
    et.begin();
    try{
       Esquema mergedEsquema = em.merge(esquema);
       em.remove(mergedEsquema);
       em.flush();
       et.commit();
       em.close();
       ModelMaker maker = ModelFactory.createModelRDBMaker(this.getJenaConnection());
       if (this.getJenaConnection().containsModel(nombre_cualificado)){
          maker.removeModel(nombre_cualificado);
       }
       if (this.getJenaConnection().containsModel(nombre_ontologia)){
          maker.removeModel(nombre_ontologia);
       }
       return 0;
      }
    catch(DataAccessException e){
       et.rollback();
       em.close();
       return 4;}
  }
```

Implementación IV-6. Método eliminarEsquema de la clase EsquemaDao

El método eliminar esquema, como su propio nombre indica es el responsable de la eliminación de los esquemas en la base de datos. A partir del nombre del esquema original se obtienen los nombres de su esquema cualificado semánticamente y su ontología de referencia. Con ayuda de las clases de Jena se realiza la petición de

borrado utilizando los nombres construidos y finalmente, si la operación ha concluido correctamente, se eliminan los datos del esquema original almacenados en la base de datos utilizando JPA.

SuperModel.java

Esta clase es la utilizada para combinar en un solo esquema todos el conocimiento contenido en todos los esquemas que utiliza la aplicación. De esta forma se consigue un único esquema sobre el que ejecutar las consultas y en el que se encuentra toda la información necesaria:

- Modelo: El esquema formado por todos los esquemas.
- EsquemaDao: Clase EsquemaDao.java para dar soporte a las conexiones que necesita la clase SuperModelo para generar dicho modelo.

```
package com.semse.search.model;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;
import com.hp.hpl.jena.shared.JenaException;
import java.sql.SQLException;
import java.util.Iterator;
import java.util.List;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import java.util.ArrayList;
import java.util.Collections;
import java.util.logging.Level;
import java.util.logging.Logger;

public class SuperModel{

   private Model model;
   private EsquemaDao esquemaDao;

   public EsquemaDao getEsquemaDao() {...}
   public void setEsquemaDao(EsquemaDao esquemaDao) {...}
   public void generate(){...}
   private List<String> busquedaOntEspecificas(String concept) {...}
   private List<String> busquedaOntMapeo_EquivForConcept(String concept) {...}
   private List<String> busquedaOntMapeo_MapeoForConcept(String concept) {...}
   private List<String> busquedaOntMapeo_ConceptForMapeo(String mapeo) {...}
   private List<String> busquedaOntMapeo_ConceptForMapeo_like(String mapeo) {...}
   private List<String> busquedaOntReferencia(String concept) {...}
   public List<Resultado> busquedaSintactica(String concept, String lang) {...}
   public List<Resultado> busquedaConceptual(String concept, String lang, Integer paso) {...}
   public List<Resultado> busquedaContextual(String concept, String lang, Integer paso) {...}
   private Resultado getInfo(String concept, String lang) {...}
}
```

Implementación IV-7. Clase SuperModel.java

Esta es la clase que implementa los métodos de búsqueda que utiliza la aplicación para realizar las búsquedas de conceptos sobre los esquemas de metadatos. Como se pude observar en la *Implementación IV-7. Clase SuperModel.java* esta clase contiene los métodos busquedaSintactica, busquedaConceptual y busquedaContextual encargados de ejecutar cada uno de los tipos de búsquedas ofrecidos por la aplicación. Sin embargo, estos métodos se basan en otros más concretos encargados de ejecutar las consultas en SPARQL que se realizan sobre el supermodelo y que se detallan a continuación.

Método busquedaOntEspecificas

```
private List<String> busquedaOntEspecificas(String concept){
List<String> resultados = new ArrayList();

String s_query =
"PREFIX semse: <http://purl.org/semse/> " +
"PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
"PREFIX skos:  <http://www.w3.org/2004/02/skos/core#>" +
"SELECT DISTINCT ?concept WHERE { " +
"{?concept rdf:type semse:SubjectConcept . FILTER regex(str(?concept), \"" + concept + "\", \"i\")} " +
"UNION " +
"{?concept rdf:type semse:SubjectProperty . FILTER regex(str(?concept), \"" + concept + "\", \"i\")} " +
"UNION " +
"{?concept semse:hasSemset ?semset . FILTER regex(str(?semset), \"" + concept + "\", \"i\")} " +
"UNION " +
"{?concept semse:hasPropertySemset ?property . FILTER regex(str(?property), \"" + concept + "\", \"i\")} " +
"UNION " +
"{?concept skos:definition ?definition . FILTER regex(str(?definition), \"" + concept + "\", \"i\")} " +
"UNION " +
"{?concept semse:hasSemset ?semset . ?semset skos:altLabel ?altlabel . FILTER regex(str(?altlabel), \"" + concept +
"\", \"i\")} " +
"UNION " +
"{?concept semse:hasSemset ?semset . ?semset skos:prefLabel ?preflabel . FILTER regex(str(?preflabel), \"" +
concept + "\", \"i\")} " +
"UNION " +
"{?concept semse:hasPropertySemset ?property . ?property skos:altLabel ?altlabel . FILTER regex(str(?altlabel), \"" +
concept + "\", \"i\")} " +
"UNION " +
"{?concept semse:hasPropertySemset ?property . ?property skos:prefLabel ?preflabel . FILTER regex(str(?preflabel),
\"" + concept + "\", \"i\")} " +
"}";

Query query = QueryFactory.create(s_query);
QueryExecution qe = QueryExecutionFactory.create(query, this.model);
ResultSet rs = qe.execSelect();

while (rs.hasNext()){
        QuerySolution soln = rs.nextSolution();
        resultados.add(soln.getResource("?concept").toString());
}
return resultados;
}
```

Implementación IV-8. Método busquedaOntEspecificas de la clase SuperModel

Este método consiste en buscar sobre las ontologías específicas y de forma sintáctica el concepto introducido por el usuario en todos los atributos que contienen los conceptos. Todos los conceptos encontrados serán recogidos en un único listado que será devuelto por el método.

Método busquedaOntMapeo_conceptForMapeo_like

```
private List<String> busquedaOntMapeo_ConceptForMapeo(String mapeo){
List<String> resultados = new ArrayList();
   String s_query =
   "PREFIX owl:   <http://www.w3.org/2002/07/owl#>" +
   "SELECT DISTINCT ?concept WHERE { " +
   "{?concept owl:equivalentClass ?mapeo . FILTER regex(str(?mapeo), \"" + mapeo + "\", \"i\")} " +
   "UNION " +
   "{?concept owl:equivalentProperty ?mapeo . FILTER regex(str(?mapeo), \"" + mapeo + "\", \"i\")} " +
   "UNION " +
   "{?concept owl:equivalentConcept ?mapeo . FILTER regex(str(?mapeo), \"" + mapeo + "\", \"i\")} " +
   "}";

   Query query = QueryFactory.create(s_query);

   QueryExecution qe = QueryExecutionFactory.create(query, this.model);
   ResultSet rs = qe.execSelect();
   while (rs.hasNext()){
            QuerySolution soln = rs.nextSolution();
            resultados.add(soln.getResource("?concept").toString());
   }
   return resultados;
   }
```

Implementación IV-9. Método busquedaOntMapeo_conceptForMapeo_like de la clase SuperModel

Este método realiza una función parecida a la del anterior, con la excepción de que en vez de realizar una búsqueda sintáctica en las ontologías específicas se realiza únicamente sobre los conceptos de la ontología de referencia que aparecen en la ontología de mapeo, de forma que se devolverán los conceptos de las ontologías específicas que son equivalentes a ellos. Este método se utiliza junto a la búsqueda sintáctica para ampliar los resultados del primer paso tanto de la búsqueda conceptual como de la contextual.

Método busquedaOntMapeo_EquivForConcept

```
private List<String> busquedaOntMapeo_EquivForConcept(String concept){
   List<String> resultados = new ArrayList();
   String s_query =
   "PREFIX owl:   <http://www.w3.org/2002/07/owl#>" +
   "SELECT DISTINCT ?equiv WHERE { " +
   "{<" + concept + "> owl:equivalentClass ?mapeo . ?equiv owl:equivalentClass ?mapeo } " +
   "UNION " +
   "{<" + concept + "> owl:equivalentProperty ?mapeo . ?equiv owl:equivalentProperty ?mapeo }" +
   "UNION " +
   "{<" + concept + "> owl:equivalentConcept ?mapeo . ?equiv owl:equivalentConcept ?mapeo }" +
   "}";

   Query query = QueryFactory.create(s_query);

   QueryExecution qe = QueryExecutionFactory.create(query, this.model);
   ResultSet rs = qe.execSelect();
   while (rs.hasNext()){
            QuerySolution soln = rs.nextSolution();
            resultados.add(soln.getResource("?equiv").toString());
   }
   return resultados;
   }
```

Implementación IV-10. Método busquedaOntMapeo_EquivForConcept de la clase SuperModel

A partir de un concepto de una ontología específica ya identificado, se localizan todos los conceptos de las ontologías específicas que comparten o son equivalentes a un mismo concepto de la ontología de referencia a partir de la definición existente en la ontología de mapeo. Este método se utiliza para obtener el segundo paso de la búsqueda conceptual y por lo tanto, es uno de los utilizados también en el segundo paso de la búsqueda contextual.

Método busquedaOntMapeo_MapeoForConcept

```
private List<String> busquedaOntMapeo_MapeoForConcept(String concept){
   List<String> resultados = new ArrayList();
   String s_query =
   "PREFIX owl:   <http://www.w3.org/2002/07/owl#>" +
   "SELECT DISTINCT ?mapeo WHERE { " +
   "{<" + concept + "> owl:equivalentClass ?mapeo } " +
   "UNION " +
   "{<" + concept + "> owl:equivalentProperty ?mapeo } " +
   "UNION " +
   "{<" + concept + "> owl:equivalentConcept ?mapeo } " +
   "}";

   Query query = QueryFactory.create(s_query);

   QueryExecution qe = QueryExecutionFactory.create(query, this.model);
   ResultSet rs = qe.execSelect();
   while (rs.hasNext()){
            QuerySolution soln = rs.nextSolution();
            resultados.add(soln.getResource("?mapeo").toString());
   }
   return resultados;
   }
```

Implementación IV-11. Método busquedaOntMapeo_MapeoForConcept de la clase SuperModel

Esta búsqueda devuelve el concepto de la ontología de referencia equivalente a un concepto de las ontologías específicas dado. Esto es útil en la búsqueda contextual para obtener un listado de los conceptos de la ontología de referencia que utilizaremos en el siguiente método.

Método busquedaOntReferencia

```
private List<String> busquedaOntReferencia(String concept){
   List<String> resultados = new ArrayList();
   String s_query =
   "PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>" +
   "SELECT DISTINCT ?sub WHERE { " +
   "{?sub rdfs:subClassOf <" + concept + "> } " +
   "UNION " +
   "{?sub rdfs:subPropertyOf <" + concept + "> } " +
   "}";

   Query query = QueryFactory.create(s_query);

   QueryExecution qe = QueryExecutionFactory.create(query, this.model);
   ResultSet rs = qe.execSelect();
   while (rs.hasNext()){
            QuerySolution soln = rs.nextSolution();
            resultados.add(soln.getResource("?sub").toString());
   }
   return resultados;
   }
```

Implementación IV-12. Método busquedaOntRefencia

Búsqueda sobre la ontología de referencia, en la que, a partir de un concepto se obtiene la clase superior de la que hereda dicho concepto. Este método es el punto clave en la búsqueda contextual y de su diferenciación respecto a la búsqueda conceptual, ya que a partir de los conceptos obtenido en esta búsqueda de realiza el camino inverso hasta buscar los conceptos de las ontologías específicas equivalentes a los obtenidos aquí.

Método busquedaOntMapeo_ConceptForMapeo

```
private List<String> busquedaOntMapeo_ConceptForMapeo(String mapeo){
   List<String> resultados = new ArrayList();
   String s_query =
   "PREFIX owl:   <http://www.w3.org/2002/07/owl#>" +
   "SELECT DISTINCT ?concept WHERE { " +
   "{ ?concept owl:equivalentClass <" + mapeo + "> } " +
   "UNION " +
   "{?concept owl:equivalentProperty <" + mapeo + "> } " +
   "UNION " +
   "{?concept owl:equivalentConcept <" + mapeo + "> } " +
   "}";

   Query query = QueryFactory.create(s_query);

   QueryExecution qe = QueryExecutionFactory.create(query, this.model);
   ResultSet rs = qe.execSelect();
   while (rs.hasNext()){
            QuerySolution soln = rs.nextSolution();
            resultados.add(soln.getResource("?concept").toString());
   }
   return resultados;
   }
```

Implementación IV-13. Método busquedaOntMapeo_ConceptForMapeo de la clase SuperModel

A partir de un concepto obtenido en la búsqueda sobre la ontología de referencia se localizan todos los conceptos de las ontologías específicas equivalentes según la ontología de mapeo. Parte final de la búsqueda contextual, en la que tras obtener un listado de conceptos de la ontología de referencia se necesita obtener un listado de los conceptos de las ontologías específicas equivalentes.

4.4.3.2 Controladores

Se les llama controladores a aquellas clases que actúan como manejadores de eventos lanzados desde las vistas y de las peticiones que se realizan sobre las clases del modelo.

Una de las grandes ventajas que aporta Spring desde su versión 2.5 y que es prácticamente de obligado cumplimiento a partir de la 3.0, es el uso de controladores etiquetados. Esto significa que, mediante una librería de etiquetas propia de Spring, cualquier clase en java con las etiquetas correspondientes puede actuar como clase controladora de una vista.

Existen dos etiquetas principales con las que transformar la clase en un controlador:

- @Controller: Es la etiqueta que debe de estar presente antes de la definición de la clase para que Spring la identifique como un controlador.
- @RequestMapping: Indica ante qué llamada reaccionará la clase, es decir, si utilizamos, por ejemplo, esta etiqueta de este modo: @RequestMapping("/buscar.htm"), Spring entenderá que cuando se llame a la página buscar.htm será esta clase la que deberá de utilizar como controlador.

  Esta etiqueta también es aplicable a métodos, de hecho, debe estar aplicada sobre dos métodos, uno para que reaccione ante el evento GET de la página mediante (method=RequestMethod.GET) y otro para que reaccione ante el POST mediante (method=RequestMethod.POST).

A continuación veremos las clases más importantes, que actúan como controlador, implementadas en este proyecto.

BuscarController.java

```
package com.semse.search.controller;
import com.semse.search.model.Busqueda;
import com.semse.search.model.Resultado;
import com.semse.search.model.SuperModel;
import java.util.List;
import javax.servlet.http.HttpSession;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
@RequestMapping("/buscar.htm")
public class BuscarController {
   private SuperModel superModel;

   public void setSuperModel(SuperModel superModel) {…}

@RequestMapping(method=RequestMethod.GET)
   public String buscar(@RequestParam(value="concept", required=false) String concept, HttpSession session,
Model model){…}
@RequestMapping(method=RequestMethod.POST)
   public String processSubmit(@ModelAttribute Busqueda busqueda, HttpSession session){…}
}
```

Implementación IV-14. Clase BuscarController.java

Esta es la clase que, como su propio nombre indica, actúa como controlador de la vista *buscar.htm*. Como se pude observar utiliza la clase supermodelo con el objetivo de poder utilizarla para ejecutar todas las búsquedas que son solicitadas desde la página. Como características especiales destaca que el método GET recoge un concepto como parámetro opcional, añadido en la URL, para ejecutar la búsqueda directamente sin necesidad de pedir el concepto por pantalla. Esto es de especial utilidad en el caso de las búsquedas en dos pasos, ya que el segundo paso se ejecuta de esta manera.

NuevoController.java

```
package com.semse.search.controller;
import com.semse.search.model.SaveResult;
import com.semse.search.model.validator.EsquemaValidator;
import com.semse.search.model.Esquema;
import com.semse.search.model.SuperModel;
import javax.servlet.http.HttpSession;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.support.SessionStatus;

@Controller
@RequestMapping("/nuevo.htm")
public class NuevoController{
   private SuperModel superModel;
   private EsquemaValidator validator;
   public SuperModel getSuperModel() {…}
   public void setSuperModel(SuperModel superModel) {… }
   public EsquemaValidator getValidator() {…}
   public void setValidator(EsquemaValidator validator) {…}

 @RequestMapping(method=RequestMethod.GET)
   public String nuevo(Model model){…}
 @RequestMapping(method=RequestMethod.POST)
   public String submitProccess(@ModelAttribute Esquema esquema, BindingResult resultado, SessionStatus
estado, HttpSession session){…}
}
```

Implementación IV-15. Clase NuevoController.java

Esta clase es la encargada de actuar como controlador de la vista nuevo.jsp y al igual que el controlador de la vista buscar.htm también utiliza la clase SuperModel. Sin embargo, la utilidad por la que se usa esta clase en este controlador no es ejecutar ninguna búsqueda, sino que, por un lado se necesita su esquemaDao para añadir el nuevo esquema a la base de datos, y por otro, se necesita llamar a su método generate para que el supermodelo se genere de nuevo teniendo en cuenta el nuevo esquema. Además, tal y como se puede apreciar, se utiliza además la clase EsquemaValidator, que es la encargada de validar los datos introducidos del nuevo esquema, y cuyos resultados son enviados a la vista mediante su vinculación en el método POST.

EliminarController.java

```
package com.semse.search.controller;
import com.semse.search.model.Esquema;
import com.semse.search.model.SuperModel;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
@RequestMapping(value="/eliminar.htm")
public class EliminarController {
   private SuperModel superModel;

   public void setSuperModel(SuperModel superModel) {...}
@RequestMapping(method=RequestMethod.GET)
   public String eliminar(@RequestParam Long id, Model model){...}
@RequestMapping(method=RequestMethod.POST)
   public String proccessSubmit(@ModelAttribute Esquema esquema) throws Exception{...}
}
```

Implementación IV-16. Clase ElminarController.java

Como controlador de la vista eliminar.jsp esta clase es la encargada de eliminar el esquema que el usuario ha seleccionado y después confirmado en dicha vista. Al igual que en el caso del controlador de la vista nuevo.jsp, el uso de la clase SuperModel es por la utilización de su EsquemaDao y de la llamada a su método generate para que el supermodelo deje de tener en cuenta el esquema eliminado.

4.4.3.3 Configuración de Spring

La configuración de Spring se realiza mediante ficheros XML que se cargan en el momento en el que el servidor Web genera el Servlet[31] que conforma la aplicación.

En este proyecto se han utilizado tres ficheros, de forma que cada uno de ellos contiene una parte, aunque dependientes entre ellas, de la configuración general de Spring mediante la definición de Beans[32].

A continuación se define la configuración contenida en cada uno de los tres ficheros.

Semsesearch-data.xml

Este fichero contiene la configuración de los Beans encargados del acceso y del manejo de la información de la base de datos, ya sea mediante la configuración de Jena o mediante JPA.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
    "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd"
    xmlns:aop="http://www.springframework.org/schema/aop">

<bean id="propertiesConfigurer"
 class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
   <property name="location" value="classpath:connection.properties" />
</bean>
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
   <property name="url" value="${jdbc.url}"/>
   <property name="username" value="${jdbc.user}"/>
   <property name="password" value="${jdbc.password}"/>
   <property name="driverClassName" value="${jdbc.driver}"/>
</bean>
```

[31] Objetos que se ejecutan en un contenedor de servlets y extienden su funcionalidad. Se puede entender como un programa que se ejecuta en un servidor, generalmente utilizado para la generación de páginas web dinámicas.

[32] Son componentes software definidos para ejecutar tareas concretas y con la particularidad de ser reutilizables.

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
   <property name="persistenceUnitName" value="UC3M" />
</bean>

<bean id="esquemaDao"
class="com.semse.search.model.EsquemaDao">
   <property name="ruta" value="rutas.properties"/>
   <property name="tipo" value="${jdbc.type}"/>
   <property name="dataSource" ref="dataSource"/>
   <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<bean id="superModel"
class="com.semse.search.model.SuperModel">
   <property name="esquemaDao" ref="esquemaDao"/>
</bean>

<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory" ref="entityManagerFactory"/>
   <property name="jpaDialect">
      <bean class="org.springframework.orm.jpa.vendor.TopLinkJpaDialect"/>
   </property>
</bean>

</beans>
```

Implementación IV-17. Fichero de configuración semsesearch-data.xml

Los Beans declarados en este fichero son:

- PropertiesConfigurer: Bean que recupera de un fichero de propiedades sus valores para almacenarlos en variables que pueden ser utilizadas en los ficheros de configuración de Spring.
- DataSource: Conexión a la base de datos mediante la URL del servidor que la alberga, el usuario, la contraseña y el driver a utilizar. Los valores son obtenidos a partir de un fichero de propiedades leído por el Bean anterior.
- EntityManagerFactory: Bean destinado a la generación de objetos Entity Manager, propios de JPA, encargados de mantener la persistencia de las clases que se indican en una unidad de persistencia declarada en un fichero XML aparte. Esto se verá con más detalle en el apartado *4.4.4.2 JPA.*
- EsquemaDao y SuperModel: Configuración del esquemaDao y el supermodelo detallados en el apartado *4.4.3.1 Modelos*.

- TransactionManager: Bean encargado de la gestión de las transacciones que realizan los Entity Manager generados mediante la Entity Manger Factory respecto a la base de datos.

Semsesearch-security.xml

En este fichero se define la configuración de Spring Security, en la que se declaran los Beans que actúan de filtro de seguridad en la aplicación y que manejan la autenticación del usuario administrador.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:security="http://www.springframework.org/schema/security"
    xsi:schemaLocation=
    "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/beans/spring-security-3.1.xsd">

<bean id="authenticationManager"
class="org.springframework.security.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref bean="daoAuthenticationProvider"/>
    </list>
  </property>
</bean>

<bean id="daoAuthenticationProvider"
class="org.springframework.security.providers.dao.DaoAuthenticationProvider">
  <property name="userDetailsService" ref="authenticationDao"/>
  <property name="passwordEncoder">
    <bean class="org.springframework.security.providers.encoding.Md5PasswordEncoder"/>
  </property>
</bean>

<bean id="authenticationDao"
class="org.springframework.security.userdetails.jdbc.JdbcDaoImpl">
   <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="FilterChainProxy"
class="org.springframework.security.util.FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
  /**=httpSessionIntegrationFilter,authenticationProcessingFilter,logoutFilter
    </value>
  </property>
</bean>
```

```
<bean id="httpSessionIntegrationFilter"
class="org.springframework.security.context.HttpSessionContextIntegrationFilter"/>

<bean id="authenticationEntryPoint"
class="org.springframework.security.ui.webapp.AuthenticationProcessingFilterEntryPoint">
   <property name="loginFormUrl" value="/logon.htm"/>
</bean>

<bean id="authenticationProcessingFilter"
class="org.springframework.security.ui.webapp.AuthenticationProcessingFilter">
   <property name="filterProcessesUrl" value="/j_security_login"/>
   <property name="authenticationFailureUrl" value="/error_login.htm"/>
   <property name="defaultTargetUrl" value="/buscar.htm"/>
   <property name="authenticationManager" ref="authenticationManager"/>
</bean>

<bean id="logoutFilter"
class="org.springframework.security.ui.logout.LogoutFilter">
   <constructor-arg value="/buscar.htm"/>
   <constructor-arg>
            <list>
        <bean class="org.springframework.security.ui.logout.SecurityContextLogoutHandler"/>
            </list>
   </constructor-arg>
   <property name="filterProcessesUrl" value="/j_security_logout"/>
</bean>

</beans>
```

Implementación IV-18. Fichero de configuración semsesearch-security.xml

Los Beans declarados en este fichero son:

- AuthenticationManager: Gestor de autenticación que acepta múltiples proveedores de autenticación, de forma que pude buscar en todos ellos hasta que al menos uno consiga autenticar al usuario. En este caso sólo se dispone de uno, el DaoAuthenticationProvider.
- DaoAuthenticationProvider: Proveedor de autenticación que recupera el usuario y la contraseña de una base de datos. En este caso se configura con el AuthenticationDao que usa y la codificación que se aplica a las contraseñas (MD5[33]).
- AuthenticationDao: Bean de acceso a la base de datos para la identificación de usuarios. En el caso de no indicar nada más, Spring Security asume que la

[33] Algoritmo de reducción criptográfico de 128 bits

base de datos dispone de una tabla Users con el usuario, la contraseña y un valor booleano que indica si el usuario está activo o no; y una tabla Authorities en la que se guardan los permisos asociados a cada usuario.

- FilterChainProxy: Bean manejador de las peticiones delegadas desde el FilterToBeanProxy, configurado en el archivo web.xml en el que se define la configuración global de la aplicación, independientemente del uso de Spring. En este Bean se configuran los múltiples filtros servlet de los que Spring Security dispone y que son usados por la aplicación, cada uno responsable de una función en concreto.
- HttpSessionIntegrationFilter: Filtro que informa el contexto de seguridad utilizando información obtenida de HttpSession.
- AuthenticationProcessingFilter: Filtro encargado de capturar el proceso de autenticación de los usuarios. En él se configura la URL que debe capturar (coincide con la acción que debe ejecutar el formulario de validación usuarios, la vista *logon.jsp*), las vistas que se muestran tanto si la autenticación se produce como si no, y el AuthenticationManager que debe utilizar para dicha autenticación.
- LogoutFilter: Filtro que captura la solicitud de finalización de la sesión de un usuario. Se le indica la URL que debe capturar, y que coincide con la acción ejecutada por el formulario de validación de usuarios.
- AuthenticationEntryPoint: Filtro en el que se define el punto de entrada por el que el usuario debe autenticarse. En este caso, el formulario que contiene la vista *logon.jsp*.

Semsesearch-servlet.xml

Este fichero contiene la configuración general del Servlet y la definición de los controladores y manejadores de eventos de la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
      xsi:schemaLocation=
      "http://www.springframework.org/schema/mvc
      http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">


<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   <property name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
   <property name="prefix"><value>/WEB-INF/views/</value></property>
   <property name="suffix"><value>.jsp</value></property>
</bean>


<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
   <property name="basenames">
      <list>
         <value>errors</value>
      </list>
   </property>
</bean>

<bean id="buscarController"
class="com.semse.search.controller.BuscarController" >
   <property name="superModel" ref="superModel"/>
</bean>

<bean id="nuevoController"
class="com.semse.search.controller.NuevoController" >
   <property name="superModel" ref="superModel"/>
   <property name="validator">
      <bean class="com.semse.search.model.validator.EsquemaValidator">
         <property name="esquemaDao" ref="esquemaDao"/>
      </bean>
   </property>
</bean>

<bean id="eliminarController"
class="com.semse.search.controller.EliminarController" >
   <property name="superModel" ref="superModel"/>
</bean>

<bean id="listadoController"
class="com.semse.search.controller.ListadoController">
   <property name="esquemaDao" ref="esquemaDao"/>
</bean>

<bean id="referenciaController"
class="com.semse.search.controller.ReferenciaController">
   <property name="modelo" ref="superModel"/>
   <property name="validator">
      <bean class="com.semse.search.model.validator.OntologiaValidator"/>
   </property>
</bean>
```

```
<bean id="mapeoController"
class="com.semse.search.controller.MapeoController">
   <property name="modelo" ref="superModel"/>
   <property name="validator">
      <bean class="com.semse.search.model.validator.OntologiaValidator"/>
   </property>
</bean>

<bean id="errorLoginController"
class="com.semse.search.controller.ErrorLoginController"/>

<bean id="exceptionResolver"
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
   <property name="exceptionMappings">
      <props>
         <prop key="java.lang.Exception">/error_desconocido</prop>
      </props>
   </property>
</bean>

</beans>
```

Implementación IV-19. Fichero de configuración semsesearch-servlet.xml

En este fichero están configurados los siguientes Beans:

- ViewResolver: Resolutorio de vistas. Indica cómo se transforman las URLs externas en internas para la correcta localización de las vistas.
- MessageSource: Bean en el que se define el origen de los ficheros de propiedades que contienen los mensajes que utiliza la aplicación. En este caso sólo se usa un único fichero con los mensajes de error de validación de los formularios.
- BuscarController: Controlador de la vista *buscar.jsp.*
- NuevoController: Controlador de la vista *nuevo.jsp.*
- EliminarController: Controlador de la vista *eliminar.jsp.*
- ListadoController: Controlador de la vista *listado.jsp.*
- ReferenciaController: Controlador de la vista *referencia.jsp.*
- MapeoController: Controlador de la vista *mapeo.jsp.*
- ErrorLoginController: Controlador de la vista *error_login.jsp.*
- ExceptionResolver: Bean que actúa como manejador de errores. En este caso se establece que vista mostrar en caso de producirse un error no controlado en la aplicación.

4.4.4 Infraestructura

En este apartado se detalla cómo se han utilizado los frameworks de Jena para el tratamiento de los esquemas de metadatos, y de JPA para gestionar la persistencia de la información en la base de datos.

4.4.4.1 Jena

Jena provee de clases y métodos muy sencillos para el tratamiento de esquemas de metadatos o modelos. En este proyecto se ha utilizado Jena para la implementación de dos funcionalidades muy concretas: La gestión de los esquemas y su persistencia en la base de datos, y la ejecución de consultas.

Respecto a la gestión de esquemas, el framework de Jena se ocupa por si sólo de la estructura de la base de datos, tan sólo con configurar una conexión, será el propio framework quien se encargue de crear las tablas necesarias y de almacenar, leer o eliminar los modelos que se soliciten.

Para el tratamiento interno de los modelos en la aplicación, Jena dispone de la clase Model, que contendrá un modelo o esquema de metadatos, y de la clase ModelFactory, encargada de la creación de objetos de la clase ModelMaker capaces de recuperar modelos nombrados de la base de datos, crearlos o eliminarlos. Por lo tanto, mediante el uso de estas tres clases es posible leer un modelo a partir de una URI, almacenarlo en la base de datos con un nombre como identificador, recuperarlo y borrarlo si se quisiera, por lo que dichas clases son suficientes, como ya se ha visto en el apartado *4.4.3.1 Modelos*, para la implementación de la funcionalidad de la gestión de esquemas y la generación del supermodelo.

En cuanto a las búsquedas y la ejecución de consultas, Jena dispone de la clase QueryFactory que genera objetos de la clase Query a partir de una consulta en SPARQL construida sobre una cadena de caracteres. Por otro lado dispone de la clase QueryExecutionFactory que genera objetos de la clase QueryExecution a partir de un objeto Query y un modelo sobre el que ejecutar dicha consulta. Finalmente, mediante

el método *execSelect* de la clase QueryExecution se ejecuta la consulta, devolviendo los resultados en un objeto de la clase ResultSet.

Como ya hemos visto en el apartado *4.4.3.1Modelos*, esta es la forma en la que se ejecutan todas las consultas sobre los modelos en la aplicación, de forma que la funcionalidad de las búsquedas se implementa mediante la creación y ejecución de dichas consultas.

4.4.4.2 JPA

La configuración de JPA reside en un fichero XML que debe de estar localizado por defecto en la ruta raíz de la aplicación o *classpath.* En el caso de este proyecto se ha utilizado el siguiente fichero de configuración, en el que se han ocultado varios de los elementos por motivos de seguridad.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

 <persistence-unit name="UC3M" transaction-type="RESOURCE_LOCAL">
  <provider>oracle.toplink.essentials.PersistenceProvider</provider>
  <class>com.semse.search.model.Esquema</class>
  <exclude-unlisted-classes>true</exclude-unlisted-classes>
  <properties>
   <property name="toplink.jdbc.user" value=""/>
   <property name="toplink.jdbc.password" value=""/>
   <property name="toplink.jdbc.url" value="jdbc:postgresql://"/>
   <property name="toplink.jdbc.driver" value="org.postgresql.Driver"/>
  </properties>
 </persistence-unit>
</persistence>
```

Implementación IV-20. Fichero de configuración persistence.xml

Como se puede observar se configura una unidad de persistencia denominada "UC3M" mediante la URL del servidor, el usuario, la contraseña y el driver a utilizar. Esta unidad de persistencia es la que se le indica al Bean EntityManagerFactory tal y como se ha visto en el apartado *4.4.3.1 Modelos*, por lo que dicho Bean estaría listo para generar objetos EntityManager encargados, en este caso, de la recuperación y

guardado de los datos de los esquemas existentes en la base de datos. Un ejemplo de cómo se han utilizado estos objetos se puede observar en los métodos de guardar

```
private int guardarEsquema(Esquema esquema) {
      EntityManager em = entityManagerFactory.createEntityManager();
      EntityTransaction et = em.getTransaction();
      et.begin();
      try{
         em.persist(esquema);
         em.flush();
         et.commit();
         em.close();
         return 1;
        }
      catch(DataAccessException e){
         et.rollback();
         em.close();
         return -1;}
   }
```

Esquema y getAllEsquemas de la clase EsquemaDao.

Implementación IV-21. Método guardarEsquema de la clase EsquemaDao

En este método, mediante el paso de un esquema como parámetro, la aplicación genera un EntityManager y un EntityTransaction a partir del EntityManagerFactory para guardar dicho esquema. Es mediante la ejecución del método *persist* del EntityManager por el que JPA guardará todas las modificaciones realizadas sobre el esquema.

```
public List<Esquema> getAllEsquemas() {
      List<Esquema> resultados = null;
      EntityManager em = entityManagerFactory.createEntityManager();


      resultados = em.createNamedQuery("Esquema.findAll")
                     .getResultList();
      em.close();
      return resultados;
   }
```

Implementación IV-22. Método getAllEsquemas de la clase EsquemaDao

Este método es un sencillo ejemplo de cómo JPA recupera los objetos de la base de datos, ya que con tan sólo ejecutar una consulta nombrada de las que se han

declarado en la clase Esquema mediante anotaciones JPA, el framework se encargará de devolvernos un listado con todos los esquemas registrados en la base de datos.

4.4.5 Persistencia

Para la implementación de la capa de persistencia y de almacenamiento dela información requerida por la aplicación se ha utilizado una base de datos de PostgreSQL en la que se almacena por un lado, la información de los esquemas originales registrados en la aplicación, por otro los propios esquemas en sí gestionados por Jena y finalmente, la información relativa a la autenticación del usuario administrador.

Las tablas que se han generado en la base de datos, exceptuando las que el framework de Jena crea y gestiona por sí sólo, se muestran a continuación en el esquema relacional de la base de datos.

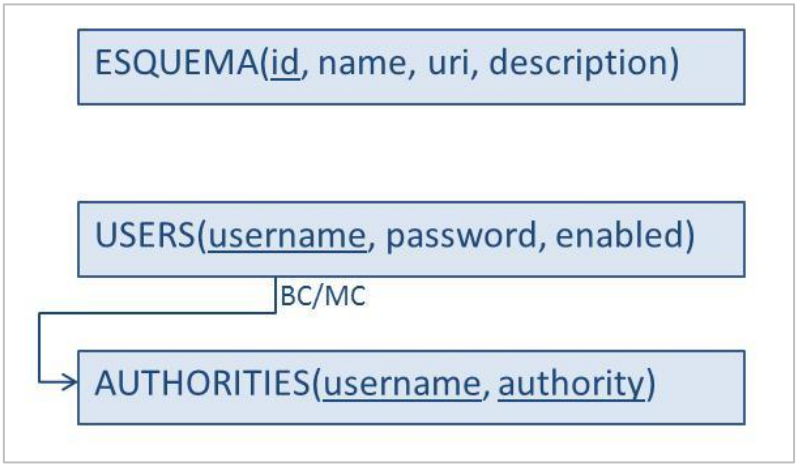

Figura IV-14. Esquema relacional de la base de datos

Tal y como se aprecia en la *Figura IV-14. Esquema relacional de la base de datos*, la base de datos contiene tres tablas:

- Esquema: Es la tabla destinada al almacenamiento de los datos de los esquemas originales. Es por ello que se guarda el nombre asociado por el usuario administrador, la URI y una descripción. Además, el sistema genera

una clave numérica de identificación por cada nueva inserción, utilizándola como clave primaria.

- Users: Tabla en la que se almacenan todos los usuarios dados de alta en la aplicación. Para cada uno de ellos se almacena su identificador unívoco utilizado como clave primaria, su contraseña y un indicador de su estado, si se encuentra activo o no. A pesar de que en el caso de esta aplicación sólo existe un único usuario, el hecho de haber utilizado esta tabla y la siguiente es no limitar el posible crecimiento de la aplicación y la inclusión en un futuro de más usuarios y más roles.
- Authorities: Por cada identificador de usuario se tienen relacionadas todas las autorizaciones con las que cuenta, esto es, todos los permisos otorgados a cada uno de los usuarios.

4.5 Manual de Usuario

En este apartado se describe un breve manual sobre el funcionamiento básico de la aplicación acompañado de capturas reales de la imagen final que tiene la aplicación en el momento de finalización de este proyecto.

4.5.1 Búsqueda

Será la funcionalidad a la que se acceda por defecto, siendo la primera página que se mostrará al usuario debido a que contiene la funcionalidad más importante de la aplicación. Desde aquí el usuario puede lanzar búsquedas de conceptos a los esquemas de metadatos almacenados en la base de datos, para ello debe primero rellenar la información necesaria en la entrada de búsqueda.

La entrada de búsqueda está compuesta de tres partes:

1. Búsqueda: Contiene el cuadro de texto en el que el usuario debe introducir el concepto que desee buscar y el botón de “Buscar” que ejecuta la búsqueda.

2. Tipo de búsqueda: En esta parte el usuario debe especificar qué tipo de búsqueda desea ejecutar: Sintáctica, Conceptual o Contextual. Por defecto aparecerá marcada la opción de búsqueda Conceptual por ser la búsqueda más completa y afinada de las tres.
3. Opciones adicionales: En esta parte el usuario puede seleccionar el idioma en el que desea ver la definición de los conceptos que aparecen en los resultados. Además, también aparece la opción de búsqueda en dos pasos. Por defecto, el lenguaje que aparece seleccionado es el inglés, por ser el lenguaje en el que están realizados prácticamente la totalidad de los esquemas de metadatos. En cuanto a la opción de la búsqueda en dos pasos, ésta aparece por defecto no seleccionada, dado que se trata de un funcionamiento opcional.

Figura IV-15. Captura de las partes de la entrada de una búsqueda

Para que el usuario pueda realizar una búsqueda sintáctica deberá de:

1. Introducir el concepto que desea buscar en el cuadro de texto indicado.
2. Seleccionar la opción de búsqueda sintáctica.
3. Elegir el idioma en el que desea ver la definición de los conceptos.
4. Seleccionar la opción de búsqueda en dos pasos si se desea.
5. Pulsar el botón “Buscar”.

A continuación se muestra una captura en la que se puede apreciar el listado de resultados que devuelve una búsqueda conceptual en un único paso para el concepto “Title”.

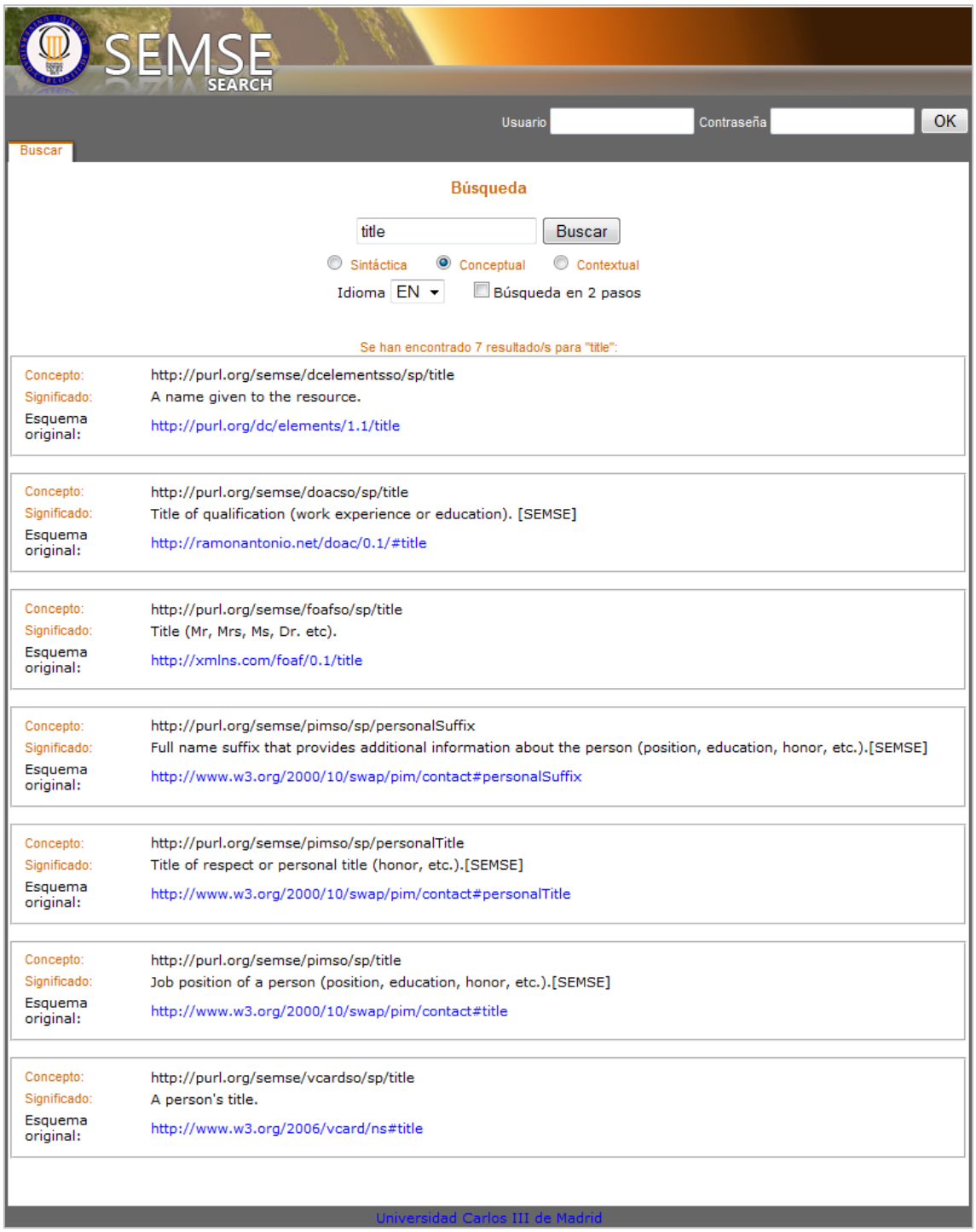

Figura IV-16. Captura de los resultado de una búsqueda

4.5.2 Autenticación de usuario

Para poder acceder a la funcionalidad propia del usuario administrador es necesario que el usuario se autentique como tal. Para hacerlo deberá de introducir su nombre de usuario y su contraseña en los cuadros de texto presentes en la parte superior derecha de la pantalla.

Figura IV-17.Captura de la introducción de usuario y contraseña

Si el usuario y la contraseña introducidos son correctos, la aplicación mantendrá al usuario en la pantalla de búsquedas, aunque desde este momento serán visibles las pestañas que permiten al usuario administrador la navegabilidad al resto de funciones.

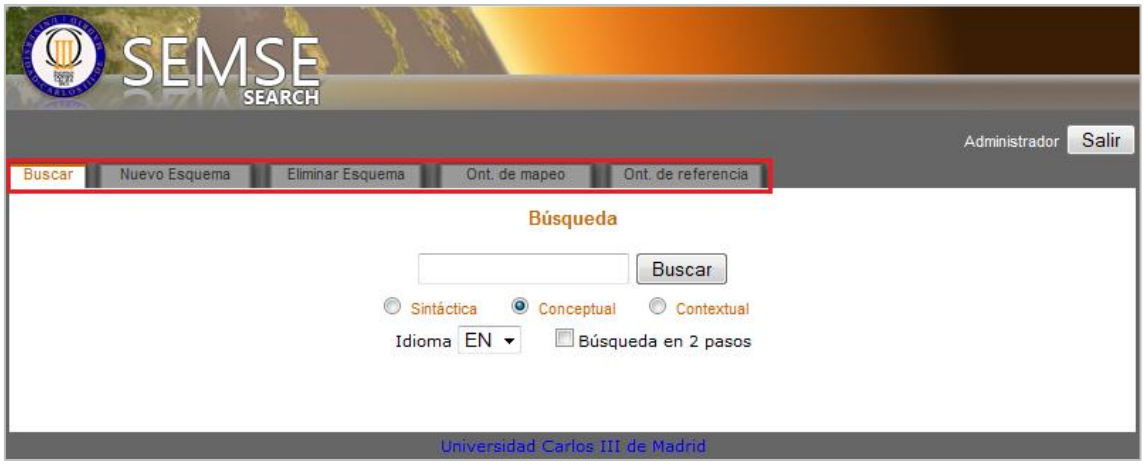

Figura IV-18. Captura de la pestañas de las funciones especiales

4.5.3 Nuevo esquema

Desde la pestaña etiquetada como "Nuevo Esquema" el usuario administrador podrá añadir un nuevo esquema a la base de datos, es decir, su esquema cualificado y su ontología específica, cuyas URIs serán construidas a partir de la URI del esquema original.

Para realizar esto el usuario debe de rellenar la siguiente información:

1. El nombre de esquema único (obligatorio).
2. La URI que lo identifica, y que además debe contener el nombre introducido anteriormente (obligatorio).
3. Una descripción del esquema (opcional).

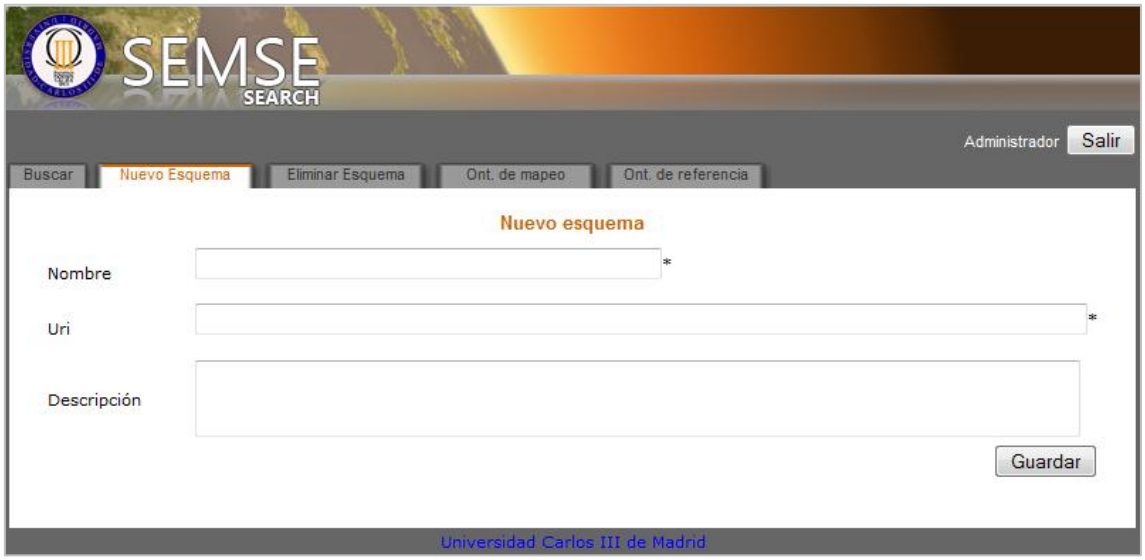

Figura IV-19. Captura de la adición de un nuevo esquema

Una vez que el usuario pulsa el botón "Guardar" la aplicación comprueba que los datos introducidos son válidos, para ello, además de comprobar que el nombre y la URI han sido introducidos por tratarse de campos obligatorios, valida que el nombre sea único y que la URI comience por la máscara http:// y termine por el nombre del esquema seguido de "/".

Si los datos son válidos, la aplicación se dispondrá a buscar el esquema original a través de su URI, además de buscar el esquema cualificado y la ontología específica. Si las tres URIs están disponibles la aplicación guarda en la base de datos estos dos últimos esquemas, informando al usuario del resultado de la operación.

Figura IV-20. Captura del mensaje de información sobre el resultado de una adición de esquemas

4.5.4 Eliminar esquema

Desde la pestaña etiquetada como “Eliminar Esquema” se muestra al usuario un listado de todos los esquemas que han sido añadidos a la base de datos y un enlace en cada uno para poder eliminarlos.

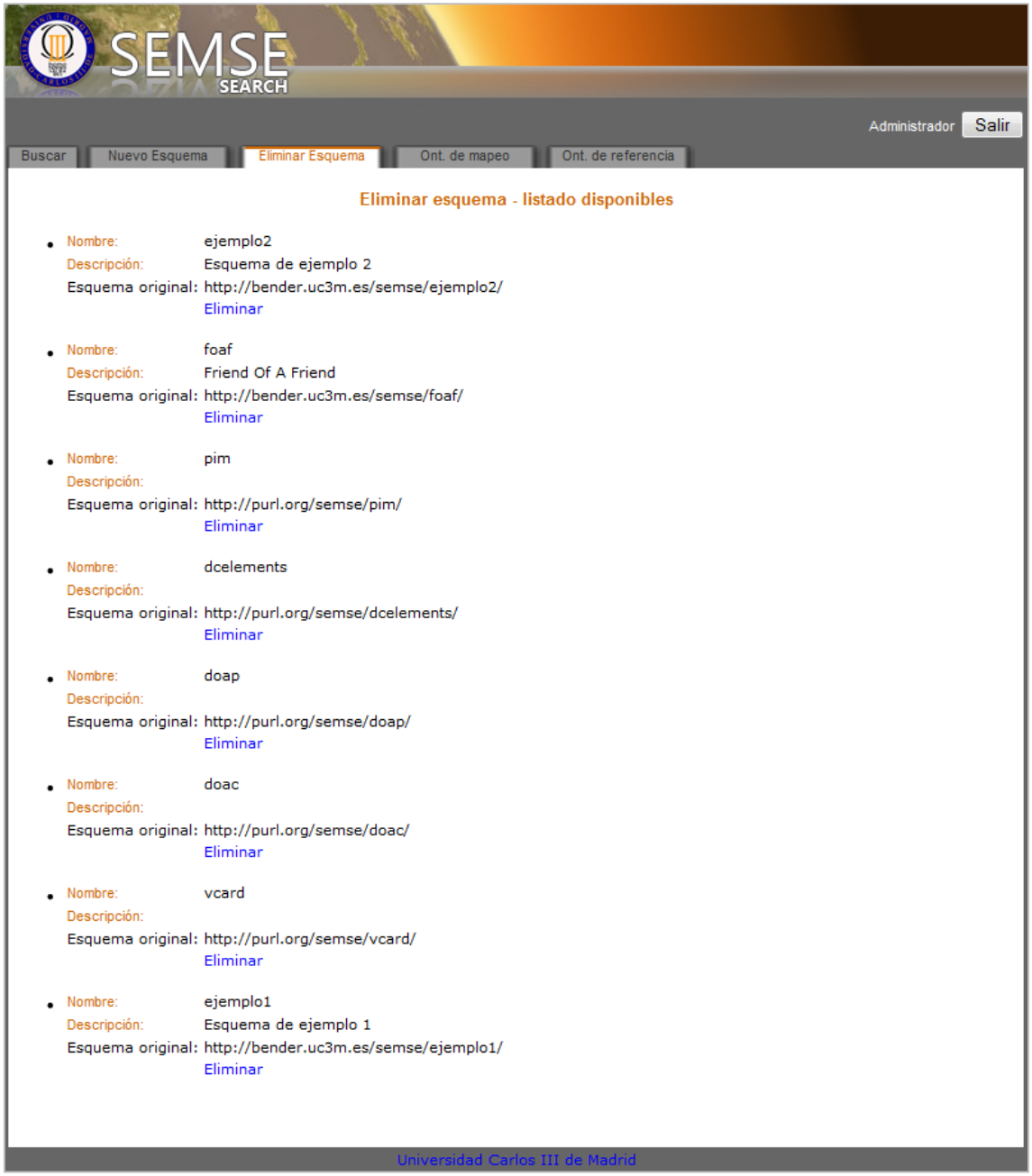

Figura IV-21. Captura del listado para eliminar un esquema

Una vez que el usuario pulsa el enlace "Eliminar" de alguno de los esquemas la aplicación le pide que confirme su decisión. Si el usuario finalmente acepta, el esquema cualificado semánticamente y la ontología de referencia asociados a dicho esquema, se eliminan de la base de datos de forma permanente.

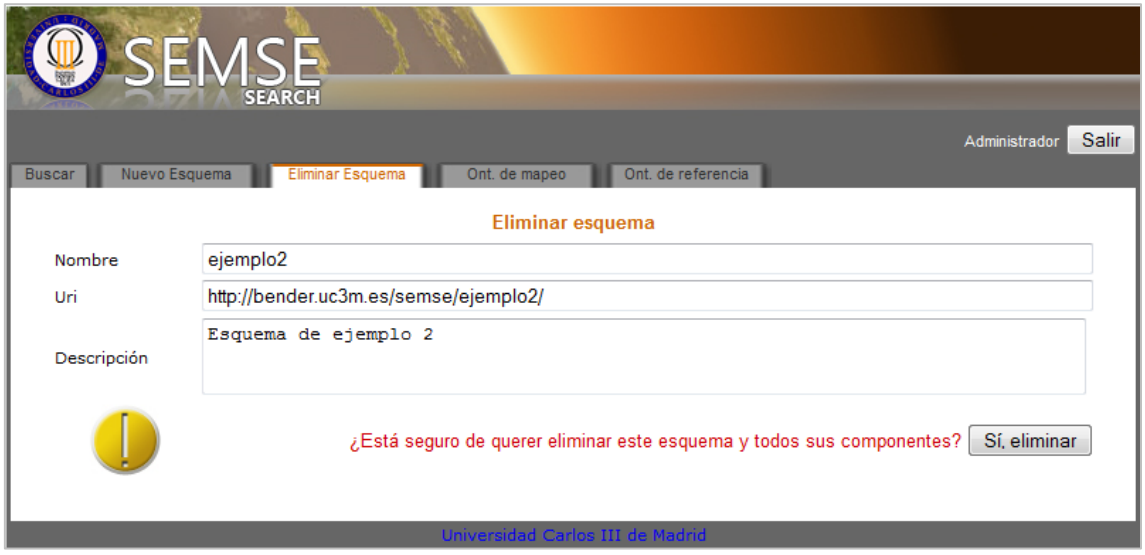

Figura IV-22. Captura de la confirmación de la eliminación de un esquema

Antes de finalizar el proceso por el que los esquemas asociados son eliminados de la base de datos e informar al usuario, la aplicación reconstruirá el SuperModelo con el fin de que las búsquedas que se realicen a partir de ese momento dejen de tener en cuenta los conceptos aportados por los esquemas asociados al esquema eliminado.

Figura IV-23. Captura del mensaje de información sobre la eliminación de un esquema

4.5.5 Consulta y modificación de la URI de la Ontología de Mapeo

Para poder consultar o modificar la URI que identifica a la ontología de mapeo es necesario acceder a la pestaña etiquetada como "Ont. De mapeo". Una vez que se ha accedido, la aplicación muestra un único cuadro de texto editable en el que por defecto aparece la URI que se tiene guardada actualmente. Si el usuario modifica dicha URI y pulsa el botón de "Guardar" la URI de la ontología de mapeo quedará modificada, por lo que a partir de ese momento se utilizará dicha URI para relacionar los conceptos de las ontologías específicas con los de la ontología de referencia.

Figura IV-24. Captura de la modificación de la URI de la ontología de mapeo

Dado que la URI ha cambiado, es necesario regenerar el SuperModelo antes de informar al usuario del éxito de la operación, con el fin de que las búsquedas que se ejecuten a partir de ese momento contemplen los cambios realizados.

Figura IV-25. Captura del mensaje informativo sobre la modificación de la URI de la ontología de mapeo

4.5.6 Consulta y modificación de la URI de la Ontología de Referencia

Finalmente, si el usuario desea modificar la URI de la ontología de referencia es necesario que acceda a la pestaña etiquetada como "Ont. de referencia". Una vez se ha accedido el procedimiento es idéntico al de la URI de la ontología de mapeo. La URI que aparece en el cuadro de texto es la URI que se tiene almacenada actualmente. Si se modifica y se pulsa el botón "Guardar" dicha URI será modifica y tenida en cuenta a partir de ese momento para obtener la jerarquización de los conceptos propios de dicha ontología.

Figura IV-26. Captura de la modificación de la URI de la ontología de referencia

Al igual que sucedía en la ontología de mapeo, si cambia la URI de la ontología de referencia es necesario regenerar el supermodelo para aplicar los nuevos cambios antes de informar al usuario del resultado de la operación.

Figura IV-27. Captura del mensaje informativo sobre la modificación de la URI de la ontología de referencia

4.6 Planificación final y Presupuesto

En este apartado se presenta un resumen de las tareas que se han realizado en este proyecto y su consecución en el tiempo. Además, basado en el número de horas que han sido necesarias para su desarrollo según la planificación final, se presenta un presupuesto.

4.6.1 Planificación final

Para ayudar a presentar la planificación final del proyecto, se ha hecho uso de un Diagrama de Gantt. La finalidad de estos diagramas es mostrar todas las fases, tareas y actividades programadas como parte de un proyecto y ordenadas en una línea temporal en la que se muestre, desde el comienzo del proyecto hasta su finalización, los plazos en los deban de ser ejecutadas, así como su duración y requisitos previos.

d	Nombre de tarea	Duración	Comienzo
1	Inicio Proyecto	0 días	lun 06/12/10
2	**1. Planificación**	**1 día**	**lun 06/12/10**
3	Estimación inicial	1 día	lun 06/12/10
4	**2. Formación Inicial**	**7 días**	**mar 07/12/1(**
5	Web Semántica	2 días	mar 07/12/1(
6	Esquemas de metadatos y ontologías	2 días	jue 09/12/10
7	Revisión documentación SEMSE	3 días	lun 13/12/10
8	**3. Especificación de requisitos de usuario**	**2 días**	**jue 16/12/10**
9	Análisis de requisitos	2 días	jue 16/12/10
10	Reunión Fin de Especificación de Requisito:	0 días	vie 17/12/10
11	**4. Análisis**	**5 días**	**lun 20/12/10**
12	Definir y desarrollar los requisitos software	3 días	lun 20/12/10
13	Definir los requisitos del interfaz	1 día	jue 23/12/10
14	Priorizar e integrar los requisitos	1 día	vie 24/12/10
15	Reunión Fin de Análisis	0 días	vie 24/12/10
16	**5. Diseño**	**5 días**	**lun 27/12/10**
17	Realizar el diseño de la arquitectura	3 días	lun 27/12/10
18	Diseñar la base de datos	1 día	jue 30/12/10
19	Diseñar la interfaz	1 día	vie 31/12/10
20	Reunión Fin de Diseño	0 días	vie 31/12/10
21	**6. Formación pre-implementación**	**13 días**	**lun 03/01/11**
22	Formación de Java	3 días	lun 03/01/11
23	Formación Spring y MVC	4 días	jue 06/01/11
24	Formación Spring Security	3 días	mié 12/01/11
25	Formación JPA	1 día	lun 17/01/11
26	Formación Jena	2 días	mar 18/01/11
27	**7. Codificación**	**96 días**	**jue 20/01/11**
28	**Desarrollo de las vistas o interfaz. JSP**	**6 días**	**jue 20/01/11**
29	Desarrollo	6 días	jue 20/01/11
30	Inspección de código	0 días	jue 27/01/11
31	**Desarrollo del modelo. Spring**	**10 días**	**vie 28/01/11**
32	Desarrollo	10 días	vie 28/01/11
33	Inspección del código	0 días	jue 10/02/11
34	**Desarrollo de la base de datos**	**1 día**	**vie 11/02/11**
35	Desarrollo	1 día	vie 11/02/11
36	Inspección de la propuesta	0 días	vie 11/02/11
37	**Desarrollo de la integración vista-modelo-controlador. Spring**	**15 días**	**lun 14/02/11**
38	Desarrollo de controladores	15 días	lun 14/02/11
39	Inspección de código	0 días	vie 04/03/11
40	**Persistencia de los esquemas originales. JP**	**5 días**	**lun 07/03/11**
41	Desarrollo	5 días	lun 07/03/11
42	Inspección del código	0 días	vie 11/03/11
43	**Persistencia de las representaciones de los esquemas. JENA**	**20 días**	**lun 14/03/11**
44	Desarrollo	20 días	lun 14/03/11
45	Inspección de código	0 días	vie 08/04/11
46	**Gestión de la ontología de alineamiento y la de referencia**	**5 días**	**lun 11/04/11**
47	Desarrollo	5 días	lun 11/04/11
48	Inspección de código	0 días	vie 15/04/11
49	**Generación y gestión del SuperModelo**	**4 días**	**lun 18/04/11**
50	Desarrollo	4 días	lun 18/04/11
51	Inspección de código	0 días	jue 21/04/11
52	**Búsquedas. SPARQL**	**20 días**	**vie 22/04/11**
53	Sintáctica	10 días	vie 22/04/11
54	Conceptual	5 días	vie 06/05/11
55	Contextual	5 días	vie 13/05/11
56	Inspección de código	0 días	jue 19/05/11
57	**Implantación de Seguridad. Spring Security**	**10 días**	**vie 20/05/11**
58	Desarrollo	10 días	vie 20/05/11
59	Inspección de código	0 días	jue 02/06/11
60	**8. Implantación**	**4 días**	**vie 03/06/11**
61	Distribución de software	1 día	vie 03/06/11
62	Instalación de software	2 días	lun 06/06/11
63	Aceptación de Software en ambiente de producción	1 día	mié 08/06/11
64	Fin de proyecto	0 días	mié 08/06/11

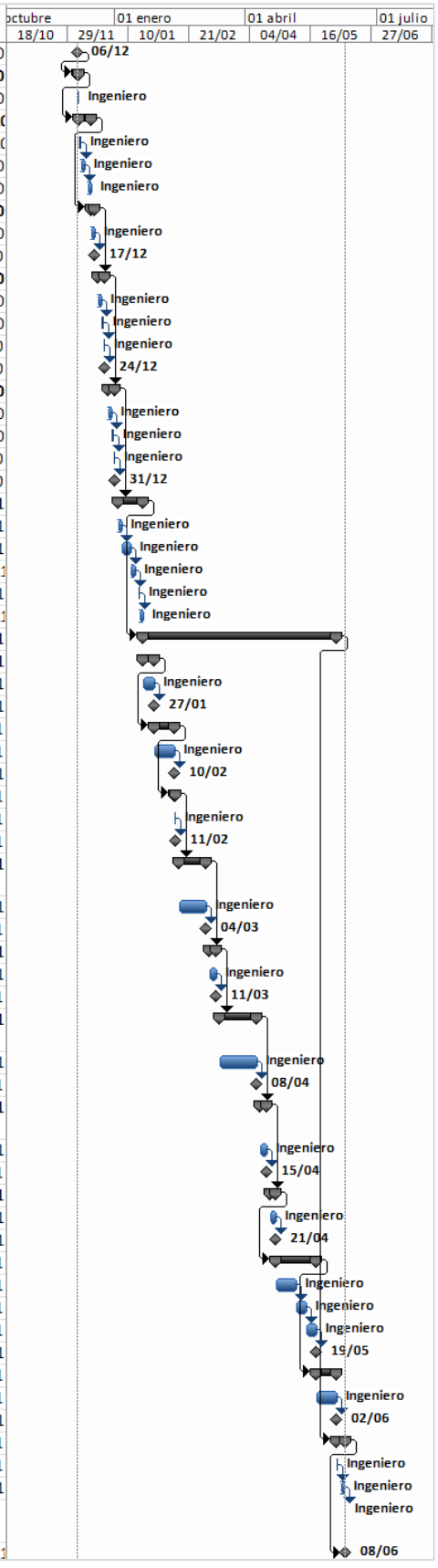

Figura IV-28. Planificación final del proyecto

4.6.2 Presupuesto

A partir de la planificación final del proyecto se ha obtenido un total de 133 días necesarios para el desarrollo del proyecto, lo que significa un total de 1.064 horas de trabajo. A continuación se detalla el presupuesto final de proyecto ajustado a 7 meses de duración.

Autor				
Miguel Zazo Torrubias				
Departamento				
Informática				
Descripción del proyecto				
Título	SEMSE search: Sistema de recuperación conceptual de esquemas de metadatos			
Duración	7 Meses			
Tasa de costes indirectos	20%			
Presupuesto Total del Proyecto				
				22.633 €
Desglose Presupuestario				
Personal				
Apellidos, Nombre	Categoría	Dedicación	Coste al mes	Coste total
Zazo Torrubias, Miguel	Ingeniero	7 meses	2.694,39 €	18.860,70 €
Equipos				
No aplica				
Subcontratación de tareas				
No aplica				
Otros costes directos del proyecto				
No aplica				
Resumen de costes				
Detalle			Costes totales	

Personal	18.861 €
Amortización	0 €
Subcontratación de tareas	0 €
Costes de funcionamiento	0 €
Costes indirectos	3.762 €
Total	**22.633 €**

Como se puede observar, se ha calculado un coste total de 22.633 € basado únicamente en el coste del personal y los costes indirectos. En cuanto al equipo necesario para la realización de este proyecto y al servidor que sirve de alojamiento para la aplicación, se trata de equipo que la universidad ya disponía de él, por lo que no se ha incluido como un gasto extra en este presupuesto. Adicionalmente, dado el tipo de desarrollo que exige este proyecto, no han sido necesario la subcontratación de tareas ni existe ningún otro tipo de gasto aplicable.

Capítulo V. Conclusiones

Una vez expuesto el trabajo realizado, en este capítulo de detallan las conclusiones a las que se han llegado mediante el desarrollo de este PFC, así como los objetivos cumplidos y los posibles desarrollos futuros.

Este proyecto, dado que forma parte del proyecto SEMSE, es el resultado de toda la investigación y trabajo realizado previamente y que ha servido como punto de partida y referente para el desarrollo de dicho proyecto. Como resultado, se ha creado una herramienta que aprovecha de una manera muy útil la base semántica desarrollada en el proyecto SEMSE y es una expresión de las múltiples utilidades que se le pueden dar a dicho trabajo. Aunque todos los recursos desarrollados en SEMSE han estado publicados desde el momento de su creación, con este proyecto se ha creado una forma de compartir todo el trabajo realizado, mientras que al mismo tiempo, se ha convertido en un servicio, ofrecido a través de Internet, que sirve de ayuda para todo aquel interesado en encontrar un esquema de metadatos adecuado a sus necesidades.

Adicionalmente, desde el punto de vista del contexto en el que se encuentra este proyecto, se puede concluir que se trata de una herramienta que forma parte de los primeros pasos por facilitar la evolución de la Web actual hacía la Web Semántica del futuro, mediante la promoción de la reutilización de esquemas de metadatos con el fin de facilitar el trabajo a los desarrolladores que estén dispuestos a aportar semántica a sus desarrollos. Por lo tanto, dado que se trataba de un objetivo fundamental del desarrollo de esta aplicación y de SEMSE en general, el éxito de esta aplicación o de sus futuras versiones basado en la utilización por parte de los usuarios, será el que determine si se ha cumplido o no este objetivo.

En cuanto a su desarrollo, se han utilizado las tecnologías más modernas destinadas tanto a la creación de aplicaciones Web, como al tratamiento de esquemas de metadatos. Aunque ya existían estudios previos sobre las recomendaciones sobre qué tecnologías usar, el hecho de usarlas ha implicado una fase de documentación y aprendizaje considerable, debido a que, aunque conocía Java como lenguaje de programación, el uso de Frameworks como Spring o JPA me eran desconocidos. Aunque quizá, lo que más dificultad ha entrañado y tiempo de desarrollo ha implicado, ha sido el uso del Framework de Jena y la realización de consultas en SPARQL, debido a que se trata de un Framework muy específico y desconocido salvo para aquellos que hayan necesitado alguna vez del tratamiento de esquemas de metadatos. Aunque por otra parte más que recomendable si es el caso.

Es por ello, que debido a la dificultad experimentada a la hora de integrar todas las tecnologías utilizadas, se ha querido dotar al apartado *4.4 Diseño Detallado* de un doble sentido. Por un lado, explicar de forma detallada como se ha desarrollado el proyecto propiamente; y por otro, de convertirlo en una pequeña guía de iniciación que pueda resultarle útil a todo aquel que quiera iniciarse en cualquiera de estas tecnologías.

5.1 Desarrollos futuros

Como primera mejora y quizá la más importante, sería la ampliación de la base ontológica de la que dispone la aplicación y que inicialmente se creó en el proyecto SEMSE. Esto es, aumentar el número de esquemas dados de alta en la aplicación, así como la actualización de la ontología de referencia y de alineamiento. El hecho de aplicar el mismo proceso de cualificación semántica que se aplicó a los esquemas originales sobre otros que gocen de popularidad y que pudieran aportar semántica de otros ámbitos, enriquecerían en gran medida los resultado devueltos por el buscador, y por tanto, aumentaría la utilidad y precisión del mismo.

Por otro lado, y orientado a la popularidad de la aplicación, sería muy conveniente la traducción a otros idiomas y que éste fuera detectable automáticamente. De esta forma no existiría la barrera idiomática, lo que facilitaría el uso de la aplicación por personas no hispanohablantes.

Finalmente, y dado que el objetivo principal de la aplicación es la reutilización de esquemas de metadatos para facilitar de esta forma el paso a la Web Semántica, sería recomendable incluir una sección, aunque bien diferenciada del buscador para no perder el verdadero objetivo de la aplicación, cuya función fuera la recopilación de documentos, guías y manuales de las mejores prácticas en cuanto a la inclusión de semántica en los recursos Web. De esta forma, se situaría a la aplicación como un referente para todo aquel implicado en este proceso.

Bibliografía

Abián, M. Á. (2005). *Ontologías: Qué son y para qué sirven*. Disponible en: http://www.wshoy.sidar.org/index.php?2005/12/09/30-ontologias-que-son-y-para-que-sirven [Consultado en Enero de 2011]

Álvarez Marañón, G. (1999). *¿Qué es Java?* Disponible en: http://www.iec.csic.es/criptonomicon/java/quesjava.html [Consultado en Febrero de 2011]

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P.y Patel-Schneider, L. A. (2004). *OWL Web Ontology Language*. Disponible en: http://www.w3.org/TR/owl-ref/ [Consultado en Enero de 2011]

Bernardo Cuenca, G., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.y Sattler, U. (2008). *OWL 2: The next step for OWL*. Disponible en: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B758F-4TP1FC8-1&_user=10&_coverDate=11/30/2008&_rdoc=1&_fmt=high&_orig=search&_origin=search&_sort=d&_docanchor=&view=c&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=4e88f31e9c5480f593242 [Consultado en Enero de 2011]

Berners-Lee, T. (1994a). *Universal Resource Identifiers in WWW*. Disponible en: http://www.ietf.org/rfc/rfc1630.txt [Consultado en Enero de 2010]

Berners-Lee, T. (1994b). *Universal Resource Identifiers*. Disponible en: http://www.ietf.org/rfc/rfc1630.txt [Consultado en Enero de 2010]

Berners-Lee, T. (2007). *Semantic Web Application Platform - SWAP*. Disponible en: http://www.w3c.org/2000/10/swap/pim/contact. [Consultado en Enero de 2011]

Berners-Lee, T., Hendle, J.y Lassila, O. (2001). *The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*. Disponible en: http://kill.devc.at/system/files/scientific-american_0.pdf [Consultado en Enero de 2011]

Blanco, M. (2008). *Almacenamiento persistente de ontologías con Jena y MySQL*. Disponible en: http://marcosblanco.blogspot.com/2008/04/almacenamiento-persistente-de-ontologas.html [Consultado en Febrero de 2011]

Booch, G. (2006). *El lenguaje unificado de modelado: guía del usuario (2ª Ed.)*. ADDISON-WESLEY. Madrid. [Consultado en Enero de 2011]

Brickley, D.y Miller, L. (2011). *Friend of a Friend*. Disponible en: http://www.foaf-project.org/about [Consultado en Abril de 2011]

Brodkin, J. (2011). *PCWorld*. Disponible en: http://www.pcworld.com/article/216728/windows_on_verge_of_dropping_below_90_market_share.html#tk.mod_rel [Consultado en Enero de 2011]

Caro, F. (2010). *WordPress*. Disponible en: http://federicojcdm.wordpress.com/2010/03/23/un-recorrido-por-spring-security-3-0/ [Consultado en Marzo de 2011]

Casanovas, J. (2004). *Usabilidad y arquitectura del software*. Disponible en: http://www.alzado.org/articulo.php?id_art=355 [Consultado en Enero de 2011]

CIST (2009). *Persistence.xml*. Disponible en: http://wiki.cetechihuahua.gob.mx/index.php/Persistence.xml [Consultado en Febrero de 2011]

Degug_mode=On (2009). *Relaciones en JPA*. Disponible en: http://es.debugmodeon.com/articulo/relaciones-en-jpa [Consultado en Febrero de 2011]

Defense Advance Research Projects Agency (2002). *DAML+OIL*. Disponible en: http://www.daml.org/2001/03/daml+oil-index.html [Consultado en Abril de 2010]

Dickinson, I. (2009). *Jena Ontology API*. Disponible en: http://jena.sourceforge.net/ontology/ [Consultado en Febrero de 2011]

doc.Ubuntu-es (2011). *Sobre Ubuntu*. Disponible en: http://doc.ubuntu-es.org/Sobre_Ubuntu [Consultado en Marzo de 2011]

Dublin Core Metadata Iniciative (2011). *Dublin Core*. Disponible en: http://dublincore.org/ [Consultado en Abril de 2011]

Dumbill, E. (2008). *DOAP*. Disponible en: http://trac.usefulinc.com/doap [Consultado en Abril de 2011]

Escartín Vigo, J. A. (2005). *UPCommons*. Disponible en: http://upcommons.upc.edu/pfc/handle/2099.1/3451 [Consultado en Enero de 2011]

Fernández-Tostado Canorea, T. (2010). *Estudio tecnológico y diseño arquitectónico de un Sistema de Gestión de Esquemas Semánticos basados en Ontologías*. Universidad Carlos III de Madrid. [Consultado en Enero de 2010]

Fowler, M. (2004). *Inversion of Control Containers and the Dependency Injection pattern*. Disponible en: http://martinfowler.com/articles/injection.html [Consultado en Febrero de 2011]

Gárate Barreiro, F. J. (2010). *Desarrollo de un Sistema de Gestión para la cualificación semántica de esquemas.* Universidad Carlos III de Madrid [Consultado en Enero de 2011]

Geraldo, Á. (2005). *Diseño y Modelación de un Proyecto de Software Utilizando el lenguaje UML*. Disponible en: http://www.monografias.com/trabajos28/proyecto-uml/proyecto-uml.shtml [Consultado en Enero de 2011]

Gruber, T. R. (1993). *A Translation Approach to Portable Ontology Specifications*. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.3273&rep=rep1&type=pdf [Consultado en Enero de 2011]

Guarino, N. (1998). *Formal Ontology and Information Systems*. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.1776&rep=rep1&type=pdf [Consultado en Enero de 2011]

Guía Ubuntu (2008). *PgAdmin III*. Disponible en: http://www.guia-ubuntu.org/index.php?title=PgAdmin_III [Consultado en Marzo de 2011]

IBM (2011). *JPA-Arquitecture*. Disponible en: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.ejbfep.multiplatform.doc/info/ae/ae/cejb_persistence.html [Consultado en Marzo de 2011]

Jena (2011). *Jena Documentation*. Disponible en: http://jena.sourceforge.net/documentation.html [Consultado en Febrero de 2011]

JGuru (2000). *jGuru: JavaServer Pages Fundamentals, Short Course Contents*. Disponible en: http://java.sun.com/developer/onlineTraining/JSPIntro/contents.html [Consultado en Febrero de 2011]

Kay, A. (2007). *Alan Kays Definition Of Object Oriented*. Disponible en: http://c2.com/cgi/wiki?AlanKaysDefinitionOfObjectOriented [Consultado en Marzo de 2011]

Kioskea (2008). *OOP-Polimorfismo*. Disponible en: http://es.kioskea.net/contents/poo/polymorp.php3 [Consultado en Febrero de 2011]

Krsulovic Morales, E. (2003). *Desarrollo de un sistema de anuario, utilizando integración semántica en la Web.* Disponible en: http://www.dcc.uchile.cl/~ekrsulov/slides/titulo/slide3-0.html [Consultado en Enero de 2011]

Kunder, M. d. (2010). *The size of the World Wide Web*. Disponible en: www.worldwidewebsize.com [Consultado en Enero de 2011]

Lamarca Lapuente, M. J. (2002). *Hipertexto: El nuevo concepto de documento en la cultura de la imagen*. Disponible en: [Consultado en Enero de 2011]

Lozano Tello, A. (2001). *Ontologías en la Web Semántica. I Jornadas de Ingeniería Web' 01*. Disponible en: http://www.anobium.es/docs/gc_fichas/doc/68ERfhjkmv.pdf [Consultado en Enero de 2011]

Madeja (2011). *JPA*. Disponible en: http://www.juntadeandalucia.es/xwiki/bin/view/MADEJA/JPA [Consultado en Febrero de 2011]

Maedche, A. (2002). *Ontology Learning for the semantic Web*. Kluwer Academic Publishers. [Consultado en Enero de 2011]

Malik, A. (2003). *Ontologies, and the semantic Web, The second generation of the web*. Disponible en: http://goliath.ecnext.com/coms2/gi_0199-2476390/XML-ontologies-and-the-Semantic.html [Consultado en Enero de 2011]

Mihalcea, R. F.y Mihalcea, S. I. (2001). *Word Semantics for Information Retrieval: Moving One Step Closer to the Semantic Web*. Disponible en: http://www.cse.unt.edu/~rada/papers/mihalcea.ictai01.pdf [Consultado en Enero de 2011]

Miller, G. A. (1995). *Wordnet*. Disponible en: http://wordnet.princeton.edu/ [Consultado en Abril de 2011]

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T.y Swartout, W. R. (1991). *Enabling Technology for Knowledge Sharing*. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.2902&rep=rep1&type=pdf [Consultado en Enero de 2011]

Netbeans (2011). *NetBeans IDE 6.9.1 Release Information*. Disponible en: http://netbeans.org/community/releases/69/ [Consultado en Febrero de 2011]

Obikto, M. (2007). *Semantic Web Architecture*. Disponible en: http://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html [Consultado en Marzo de 2011]

Ogden, C. K.y Richards, I. A. (1923). *Meaning of meaning*. New York: Harcourt & Brace. [Consultado en Enero de 2011]

Palacios Madrid, V. (2010). *Sistema de Recuperación Conceptual mediante Niveles Semánticos en la representación de Esquemas de Metadatos*. Departamento de Biblioteconomía y Documentación de la Universidad Carlos III de Madrid [Consultado en Abril de 2011]

Parada, R. A. (2008). *DOAC: Description of a Career*. Disponible en: http://ramonantonio.net/doac/ [Consultado en Abril de 2011]

Paredes, A. (2010). *Introducción a las Ontologías en la Web Semántica*. Disponible en: http://personal.us.es/aparedes/ws.pdf [Consultado en Enero de 2011]

Patel, V. (2008). *Java Virtual Machine, An inside story!!* Disponible en: http://viralpatel.net/blogs/2008/12/java-virtual-machine-an-inside-story.html [Consultado en Marzo de 2011]

Protégé (2011). *Protégé*. Disponible en: http://protege.stanford.edu/overview/index.html [Consultado en Marzo de 2011]

SEMSE (2008). *SEMSE: SEmantic Metadata SEarch*. Disponible en: http://163.117.147.101:9000/SEMSE/index.php [Consultado en Enero de 2011]

Spring Source (2010). *About Spring*. Disponible en: http://www.springsource.org/about [Consultado en Marzo de 2011]

Spring Source (2011). *Spring Security*. Disponible en: http://static.springsource.org/spring-security/site/ [Consultado en Marzo de 2011]

Studer, R., Benjamins, V.y Fensel, D. (1998). *Knowledge Engineering: Principles and Methods*. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.5463&rep=rep1&type=pdf [Consultado en Enero de 2011]

Terziev, I., Kiryakov, A.y Dimitar, M. (2005). *Base upper-level ontology (BULO) Guidance*. Disponible en: http://proton.semanticweb.org/D1_8_1.pdf [Consultado en Enero de 2011]

Umbel.org (2008). *Umbel*. Disponible en: http://www.umbel.org/ [Consultado en Abril de 2011]

The Unicode Consortium (2010). *The Unicode Consortium*. Disponible en: http://www.unicode.org/ [Consultado en Enero de 2011]

Uschold, M.y Jasper, R. (1999). *A Framework for Understanding and Classifying Ontology Applications*. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.6456&rep=rep1&type=pdf [Consultado en Febrero de 2011]

Vargas Romo, A. R. (2009). *Glassfish*. Disponible en: http://blogs.sun.com/AlanVargas/entry/qu%C3%A9_es_glassfish [Consultado en Marzo de 2011]

Walls, C. (2008). *Spring*. Anaya. Madrid. ISBN: 978-84-415-2497-2 [Consultado en Marzo de 2011]

webKB.org (2003). *webKB*. Disponible en: http://www.webkb.org/ [Consultado en Abril de 2011]

Webtaller (2011). *Manual de Java*. Disponible en: http://www.webtaller.com/manual-java/caracteristicas-java.php [Consultado en Marzo de 2011]

Wikipedia (2010a). *Clase (Informática)*. Disponible en: http://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica) [Consultado en Marzo de 2011]

Wikipedia (2010b). *Glassfish*. Disponible en: http://es.wikipedia.org/wiki/GlassFish [Consultado en Marzo de 2011]

Wikipedia (2011a). *AOP*. Disponible en: http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_aspectos [Consultado en Febrero 2011 de]

Wikipedia (2011b). *IOC*. Disponible en: http://es.wikipedia.org/wiki/Inversi%C3%B3n_de_Control [Consultado en Febrero de 2011]

Wikipedia (2011c). *Servidor de Aplicaciones*. Disponible en: http://es.wikipedia.org/wiki/Servidor_de_aplicaciones [Consultado en Enero de 2011]

Wikipedia (2011d). *Servidor Web*. Disponible en: http://es.wikipedia.org/wiki/Servidor_web [Consultado en Enero de 2011]

Wikipedia (2011e). *World Wide Web*. Disponible en: http://es.wikipedia.org/wiki/World_Wide_Web [Consultado en Enero de 2011]

Wikipedia (2011f). *WWW*. Disponible en: http://es.wikipedia.org/wiki/World_Wide_Web [Consultado en Enero de 2011]

World Wide Web Consortium (1998). *Document Object Model*. Disponible en: http://www.w3.org/TR/REC-DOM-Level-1/ [Consultado en Enero de 2011]

World Wide Web Consortium (1999a). *HTML 4.01 Specification*. Disponible en: http://www.w3.org/TR/html401/ [Consultado en Enero de 2011]

World Wide Web Consortium (1999b). *Resource Description Framework (RDF)*. Disponible en: http://www.w3.org/RDF/ [Consultado en Enero de 2011]

World Wide Web Consortium (1999c). *XML Path Language (XPath)*. Disponible en: http://www.w3.org/TR/xpath/ [Consultado en Enero de 2011]

World Wide Web Consortium (1999d). *XSL Transformations (XSLT)*. Disponible en: http://www.w3.org/TR/xslt [Consultado en Enero de 2011]

World Wide Web Consortium (2001). *XML Linking Language (XLink)*. Disponible en: http://www.w3.org/TR/xlink/ [Consultado en Enero de 2011]

World Wide Web Consortium (2002). *XHTML 1.0 The Extensible HyperText Markup Language*. Disponible en: http://www.w3.org/TR/xhtml1/ [Consultado en Enero de 2011]

World Wide Web Consortium (2003). *XPointer Framework*. Disponible en: http://www.w3.org/TR/xptr-framework/ [Consultado en Enero de 2011]

World Wide Web Consortium (2004a). *RDF Primer*. Disponible en: http://www.w3.org/TR/rdf-primer/ [Consultado en Enero de 2011]

World Wide Web Consortium (2004b). *RDF Vocabulary Description Language 1.0: RDF Schema*. Disponible en: http://www.w3.org/TR/rdf-schema/ [Consultado en Enero de 2011]

World Wide Web Consortium (2004c). *XML Information Set (Second Edition)*. Disponible en: http://www.w3.org/TR/xml-infoset/ [Consultado en Enero de 2011]

World Wide Web Consortium (2004d). *XML Schema*. Disponible en: http://www.w3.org/XML/Schema [Consultado en Enero de 2011]

World Wide Web Consortium (2006a). *Extensible Stylesheet Language (XSL) Version 1.1*. Disponible en: http://www.w3.org/TR/xsl11/ [Consultado en Enero de 2011]

World Wide Web Consortium (2006b). *Naming and Addressing: URIs, URLs, ...* Disponible en: http://www.w3.org/Addressing/ [Consultado en Enero de 2011]

World Wide Web Consortium (2009a). *Namespaces in XML 1.0*. Disponible en: http://www.w3.org/TR/REC-xml-names/ [Consultado en Abril de 2011]

World Wide Web Consortium (2009b). *OWL 2 Web Ontology Language*. Disponible en: http://www.w3.org/TR/2009/REC-owl2-overview-20091027/ [Consultado en Enero de 2011]

World Wide Web Consortium (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Disponible en: http://www.w3.org/TR/REC-xml/ [Consultado en Enero de 2011]

World Wide Web Consortium (2010a). *Cascading Style Sheets Level 2 Revision 1*. Disponible en: http://www.w3.org/TR/CSS2/ [Consultado en Enero de 2011]

World Wide Web Consortium (2010b). *Representing vCard Objects in RDF*. Disponible en: http://www.w3.org/Submission/vcard-rdf/ [Consultado en Abril de 2011]

World Wide Web Consortium (2011). *HTTP - Hypertext Transfer Protocol*. Disponible en: http://www.w3.org/Protocols/ [Consultado en Enero de 2011]