UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES, SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Desarrollo de un Sistema de Gestión para la Cualificación Semántica de Esquemas

Autor: Francisco José Gárate Barreiro

Tutor: Vicente Palacios Madrid

Diciembre 2010

Agradecimientos

En primer lugar quisiera agradecer a mis padres la paciencia que han tenido y el apoyo que me han dado en los buenos y en los malos momentos.

También agradezco a mi tutor Vicente toda la ayuda que me ha prestado.

Por último no quiero olvidarme de todos los amigos que he conocido durante todos estos años, compañeros de clase y del servicio de informática, solo por eso ya ha merecido la pena.

Resumen

Las ontologías son una parte fundamental de la web semántica al permitir relacionar la información de la web con su significado. Este proceso de cualificación semántica es necesario para lograr la recuperación semántica de información en la web. El presente trabajo, englobado dentro del proyecto SEMSE, tras realizar un análisis de la problemática existente en la Web actual y la identificación de elementos que pudieran solventar esta situación, se centra en el desarrollo de una aplicación Web que sirva como herramienta para la gestión de esquemas de metadatos y facilite el proceso de cualificación semántica.

Índice general

Índices de figuras

Índice de códigos fuente

Índice de tablas

Capítulo1. Introducción

El enfoque original de la Web, orientado a su utilización por personas en lugar de aplicaciones, ha limitado las posibilidades de las aplicaciones Web, tales como la recuperación de información, los sistemas pregunta-respuesta o el razonamiento mediante inferencias, impidiendo que las máquinas tengan un nivel de comprensión de la Web suficiente como para hacerse cargo de una parte. Estas limitaciones están relacionadas con: la tecnología utilizada, el número y calidad de los metadatos y el tamaño del Web. La solución más adecuada consiste en transformar la información en un formato más comprensible para el software, con una mayor carga de semántica codificada. Para conseguir este objetivo es necesario crear una "*capa semántica*" que pueda ser "*comprendida*" por el software. Este es el objetivo de la Web Semántica, y para conseguirlo utiliza como elemento clave las ontologías.

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 e incluye definiciones de conceptos básicos en el dominio y sus relaciones de forma que puedan ser interpretados por una máquina.

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

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 los esquemas de metadatos 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 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. De este modo, se 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 conseguirá un mapa conceptual y relacional que incluirá y permitirá equivalencias léxicas entre los términos de las etiquetas de los documentos y la estructura ontológica general.

1.1- Objetivos

El objetivo principal es facilitar la gestión y manejo, de forma manual, de esquemas, relacionados con la ingeniería del conocimiento.

La totalidad de sus funciones están relacionadas con el concepto de reutilizar la información de un dominio determinado. Esta aplicación permite compartir y gestionar esquemas, que servirán de base a futuros proyectos que consigan cumplir los objetivos propuestos en el proyecto SEMSE.

Puesto que el producto está destinado a un amplio abanico de usuarios, que no tienen por qué ser expertos en el manejo de este tipo de herramientas, la actual propuesta pretende ser intuitiva y de fácil manejo, prescindiendo de adornos que retrasen o dificulten la interacción. La aplicación pretende tener un diseño minimalista que favorezca la concentración en la gestión de esquemas y ontologías relacionados con la ingeniería del conocimiento.

Este PFC está englobado dentro del proyecto SEMSE y se centrará en la implementación de la gestión y consulta de esquemas, tanto originales como cualificados, siendo los principales requisitos a satisfacer:

- **La aplicación permitirá la consulta de esquemas por metadatos.** Pudiendo ser consultados por distintos parámetros, nombre del esquema, autor, URI, Dominio o fecha.

- **La aplicación permitirá la gestión de esquemas.** Pudiéndose dar de alta, baja, edición y validación.

- **La aplicación permitirá abrir y descargar esquemas.** Pudiéndose abrir o descargar los distintos esquemas.

- **La aplicación permitirá que el usuario pueda consultar la ayuda.**

Para ello, la aplicación Web deberá ofrecer las siguientes funcionalidades:

- **Autenticación de usuarios y control de acceso** Definición de roles y funcionalidades permitidas a cada uno de ellos.
- **Interfaz sencilla** que permita facilitar las distintas operaciones sobre los esquemas.
- **Visualización de los esquemas existentes** para poder consultar en cualquier momento el estado de los mismos

Para lograr estos objetivos, se aplicará un ciclo de vida en cascada, con las siguientes fases, tal y como se detallará en el Capítulo 4 "*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.

1.2- Estructura del Proyecto

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

En el siguiente capítulo, Capitulo 2 "*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 3 "*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 4 *"Desarrollo del proyecto"* es la síntesis del esfuerzo de elaboración de este trabajo. Se recoge el proceso de desarrollo del sistema realizado hasta el momento.

En el Capítulo 5 *"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.

Capítulo 2. Estado del Arte

Tal y como se ha planteado en la introducción de este trabajo, el PFC consiste en el desarrollo de un sistema informático, accesible vía Web, que permita la reutilización de esquemas, que sea comprensible por el usuario, potenciando la usabilidad mediante una interfaz sencilla y amigable.

En este Capítulo se realizará la presentación de las distintas tecnologías relacionadas con el PFC. Partiendo de la descripción de la Web actual junto con los problemas asociados que derivan de ella y como la Web Semántica pretende resolverlos basándose en el uso de ontologías, posteriormente se tratarán los esquemas de metadatos y la ontología de referencia PROTON.

A continuación se hablará de las aplicaciones Web y los requisitos que necesitan cumplir para poder ser usables, objetivo a cumplir por el sistema informático que soportará al proyecto SEMSE.

Finalmente se presentan los conceptos de desarrollo de software, lenguaje de modelado unificado y los distintos patrones que se implementarán en el diseño del sistema.

2.1- La Web Semántica

Según (LOZANO, 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.

Los actuales buscadores Web realizan la búsqueda de información mediante palabras clave que aparecerán en el código HTML[1] (Lenguaje de Marcado de Documentos para Hipertexto) de las páginas Web dispersas en Internet. En los últimos años, algunas empresas están realizando anotaciones de datos introducidas dentro de este código HTML, siguiendo algún esquema de anotación común, normalmente basado en XML[2].

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 a lo que estamos buscando. Además no todas las páginas proporcionan igual cantidad de información, debido precisamente a que no existe un formato o convenio que nos diga qué contenido se debe añadir a 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.

[1] HTML es el lenguaje de marcado empleado para la elaboración de páginas Web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

[2] XML es un metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Las ventajas que ofrece Internet son enormes a la hora de buscar información, pero adolece de una manera de encontrar información de forma precisa y de poder realizar deducciones con la información existente.

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.

2.1.1- 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. 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[3] (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

[3] WWW es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza sitios Web compuestos de páginas Web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

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 los documentos XML, deberá de añadirse 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, 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 ordenadores 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.

2.1.2- De la Web actual a la Web Semántica

En este apartado, se verá cuál es el estado actual de la Web, las bases de su funcionamiento, y sus principales problemas. A partir de ahí se comentará porque parece necesario el salto a otro tipo de Web, la Web Semántica.

2.1.2.1- La Web actual

La Web se puede ver como un conjunto de documentos enlazados unos con otros (hipertexto), disponibles en una red de ordenadores (Internet).

El término "*hipertexto*" fue acuñado por Ted Nelson en 1965, en su artículo "*A File Structure for the Complex, the Changing, and the Indeterminate*", que leyó durante la vigésima conferencia anual de la Association of Computer Machinery (ACM).

Ted Nelson ideó un modelo para la interconexión de documentos electrónicos. La World Wide Web fue inventada en 1989 por Tim Berners-Lee, un informático del CERN (Organización Europea de Investigación Nuclear). Era un sistema de hipertexto para compartir información basado en Internet, concebido originalmente para servir como herramienta de comunicación entre los científicos nucleares del CERN.

Funcionamiento tecnológico

Se pueden distinguir dos tecnologías principales que intervienen en el funcionamiento de la Web, el protocolo de aplicación HTTP[4] y el lenguaje de intercambio HTML.

La Web funciona sobre internet, por lo que está "*montada*" sobre la pila de protocolos TCP/IP[5], protocolos de transporte y red respectivamente. HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura Web para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. A la información transmitida se la llama recurso y se la identifica mediante una URL[6].

La principal característica de la Web es su estructura de hipertexto. Ésta permite una lectura no lineal de las páginas Web, en la que el lector puede saltar a información relacionada con el documento actual. Para describir la estructura y el contenido de los documentos, se utiliza el lenguaje HTML. Con este lenguaje, se define el contenido de las páginas Web, así como la información que necesitan los navegadores sobre cómo mostrarla en pantalla y los vínculos a otras páginas, en los que se basa el hipertexto.

Problemas de la Web actual

En la Web actual se pueden distinguir principalmente dos problemas:

[4] HTTP es el protocolo usado en cada transacción de la WWW. 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 sigue el esquema petición-respuesta entre un cliente y un servidor.

[5] TCP/IP es la base de Internet, y sirve para enlazar computadoras que utilizan diferentes sistemas operativos, sobre redes de área local (LAN) y área extensa (WAN).

[6] URL 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.

No incorpora mecanismos que permiten el procesado automático de la información. 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 hace que no se puedan hacer búsquedas en la Web con un contenido semántico.

No incluye mecanismos para la interoperabilidad completa de los sistemas de información. No facilita la comprensión común y compartida de un dominio, de forma que esta no puede ser utilizada por personas, organizaciones y máquinas. Para llegar a esa comprensión, hay que llegar a tener tres tipos de interoperabilidad.

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

- **Interoperabilidad sintáctica**: Es la capacidad de un SI de leer datos procedentes de otro. 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 común significado de los términos y expresiones que usan, es decir consistente en el significado que tiene la información.

La falta de interoperabilidad en la Web, hace que surjan varios problemas. A continuación se verán los dos problemas más relevantes, la dificultad para encontrar información y la dificultad para implantar comercio electrónico B2B.

Dificultad para encontrar 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, depende 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, 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 que nos interesan no usan el mismo vocabulario que el empleado en la búsqueda, nunca aparecerán.

Dificultad para implantar comercio electrónico B2B

Un reto con el que se enfrentan las empresas es la integración con otras, ya sean clientes, proveedores, fabricantes, etc. Para lograra esto, los SI de las empresas deben permitir el intercambio de información interempresarial.

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[7] (Electronic Data Interchange).

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 inflexibilidad.

2.1.2.1- La Web Semántica

A partir del esquema (RICHERO, 2007) se verá cómo enmarcar la Web Semántica dentro de otros conceptos como Web 2.0 y Web 3.0.

- **Web 1.0.** Personas conectándose a la Web

- **Web 2.0**. Personas conectándose a personas - redes sociales, wikis, colaboración, posibilidad de compartir.

[7] EDI es el intercambio automatizado de documentos comerciales electrónicos entre una organización y sus socios comerciales, de modo que no se requiera intervención humana.

- **La Web Semántica.** Vendría a ser una extensión de la Web actual dotada de significado, esto es, un espacio donde la información tendría un significado bien definido, de manera que pudiera ser interpretada tanto por agentes humanos como por agentes software. Es decir, como vimos en el apartado anterior, dar a la Web interoperabilidad semántica.

- **Web 3.0**. Aplicaciones Web conectándose a aplicaciones Web, a fin de enriquecer la experiencia de las personas, a esto agrega: estado de conciencia del contexto en la Web Geoespacial, autonomía respecto del navegador y construcción de la Web Semántica. No existe un acuerdo unánime a la hora de definir que es la Web 3.0 y cuáles serán los caminos más adecuados para su desarrollo, algunos autores consideran la Web 3.0 y la Web Semántica como sinónimos, pero se pueden diferenciar en el hecho que la Web 3.0 se concibe como un estadio a ser alcanzado en mayor o menor plazo, mientras que la Web Semántica es un proceso evolutivo en construcción permanente.

¿Cómo llegar a la Web Semántica?

Llegar a la Web Semántica no es un salto que se pueda dar de un día para otro, de hecho es algo que ya ha empezado. Actualmente en la mayoría de las páginas ya hay metadatos con información sobre el contenido de éstas. El problema es que no es información estandarizada, ni que pueda ser entendida por todos.

La Web Semántica ha sido impulsada por Tim Berners-Lee, creador de la WWW, y otras personas relacionados con el W3C[8] (World Wide Web Consortium). El primer avance en este sentido, fue la publicación en septiembre de 1998, por parte de Berners-Lee de 2 documentos denominados "*Semantic Web Road Map*" y "*What the Semantic Web can represent*". Según Berners-Lee, la arquitectura de la Web Semántica se podría representar de la siguiente forma:

[8] W3C es un consorcio internacional donde las organizaciones miembro, personal a tiempo completo y el público en general, trabajan conjuntamente para desarrollar estándares Web.

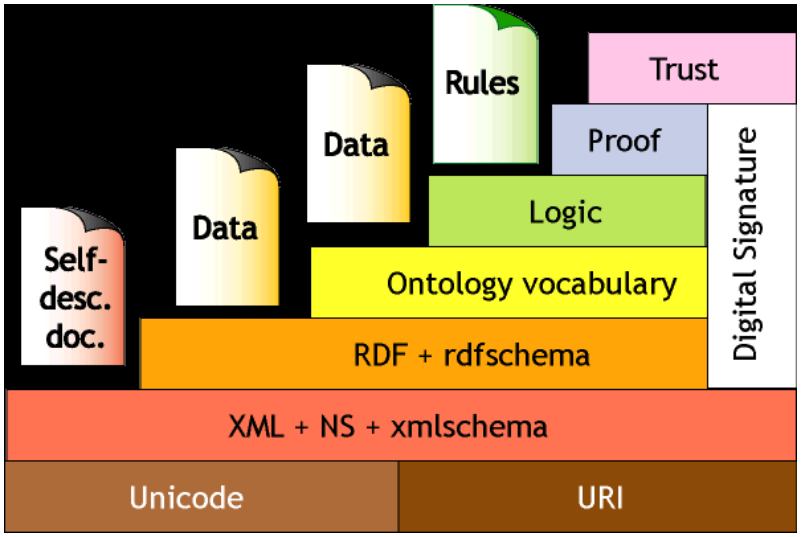

Figura 2.1. Arquitectura de la Web Semántica

El desarrollo de la Web Semántica requiere la utilización de otros lenguajes como el lenguaje estructurado XML y el lenguaje RDF[9] (Resource Description Framework) que puedan dotar a cada página, archivo y recurso o contenido de la red, de una lógica y una semántica que permitan a los ordenadores conocer el significado de la información que manejan, con el fin de que esta información pueda no sólo ser presentada en pantalla, sino también que pueda ser integrada y reutilizada.

XML ha logrado convertirse en un lenguaje estándar. Se trata de un subconjunto del complejo y sofisticado lenguaje SGML que aporta datos estructurados a la Web y que se ha convertido en la infraestructura preferida para el intercambio de datos. Además, las páginas XML pueden ubicar metadatos, esquemas XML y esquemas RDF, que aportan un mecanismo para que los programas puedan interpretar y comprender documentos con un vocabulario descriptivo.

Para poder explotar la Web Semántica, se necesitan lenguajes semánticos más potentes, esto es, lenguajes de marcado capaces de representar el conocimiento basándose en el uso de metadatos y ontologías. Utilizando anotaciones RDF y RDF

[9] RDF es un modelo para representar propiedades designadas y valores de propiedades, se basa en principios perfectamente establecidos de varias comunidades de representación de datos. La propiedades RDF pueden recordar a atributos de recursos y en este sentido corresponden con los tradicionales pares de atributo-valor.

Schema[10] se pueden presentar algunas facetas sobre conceptos de un dominio del conocimiento y se puede, mediante relaciones taxonómicas, crear una jerarquía de conceptos. Pero se precisan lenguajes de marcado (basados en RDF) con una mayor expresividad y capacidad de razonamiento para representar los conocimientos que contienen las ontologías. Además, estos lenguajes deben ser estandarizados y formalizados para que su uso sea universal, reutilizable y compartido a lo largo y ancho de la Web.

Se necesita un lenguaje común basado en Web, con suficiente capacidad expresiva y de razonamiento para representar la semántica de las ontologías. De esta forma, la utilización de lenguajes tales como OWL[11] es un paso más en la consecución de la Web Semántica.

Es necesario, pues, crear una ontología o biblioteca de vocabularios descriptivos/semánticos, definidos en formato RDF y ubicados en la Web para determinar el significado contextual de una palabra por medio de la consulta a la ontología apropiada.

De esta forma, agentes inteligentes y programas autónomos podrían rastrear la Web de forma automática y localizar, exclusivamente, las páginas que se refieran a la palabra buscada con el significado y concepto precisos con el que se interpreta ese término, por lo tanto, para potenciar el uso de ontologías en la Web, se necesitan aplicaciones específicas de búsqueda de ontologías, que indiquen a los usuarios las ontologías existentes y sus características para utilizarlas en su sistema.

[10] RDFSchema es una extensión semántica de RDF. Provee de mecanismos para especificar que clases y propiedades son parte de un vocabulario y de cómo se espera su relación. También permite definir a los recursos como instancias de una o más clases., además permite que las clases puedan ser organizadas en forma jerárquica.

[11] El Lenguaje de Ontologías Web (OWL) está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente representar información para los humanos. OWL facilita un mejor mecanismo de interpretabilidad de contenido Web que los mecanismos admitidos por XML, RDF, y esquema RDF (RDF-S) proporcionando vocabulario adicional junto con una semántica formal.

2.1.3- RDF

El origen de RDF (Resource Description Framework, marco de descripción de recursos) se debe a Ramanathan V.Guha en la especificación REC-rdf-syntax-19990222[12] en el seno del consorcio Web W3C, es un lenguaje para representar información sobre recursos de la Web (metadatos). El objetivo de RDF es especificar la semántica para los datos basados en XML, de manera interoperable y estandarizada. XML impone la necesidad de una restricción estructural para proporcionar métodos inequívocos de expresión semántica. RDF proporciona la infraestructura que permite esa restricción gracias a la codificación, reutilización e intercambio de metadatos estructurados. Con estas prerrogativas, interoperabilidad y estructuración, RDF es el modelo más prometedor para asociar información sobre el contenido de los recursos Web.

RDF es una base para procesar metadatos y proporciona interoperabilidad entre aplicaciones que intercambian información legible por máquinas en la Web. RDF destaca por la facilidad para habilitar el procesamiento automatizado de los recursos Web.

RDF puede utilizarse en distintas áreas de aplicación; por ejemplo: en recuperación de recursos para proporcionar mejores prestaciones a los motores de búsqueda, en catalogación para describir el contenido y las relaciones de contenido disponibles en un sitio Web, una página Web, o una biblioteca digital particular, por los agentes de software inteligentes para facilitar el intercambio y para compartir conocimiento; en la calificación de contenido, en la descripción de colecciones de páginas que representan un "*documento*" lógico individual, para describir los derechos de propiedad intelectual de las páginas Web y para expresar las preferencias de privacidad de un usuario, así como las políticas de privacidad de un sitio Web. RDF

[12] Esta especificación describe cómo usar RDF para describir vocabularios RDF. La especificación define también un vocabulario básico para este propósito, así como un mecanismo de extensibilidad para anticipar futuras extensiones a RDF.

junto con las firmas digitales será la clave para construir una Web segura para el comercio electrónico, la cooperación y otras aplicaciones.

2.1.3.1- Modelo RDF

El modelo de datos básico consiste en tres tipos de objetos:

- **Recursos**. Todo lo descrito por expresiones RDF se denominan recursos. Un recurso puede ser una página Web completa, una colección completa de páginas, un objeto que no sea directamente accesible vía Web, p. e. Un libro impreso. Los recursos se designan siempre por URIs[13] más identificadores de anclas opcionales.

- **Propiedades.** Una propiedad es un aspecto específico, característica, atributo o relación utilizado para describir un recurso. Cada propiedad tiene un significado específico, define sus valores permitidos, los tipos de recursos que puede describir, y sus relaciones con otras propiedades.

- **Sentencias.** Un recurso específico junto con una propiedad denominada, más el valor de dicha propiedad para ese recurso es una sentencia. Estas tres partes individuales de una sentencia se denominan, respectivamente, **sujeto**, **predicado** y **objeto**. El objeto de una sentencia (es decir, el valor de la propiedad) puede ser otro recurso o puede ser un literal; es decir, un recurso (especificado por un URI) o una cadena simple de caracteres (string) u otros tipos de datos primitivos definidos por XML.

En RDF la construcción básica es la tripleta sujeto, predicado y objeto que forman una sentencia, la parte que identifica a qué se refiere la sentencia es el sujeto;

[13] URI Identificador uniforme de recurso, definido en RFC 2396, es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

la parte que identifica las características del sujeto a la que se refiere la sentencia es la propiedad o predicado y la parte que identifica el valor de la propiedad es el objeto.

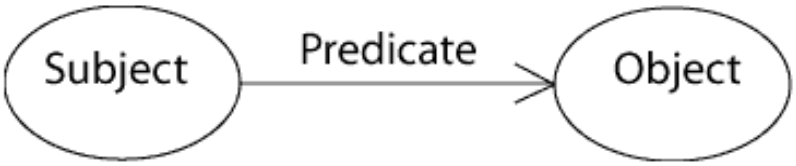

Figura 2.2. Tripleta RDF

Los sujetos, las propiedades y los objetos son recursos. Los recursos con que trabaja RDF no son necesariamente recursos presentes en la Web, pueden ser personas, animales, objetos materiales, números de teléfono, etc. En general, un recurso RDF es cualquier cosa con identidad. Para identificarlos se recurre a los URI (Uniform Resource Identificator, Identificadores Uniformes de Recursos). Estos Localizadores son como unas URL universales, que no se limitan a identificar entidades que tengan localizaciones en la Web o que sean accesibles mediante aplicaciones informáticas.

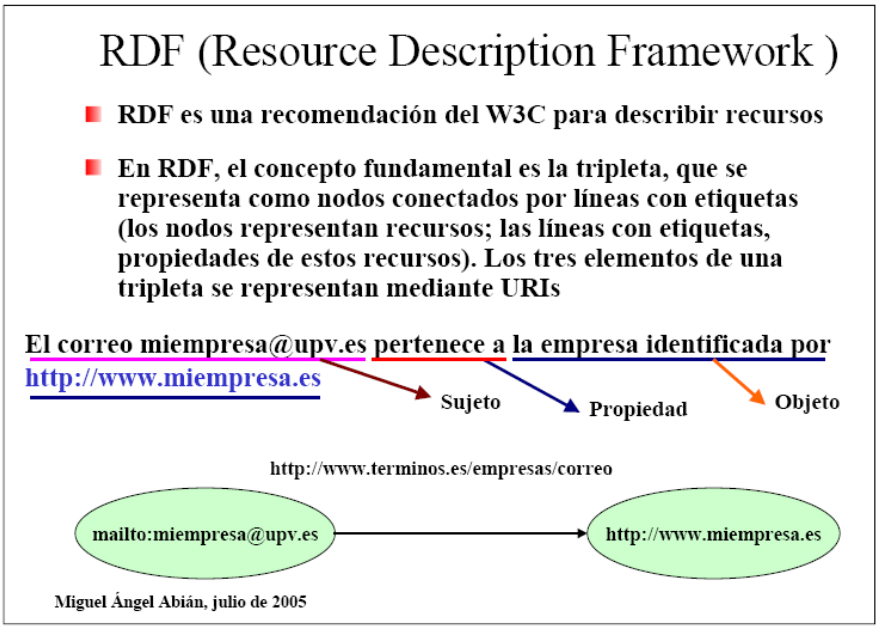

Figura 2.3. Representación gráfica de una tripleta RDF

Cualquier organización o persona puede crear URIs y usarlos para trabajar con sus dominios de interés. Los URIs permiten referirse a un mismo recurso ubicado en distintas localizaciones.

Los URIs no tienen significado por sí mismos, son identificadores, y quien los crea se responsabiliza de darles significado.

Como ejemplo, se puede considerar la siguiente sentencia extraída de la recomendación W3C 22, febrero 1999 :

Ora Lassila es el creador [autor] del recurso http://www.w3.org/Home/Lassila.

Sujeto (Recurso): Http://www.w3.org/Home/Lassila

Predicado (Propiedad): Creador

Objeto (Literal): "*Ora Lassila*"

Se puede representar gráficamente una sentencia RDF usando grafos etiquetados.

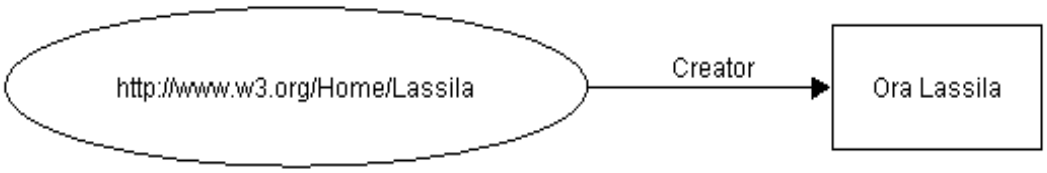

Figura 2.4. Ejemplo de grafo dirigido, empieza en el sujeto y apunta hacia el objeto de la sentencia

Si se quiere decir algo más sobre las características del creador.

El individuo cuyo nombre es Ora Lassila, correo electrónico <lassila@w3.org>, es el creador de http://www.w3.org/Home/Lassila.

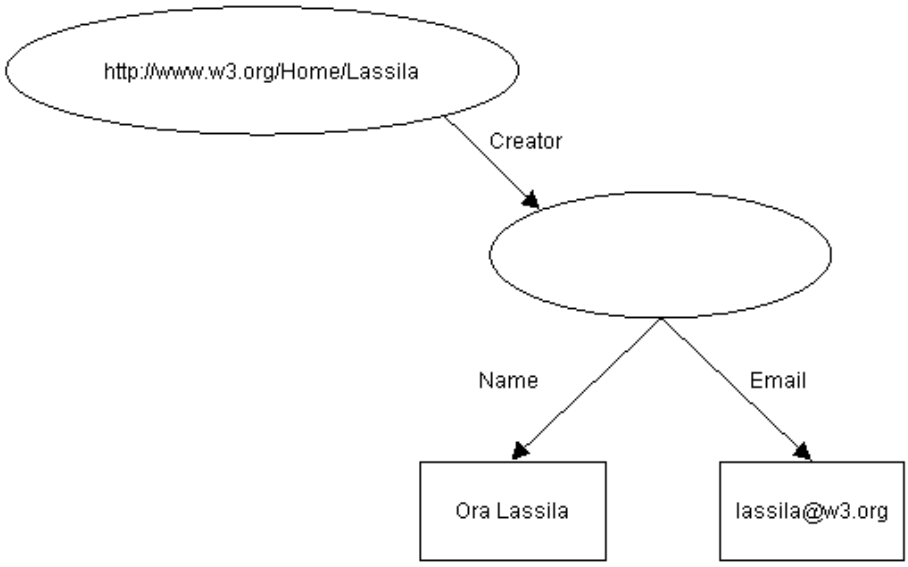

Figura 2.5. Propiedad con valor estructurado

La intención de la frase es precisar el valor de la propiedad Creador una entidad estructurada. En RDF tal entidad se representa como un recurso, la sentencia no proporciona un nombre a ese recurso por lo que en el gráfico se representa como un óvalo vacio.

Si el identificador empleado se utiliza como único identificador para el recurso "*persona*". Los URIs que sirven como clave para cada empleado podrían ser algo como: http://www.w3.org/staffId/85740. Se pueden escribir dos sentencias:

El individuo al que se refiere el identificador de empleado id 85740 se llama Ora Lassila y tiene la dirección de correo lassila@w3.org. Ese individuo creó el recurso http://www.w3.org/Home/Lassila

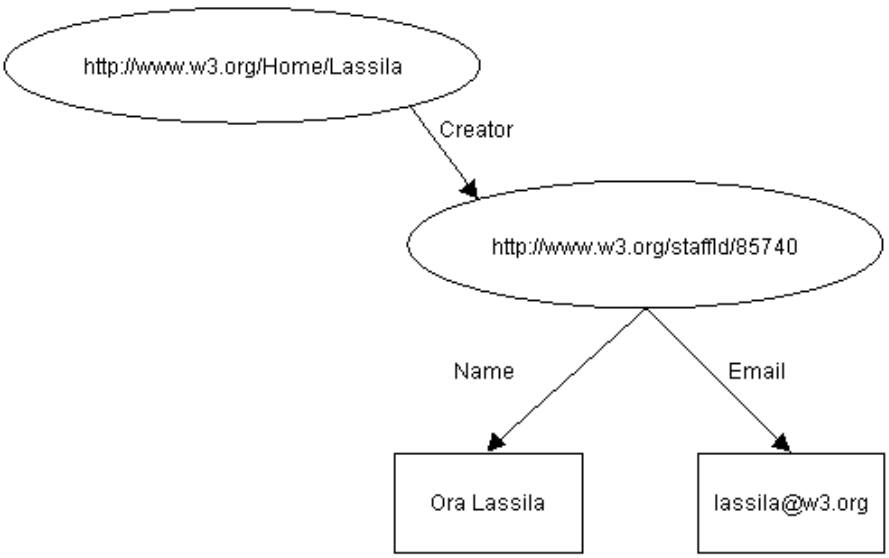

Figura 2.6. Valor estructurado con identificador

2.1.3.2- Sintaxis básica de RDF

El modelo de datos RDF proporciona un marco abstracto y conceptual para definir y utilizar metadatos. Necesita de una sintaxis concreta para crear e intercambiar metadatos. Esta especificación de RDF utiliza XML codificado como su sintaxis de intercambio. RDF necesita también la posibilidad de definir espacios de nombre "*XML namespace facility*" para asociar con precisión cada propiedad con el esquema que define dicha propiedad.

Se pueden distinguir dos tipos de construcciones sintácticas para codificar RDF. Por un lado la serializada, que expresa de una forma regular todas las capacidades de un modelo de datos RDF; y por otro la sintaxis abreviada que incluye construcciones adicionales. Los intérpretes de RDF se han anticipado a implementar ambas sintaxis, la serializada completa y la abreviada. Así, los autores (creadores) de metadatos pueden mezclar ambas libremente

A pesar de todo, el modelo y la sintaxis, no facilitan los mecanismos para definir esas propiedades ni las relaciones entre esos predicados y otros recursos o sujetos; por ello se ha definido también una especificación para definir los esquemas. Un esquema RDF es un conjunto de informaciones relativas a las clases de recursos, y sirve para explicitar las relaciones jerárquicas que establecen entre ellos, o bien para matizar el

carácter obligatorio u opcional de las propiedades y otras restricciones como el número de ocurrencias, etc.

En RDF el significado se expresa a través de un esquema. Se puede pensar en un esquema como en una especie de diccionario. Un esquema define los términos que se utilizarán en una sentencia (declaración) RDF y le otorgará significados específicos. Con RDF se pueden utilizar una gran variedad de formas de esquema, incluyendo una forma específica definida en un documento aparte RDFSchema que tiene algunas características para ayudar a la automatización de tareas usando RDF.

En un esquema se documentan las definiciones y restricciones de uso de las propiedades. Para evitar confusiones entre definiciones independientes (y posiblemente conflictivas) del mismo término. RDF utiliza la facilidad de los espacios de nombres de XML. Los espacios de nombres son una forma de asociar el uso específico de una palabra en el contexto del diccionario (esquema) en que se puede encontrar una definición determinada. En RDF, cada predicado utilizado en una sentencia (declaración) debe ser identificado con un sólo espacio de nombres, o esquema.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs = "http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#"
          xmlns:dc = "http://purl.org/dc/elements/1.0/">
<HEAD>
<TITLE>Resource Description Framework (RDF) Schema
Specification</TITLE>
<STYLE TYPE="text/css"> .EXAMPLE { margin-left: 1em } </STYLE>
<LINK rel="stylesheet" type="text/css" media="screen"
href="/StyleSheets/TR/W3C-PR">

<rdf:RDF>
<rdf:Description about=""

xmlns:ddc="http://purl.org/net/ddc#"
dc:Title="Resource Description Framework (RDF) Schema Specification"
```

```
dc:Description="The Resource Description Framework (RDF) is a
foundation for processing metadata; it provides interoperability
between applications that exchange machine-understandable information
on the Web.  RDF emphasizes facilities to enable automated processing
of Web resources."
dc:Publisher="World Wide Web Consortium"
dc:Date="1999-03-03"
dc:Format="text/html"
dc:Type="technical specification"
dc:Language="en">
<dc:Subject resource="http://purl.org/net/ddc/025.30285"/>
<dc:Subject resource="http://purl.org/net/ddc/025.316"/>
<dc:Subject ddc:Class="025.302855741"
ddc:Heading="Applications of computer file organization and access
methods"/>
<dc:Creator>
       <rdf:Bag rdf:_1="Dan Brickley"
                rdf:_2="R.V. Guha" /></dc:Creator>
<rdfs:seeAlso  rdf:resource="http://www.w3.org/1999/.status/PR-rdf-
schema-
19990303/status"/>

</rdf:Description>
</rdf:RDF>
</HEAD>
```

Código Fuente 2.1. Fragmento de código RDF

En el fragmento de código anterior se puede ver como el código RDF con sintaxis abreviada se encuentra embebido dentro del código HTML. El elemento Description <rdf:description> con el atributo "*about*", se utiliza para identificar (URI/URL) el recurso que se está describiendo que, en este caso, es además implícitamente el propio documento de la propuesta de recomendación del esquema.

Dentro de las etiquetas <rdf:Description>...</rfd:Descripion> se encuentran todas las propiedades (con el prefijo DC, según la declaración previa del espacio de nombres) con sus valores. Dentro de la descripción se declara otro espacio de nombres

(xmlns:ddc="http://purl.org/net/ddc#") que cualificará a su vez el elemento DC:subject, según la Clasificación Decimal de Dewey (DDC).

2.1.3.3- Contenedores

Normalmente es necesario referirse a una colección de recursos, por ejemplo, para expresar que un trabajo se creó por más de una persona, o para enumerar los estudiantes de un curso, o los módulos de un software en un paquete. Los contenedores RDF se usan para mantener tales listas de recursos o literales.

RDF define tres tipos de objetos contenedores:

- **Bag:** Una lista desordenada de recursos o literales. Los Bags se utilizan para indicar que una propiedad tiene múltiples valores y que no es significativo el orden en que se den tales valores

- **Sequence:** Una lista ordenada de recursos o literales. Sequence se usa para manifestar que una propiedad tiene múltiples valores y que el orden de los valores es significativo.

- **Alternative:** Una lista de recursos o literales que representan alternativas para un valor (individual) de una propiedad.

En la siguiente sentencia, extraída de W3C Recommendation 10 February 2004.

Course 6.001 has the students Amy, Mohamed, Johann, Maria, and Phuong

Figura 2.7. Descripción de un contenedor bag

EL valor de la propiedad students es Bag por lo que no es significativo el orden en el que se introduzcan los URIs de los estudiantes.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
         <rdf:Bag>
            <rdf:li rdf:resource="http://example.org/students/Amy"/>
            <rdf:li
rdf:resource="http://example.org/students/Mohamed"/>
            <rdf:li
rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li
rdf:resource="http://example.org/students/Phuong"/>
         </rdf:Bag>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

Código Fuente 2.2. Fragmento de código RDF que representa un contenedor bag

2.1.3.4- Descripción de RDFS

El modelo de datos de RDF, tal y como se especifica en (W3C 27. 2000) define un modelo simple para describir las relaciones entre recursos en términos de propiedades y valores designadas. Las propiedades RDF pueden entenderse como atributos de un recurso y en este sentido corresponden con los tradicionales pares atributo-valor. Las propiedades RDF también representan relaciones entre recursos. De esta forma, el modelo de datos RDF puede parecer un diagrama entidad-relación. El modelo de datos RDF, sin embargo, no proporciona mecanismos para declarar estas propiedades, ni proporcionan ningún mecanismo para definir las relaciones entre estas propiedades y otros recursos. Este es el papel del Esquema RDF.

Cada persona u organización puede definir su propia terminología o vocabulario mediante RDFS, que se define en sentencias RDF. Además, RDFS permite especificar las entidades a las que puede aplicarse los atributos del vocabulario. Al disponer de un esquemas RDF, se puede comprobar si las propiedades aplicadas a los recursos son correctas y si los valores vinculados a las propiedades tienen sentido. RDFS permite controlar la validez de los valores y restringir las entidades a las cuales pueden aplicarse ciertas propiedades.

Pero RDFS tiene algunas carencias:

- No se pueden declarar restricciones de rango que sean validas sólo para algunas clases.
- No se puede reflejar que unas determinadas clases son disjuntas.
- No se pueden representar algunas características de las propiedades.
- No permite expresar restricciones de cardinalidad.
- Existen algunas expresiones cuya semántica no es estándar.

Para solventar estas carencias han surgido distintos lenguajes como OWL

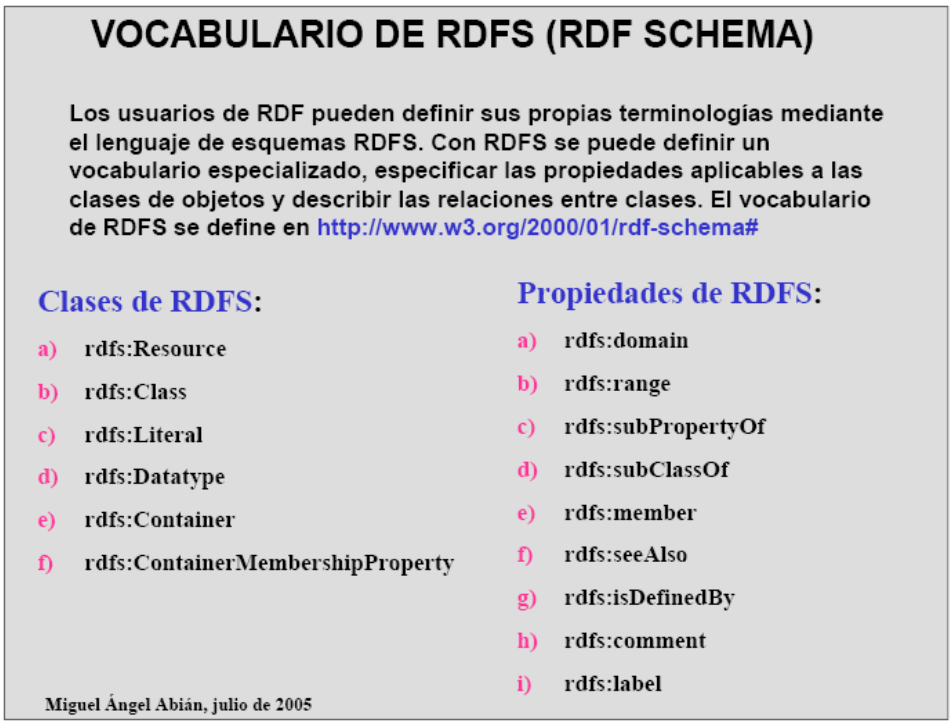

Figura 2.8. Resumen de propiedades y clases de RDFS

2.1.4- Esquemas de metadatos

Metadatos es un término que se refiere a toda aquella información que habla sobre los propios datos, es decir, información que define la propia información. Como ejemplo se puede tomar la ficha técnica de una enciclopedia, en esta ficha aparecerán datos tales como: autores, número de entradas, número de páginas, fecha de edición, etc. Toda esta información describe cómo es otro conjunto de información como puede ser la enciclopedia.

Esta información sobre los datos es lo que se denomina metadatos. Los metadatos permiten recuperar cierta información del tipo: qué, cómo, cuándo, dónde, quién, por qué, para qué. Esta capacidad para definir la semántica de la información abre un horizonte de posibilidades, en cuanto al manejo automático de la información se refiere, y más específicamente para la recuperación de información.

Cuando Internet empezó a crecer, la cantidad de información disponible aumentó desmesuradamente, surgió el problema de clasificarla e identificarla de manera eficiente. Partiendo de ese problema, se comenzó a utilizar los metadatos para facilitar la catalogación y proporcionar la información semántica asociada.

2.1.4.1- Tipos de metadatos

Existen diversos tipos de metadatos, cada uno con su propio formato para describirlos. Según la información que proporcionen, existen metadatos sobre: el contenido, aspectos formales, derechos de autor y el contexto.

Según la función que proporcionan, se pueden clasificar en:

Tipo	objetivo	Ejemplos
Descriptivos	Describen e identifican recursos de información. Permite a los usuarios la búsqueda y recuperación de la información.	Dublin Core o Etiquetas META de HTML
Estructurales	Facilitan la navegación y la presentación de los recursos. Proporcionan información sobre la estructura interna de los documentos, así como la relación entre ellos.	XML y RDF o SGML
Administrativos	Facilitan la gestión de conjuntos de recursos. Incluye la gestión de derechos y sobre control de acceso y uso.	MOA2

Tabla 2.1. Clasificación de los metadatos

2.1.4.2- Descripción de Dublin Core

La iniciativa Dublin Core [DUBLIN. 2009] se creó en 1995 con el propósito de crear estándares que facilitaran la descripción y recuperación de recursos de información, al tratarse de uno de los estándares más conocidos y utilizados se realizará una descripción detallada de sus características y elementos.

Los metadatos Dublin Core se diseñaron para facilitar la recuperación de recursos electrónicos de forma similar a una ficha de catálogo en las bibliotecas intentando establecer en la red los datos necesarios para describir, identificar y encontrar un documento.

Los elementos poseen nombres descriptivos que pretenden transmitir un significado semántico a los mismos. Para promover una interoperabilidad global, una descripción del valor de algunos elementos podrá ser asociada a vocabularios controlados. Se asume que otros vocabularios controlados serán desarrollados para asegurar esta interoperabilidad en dominios específicos.

Cada elemento es opcional y puede repetirse. Además, los elementos pueden aparecer en cualquier orden.

Aunque algunos entornos, como HTML, no diferencian entre mayúsculas y minúsculas, es recomendable escribir correctamente cada metadato según su definición para evitar conflictos con otros entornos, como XML.

Podemos clasificar estos elementos en tres grupos que indican la clase o el ámbito de la información que se guarda en ellos:

1. Elementos relacionados principalmente con el contenido del recurso.

2. Elementos relacionados principalmente con el recurso cuando es visto como una propiedad intelectual.

3. Elementos relacionados principalmente con la instanciación del recurso.

Contenido	Propiedad intelectual	Instanciación
Title	Creator	Date
Subject	Publisher	Type
Description	Contributor	Format
Source	Rights	Identifier

Language
Relation
Coverage

Tabla 2.2. Clasificación de los elementos de Dublin Core

Descripción de los elementos

- *Título*

  Etiqueta: DC.Title

  El nombre dado a un recurso, usualmente por el autor.

- *Autor o Creador*

  Etiqueta: DC.Creator

La persona u organización responsable de la creación del contenido intelectual del recurso. Por ejemplo, los autores en el caso de documentos escritos, artistas, fotógrafos e ilustradores en el caso de recursos visuales.

- *Claves*

  Etiqueta: DC.Subject

Los tópicos del recurso. Típicamente, Subject expresa las claves o frases que describen el título o el contenido del recurso. Se fomenta el uso de vocabularios controlados y de sistemas de clasificación formales.

- *Descripción*

  Etiqueta: DC.Description

Una descripción textual del recurso, un resumen en el caso de un documento o una descripción del contenido en el caso de un documento visual.

- *Editor*

  Etiqueta: DC.Publisher

La entidad responsable de hacer que el recurso se encuentre disponible en la red en su formato actual, por ejemplo la empresa editora, un departamento universitario u otro tipo de organización.

- *Otros Colaboradores*

  Etiqueta: DC.Contributor

Persona u organización que haya tenido una contribución intelectual significativa en la creación del recurso pero cuyas contribuciones son secundarias en comparación a las de las personas u organizaciones especificadas en el elemento Creator.

- *Fecha*

  Etiqueta: DC.Date

Fecha en la que el recurso se puso a disposición del usuario en su forma actual. Esta fecha no ha de confundirse con la que pertenece al elemento Coverage, que sería asociada con el recurso sólo en la medida en que el contenido intelectual está de algún modo relacionado con esa fecha.

- *Tipo del Recurso*

  Etiqueta: DC.Type

Categoría del recurso, por ejemplo página personal, romance, poema, minuta, diccionario. Para asegurar la interoperabilidad, Type debería ser seleccionado de entre una lista de valores que actualmente se encuentra bajo desarrollo en un grupo de trabajo.

- *Formato*

  Etiqueta: DC.Format

Formato de datos de un recurso, usado para identificar el software y posiblemente, el hardware que se necesitaría para mostrar el recurso. Para asegurar la interoperabilidad, los valores de Format deben ser seleccionados de entre una lista de valores que actualmente se encuentra bajo desarrollo en un grupo de trabajo.

- *Identificador del Recurso*

  Etiqueta: DC.Identifier

Secuencia de caracteres usados para identificar unívocamente un recurso. Para recursos en línea pueden ser URLs y URNs (cuando estén implementados). Para otros recursos pueden ser usados otros formatos de identificadores, como por ejemplo ISBN ("*International Standard Book Number*" - Número Internacional Normalizado para Libros)

- *Fuente*

  Etiqueta: DC.Source

Secuencia de caracteres utilizado para identificar unívocamente un trabajo a partir del cual proviene el recurso actual. Por ejemplo, es posible usar Source con la fecha de 1603 como descripción de una película basada en una obra de Shakespeare, pero es preferible, en ese caso, usar Relation "*IsBasedOn*" con una referencia a un recurso distinto cuya descripción contenga el elemento Date con valor 1603.

- *Lengua*

  Etiqueta: DC.Language

Lengua/s del contenido intelectual del recurso. El contenido de este campo debería coincidir con los de la RFC.

- *Relación*

  Etiqueta: DC.Relation

Un identificador de un segundo recurso y su relación con el recurso actual. Este elemento permite enlazar los recursos relacionados y las descripciones de los recursos.

Para asegurar la interoperabilidad, las relaciones deberían ser seleccionadas de una lista de elementos que actualmente se encuentra bajo desarrollo en un grupo de trabajo.

- *Cobertura*

  Etiqueta: DC.Coverage

La característica de cobertura espacial y/o temporal del contenido intelectual del recurso.

- *Derechos*

  Etiqueta: DC.Rights

Una referencia (URL, por ejemplo) para una nota sobre derechos de autor, para un servicio de gestión de derechos o para un servicio que dará información sobre términos y condiciones de acceso a un recurso. Una especificación formal del elemento Rights se encuentra actualmente en discusión y por lo tanto su uso se considera experimental.

2.1.4.3- Descripción de FOAF

FOAF (Friend Of A Friend) es una aplicación basada en XML, RDF y OWL, tecnologías recomendadas por el W3C. (LEANDRO, 2002) explica que se trata de un archivo XML con el cual se describen personas, documentos o cualquier otra cosa. En el archivo denominado FOAF, se indicará a qué personas se conoce, quiénes son amigos y quiénes no, qué grado de interés se tiene por unos temas u otros, etc. Casi todo puede ser definido.

Según (DODDS, 2002) FOAF es una manera de describirse a uno mismo, nombre, dirección de email, y la gente de quien se es amigo, usando XML y RDF. Esto permite al software procesar estas descripciones, tal vez como parte de un motor de búsqueda automatizado, para descubrir información acerca suyo y de las comunidades

de las cuales es miembro. FOAF tiene el potencial de conducir a nuevos desarrollos en comunidades online

2.1.4.4- Descripción de vCard

vCard es un formato estándar para el intercambio de información personal, específicamente tarjetas personales electrónicas (electronic business cards). Las vCards son usualmente adjuntadas a mensajes de e-mail, pero pueden ser intercambiadas en muchas otras formas, como en la World Wide Web. Pueden contener nombre, dirección, números telefónicos, URLs, logos, fotografías, e incluso clips de audio.

2.1.4.5- Descripción de DOAC

DOAC (Description Of A Career) es una extensión del vocabulario FOAF, creada por Ramón A. Parada, que permite compartir un curriculum de forma que pueda ser procesado por cualquier aplicación. Ha sido diseñado para ser compatible con el currículum europeo (Europass) así que este puede ser generado a partir de un archivo FOAF+DOAC. Incluye información sobre educación, experiencia laboral, publicaciones, idiomas y otras habilidades.

Los empresarios serán capaces de buscar en internet para encontrar a un trabajador que se adapte a sus necesidades y será tan fácil como cualquier búsqueda en Google.

2.1.4.6- Descripción de DOAP

DOAP (Description Of A Project) es un proyecto para crear un vocabulario XML RDF que permita describir los proyectos de software, particularmente de código abierto. Además de desarrollar un esquema RDF y ejemplos, el proyecto DOAP tiene como objetivo proporcionar las herramientas de apoyo en todos los lenguajes de programación (PROYECTO DOAP, 2010).

2.1.5- 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 (OGDEN, 1923). "*El triangulo del significado*", que se muestra en la Figura 2.9. "*Triángulo del Significado*", define la interacción entre símbolos o palabras, pensamientos y cosas del mundo real.

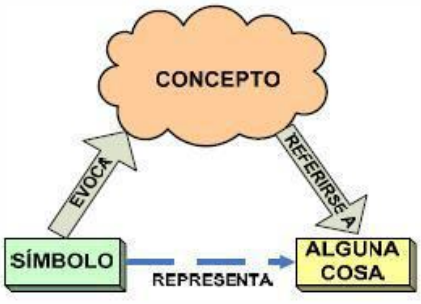

Figura 2.9. Triángulo del Significado

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, 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, Gruber (GRUBER, 1993) dio una de las definiciones más empleadas: *"las ontologías se definen como una especificación explícita de una conceptualización"*

En 1998, Studer (STUDER, 1998) modificó 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.

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 Jasper y Uschold (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 2.10. Ejemplo de uso de ontologías

2.1.5.1- Utilidad de las ontologías

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).

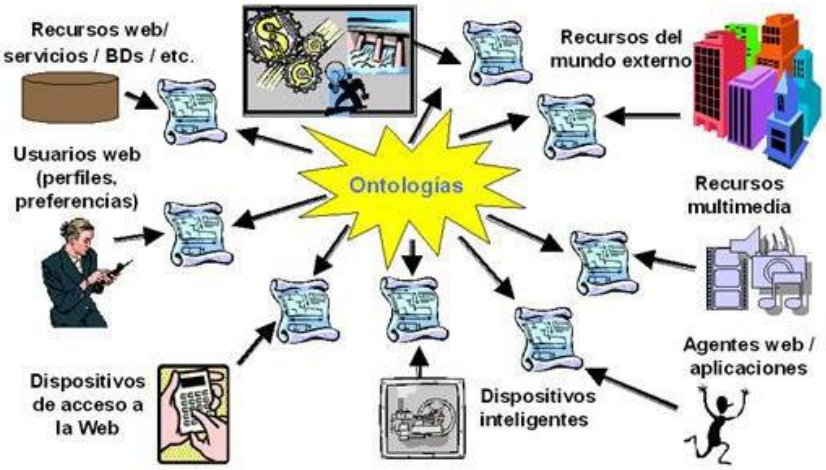

Figura 2.11. Ejemplos de posibles usos de las ontologías

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.

2.1.5.2- Lenguajes de representación

Existen diversos lenguajes que pueden ser utilizados para representar ontologías. Los más utilizados en el entorno de la Web son RDFS y OWL

A pesar de que RDFS es un modelo de representación de metadatos, puede usarse como lenguaje general para la representación del conocimiento. OWL (Web Ontology Languaje) es una ampliación de RDFS desarrollada por W3C que emplea el modelo de tripletas de RFS. Posee una sintaxis abstracta basada en XML y otra en RDF. Permite la utilización de propiedades para:

- **Restringir las instancias de una clase:**

  - ***unionOf*** (expresa uniones de clases)

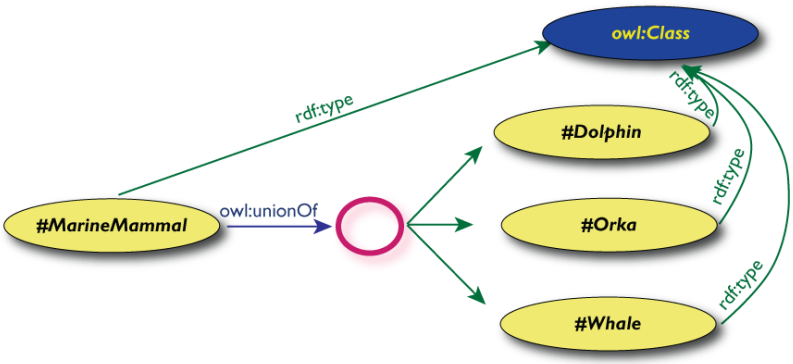

Figura 2.12. Ejemplo de uso de unionOf

- ***oneOf*** (expresa uniones de clases)

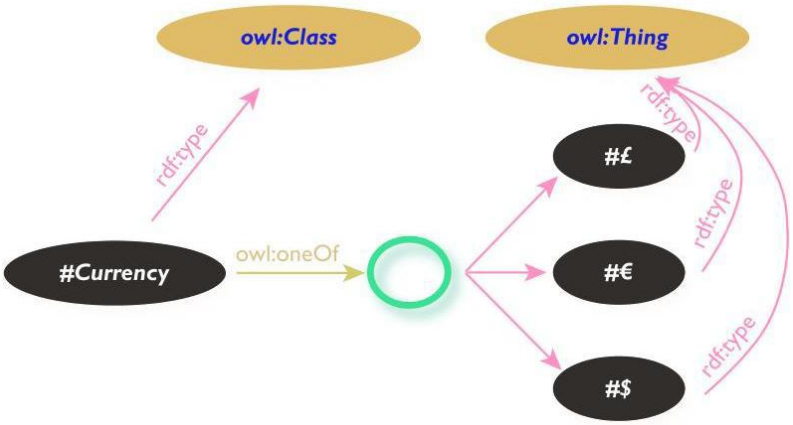

Figura 2.13. Ejemplo de uso de oneOf

- **Restringir las los valores de las propiedades:**
  - ***rdfs:domain*** (restringe valores de las propiedades)
  - ***rdfs:range*** (restringe valores de las propiedades)
  - ***TransitiveProperty*** (propiedad transitiva)
  - ***inverseOf*** (propiedad inversa de otra)
  - ***minCardinality*** (número mínimo de elementos de una relación)
  - ***maxCardinality*** (número máximo de elementos de una relación)

Además, OWL posee propiedades para facilitar la importación y exportación de clases, permitiendo por ejemplo detectar si dos entidades son iguales, algo muy útil cuando se mezclan ontologías. Algunas de estas propiedades son:

- **sameAs**
- **equivalentClass**
- **equivalentProperty**
- **differentFrom**

Existen varios editores gratuitos como pueden ser Protegé, OILed y ORIENT. Todos excepto OILed se basan en Java y ORIENT es un *plug-in* de Eclipse.

.

2.1.5.3- Editores de ontologías

En el momento actual existe una gran variedad de editores de ontologías diferentes, en este apartado se describirán las características más representativas de algunos de los editores más empleados.

LinkFactory

LinkFactory es una herramienta que ofrece una enorme variedad de funcionalidades usadas para construir sistemas comprensivos de terminología corporativa, capaz de extraer significado de grandes almacenes de datos desestructurados, escondidos en documentos y bases de datos de una compañía. LinkFactory se ha diseñado específicamente para construir ontologías que superen los millones de conceptos.

Además, LinkFactory permite crear, mantener y alcanzar sistemas de terminología multilingüe y ontologías (inglés, español, francés, etc.). Esto se puede aplicar a dominios no médicos y ha sido probada con éxito en aplicaciones de Defensa, de Cumplimiento de Leyes, etc. También incluye varias herramientas que reducen la carga de trabajo del ingeniero de conocimiento.

Las principales funcionalidades que implementa son las siguientes:

- Modelado de conceptos con todas relaciones y definiciones relevantes.
- Mapeo contra sistemas de codificación externos.
- Control de versiones para asegurar que las referencias pueden hacerse sobre versiones más antiguas de objetos, sin pérdida de información.

- Representación multilingüe de los conceptos de lenguaje independiente, para que sepuedan almacenar conceptos con varias representaciones superficiales en cada uno de los idiomas.

- Arquitectura multiusuario, que asegure la integridad de la ontología mientras se esté editando con diferentes modeladores.

NeOn Toolkit

NeOn Toolkit es un entorno extensible de Ingeniería Ontológica. Es parte de la implementación de referencia de la arquitectura NeOn. Contiene plugins para gestión y visualización de ontologías. Las principales funcionalidades que incluye son:

- Edición básica: Esquema de edición.
- Visualización / Navegación.
- Importación / Exportación: FLogic, (subconjuntos de) RDF (S) y OWL.

Existen varios plugins comerciales que extienden la herramienta en algunas funcionalidades, incluyendo:

- Soporte de reglas: Edición gráfica/textual, debugging.

- Mediación: Editor de mapeo gráfico, interpretación de mapeos.

- Integración de la base de datos: Importación de esquemas de bases de datos, acceso a base de datos.

- Consultas: Editor de consultas y consultas persistentes.

OntoStudio

OntoStudio es un entorno de desarrollo profesional para soluciones basadas en ontologías, sucesor de OntoEdit (anterior nombre de la herramienta).

- Combina herramientas de modelado para ontologías y reglas, con componentes para la integración de fuentes de datos heterogéneas.

- Se puede enriquecer con módulos de autodesarrollo y se puede configurar bajo las necesidades del usuario.

- Soporta lenguajes de ontologías estándar de la W3C: OOXML, OWL, RDF, RDFS, FLogic, NTriples y N3.

Altova SemanticWorks

Altova SemanticWorks 2008 es un editor para Web Semántica de RDF/OWL. El lenguaje en el que se ha implementado este editor de ontologías es Java. Gráficamente diseña documentos de instancia RDF, vocabularios RDFS y ontologías OWL. Luego los imprime tanto en RDF/XML como en formato NTriples. SemanticWorks facilita el trabajo con etiquetas para instancias, propiedades, clases, gestión de integridad y otras funcionalidades que se definen a continuación.

Altova creó esta herramienta para ayudar a sus clientes a aprender y trabajar con las nuevas tecnologías de Web Semántica de una forma intuitiva. Mientras se está trabajando, se puede cambiar de la vista de diseño gráfico a la vista textual para visualizar el correspondiente, autogenerado, código RDF/XML o NTriples. También se puede exportar de RDF/XML a NTriples o viceversa en cualquier momento.

COE (CmapTools Ontology Editor)

COE es una versión especializada del CmapTools de IHMC (Institute for Human and Machine Cognition). COE está basado en la idea de mapas conceptuales. Un mapa conceptual es un diagrama gráfico que muestra las relaciones entre conceptos. Los

conceptos están conectados con flechas etiquetadas con las relaciones, generando una estructura jerárquica. COE es un conjunto de herramientas software integrado para construir, compartir y visualizar ontologías OWL.

En estos momentos, se está focalizando el trabajo en el desarrollo de estándares para construir nuevas ontologías Cmap en OWL, que ayuden y guíen a los usuarios en el proceso de creación de clases y relaciones de propiedades entre los conceptos de una ontología. Además, se está trabajando en la agrupación y búsqueda de técnicas que soporten la reutilización de ontologías ya existentes.

COE está implementado en el lenguaje Java y está disponible vía Web. Además, se puede utilizar COE como:

- Herramienta para visualizar una ontología.
- Editor de ontologías.
- Motor de búsqueda para conceptos.

OILEd

OilEd es un editor de ontologías que permite a sus usuarios construir ontologías usando DAML-OIL[14]. OilEd está disponible vía Web.

La primera intención que perseguía OilEd era la de proporcionar un editor sencillo que demostrara el uso y estimulara el interés en el lenguaje OIL. Las actuales versiones de OilEd no ofrecen un desarrollo completo del entorno de ontologías: permite de manera activa el soporte para el desarrollo de ontologías de gran escala, la migración e integración de ontologías, control de versiones, argumentación y muchas otras actividades que están involucradas en la construcción de ontologías. Sin embargo, este es el "*NotePad*" de los editores de ontologías, ofreciendo las funcionalidades suficientes para permitir a los usuarios construir ontologías y

[14] DAML-OIL procede de la unificación de los lenguajes DAML y OIL. En el desarrollo de este lenguaje participaron tanto el grupo de trabajo de DARPA (Defense Advanced Research Projects Agency) como el grupo de trabajo que desarrolló el lenguaje OIL (Ontology Inference Layer).

demostrar cómo se puede usar el razonador FaCT para chequear la consistencia de las ontologías.

Es uno de los editores más usados por empresas e instituciones con propósitos tanto de investigación como de enseñanza, debido a su sencillez y claridad.

Apollo CH

Apollo CH es un editor de ontologías que utiliza su propio modelo de conocimiento. Dicho modelo de conocimiento está basado en marcos y es independiente de cualquier lenguaje de modelado de conocimiento. El uso de plugins de entrada y salida permite tratar con cualquiera de los lenguajes de modelado de conocimiento particulares.

Apollo CH está equipado con funcionalidades adicionales categorizadas en los siguientes tres apartados:

- Funcionalidades dirigidas hacia la *mejora de la eficiencia* del trabajo de los usuarios cuando se editan grandes ontologías/bases de conocimiento. Entre estas funcionalidades se incluyen nuevas formas de navegación en grandes ontologías/bases de conocimiento.

- Funcionalidades que buscan permitir al usuario *consolidar ontologías en un nivel alto de abstracción*. La metodología para la creación colaborativa de bases de conocimiento espera que el usuario utilice herramientas de un nivel bastante bajo para comparar ontologías e integrar la contribución de los usuarios con la base de conocimiento compartida consolidada. Apollo CH contiene módulos que permiten al usuario llevar a cabo esta tarea tan complicada en un nivel superior de abstracción de una manera más sencilla, que ofrece más confianza y más eficiencia.

- Funcionalidades dirigidas hacia el desarrollo de una base de conocimiento robusta, aseguran que la base de conocimiento producida por Apollo CH reúne las restricciones necesarias.

Apollo CH se ha desarrollado en el Laboratorio Gerstner de la Universidad Técnica Checa (Czech Technical University), en Praga, República Checa. Está basada en la herramienta de modelado de conocimiento Apollo, que fue desarrollada en KMI (Knowledge Media Institute).

Protégé

Protégé es una plataforma gratuita, de código abierto que ofrece un conjunto de herramientas para construir modelos de dominio y aplicaciones basadas en conocimiento con ontologías, al tratarse del editor de ontologías empleado en el desarrollo del proyecto se ampliará esta descripción en el Apartado 3.4.3. *"Protégé"*.

Protégé implementa un conjunto de estructuras de modelado de conocimiento y acciones que soportan la creación, visualización, y manipulación de ontologías en varios formatos de representación. Protégé puede ser configurado para ofrecer soporte para creación de modelos de conocimiento y datos. También se puede extender Protégé a través de un plugin de arquitectura y una API[15] Java para construir herramientas y aplicaciones basadas en conocimiento.

Protégé proporciona tres componentes que se pueden descargar de manera conjunta o por separado:

- El editor ProtégéFrames.
- El editor ProtégéOWL.
- Varios plugins opcionales.

La plataforma Protégé ofrece dos formas para modelar ontologías:

[15] Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Editor ProtégéFrames:

Permite a los usuarios construir y poblar ontologías que están basadas en *frames*, en concordancia con el protocolo OKBC[16] (Open Knowledge Base Connectivity). En este modelo, una ontología consiste en un conjunto de clases organizadas en una jerarquía para representar los conceptos de un dominio, un conjunto de slots asociados con clases para describir sus propiedades y relaciones, y un conjunto de instancias de esas clases.

Editor ProtégéOWL:

Permite a los usuarios construir ontologías para la Web Semántica, en particular en el Lenguaje de Ontologías Web (OWL) del W3C. Una ontología OWL puede incluir descripciones de clases, propiedades y sus instancias. Dada una ontología, las semánticas formales de OWL especifican como derivar sus consecuencias lógicas, es decir, hechos que no están literalmente presentes en la ontología, pero supuestos por las semánticas. Estas suposiciones pueden estar basadas en un solo documento o múltiples documentos distribuidos que se han combinado usando mecanismos definidos de OWL.

2.1.6- PROTON

PROTON [TERZIEV, I.2005] es un desarrollo de la ontología KIMO, que fue creada y utilizada en el ámbito de aplicación de la plataforma KIM para anotación, indexación y recuperación semántica.

La ontología PROTON contiene alrededor de 300 clases y 100 propiedades, proporcionando la cobertura de los conceptos generales necesarios para una amplia gama de tareas, incluida la anotación, la indización y recuperación semántica de los documentos. Los principios de diseño pueden ser resumidos como sigue:

[16] OKBC es una interfaz de programación de aplicaciones para acceder a bases de conocimientos almacenados en los sistemas de representación del conocimiento

1. Independencia de dominio.
2. Definiciones lógicas *"Ligth-weight".*
3. Alineación con los estándares más populares.
4. Buena cobertura de llamada de entidades y dominios concretos (por ejemplo, las personas, organizaciones, lugares, números, fechas, direcciones).

PROTON no proporciona apoyo específico a los conceptos generales, tales como "*Manzana*", "*amor*", etc. Sin embargo, el nivel superior de la ontología está diseñado para permitir una fácil extensión en esta dirección.

Con el fin de satisfacer las necesidades de los escenarios de uso y para garantizar una comprensión fácil y gradual, PROTON se divide en cuatro módulos:

- **SYSTEM module.** Contiene primitivas de nivel (6 clases y 7 propiedades). Introduce el concepto de entidad. En general, el System module de PROTON se hace referencia a través del prefijo "protons:".

- **TOP module.** El más alto, más general, el nivel conceptual, que consta de unas 20 clases que aportan un buen equilibrio de utilidad, independencia de domino y facilidad de comprensión y uso. La capa superior es generalmente el mejor nivel para establecer la adaptación a las ontologías y otros esquemas. El *Top module* de PROTON se hace referencia a través del prefijo "protont:".

- **UPPER module.** Más de 200 clases generales de las entidades, que a menudo aparecen en varios dominios. el Upper module de PROTON se hace referencia a través del prefijo "protonu:".

- **KM (Knowledge Management) module.** Contiene 38 clases de entidades especializadas que son específicas para las típicas tareas de Gestión del Conocimiento y de aplicaciones. El módulo de PROTON KM se hace referencia a través del prefijo "protonkm:".

2.1.6.1- Arquitectura de PROTON

PROTON está organizado en tres niveles (incluyendo cuatro módulos), como se muestra en la Figura. 2.14. "*Arquitectura de PROTON*" El System module ocupa el primer lugar, la capa de base. Luego el Top module, Upper module y KM module se actualizan en la parte superior de la misma forma modular. Permite una fácil integración, ampliación y remodelación.

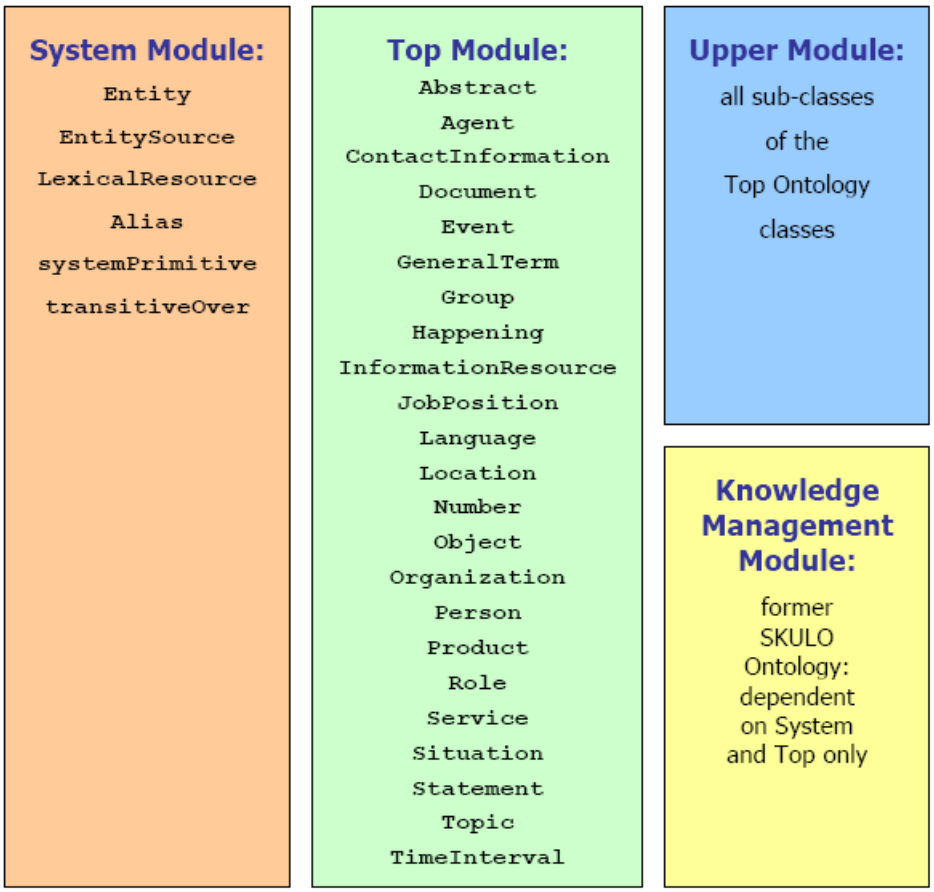

Figura 2.14. Arquitectura de PROTON

Cobertura del System module de PROTON

El *System module* de PROTON proporciona un sistema de alto nivel o metaprimitivas. Es el único componente de PROTON que no se va a cambiar a los efectos de la ontología de extensión. El System module de PROTON incluye las siguientes clases: protons:LexicalResource;protons:Alias; protons:EntitySource; protons:Entity.

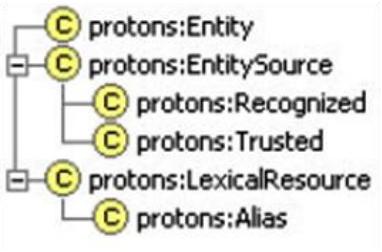

Figura 2.15. Jerarquía de clases del System module de PROTON

La rama protons:Entity representa la raíz de la "*verdadera*" ontología. Las instancias de la súper-clase protons:EntitySource se emplean para separar la información de confianza en el KB. La súper-clase protons:LexicalResource se emplea principalmente a la codificación de los distintos datos, relacionados con la IE y los procesos de minería de datos. Protons:Alias es una importante clase dentro de la rama, representa los nombres de la instancia de la clase protons:Entity.

Cobertura del Top module de PROTON

El Top module de PROTON representa las clases más generales.

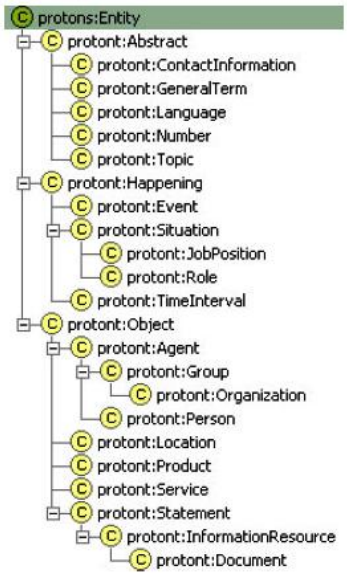

Figura 2.16. Clases del Top module de PROTON

Cobertura del Upper module de PROTON

El Upper module de PROTON es una extensión del Top module. En términos generales, las sub-clases de las ramas de las clases del Top module, así como las propiedades correspondientes y los axiomas.

Convenciones de la nomenclatura

Las convenciones de la nomenclatura en PROTON, concernientes a las clases, las relaciones, y los atributos, son los siguientes:

La etiqueta de una clase se compone de una o más palabras, escritas con la primera letra en mayúscula, y sin ningún tipo de intervalos o símbolos alfanuméricos entre ellas, (por ejemplo la instancia protont:InformationResource). Las etiquetas de las relaciones y los atributos siguen la misma regla, a excepción de que la primera letra que no es mayúscula de la relación / atributo, (por ejemplo protont:locatedIn).

2.1.6.2- Definiciones del *Top module* y ramas del *Upper module*

El *Top module* de PROTON contiene una serie de clases generales, que serán utilizadas en tres casos de uso, el descubrimiento de conocimiento, la generación de metadatos, y herramientas de acceso inteligente del conocimiento. Estas clases de nivel superior representan las nociones más comunes, básicas y globales del conocimiento mundial.

Rama Object

Un objeto puede desempeñar un papel en algunos acontecimientos. Los objetos podrían ser sustancialmente reales o casi imperceptibles. Las propiedades de las relaciones y los atributos de protont:Object son:

- **protont:hasContactInfo**: Esta propiedad permite que las relaciones entre protont:Object and protont:ContactInformation.

- **protont:isOwnedBy:** Esta propiedad relaciona una organización en particular con los agentes que son miembros de esa organización. Este predicado indica un tipo genérico de afiliación, aunque puede haber tipos especializados de pertenencia a la misma organización.

El protont:Object es una súper-clase con las siguientes clases Top module de PROTON:

- **protont:Agent**
- **protont:Person**
- **protont:Group**
- **protont:Organization**
- **protont:Location**
- **protont:Statement**
- **protont:InformationResource**
- **protont:Document**
- **protont:Product**
- **protont:Service**

Rama Happening

protont:Happening es algo que sucede. Puede ser dinámico o estático. En todos los casos, un acontecimiento tiene un cierto posicionamiento temporal.

Generalmente es el caso en el que algunas entidades toman parte en el acontecimiento, es decir, están involucrados, o desempeñar un cierto papel en el.

El protont:Happenig es una súper-clase con las siguientes clases Top module de PROTON:

- **protont:Event**
- **protont:Situation**
- **protont:TimeInterval**

- **protont:JobPosition**

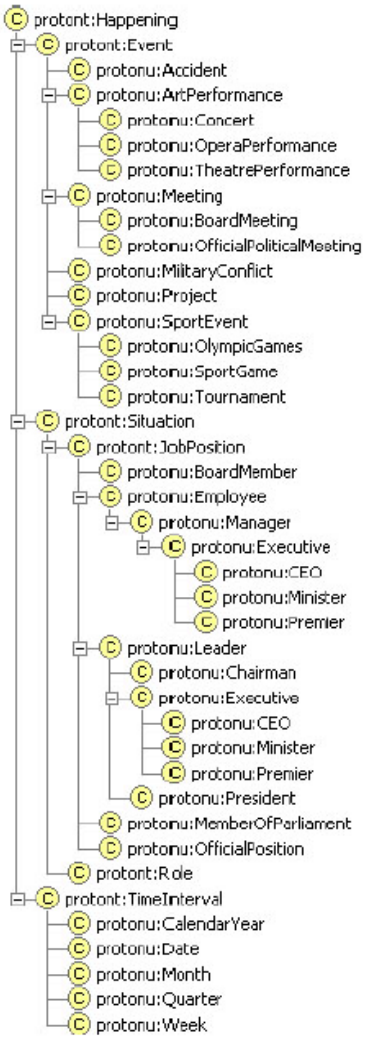

Figura 2.17. Clases Happening del Top module de PROTON y las ramas del Upper

Rama Abstract

protont: Abstract denota una abstracción: es decir, algo, que no sucede, ni existe. En algunos casos, cuando una clase tiene datos concretos y bien definidos, que sin embargo no pueden ser modelados como sub-clases, los casos están incluidos en la base de conocimientos.

El protont:Abstract es una súper-clase con las siguientes clases Top module de PROTON:

- **protont:Number**
- **protont:ContactInformation**
- **protont:Language**
- **protont:Topic**

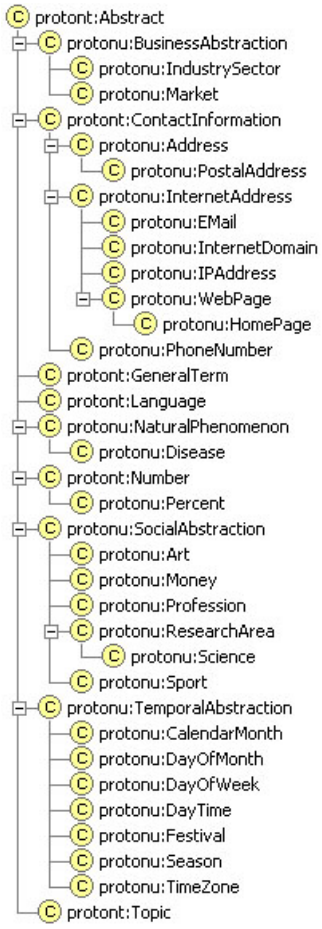

Figura 2.18. Jerarquía de clases de protont:Abstract

2.1.6.3- Clases del Knowledge Management module de PROTON

EL Knowledge Management (KM) module de PROTON es independiente de los System modules y de los Top modules. En realidad, este módulo es la antigua ontología SKULO (SEKT Knowledge Management Upper Level Ontology), que se ha desarrollado e integrado en PROTON.

Clases del KM module de PROTON:

- **protonkm:InformationSpace**

- **protonkm:SoftwareAgent**
- **protonkm:User**
- **protonkm:Profile**
- **protonkm:InformationSpaceProfile**
- **protonkm:UserProfile**
- **protonkm:Mention**
- **protonkm:WeightedTerm**
- **protonkm:Device**

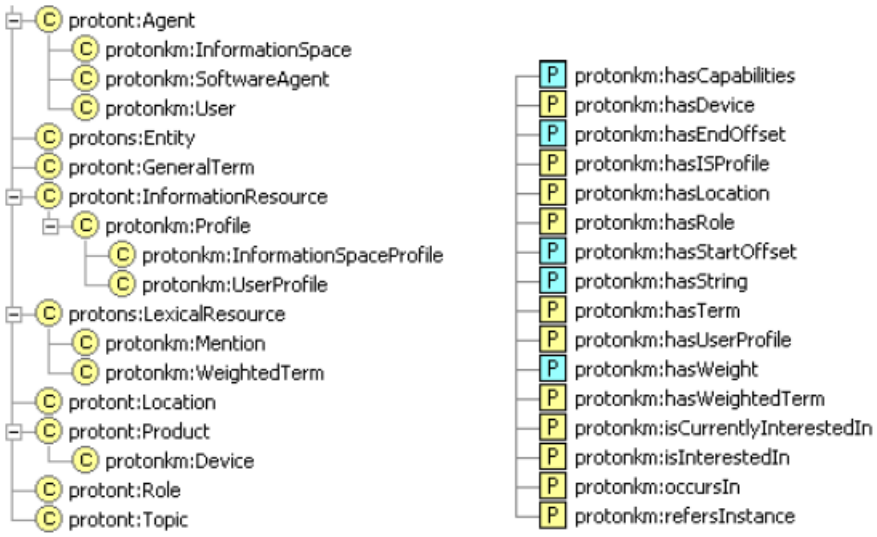

Figura 2.19. Clases y propiedades del Knowledge Management Module de PROTON

Este módulo no se ha empleado en el desarrollo del proyecto, la ampliación de la ontología se ha realizado de una forma similar, mediante la generación de un módulo que amplia y redefine conceptos en PROTON. No se entrará más en detalle en este punto pues este desarrollo queda fuera del ámbito del PFC.

2.1.6.4- Relaciones con otros estándares

En el proceso de desarrollo de PROTON se han consultado los estándares más importantes de ontologías y metadatos. Como resultado, conceptos importantes han sido importados directamente desde ellos.

Muchos de los conceptos de PROTON se han comentado de manera informal a través de descripciones.

Dublin Core

La elaboración de modelos de documentos y otros tipos de recursos de información en PROTON sigue Dublin Core, proporcionando asignación directa de todos sus elementos. Sin embargo, algunos de los nombres de los elementos o tipos son alterados por dos razones:

- Para que coincida con la nomenclatura de PROTON.

- Para evitar la ambigüedad con las nociones que faltan en DC, pero están presentes en PROTON.

Wordnet

Aunque Wordnet no es una verdadera ontología, ha sido consultada en el desarrollo de PROTON y se refiere en muchas de las descripciones.

Librerías de Alexandria Digital y GNS

La rama proton:Location de PROTON contiene cerca de 80 clases alienadas con ADL , que a su vez está alienada con designadores de las características geográficas, de la base de datos GNS de NIMA.

OpenCyc

OpenCyc ha sido consultado durante la elaboración de PROTON y se hace referencia en muchas de las descripciones. A continuación se hace un breve resumen de las diferencias entre OpenCyc y PROTON:

- OpenCyc proporciona muchas definiciones lógicas de sus elementos.

- OpenCyc tiene una cobertura mucho más amplia. Sin embargo, hay elementos en PROTON que faltan en OpenCyc.

- OpenCyc está tratando de proveer de modelización formal de muchas de las nociones generales, tales como el espacio y el tiempo. PROTON no tiene una cobertura de los conceptos generales

En resumen, OpenCyc es lógicamente más amplio, lo que lo hace adecuado para ingeniería del conocimiento "*de peso pesado*". Sin embargo, hace que las clases de la ontología sean más complejas y abstractas, lo que hace que no sea idóneo para tareas de Knowledge Management.

2.2- 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.

2.2.1- Elementos clave de sitios Web centrados en usuario

Dado que la aplicación que se ha desarrollado está claramente orientada al usuario se ha intentado 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 Microsoft Internet Explorer 7 y 8. El cliente correrá en dicho navegador, y el servidor será una aplicación a desplegar en un entorno Java, por lo que será multiplataforma.

2.2.2- 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.

(ALARCÓN, 2007) 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.

2.2.2.1- Diagramas UML

Un diagrama (HERNANDEZ, 2000) 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.

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 2.20. *"Ejemplo de diagrama de casos de uso"*, se muestra un ejemplo.

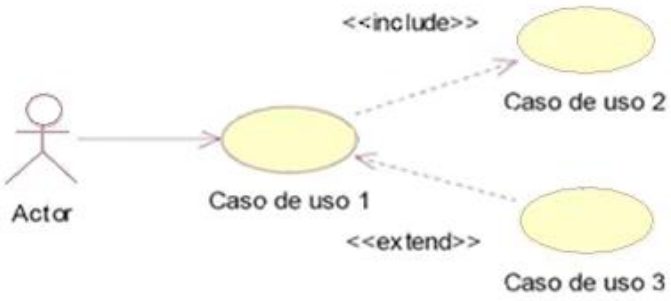

Figura 2.20. Ejemplo de diagrama de casos de uso

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.

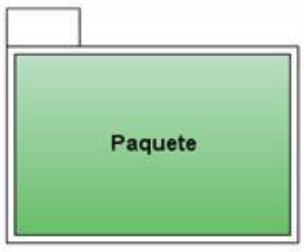

Figura 2.21. Figura de un paquete

- **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.

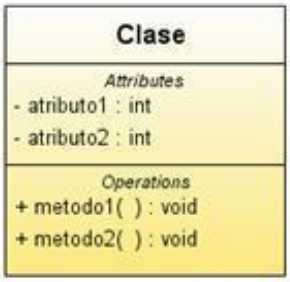

Figura 2.22. Figura de una clase

- **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.

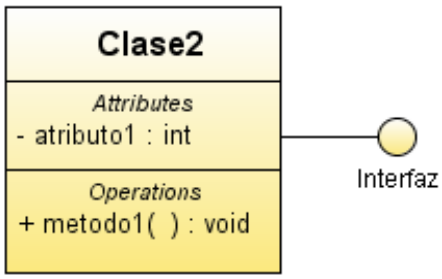

Figura 2.23. Figura de una interfaz

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 2.24. *"Ejemplo de diagrama de clases"*.

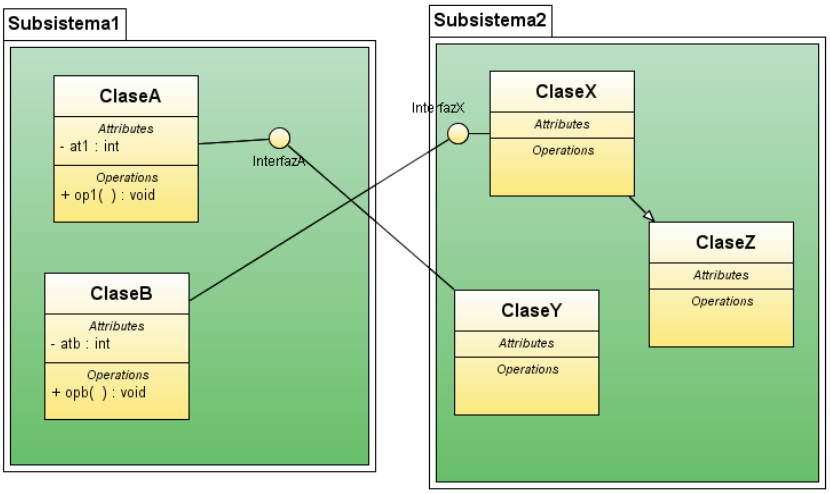

Figura 2.24. Ejemplo de diagrama de clases

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.

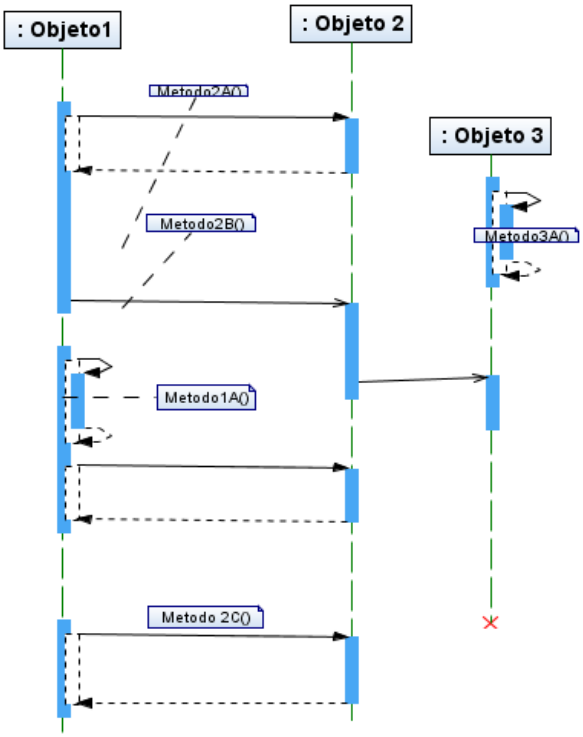

Figura 2.25. Ejemplo diagrama de secuencia

2.2.3- 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 patrones arquitectónicos y los patrones de diseño que se han aplicado en el proyecto.

2.2.3.1- 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 describen los patrones MVC y Layers.

MVC

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.

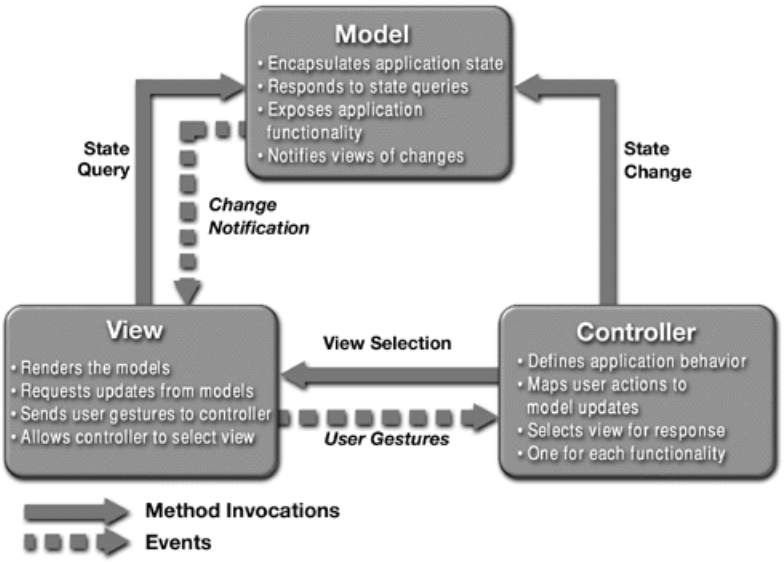

Figura 2.26. Modelo Vista Controlador

Este patrón es muy popular y ha sido portado a una gran cantidad de entornos y frameworks[17] 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.

[17] 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[18] 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[19]), y se transforma el modelo para ser desplegado una vez más.

[18] 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,

[19] 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.

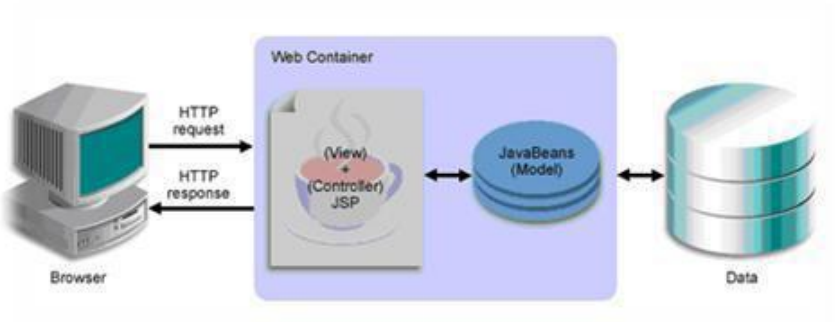

Figura 2.27. Arquitectura MVC modelo1

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.

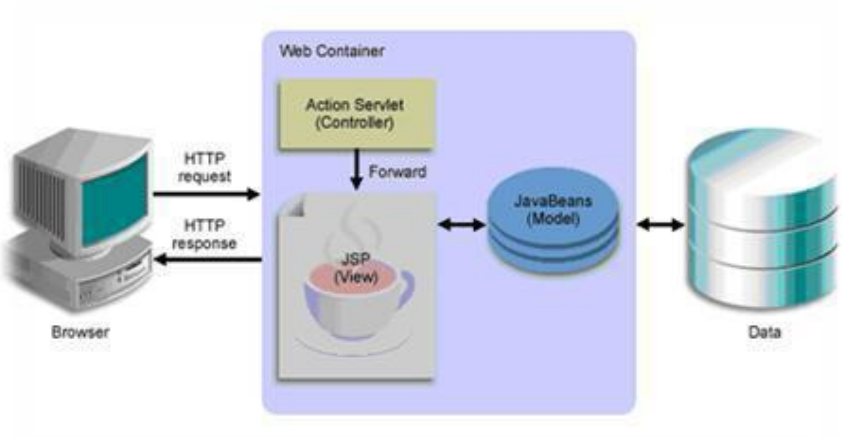

Figura 2.28. Arquitectura MVC modelo2

Layers

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

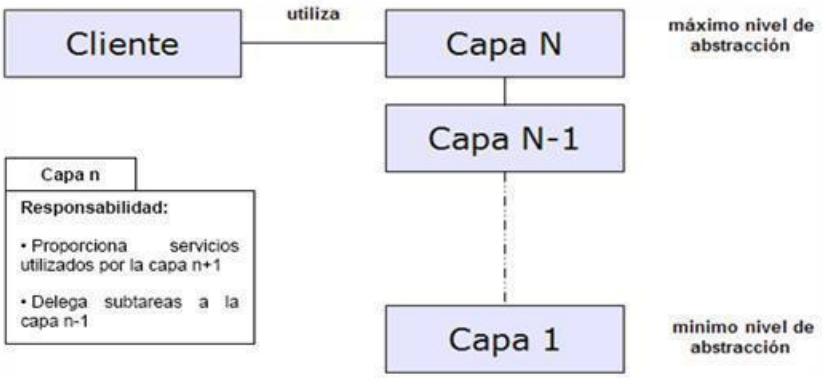

Figura 2.29. Estructura del patrón capas

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.

2.2.3.1- 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.

Value Object

Value Object (o Transfer Object) es una forma compacta y organizada de pasar datos de una capa a otra. Un Value Object no es más que un objeto que "*empaqueta*" datos para que puedan viajar entre las capas. Dicho objeto contendrá todos los datos que nos interesen, procecentes de uno o varios objetos de negocio, accesibles mediante getters y setters. Nótese que aunque los Value Objects están directamente relacionados con los objetos del dominio, no se trata de los mismos objetos. Los objetos del dominio pueden contener lógica de negocio mientras que los Value Objects son meros almacenes de datos. Además no tiene por qué haber una relación uno-a-uno entre objetos del dominio y Value Objects. Se destacan algunos puntos importantes:

- Por cada objeto de negocio puede haber más de un Value Object. De hecho, una estrategia muy común es usar un Value Object distinto para cada caso de uso.

- Un problema importante es el de la sincronización entre los valores del Value Object y los del objeto del dominio que representa. Hay que asegurarse de que dichos valores están actualizados o que una falta de actualización de los mismos no conlleve consecuencias graves (caso típico de las operaciones de solo lectura). En caso de usar los Value Object (VO) tanto para mostrar datos como para almacenarlos (VOs actualizables) habrá que tener sumo cuidado para sincronizar la información de los VOs que hayan cambiado.

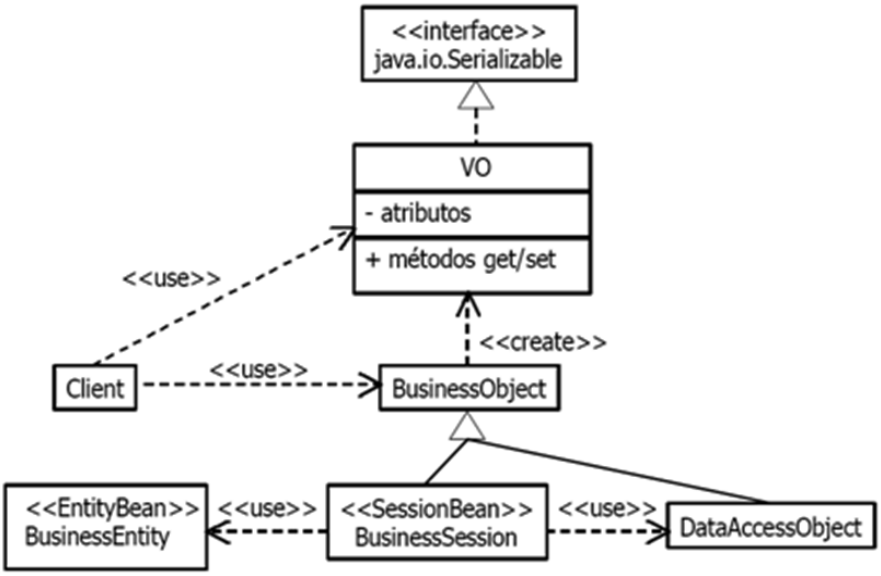

Figura 2.30. Estructura del Patrón Value Object

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[20] 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.

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

- 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.

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.

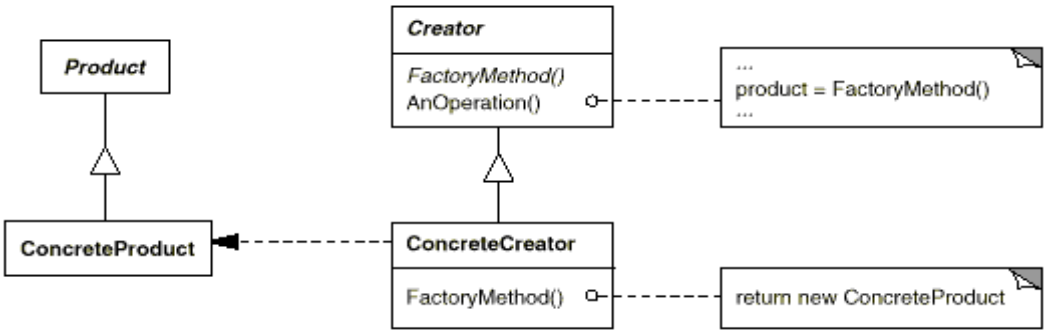

Figura 2.31. Estructura del Patrón Factory

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.

Facade

Este patrón proporciona una interfaz unificada (fachada) para un conjunto de interfaces en un subsistema, es decir, define una interfaz de nivel más alto haciendo al subsistema más fácilmente utilizable. El patrón Facade le oculta al cliente el conjunto de clases de servicio, separando la *funcionalidad*, de las clases que la utilizan o *clientes*. Dicho patrón reduce al mínimo la comunicación y dependencias entre subsistemas.

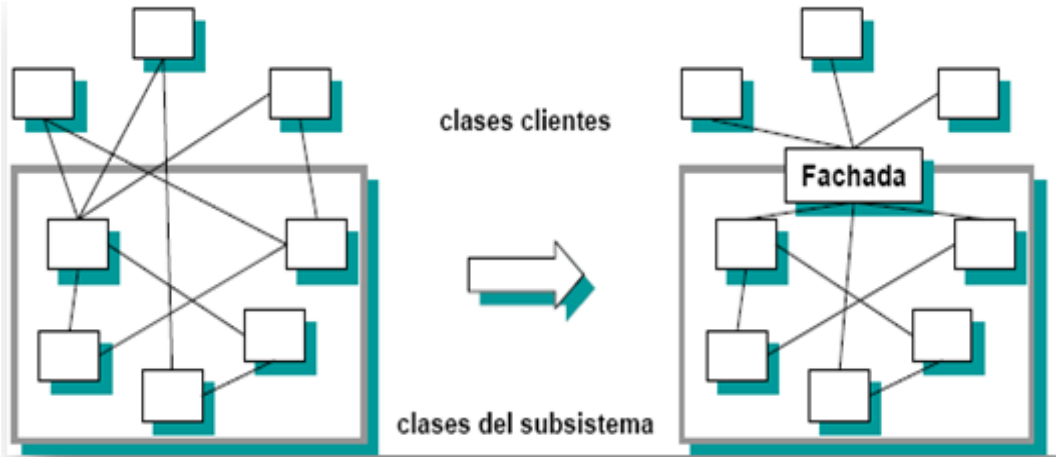

Figura 2.32. Propósito del Patrón Facade

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

- Facade

  - Conoce que clases del subsistema son responsables de una petición.
  - Delega las peticiones de los clientes en los objetos del subsistema.

- Clases del subsistema

  - Implementan la funcionalidad del subsistema.
  - Manejan el trabajo asignado por el objeto Facade.
  - No tienen ningún conocimiento del Facade (no guardan referencia de éste).

Mediante esta estructura los clientes se comunican con el subsistema a través de la fachada (facade) que reenvía las peticiones a los objetos del subsistema apropiados pudiendo realizar también algún trabajo de traducción. El patrón Facade proporciona los siguientes beneficios:

- Oculta a los clientes de la complejidad del subsistema y lo hace más fácil de usar.

- Favorece un acoplamiento débil entre el subsistema y sus clientes, consiguiendo que los cambios de las clases del sistema sean transparentes a los clientes.

- Facilita la división en capas y reduce dependencias de compilación.

- No se impide el acceso a las clases del sistema.

Capítulo 3. Herramientas para la elaboración del proyecto

El objetivo de este Capítulo es el de analizar las herramientas y tecnologías que se han empleado en el desarrollo del proyecto.

El Capítulo está dividido en cuatro secciones; en primer lugar se describe la infraestructura del servidor, el sistema operativo, el servidor Web, el servidor de bases de datos y el repositorio semántico. Posteriormente se analiza el entorno de trabajo, lenguaje de programación, IDE y sistema de control de versiones empleados. A continuación se hace una breve descripción de los framework empleados en el desarrollo, desde la capa de presentación hasta la persistencia. Por último se presentan de las herramientas que proporcionan servicios añadidos y que están fuera de las secciones 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

Como sistema operativo se ha optado por utilizar GNU/Linux, pues se adapta perfectamente a las necesidades del proyecto, seguridad, fiabilidad y robustez.

A continuación se enumeran las características más destacables que han motivado la elección de GNU/Linux frente a otras alternativas.

- **Seguro:** Ya que la gran mayoría de los ataques de hackers son dirigidos a servidores Windows al igual que los virus, los cuales se enfocan principalmente a servidores con este sistema operativo. La plataforma Linux es más robusta lo cual hace más difícil que algún intruso pueda violar la seguridad del sistema.

- **Rápido:** Al tener una plataforma más estable, se favorece la utilización de aplicaciones de todo tipo. La eficiencia de su código fuente hace que la velocidad de las mismas aplicaciones corriendo en Linux sean superiores a las que corren sobre Windows.

- **Económico:** Ya que requiere menor mantenimiento. Los servidores Windows son más costosos debido a que es necesaria una frecuente atención y la

monitorización contra ataques de virus, hackers y errores de código. El software Linux, así como también un sin fin de aplicaciones, son de código abierto y están protegidas por la licencia GPL1[21], motivo por el que son distribuidas gratuitamente. No requieren supervisión constante ni pagos de mantenimiento para obtener Service Packs,

Dentro de los SSOO GNU/Linux, la distribución elegida ha sido Ubuntu en su versión estable a fecha de instalación de este sistema operativo: Ubuntu 9.04 Jaunty Jackalope. Ubuntu 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.

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 (SERVIDOR WEB, 2009) es un programa que implementa el protocolo HTTP (hypertext transfer protocol). Este protocolo está diseñado para

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

transferir lo que llamamos hipertextos, páginas Web o páginas HTML (hypertext markup language): es decir: 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.

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. 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[22], 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[23].

- **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.

[22]JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos.

[23] Es una aplicación que se relaciona con otra para aportarle una función nueva y especifica.

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.

3.1.2- Servidor de aplicaciones

Se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. 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.

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

3.1.2.1- GlassFish v2

GlassFish (GlASSFISH, 2009) es un servidor de aplicaciones de código abierto 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, 2008) 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.

3.1.3- Sistema Gestor de Base de Datos

Una Base de Datos[24] 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*.

- **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

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

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.

3.1.3.1- 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[25]. Que presenta las siguientes características:

[25] 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.

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[26].

[26] 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.

Claves ajenas

También denominadas Llaves ajenas o Claves Foraneas (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.

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.

3.1.4- Gestión de ontologías

Para la gestión de las ontologías se ha elegido Jena[27], esta elección se ha basado en la compatibilidad con las últimas versiones de Java, en la facilidad de uso, en el amplio interfaz de programación (API) que proporciona para la gestión de ontologías y en las facilidades que incluye para la creación de recursos compartidos en Web. Jena se instalará sobre el gestor de Base de Datos PostgreSQL 8.4. De este modo, el servidor Jena se encarga de la gestión de consultas y actualizaciones en diversos lenguajes, delegando las tareas de persistencia a PostgreSQL.

[27] Jena es un framework Java para la creación de aplicaciones relacionadas con la Web Semántica. Proporciona un entorno de programación para RDF, RDFS y de OWL, SPARQL, e incluye un motor basado en reglas de inferencia.

3.2- Entorno de trabajo

En el desarrollo del proyecto se han utilizado herramientas basadas en software libre. Debido a que el proyecto SEMSE es desarrollado de forma colaborativa es de especial importancia la utilización de una herramienta de control de versiones, para tal fin se ha utilizado subversión[28].

3.2.1- Java Standard Edition 6

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:

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, como los punteros.

Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

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 herencia, polimorfismo, abstracción, concurrencia, etc. El lenguaje va acompañado de numerosas librerías de objetos que cubren todas las

[28] Es un sistema de control de versiones iniciado en el año 2000 por CollabNet Inc. Se emplea para mantener las versiones actualizadas y mantener un histórico de los archivos. Tiene como objetivo ser el sucesor de CVS (Concurrent Versions System).

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.

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

Dada la naturaleza distribuida de Java, donde los applets se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia, 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 del sistema operativo.

Todas estas restricciones se ponen de manifiesto en las restricciones que se imponen en la creación de applets, como son: la imposibilidad de acceder a un disco duro de una máquina local, el no poder acceder a un servidor.

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[29], que sí es dependiente de la máquina (Máquina virtual de Java).

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 (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.

[29] 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

.

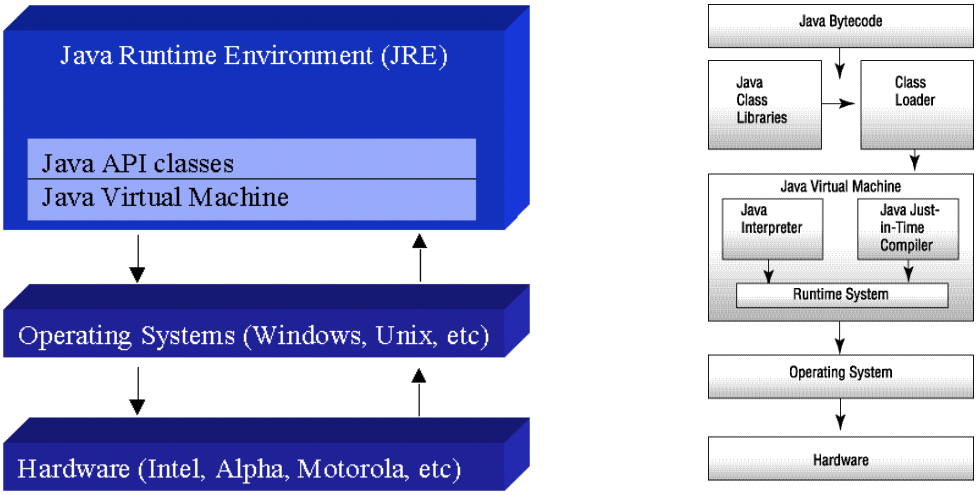

Figura 3.1. Estructura de java

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- Conceptos importantes de java

Debido a la especial importancia que ha tenido java en el desarrollo del proyecto en este apartado se describirán algunos de los conceptos más representativos de este lenguaje como son la programación orientada a objetos, la herencia y el polimorfismo. Estos conceptos, con algunos matices también serian aplicables a otros lenguajes de programación.

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 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, doublé…). 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, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto.

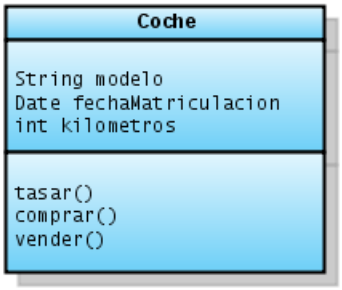

Figura 3.2. Ejemplo de la clase coche

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.

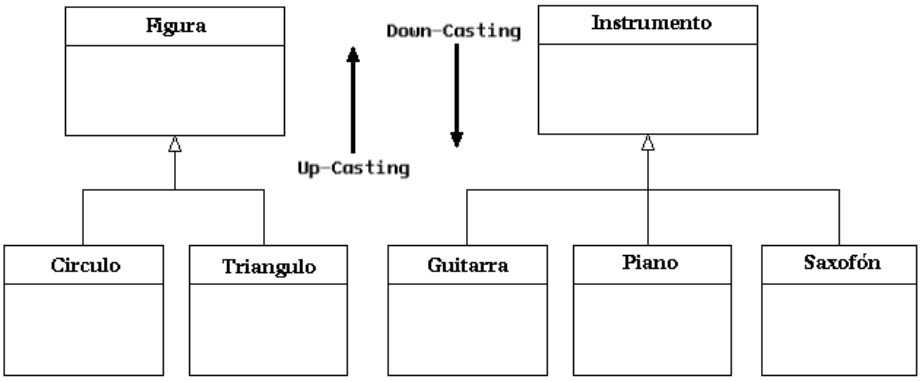

Figura 3.3. Herencia y polimorfismo en java

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.

3.2.2- NetBeans

Para el desarrollo de la aplicación se hará uso de uno de los entornos de desarrollo integrado (IDE) [30] más completos y de software libre que se encuentran en la actualidad, Netbeans.

[30] Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

Netbeans (NETBEANS, 2008) es un IDE, desarrollado en Java, que surgió como un proyecto de estudiantes de la república Checa, y que se llamó originalmente "*Xelfi*". Pretendían hacer un entorno de programación como Delphi pero para Java. Xelfi fue el primer IDE para Java escrito en Java, con una primera versión disponible en 1997. Xelfi fue un proyecto bastante productivo y novedoso, ya que el espacio de los IDEs desarrollados en Java era un territorio sin explorar hasta entonces.

El proyecto atrajo tanto interés para estos estudiantes, que una vez graduados decidieron hacerlo un producto comercial, y crearon una empresa para poder distribuirlo. Estos desarrolladores a día de hoy siguen contestando a la gente en las listas de distribución. Pronto recibieron una oferta de Roman Stanek un emprendedor que ya había participado en diversos proyectos en la república Checa con muy buenos resultados. Buscaba una gran idea en la que invertir y encontró a Xelfi en el camino.

La idea original para la empresa fue desarrollar componentes JavaBeans preparados para redes (Network-Enabled). De ahí el nombre de Netbeans. Cuando las especificaciones para las Entrerprise Java Beans (EJB) fueron liberadas, se pensó que sería mejor trabajar con ellas, más que competir con ellas. Cambió la filosofía pero el nombre continuó siendo el mismo.

En la primavera de 1999 salió a la luz "*Netbeans DeveloperX2"*, que incluía soporte para Swing[31]. El avance en el rendimiento que java ofreció en su versión JDK1.3 hizo que Netbeans se convirtiera en una herramienta de desarrollo viable y ampliamente utilizada. A partir de ese momento, los desarrolladores decidieron reestructurar la plataforma, modularizandola y convirtiéndola en la beta de "*Netbeans Developer"*.

Entró en juego entonces Sun Microsystems, que compró la compañía y decidió llamar a este IDE *Forté*, desapareciendo el nombre de Netbeans por entonces. Sun necesitaba herramientas de desarrollo para Java y se interesó por este IDE.

[31] Swing es una biblioteca gráfica para Java que forma parte de las Java Foundation Classes (JFC). Incluye widgets para la interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Siempre ha habido un gran interés por parte de los desarrolladores de Netbeans en el Software Libre, los desarrolladores eran jóvenes y habían estado involucrados en proyectos de software libre durante el desarrollo de sus carreras universitarias. Por eso, la decisión conjunta por Sun y el equipo de Netbeans de convertir Netbeans en un proyecto Open Source fue bienvenida, tanto por los desarrolladores, como por la extensa comunidad de usuarios que utilizaban este entorno.

Este fue el primer proyecto Open Source que Sun Microsystems patrocinaba ya que invertía en el desarrollo funcional y en toda la infraestructura del proyecto. Surgió así el proyecto "*Netbeans.org"* en Junio del 2000 empezando un ciclo que actualmente sigue en desarrollo.

3.2.2.1- NetBeans 6.5

Para la realización de este proyecto se ha utilizado NetBeans 6.5. NetBeans 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 móviles.

Entre las mejoras de esta nueva versión se encuentra soporte completo a PHP[32] ofreciendo dispositivos como compleción de código, codificación coloreada por semántica e integración a bases de datos. También hay elementos mejorados Ruby[33] en el editor, debugger y Rake (variante de codificación Ruby). Soporta Groovy[34] y Grails[35] en el editor.

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

[33] Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos

[34] Groovy es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java.

[35] Grails es un framework open source para la realización de aplicaciones web utilizando Groovy

El rendimiento del código se optimizó 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. NetBeans 6.5 soporta desarrollo con JavaFX[36].

Otra novedad es 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 proporciona soporte ampliado para Spring, Hibernate[37], Java Server Pages y la API Java Persistence. Esta versión viene con 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.

[36] Es una familia de productos y tecnologías de Sun Microsystems, adquirida por Oracle Corporation, para la creación de Rich Internet Applications.

[37] 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.

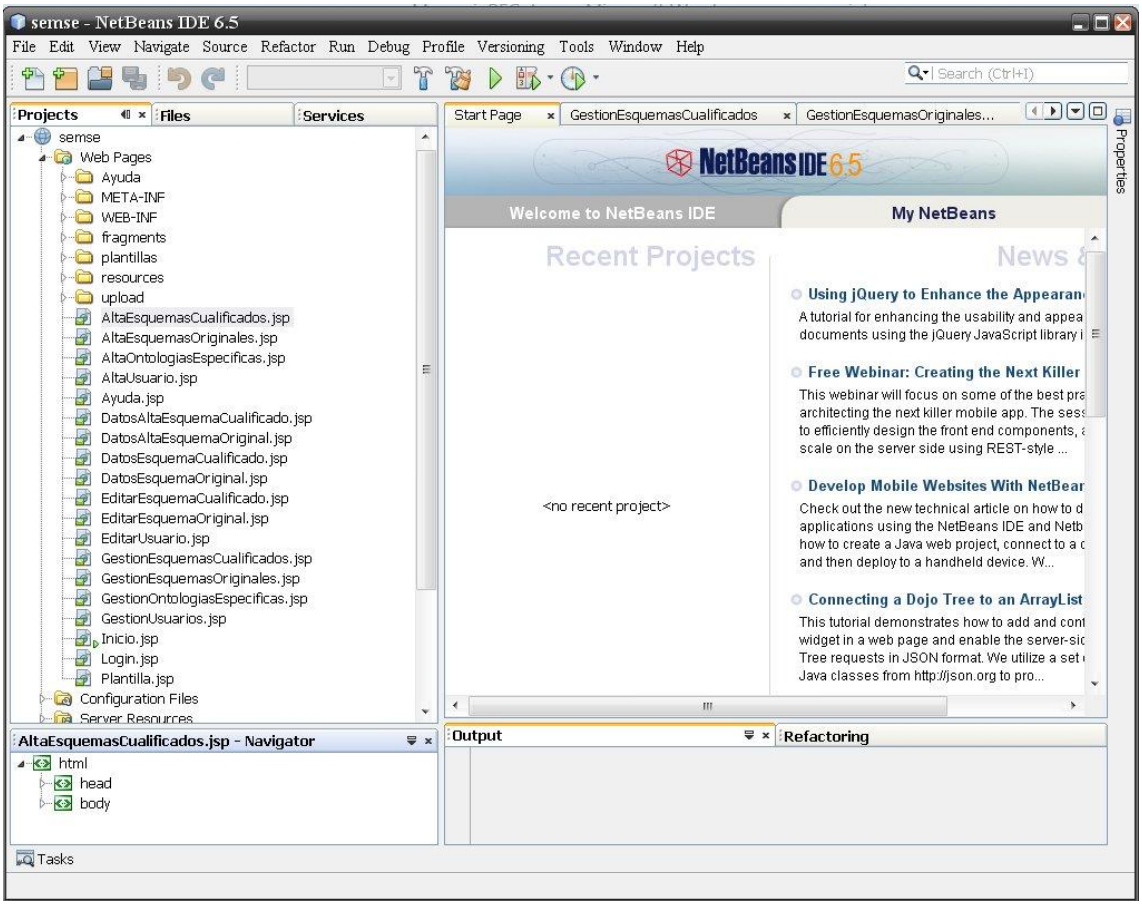

Figura 3.4. NetBeans 6.5

3.2.3- Subversion

En el desarrollo de este proyecto se ha utilizado Subversion versión 1.5 como sistema de control de versiones, que facilita el desarrollo colaborativo de aplicaciones. De esta forma el desarrollo se ha podido llevar a cabo desde distintas ubicaciones y entre varios desarrolladores, y además ha facilitado el poder volver atrás en el desarrollo cuando ha sido necesario.

Subversion (COLLINS-SUSSMAN, 2008) es un sistema de control de versiones libre y de código fuente abierto. La estructura de subversión consiste en un árbol de ficheros en un repositorio central, que recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Podemos acceder al repositorio a

través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores.

Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados específicamente para la administración de árboles de código fuente, y tienen muchas características que son específicas del desarrollo de software, tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software. Subversion es un sistema general que puede ser usado para administrar *cualquier* conjunto de ficheros.

3.2.3.1- Arquitectura de Subversion

En un extremo se encuentra un repositorio de Subversion que conserva todos los datos versionados. Al otro lado, hay un programa cliente Subversion que administra réplicas parciales de esos datos versionados (llamadas "*copias de trabajo*"). Entre estos extremos hay múltiples rutas a través de varias capas de acceso al repositorio (AR). Algunas de estas rutas incluyen redes de ordenadores y servidores de red que después acceden al repositorio. Otras pasan por alto la red y acceden al repositorio directamente.

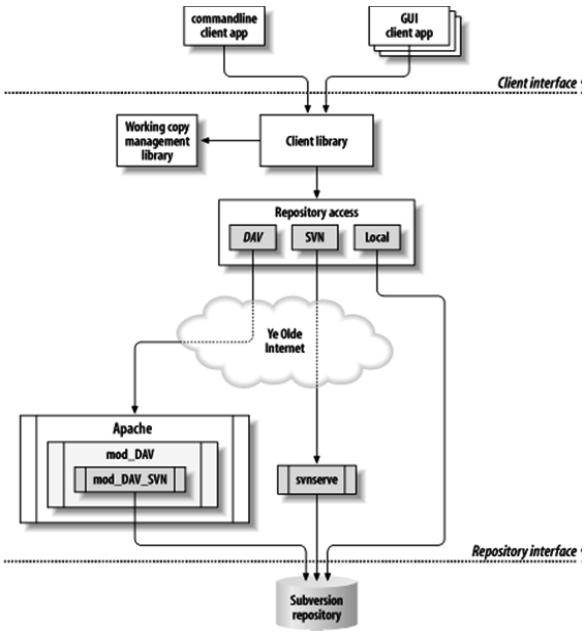

Figura 3.5. Arquitectura de Subversion

3.2.3.2- Conceptos básicos de subversión

El repositorio

Subversion es un sistema centralizado para compartir información. La parte principal de Subversion es el repositorio, el cual es un almacén central de datos. El repositorio guarda información en forma de árbol de archivos, formado por ramas (archivos o directorios). Cualquier número de clientes puede conectarse al repositorio y luego leer o escribir en esos archivos.

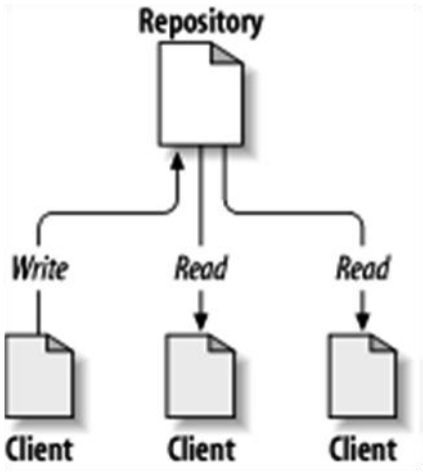

Figura 3.6. El repositorio en subversión

Un repositorio puede estar estructurado como el usuario crea conveniente. Sin embargo, como podemos ver en la Figura 3.7. *"Ejemplo de estructura de repositorio".*

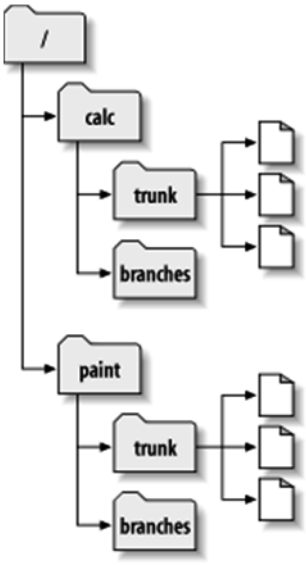

Figura 3.7. Ejemplo de estructura de repositorio

La estructura que tendría este repositorio, sería un directorio principal dónde se desarrollaría el proyecto, y los directorios branches o ramas, serían copias de calc/trunk para que los desarrolladores puedan realizar pruebas en paralelo, sin afectarse entre sí.

Ramas

Una rama es una línea de desarrollo que existe de forma independiente a otra, dentro de un repositorio, pero comparte una historia común si mira suficientemente atrás en el tiempo. Una rama siempre nace como una copia de algo, y a partir de ahí, pasa a generar su propia historia.

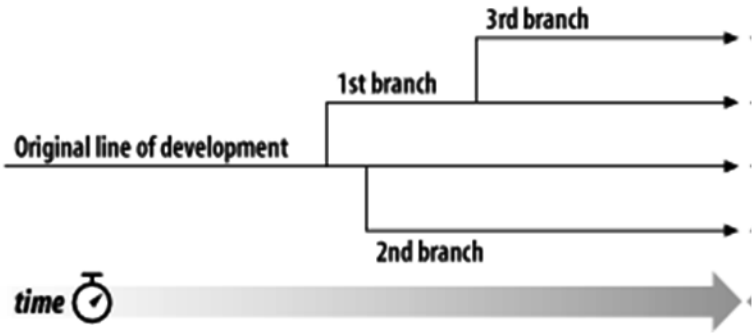

Figura 3.8. Ramas en subversión

Subversion tiene comandos para ayudarle a mantener ramas paralelas de sus ficheros y directorios. Permite crear ramas copiando sus datos, y recordando que las copias están relacionadas unas a otras. También le ayuda a duplicar cambios de una rama a otra. Finalmente, puede reflejar el contenido de diferentes ramas en partes de su copia de trabajo local, para que pueda "*mezclar y probar*" diferentes líneas de desarrollo.

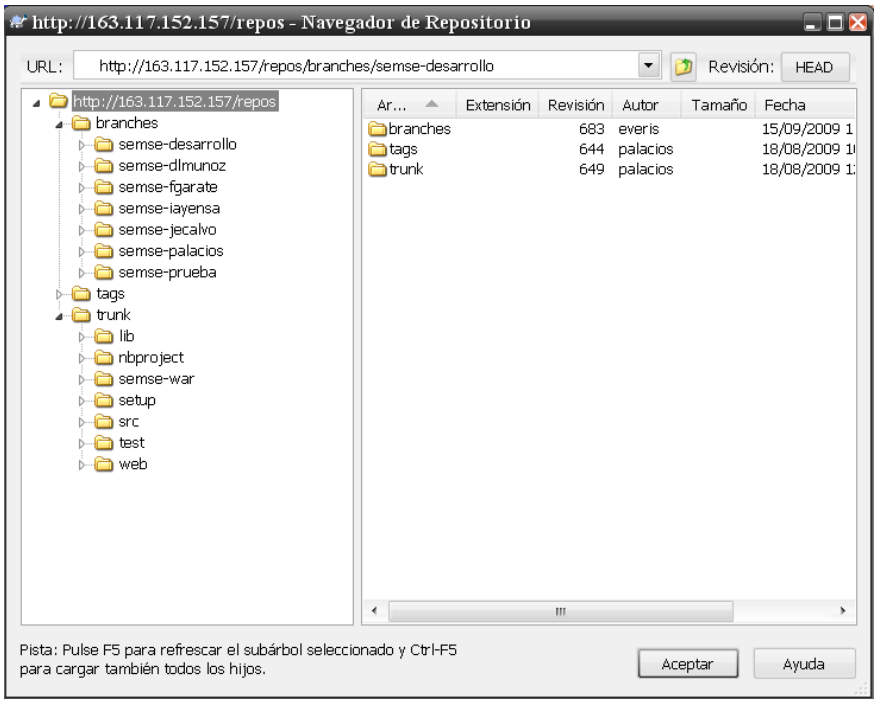

Figura 3.9. Ramas empleadas en el desarrollo del proyecto SEMSE

3.2.3.3- Tortoise

En el desarrollo del proyecto se ha utilizado TortoiseSVN (TORTOISE, 2009) como cliente Subversion, esta implementado como una extensión al shell de Windows que permite ser integrado en el explorador de Windows, se trata de software libre liberado bajo la licencia GNU GPL.

Aunque Totoise es utilizado principalmente por desarrolladores software, también puede ser útil para otro tipo de actividades, como escritores, traductores, etc.

Mediante Tortoise se han podido realizar todas las acciones que permite realizar subversión de una forma gráfica e integrada en las ventanas de Windows.

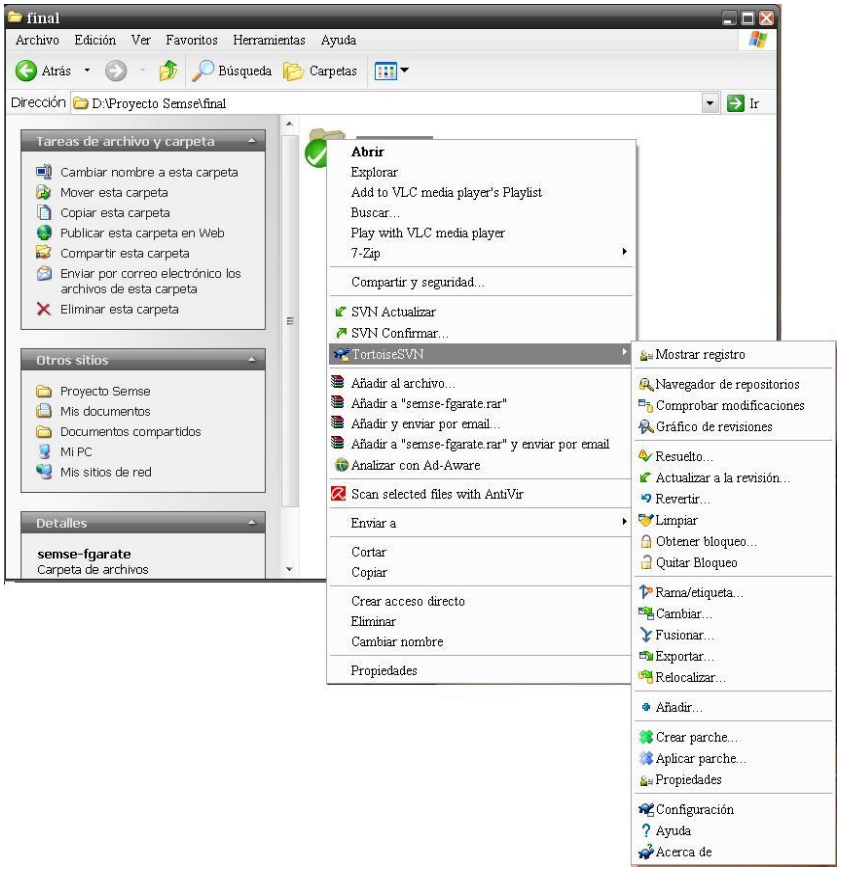

Figura 3.10. Uso de TortoiseSVN

3.3- Frameworks empleados

El término framework o marco de trabajo se ha popularizado en los últimos años dentro del ambiente del desarrollo de software. Un framework, en el desarrollo de software, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En este apartado se describen los distintos frameworks empleados en el desarrollo del proyecto, comenzando desde los empleados en la capa de vista hasta llegar finalmente a la conexión con la base de datos.

3.3.1- ICEFaces (AJAX, JavaScript)

ICEFaces es un framework AJAX de código abierto que permite a los desarrolladores de Java EE crear y desplegar fácilmente aplicaciones tipo RIA (Rich Internet Application).

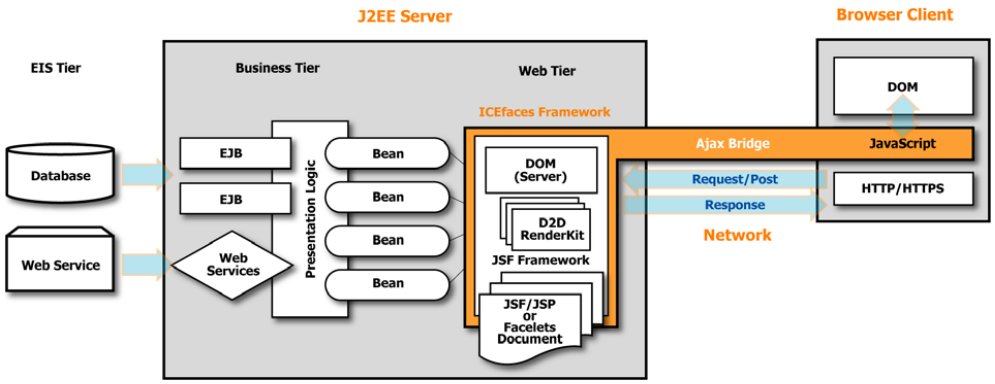

Figura 3.11. Arquitectura básica de ICEFaces

Las aplicaciones desarrolladas en ICEFaces no necesitan plugins de navegador o applets para ser vistas. Estas aplicaciones están basadas en JavaServer Faces (JSF), así

que permite el desarrollo de aplicaciones Java EE con la posibilidad de utilizar de forma fácil desarrollos basados en JavaScript.

Entorno a AJAX[38] han surgido varios frameworks (Prototype, DWR, GWT, etc) que, si bien aportaban facilidad de uso, no acababan de convencer a la comunidad de programadores. Algunos porque sólo eran clientes JavaScript, otros porque, aunque integraban la parte de servidor con la de cliente, no eran realmente frameworks, sino librerías de comunicación. Además, no estaba claro cómo unirlos con la arquitectura JEE.

Con la llegada de JSF, se empezó a vislumbrar posibilidades de integración. Si JSF permitía al desarrollador aislarse de la arquitectura Web y ver sus aplicaciones como algo parecido a una aplicación de escritorio, debería entonces ser sencillo utilizar AJAX para hacer estos controles más funcionales. Y así fue, empezaron a aparecer AJAX4JSF, ICEFaces, Tobago, ...

Sin embargo, de estas propuestas, ICEFaces fue una de las más acogidas ya que aísla completamente al desarrollador de AJAX. No hacen falta etiquetas especiales: se ponen los controles en la pantalla e ICEFaces se encarga de enviar entre cliente y servidor sólo la información necesaria.

Algunos de los beneficios de ICEFaces son:

- **Experiencia de usuario Enriquecida:** permite generar interfaces semejantes a las de las aplicaciones de escritorio de forma rápida haciendo uso de la Suite de Componentes.

- **Código Abierto:** la licencia de ICEFaces es MPL o Mozilla Public License.

- **Basado en Estándares:** quienes deseen empezar con ICEFaces sólo necesitan tener conocimientos básicos de desarrollo de aplicaciones Web y Java.

- Se puede usar cualquier IDE para programar en ICEFaces.

[38] Acrónimo de Asynchronous JavaScript And XML, es una técnica de desarrollo web para crear aplicaciones interactivas o RIA

- **AJAX "*Transparente*":** desarrollar aplicaciones Web interactivas es sencillo con ICEFaces. No es necesario agregar código JavaScript adicional a las páginas para enviar los requerimientos al servidor, ICEFaces se encarga de ello.

- **Interoperabilidad:** Muchos servidores de aplicaciones soportan ICEFaces, se puede cambiar, por ejemplo, de server GlassFish a Tomcat sin necesidad de modificar el código.

- **Seguridad:** ICEFaces es compatible con SSL y además no es posible a través del browser ver o modificar el código de la aplicación generada. La Figura 3.12. *"Elementos de la arquitectura ICEFaces"* presenta la arquitectura de una aplicación en JSF integrada con ICEFaces:

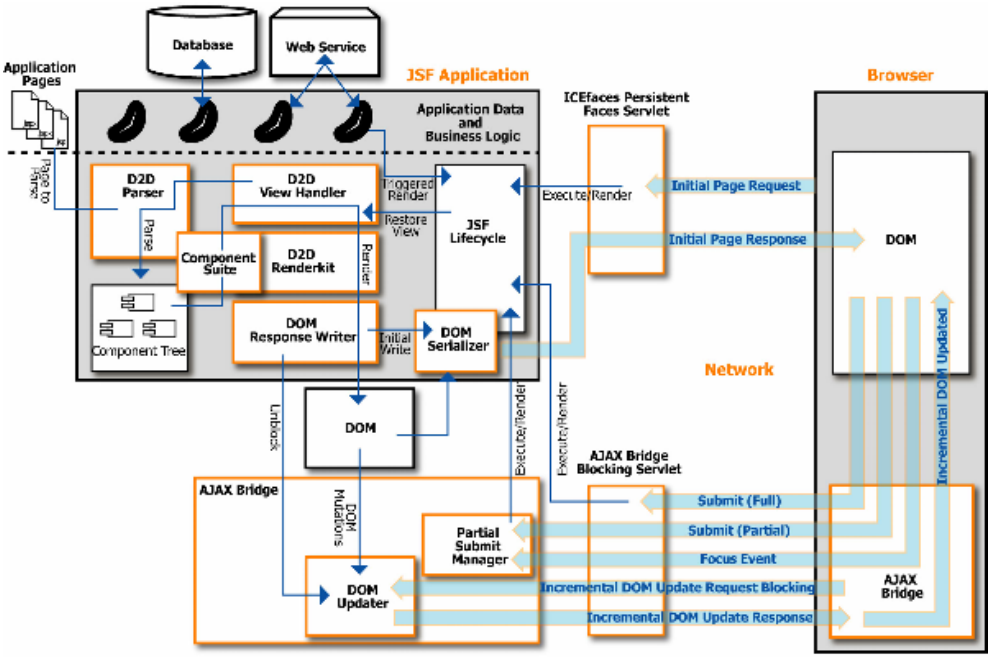

Figura 3.12. Elementos de la arquitectura ICEFaces

Los principales elementos de la arquitectura ICEFaces incluyen:

- **Persistent Faces Servlet:** Las URLs con extensión ".iface" son mapeadas por el servlet 'Persistent Faces Servlet'. Cuando se realiza una petición de la página inicial en la aplicación, este servlet se hace responsable de la ejecución del ciclo de vida JSF para petición asociada.

- **Blocking Servlet:** Se encarga de la gestión de todas las peticiones de bloqueo y no-bloqueo después de las primeras páginas.

- **D2D ViewHandler:** Se encarga de establecer el Direct-to-DOM, incluyendo la inicialización de la 'DOM Respuesta Writer'. El ViewHandler también invoca al Parser para analizar el árbol de componentes JSF en la página inicial.

- **Parseador D2D:** Responsable del montaje de un componente de documentos JSP. El Parser ejecuta la etiqueta de JSP de procesamiento del ciclo de vida con el fin de crear el árbol, pero lo hace sólo una vez para cada página. La compilación del estándar JSP y el proceso de análisis no es compatible con ICEFaces.

- **DOM Response Writer:** Se encarga de la escritura en el DOM[39]. También inicia la serialización DOM para la primera prestación, y desbloquea el DOM Updater para actualizaciones incrementales.

- **Component Suite:** Ofrece un conjunto de componentes 'rich JSF' con influencia AJAX y características del puente, proporcionando los elementos básicos para aplicaciones ICEFaces.

[39] Document Object Model es esencialmente un modelo computacional a través del cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML y XML. Su objetivo es ofrecer un modelo orientado a objetos para el tratamiento y manipulación en tiempo real (o de forma dinámica) a la vez que de manera estática de páginas de internet.

- **Client-side AJAX Bridge:** Responsable de la actualización DOM en curso generada por la solicitud y la respuesta del proceso. También es el encargado de centrar la gestión y de presentar el proceso.

- **DOM Serializer:** Responsable de la serialización del DOM de la página inicial.

- **DOM Updater:** Se encarga de conjuntar las de las 'DOM mutations' en una única actualización DOM.

3.3.1.1 - Suite de componentes ICEFaces

La suite de componentes ICEFaces proporciona los bloques necesarios para desarrollar una sofisticada interfaz de usuario en aplicaciones JSF.

La amplia gama de componentes ICEFaces se puede ver en acción en ICEFacesComponent Showcase.

Atributos más comunes

- **renderedOnUserRole:** Si el usuario esta en el rol indicado en el valor del atributo el componente se utilizará normalmente. Si no, no se hace nada y el cuerpo de esta etiqueta se salta.

- **enabledOnUserRole:** Si el usuario esta en el rol indicado en el valor del atributo el componente se utilizará normalmente. Si no, el componente pasará a estado deshabilitado.

- **visible:** Se utiliza para permitir la visibilidad del atributo en el elemento raíz, por defecto es visible.

- **disabled:** Es un flag que indica que el elemento no debe ser atendido ni incluido en posteriores presentaciones.

- **partialSubmit:** permite a un componente hacer una presentación parcial en el evento apropiado.

ICEFaces Components	renderedOnUserRole	enabledOnUserRole	visible	disabled	partialSubmit
commandButton	*	*	*	*	*
commandLink	*	*	*	*	*
commandSortHeader	*	*		*	
dataPaginato	*	*		*	
dataTable	*				
form	*				*
graphicImage	*		*		
inputFile	*	*		*	
inputSecret	*	*	*	*	*
inputText	*	*	*	*	*
inputTextarea	*	*	*	*	*
menuBar	*				
menuItem	*				
menuItems	*				
menuItemSeparator	*				
message	*		*		
messages	*		*		
outputConnectionStatus	*				
outputDeclaration	*				
outputLabel	*		*		
outputLink	*	*	*	*	
outputProgress	*				
outputText	*		*		
panelBorder	*				
panelGrid	*		*		
panelGroup	*		*		
panelPopup	*		*		
panelSeries	*				
panelStack	*				
panelTabSet	*				

panelTab	*	*		*	
selectBooleanCheckbox	*	*	*	*	*
selectInputDate	*	*		*	
selectInputText	*	*		*	
selectManyCheckbox	*	*	*	*	*
selectManyListbox	*	*	*	*	*
selectManyMenu	*	*	*	*	*
selectOneListbox	*	*	*	*	*
selectOneMenu	*	*	*	*	*
selectOneRadio	*	*	*	*	*

Tabla 3.1. Distintos componentes ICEFaces

3.3.1.2 – Tecnología AJAX

El término AJAX se presentó por primera vez en el artículo "*AJAX: A New Approach to Web Applications*" publicado por Jesse James Garrett el 18 de Febrero de 2005. Hasta ese momento, no existía un término normalizado que hiciera referencia a un nuevo tipo de aplicación Web que estaba apareciendo.

En realidad, el término AJAX es un acrónimo de Asynchronous JavaScript + XML, que se puede traducir como "*JavaScript asíncrono + XML*".

El artículo define AJAX de la siguiente forma: *"AJAX no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes."*

Las tecnologías que forman AJAX son:

- **XHTML y CSS**, para crear una presentación basada en estándares.
- **DOM**, para la interacción y manipulación dinámica de la presentación.
- **XML, XSLT y JSON**, para el intercambio y la manipulación de información.
- **XMLHttpRequest,** para el intercambio asíncrono de información.

- **JavaScript**, para unir todas las demás tecnologías.

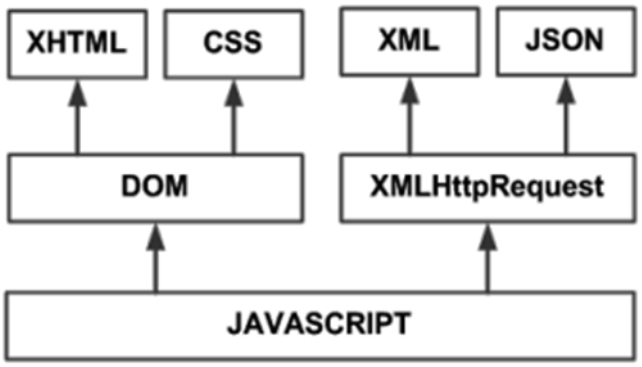

Figura 3.13. Tecnologías agrupadas bajo el concepto de AJAX

En las aplicaciones Web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

En el siguiente esquema, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones Web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX:

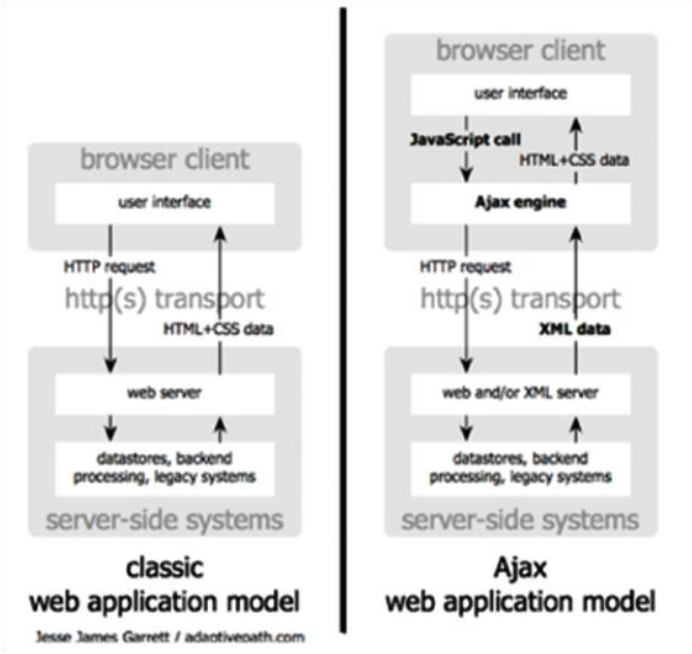

Figura 3.14. Modelo AJAX frente al modelo tradicional

Esta técnica tradicional para crear aplicaciones Web funciona correctamente, pero no crea una buena sensación al usuario. Al realizar peticiones continuas al servidor, el usuario debe esperar a que se recargue la página con los cambios solicitados. Si la aplicación debe realizar peticiones continuas, su uso se convierte en algo molesto.

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor. Una aplicación AJAX elimina el start-stop-start-stop natural de las iteraciones en la Web introduciendo un intermediario (un motor de AJAX) entre el usuario y el servidor.

En lugar de cargar una página Web, al comenzar la sesión, el navegador carga un motor de AJAX, (escrito en JavaScript y normalmente escondido en un marco oculto). Este motor es responsable tanto de presentar la interfaz que el usuario ve como la comunicación con el servidor. El motor de AJAX permite que la iteración del usuario con el servidor sea asíncrona. De modo que el usuario nunca este viendo la ventana del navegador en blanco a la espera de que el servidor haga algo.

En la Figura 3.15. *"Comunicación asíncrona de AJAX frente a la síncrona tradicional"* se muestra la diferencia más importante entre una aplicación Web tradicional y una aplicación Web creada con AJAX. La imagen superior muestra la iteración síncrona propia de las aplicaciones Web tradicionales. La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX.

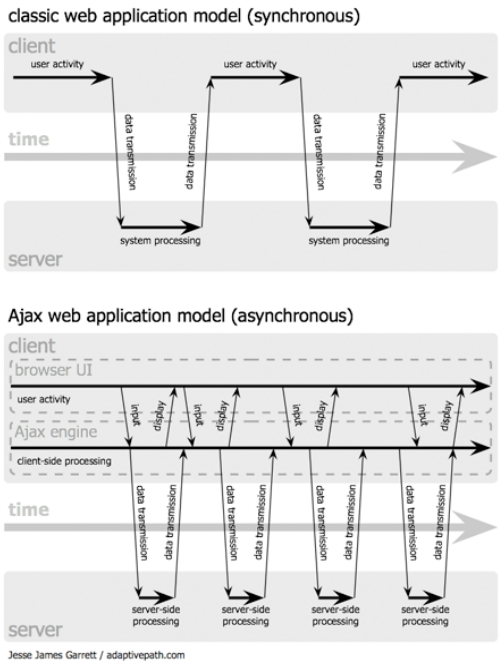

Figura 3.15. Comunicación asíncrona de AJAX frente a la síncrona tradicional

Las peticiones HTTP al servidor se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor

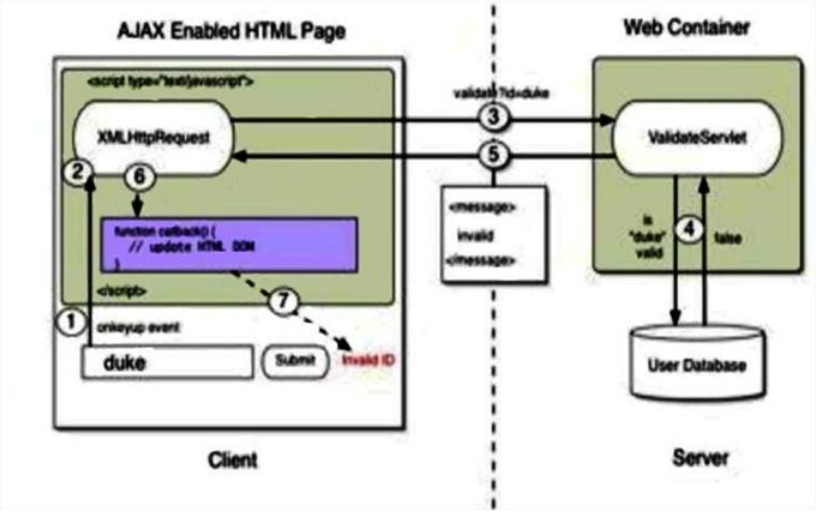

Figura 3.16. Peticiones JavaScript en AJAX

3.3.1.3 – JavaScript

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. Su sintaxis es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas:** como sucede con XHTML[40], el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para su manejo (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.).

- **Se distinguen las mayúsculas y minúsculas:** al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente y el único problema es que la página no valida. Por el contrario, si en JavaScript se intercambian mayúsculas y minúsculas, las aplicaciones no funcionan correctamente.

- **No se define el tipo de las variables:** al definir una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del programa.

- **No es obligatorio terminar cada sentencia con el carácter del punto y coma (;).** No obstante, es muy recomendable seguir la tradición de terminar cada sentencia con el carácter ;

- **Se pueden incluir comentarios:** los comentarios se utilizan para añadir alguna información relevante al código fuente del programa. Aunque no se visualizan por pantalla, su contenido se envía al navegador del usuario junto con el resto

[40] Acrónimo de eXtensible Hypertext Markup Language, es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web

del programa, por lo que es necesario extremar las precauciones sobre el contenido de los comentarios.

Como se comenta en (EGUÍLUZ, 2008) JavaScript se utiliza principalmente para crear páginas Web dinámicas que procesen la entrada del usuario y que sean capaces de gestionar datos persistentes usando objetos especiales, archivos y bases de datos relacionales. Es el lenguaje dinámico del lado del cliente por excelencia. JavaScript se puede incluir en cualquier documento HTML, o todo aquel que termine traduciéndose en HTML en el navegador del cliente mediante scripts.

El lenguaje fue inventado por Brendan Eich en la empresa Netscape con el nombre de LiveScript y fue rebautizado como JavaScript en un anuncio conjunto entre Sun Microsystems y Netscape. Posteriormente fue adoptado como estándar en la ECMA bajo el nombre ECMAScript. Poco después también se incorporó como un estándar ISO.

En la actualidad la última versión es JavaScript 1.8 que aporta las siguientes características al lenguaje:

- Permite declarar funciones que devuelvan datos sin necesidad de llaves (*{..}*), ni de devolver directamente el resultado con *return*.

- Permite generar expresiones como arrays o matrices de forma rápida y elegante.

- Existen dos nuevos métodos de iteración para los arrays:

  - **reduce ():** Ejecuta una función en cada elemento en el array y colecciona los resultados de llamadas previas.

  - **reduceRight ():** Ejecuta una función en cada objeto del array y colecciona los resultados de llamadas previas, pero en orden inverso.

3.3.2 – JSF (Java Server Faces)

JavaServer Faces (JSF) constituye un framework de interfaces de usuario del lado de servidor para aplicaciones Web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador). JSF es un framework de Sun Microsystems y tiene especificaciones desarrolladas por la Java Community Process: la especificación JSR 127, que define JSF 1.0 y JSF1.1, y la especificación JSR 252, que define JSF 1.2.

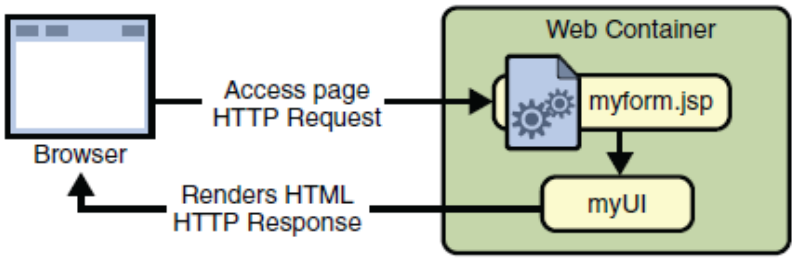

Figura 3.17. En JSF las UIs corren en el servidor de aplicaciones

Como se describe en (JENDROCK, 2008) los principales componentes de esta tecnología son:

- Una API y una implementación de referencia para:

  - Representar componentes de interfaz de usuario (UI) y manejar su estado

  - Manejar eventos, validar en el lado del servidor y convertir datos

  - Definir la navegación entre páginas

  - Soportar internacionalización y accesibilidad

  - Proporcionar extensibilidad para todas estas características

- Dos librerías de etiquetas JavaServer Pages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido y las librerías de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones Web con UIs en el lado servidor. Con un mínimo esfuerzo, es posible:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos en el lado servidor.
- Construir una interfaz de usuario con componentes reutilizables y extensibles.

JSF presenta dos nuevos términos: *managed bean* (bean manejado) y *backing bean* (bean de respaldo). Los objetos JavaBean gestionados por una implementación JSF se llaman beans manejados. Un bean manejado describe como se crea y se maneja un bean. No tiene nada que ver con las funcionalidades del bean.

El bean de respaldo define las propiedades y las lógicas de manejo asociadas con los componentes UI utilizados en la página. Cada propiedad del bean de respaldo está unida a un ejemplar de un componente o a su valor. Un bean de respaldo también define un conjunto de métodos que realizan funciones para el componente, como validar los datos del componente, manejar los eventos que dispara el componente, y realizar el procesamiento asociado con la navegación cuando el componente se activa.

3.3.2.1 - Beneficios de la tecnología JSF

JSF está definida por una especificación lo que le convierte en un estándar. Una de las ventajas de que JSF sea una especificación estándar es que pueden encontrarse implementaciones de distintos fabricantes. Esto permite no vincularse exclusivamente

con un proveedor concreto, y poder seleccionar el que más interese en cada caso según el número de componentes que suministra, el rendimiento de éstos, soporte proporcionado, precio, política de evolución, etc.

JSF trata la vista (la interfaz de usuario) de una forma algo diferente a lo acostumbrados en las aplicaciones Web, ya que este tratamiento es mucho más cercano al estilo de Java Swing, Visual Basic o Delphi, donde la programación de la interfaz se hace a través de componentes y está basada en eventos (pulsación de un botón, cambio en el valor de un campo, etc.).

JSF es muy flexible. Por ejemplo nos permite crear nuestros propios componentes, y/o crear nuestros propios renderizadores para pintar los componentes en la forma que más nos convenga.

Una de las grandes ventajas de la tecnología JSF es que ofrece una clara separación entre el comportamiento y la presentación. Las aplicaciones Web construidas con tecnología JSP conseguían parcialmente esta separación. Sin embargo, una aplicación JSP no puede mapear peticiones HTTP al manejo de eventos específicos de los componentes o manejar elementos UI como objetos con estado en el servidor. La tecnología JSF permite construir aplicaciones Web que introducen realmente una separación entre el comportamiento y la presentación, separación sólo ofrecida tradicionalmente por arquitecturas UI del lado del cliente.

Separar la lógica de negocio de la presentación también permite que cada miembro del equipo de desarrollo de la aplicación Web se centre en su parte asignada del proceso diseño, y proporciona un modelo sencillo de programación para enlazar todas las piezas. Por ejemplo, personas sin experiencia en programación pueden construir páginas JSF usando las etiquetas de componentes UI que esta tecnología ofrece, y luego enlazarlas con código de la aplicación sin escribir ningún script.

Otro objetivo importante de la tecnología JSF es mejorar los conceptos familiares de componente-UI y capa-Web sin limitarnos a una tecnología de script particular o un lenguaje de marcas. Aunque la tecnología JSF incluye una librería de etiquetas JSP personalizadas para representar componentes en una página JSP, las

APIs de JSF se han creado directamente sobre el API JavaServlet. Esto nos permite, teóricamente, hacer algunas cosas avanzadas: usar otra tecnología de presentación junto a JSP, crear nuestros propios componentes personalizados directamente desde las clases de componentes, y generar salida para diferentes dispositivos cliente, entre otras.

En definitiva, la tecnología JSF proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario, y manejar eventos.

A día de hoy está disponible la versión JSF 1.2 que añade las siguientes características:

- Integración con la tecnología JSP 2.1 mediante el uso del lenguaje de expresión (EL) unificado.
- Posibilidad de añadir mensajes personalizados de manera más sencilla.
- Nueva etiqueta setPropertyActionListener

3.3.2.2 - Visual JSF

Visual JSF (WINSTON, 2006) es un framework de J2EE basado en JavaServer Faces (JSF). Con Visual JSF se pueden generar páginas Web visualmente. Visual JSF introduce algunas bibliotecas de extensión para dar soporte al desarrollo de aplicaciones JSF:

- **DataProvider:** para el acceso uniforme de los datos, independientemente de si los datos se han obtenido a través de "*CachedRowset*" o si los datos se almacenan como arrays o lista de objetos
- **Application Model:** para facilitar la integración de los usuarios en la lógica de negocio a lo largo de todo el ciclo de vida JSF (sin la necesidad de comprender las complejas fases del ciclo de vida de JSF)

- **Designtime API:** para integrar los componentes JSF

- **ErrorHandler:** para apoyar la adecuada reorientación de la página cuando se produce un error en la aplicación

Visual JSF genera todo el código necesario para una aplicación basada en JSF:

- Creación de un fichero JSP (Page1.jsp).

- Creación de un fichero Java (Page.java), utilizado como "*backing bean*" por Page1.jsp, que contiene los métodos de Application model.

- Creación de tres beans: RequestBean, SessionBean, ApplicationBean.

- Bibliotecas para el proyecto:

    - Componentes Woodstock (Webui-JSF, dojo, Jason, etc)
    - DataProvider (dataprovider.jar)
    - ErrorHandler (errorhandler.jar)
    - Application model (appbase.jar)

- Inyección de entradas a WEB-INF/Web.xml

    - JSF Servlet
    - Parámetros de Contexto para personalizar la aplicación JSF
    - Theme Servlet
    - File Upload Filter
    - Error Handler Servlet

- Información de la pagina inicial

- Inyección de entradas a WEB-INF/faces-config.xml

  - Información del managed bean

En resumen, en una aplicación visual JSF, el IDE de desarrollo, al agregar una nueva página, nos genera el código JSP necesario para generar la respuesta HTML al cliente. Se puede desarrollar un portal al mejor estilo drag & drop, editando las características de los componentes desde la pestaña "*propiedades"* del editor.

3.3.3 – Spring

Spring es un framework libre para el desarrollo de aplicaciones Java. La primera versión fue escrita por Rod Johnson, quien lo lanzó primero con la publicación de su libro "*Expert One-on-One Java EE Design* and Development" (JOHNSON, 2002).

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.

Como se comenta en (SPRING, 2008) este framework fue creado basándose en los siguientes principios:

- El buen diseño es más importante que la tecnología subyacente
- Los JavaBeans ligados de una manera más libre son un buen modelo
- El código debe ser fácil de probar

Spring está diseñado en módulos y proporciona:

- Un contenedor ligero que permite una potente gestión de configuración basada en JavaBeans, aplicando los principios de Inversión de Control (*IoC*). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener singletons[41] ni ficheros de configuración, una aproximación consistente y elegante. Estas definiciones de beans se realizan en lo que se llama el contexto de aplicación.

- Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción añadibles (*pluggables*), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. Se incluyen estrategias genéricas para JTA[42] y un único JDBC[43] DataSource. En contraste con el JTA simple o EJB CMT[44], el soporte de transacciones de *Spring* no está atado a entornos J2EE.

- Un framework MVC, construido sobre el núcleo de Spring. Este framework es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, Velocity, Tiles o iText. De cualquier manera una capa modelo realizada con *Spring* puede ser fácilmente utilizada con una capa Web basada en cualquier otro framework MVC, como Struts, WebWork o Tapestry.

A día de hoy la última versión de Spring es la 2.5. Algunas de las características que incluye esta nueva versión son:

[41] Objeto basado en la implementación del patrón con el mismo nombre, asegura que una clase solo tiene una instancia (un único objeto) y proporciona un punto global de acceso a ella.

[42] Java Transaction API o API para transacciones en Java.

[43] Java Database Connectivity o Conectividad a la Base de Datos de Java.

[44] Enterprise Java Beans Container-Managed Transactions: EJB con transaccionalidad manejada por el container.

- Soporte para Web Services (API JAX-WS).
- Inyección de dependencia vía anotaciones.
- Anotaciones disponibles para controladores MVC.
- TestContext Framework. Un marco completo para probar los test de nuestras aplicaciones de manera independiente del sistema utilizado, junit o testing, y que hace más fácil integrar los test, encargándose el automáticamente de tareas que antes estamos obligados a realizar nosotros, como la carga de los contextos, cerrar transacciones, etc.
- Es posible desplegar contextos de Spring en un servidor de aplicaciones como archivo rar.
- Es posible desplegar contextos de Spring en un servidor de aplicaciones como un recurso JCA (J2EE Connector Architecture).
- Mayor control sobre el ciclo de vida de un componente.
- Nuevos contextos. Es posible insertar y obtener objetos de una petición Web de manera declarativa.

Los conceptos más importantes en Spring son inversión de control, inyección de dependencia, programación orientada a aspectos, persistencia (ORM[45]), Spring Security y mock testing que se comentarán en los siguientes apartados.

3.3.3.1 – Estructura de Spring

La arquitectura en capas de *Spring* ofrece mucha de flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar

[45] Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia.

la gestión de configuración basada en JavaBeans sin utilizar el framework MVC o el soporte AOP[46].

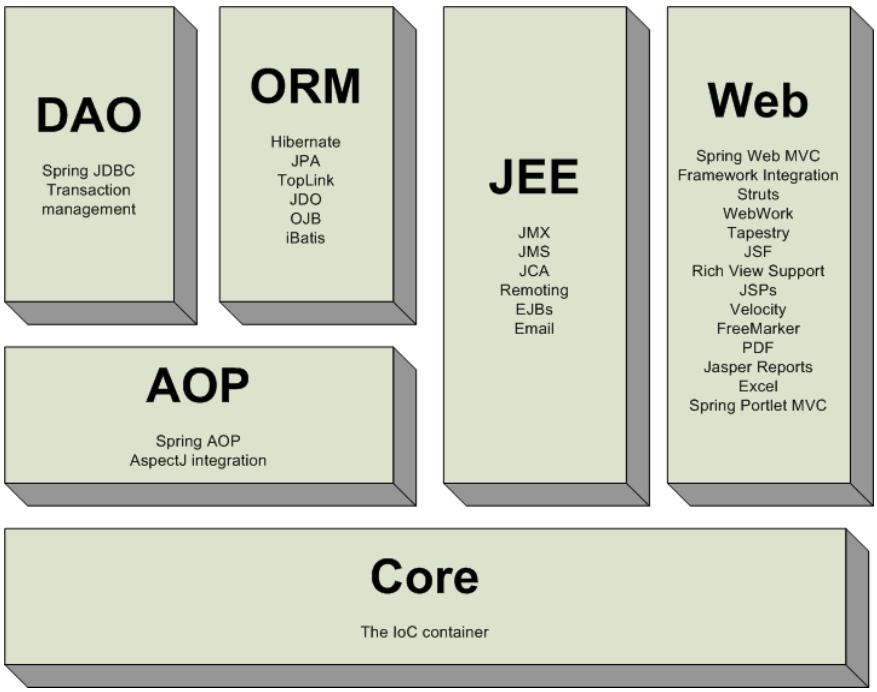

Figura 3.18. Arquitectura en capas de Spring

El paquete *Core*

Es 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[47], 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.

[46] Aspect Oriented Programming o Programación Orientada a Objetos.

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

El paquete *Context*

Está construido sobre una solida 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.

El paquete *DAO*

Provee una capa de abstracción a JDBC, eliminando la tediosa codificación propia de JDBC y el parseo de los códigos de errores específicos de cada proveedor de base de datos. Del mismo modo, este paquete proporciona mecanismos de manejo de transacciones tanto programáticamente como declarativamente, cuyo manejo no está restringido a clases que implementen algún tipo de interfaz especial, sino que también está pensado para cualquiera de nuestros POJOs.

El paquete *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 Spring ofrece.

El paquete *AOP*

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.

El paquete *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. El paquete *MVC* de Spring, provee una implementación del patrón MVC (*Moldeo-Vista-Controlador*) para las aplicaciones Web.

Spring MVC

Provee una limpia separación entre el código del modelo de dominio y los formularios Web, integrándose a todas las otras características que este framework posee.

Inversión de Control (IoC)

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. Es por esto que se le llama "i*nversión*", ya que invierte la acción de llamada a alguna librería externa (JOHNSON, 2007).

Inyección de dependencias

Normalmente (WALLS, 2008) se suelen extender clases de manera innecesaria. Especialmente problemática es esta práctica cuando se trata de objetos DAO (Data Access Object) que crean una conexión a la Base de Datos ya que si cada usuario crea una conexión a la Base de Datos y el número de usuarios es alto pueden hacer que el servidor se caiga.

Para evitar esto, se propone como patrón la inyección de dependencias que radica 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, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los

usuarios y así se evita tener que andar extendiendo clases o saturar el servidor de la BD.

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.

Programación Orientada a Aspectos (AOP)

AOP (Aspect-Oriented Programming) es una técnica que permite modularizar las aplicaciones:

- Por un lado funcionalidades comunes utilizadas a lo largo de la aplicación.
- Por otro lado 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:** Especificando que un advice16 debe causar que un objeto implemente interfaces adicionales.
- **Pointcuts dinámicos y estáticos:** para especificar los puntos en la ejecución del programa donde debe haber intercepción.

3.3.4 - Hibernate

Hibernate (HIBERNATE, 2008) es una herramienta de mapeo objeto/relacional (ORM) de software libre para Java aunque también está disponible para .Net con el nombre de NHibernate. Fue una iniciativa de un grupo de desarrolladores conducidos por Gavin King.

Hibernate permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones y un gran número de tipos de datos. Hibernate

puede expresar consultas tanto en su lenguaje de consulta llamado HQL (*Hibernate Query Languaje*) como en SQL.

Hibernate (CARO,2008) presenta las siguientes ventajas:

- **Productividad:** Evita parte del código farragoso de la capa de persistencia, permitiendo centrarse en la lógica de negocio. Permite una estrategia de desarrollo de aplicaciones topdown (empezar con el modelo de entidades) o bottom-up (trabajar con un modelo de datos existente).

- **Mantenibilidad:** Al tener pocas líneas de código permite que el código sea más comprensible.

- **Rendimiento:** Existe la tendencia a pensar que una solución "*manual*" es más eficiente que una "*automática*". Hay que tener en cuenta que una solución automática, permite que se dedique más tiempo a optimizaciones. Por otra parte, por ejemplo el pooling de "*PreparedStatement*" son más óptimos para un driver DB2, mientras que menos para Interbase. Actualizar las columnas que cambian en una sentencia update es más rápido en unas bases de datos, pero más lentas en otras. Todo está lógica está embebida en el motor ORM. El motor está desarrollado por programadores con altos conocimientos de los SGBD y la conectividad con Java (JDBC y drivers).

- **Indepencia de vendedor:** Una solución ORM abstrae del SGBD. Permite desarrollar en local con bases de datos ligeras sin implicación en el entorno de producción.

Hibernate presenta distintos módulos.

Hibernate Core

Conocido como Hibernate 3.x. Corresponde con el servicio base de persistencia con una API propia y los ficheros de mapping residen en XML. Actualmente está disponible Hibernate 3.3. Entre las novedades frente a la versión 3.2 se destacan:

- Migración a un sistema de construcción con Maven.

- División del proyecto en varios módulos jar (al estilo de módulos Maven), lo que facilita el ver y administrar las dependencias.

- Rediseño de las SPI (Interfaces de Servicio) para el caché de segundo nivel.

- Integración con JBossCache 2.x como proveedor de caché de segundo nivel.

Hibernate annotations

Permite definir anotaciones disponibles en JDK5.0 que son embebidas directamente en el código Java evitando disponer de ficheros XML de mapeo.

Las anotaciones son un conjunto de anotaciones básicas que implementa el estándar JPA, y además incluye un conjunto de extensiones que dan cabida a funcionalidades más avanzadas propias de Hibernate (tunning y mapping).

Hibernate EntityManager La especificación JPA define un conjunto de interfaces, reglas para el ciclo de vida de una entidad persistente y características de las consultas. La implementación de Hibernate para esta parte de la especificación de JPA es cubierta con Hibernate EntityManager. Las características de Hibernate son un superconjunto de las especificadas por JPA.

Hibernate EntityManager

La especificación JPA define un conjunto de interfaces, reglas para el ciclo de vida de una entidad persistente y características de las consultas. La implementación de Hibernate para esta parte de la especificación de JPA es cubierta con Hibernate EntityManager. Las características de Hibernate son un superconjunto de las especificadas por JPA.

.
.

```
.
@Entity

@Table(name = "esquemacualificado", schema = "public")

public class Esquemacualificado implements java.io.Serializable {

    private long esquemacualificadoid;

    private Esquemaoriginal esquemaoriginal;

    private long esquemaid;

    private Set<Ontologiaespecifica> ontologiaespecificas = new
HashSet<Ontologiaespecifica>(0);



public Esquemacualificado() {

}

public Esquemacualificado(long esquemacualificadoid,
Esquemaoriginal esquemaoriginal, long esquemaid) {

        this.esquemacualificadoid = esquemacualificadoid;

        this.esquemaoriginal = esquemaoriginal;

        this.esquemaid = esquemaid;


.
.
.


@Id

@Column(name = "esquemacualificadoid", unique = true, nullable =
false)

public long getEsquemacualificadoid() {

        return this.esquemacualificadoid;

}

@OneToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "esquemacualificadoid", unique = true, nullable =
false, insertable = false, updatable = false)

public Esquemaoriginal getEsquemaoriginal() {

        return this.esquemaoriginal;

}


.
.
.
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy
= "esquemacualificado")

public Set<Ontologiaespecifica> getOntologiaespecificas() {

        return this.ontologiaespecificas;
```

```
}

.
.
.
```

Código Fuente 3 1. Fragmento de la clase EsquemaCualificado.java

En este fragmento de código se puede observar el aspecto que presenta la entidad EsquemaCualificado, sus distintos atributos y las relaciones con otras entidades definidas en las anotaciones. En la Figura 3.19. "*Papel de Hibernate en las aplicaciones Java*" se puede apreciar el lugar que ocupa Hibernate dentro de las aplicaciones Java.

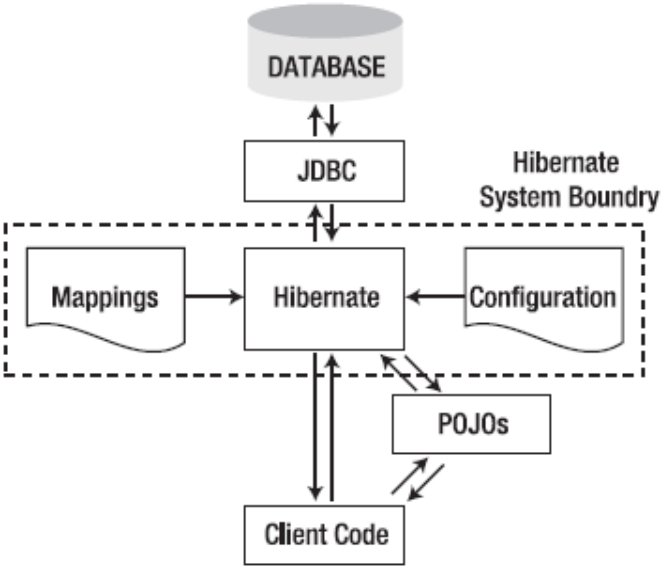

Figura 3.19. Papel de Hibernate en las aplicaciones Java

3.3.4.1– Arquitectura de Hibernate

Este diagrama (Red Hat, 2004) muestra a Hibernate utilizando datos de configuración y la base de datos para proveerle servicios de persistencia (y objetos persistentes) a la aplicación.

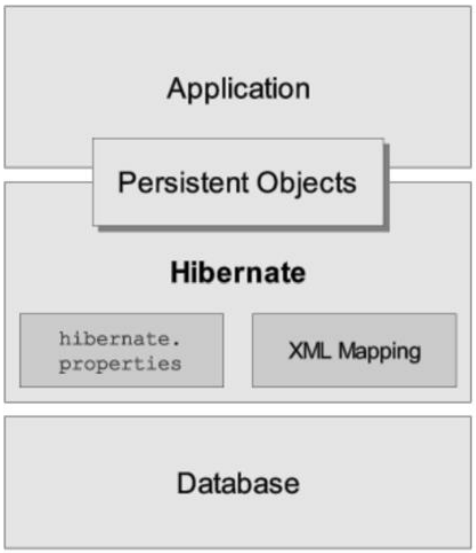

Figura 3.20. Arquitectura básica de Hibernate

Hibernate es muy flexible y soporta múltiples arquitecturas en tiempo de ejecución, en la Figura 3.21. "*Arquitectura de Hibernate* " se muestra una arquitectura que abstrae a la aplicación, alejándola de las capas subyacentes de JDBC/JTA, y deja que Hibernate se encargue de los detalles.

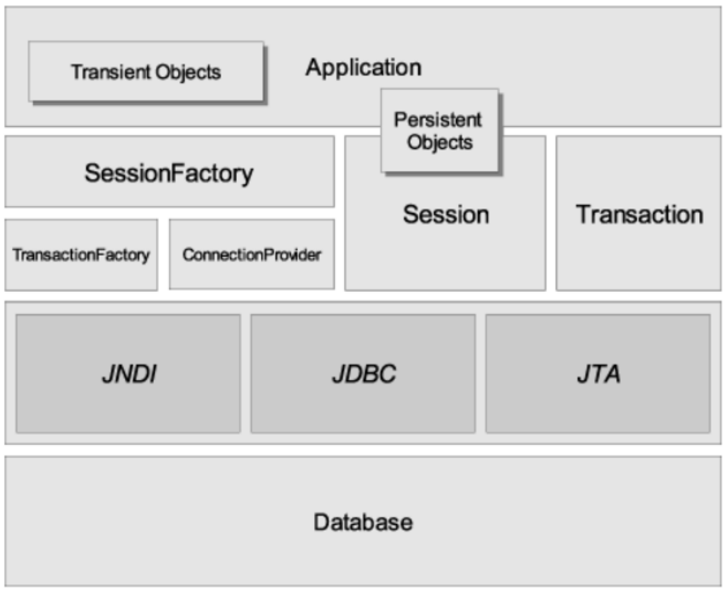

Figura 3.21. Arquitectura de Hibernate

SessionFactory (org.Hibernate.SessionFactory)

Un caché thread-safe e inmutable de mapeos, compilados para una base de datos en particular. Una fábrica (factory) de sesiones, y cliente de ConnectionProvider. Puede contener un caché optativo (llamado "*de segundo nivel*") que es reusable entre transacciones, a nivel de proceso o de cluster.

Session (org.Hibernate.Session)

Un objeto de thread simple y de vida corta, que representa una conversación entre la aplicación y el repositorio persistente. Envuelve a una fábrica de conexiones JDBC por transacción. Contiene un caché obligatorio (llamado "*de primer nivel*") de objetos persistentes, que se usa al navegar el árbol de objetos, o cuando se buscan los objetos por identificador.

Objetos y colecciones persistentes

Objetos de un solo thread y de vida corta, que contienen "*estado*" persistente y cumplen una función de negocio. Pueden ser JavaBeans comunes, o puros objetos de Java (POJOs por sus siglas en inglés), la única característica notable que tienen, es que están asociados con una sesión. En cuanto la sesión se cierra, serán desprendidos, y quedarán listos para ser usados en cualquier capa de la aplicación (por ejemplo, directamente como objetos de transmisión de datos o "*DTOs*", desde y hacia la capa de presentación).

Objetos y colecciones transitorios y desprendidos

Instancias de clases persistente que, por el momento, no están asociadas con una sesión. Pudieron haber sido instanciadas por la aplicación, y aún no haber sido asociadas con una sesión, o bien haber sido instanciadas por una sesión que en ese momento esté cerrada.

Transacción (org.Hibernate.Transaction)

Objeto opcional de un solo thread y de vida corta, usado por la aplicación para especificar unidades atómicas de trabajo. Abstrae a la aplicación de la transacción JDBC, JTA o CORBA subyacente. En algunos casos una sesión puede extenderse a lo largo de varias transacciones.

ConnectionProvider (org.Hibernate.connection.ConnectionProvider)

Una fábrica y repositorio (pool) de conexiones JDBC. Abstrae la aplicación de la fuente de datos (Datasource) o del driver (DriverManager) subyacentes. No está expuesto directamente a la aplicación, pero puede ser implementado o extendido por el programador.

TransactionFactory (org.Hibernate.TransactionFactory)

Una fábrica de instancias de Transaction. No está expuesta directamente a la aplicación, pero puede ser implementada o extendida por el programador.

3.3.4.2 - Configuración de Hibernate

Por defecto el componente SessionFactory va buscando un recurso llamado Hibernate.cfg.xml dentro del ClassLoader (cargador de clases).

Dentro del fichero Hibernate.cfg.xml se configura:

- Listado de ficheros de mapeo
- Dialecto de la BBDD. Engloba aquellas particularidades del SGBD.
- Cadena de conexión a la BBDD.
- Propiedades del pool de conexiones (C3PO, Apache DBCP, …)
- Propiedades adicionales:

- show_sql: Muestras las sentencias SQL emitidas por el motor de persistencia
- format_sql: Formatea las cadenas de las consultas para que sean legibles.

Una vez obtenido el SessionFactory, por cada petición al motor de persistencia, se obtiene un componente Session que se convierte en el interfaz para cargar y persistir objetos.

```
<?xml version=1.0" emcoding="UTF-8" ?>
<!DOCTYPE Hibernate-configuration>

<Hibernate-configuration>
<session-factory>
<property
   name="Hibernate.dialect">org.Hibernate.dialect.PostgreSQLDialect
</property>
<property
    name="Hibernate.connection.driver_class">org.postgresql.Driver
</property>
<property
name="Hibernate.connection.url">jdbc:postgresql://163.117.152.***:5432/SEMSE
</property>
<property name="Hibernate.connection.username">***</property>
<property name="Hibernate.connection.password">***</property>
<property name="Hibernate.show_sql">true</property>

  <mapping class="semse.Hibernate.Infousuario" />
  <mapping class="semse.Hibernate.Metadatos" />
  <mapping class="semse.Hibernate.Perfil" />
  <mapping class="semse.Hibernate.Ontologiaespecifica" />
  <mapping class="semse.Hibernate.JenaGraph" />
  <mapping class="semse.Hibernate.Documento" />
  <mapping class="semse.Hibernate.Esquema" />
  <mapping class="semse.Hibernate.Ontologiareferencia" />
  <mapping class="semse.Hibernate.Authorities" />
  <mapping class="semse.Hibernate.Ontologias" />
  <mapping class="semse.Hibernate.Esquemacualificado" />
  <mapping class="semse.Hibernate.Users" />
  <mapping class="semse.Hibernate.Esquemaoriginal" />
</session-factory>
```

```
</Hibernate-configuration>
```

Código Fuente 3.2. Fichero de configuración Hibernate.cfg.xml

3.3.5 - Jena

Jena es un Framework de software libre, desarrollado por HP Labs, para el desarrollo de aplicaciones Java relacionadas con la Web Semántica. Entre las características de Jena cabe destacar que permite gestionar todo tipo de ontologías, almacenarlas y realizar consultas contra ellas. Soporta distintos lenguajes como RDF, DAML y OWL, siendo independiente del lenguaje. Actualmente está disponible Jena 2 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.

Algunos de estos componentes que incluye Jena 2 (JENA, 2008) se comentan a continuación.

3.3.5.1 - API para RDF

Permite crear y manipular modelos RDF desde una aplicación Java. La utilización de estos modelos es aceptada por la W3C. RDF es un lenguaje semejante a XML que permite la creación de información semántica utilizando la estructura XML. Define una estructura para implementar la semántica de la información identificando objetos y sus propiedades. Provee primitivas para utilizar expresiones que poseen (sujeto, predicado, objeto).

Esta API proporciona clases Java para representar:

- **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:

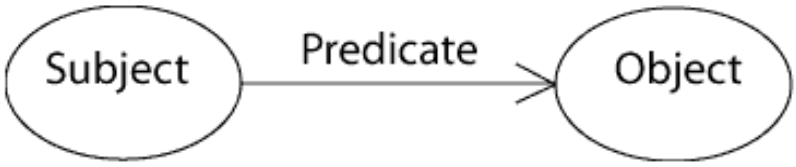

Figura 3.22. Tripleta RDF

  - 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
  - Carga en memoria de modelos
  - Navegar un modelo a partir de la URI22 de un recurso

- Consultar un modelo: se podrá buscar información del modelo y realizar consultas avanzadas.
- Operaciones sobre modelos: unión, intersección y diferencia.

3.3.5.2 - API para OWL

Como se ha comentado anteriormente existen diferentes lenguajes RDF, RDF Schema, OWL y SPARQL. El API de OWL de Jena utiliza un lenguaje neutral, es decir, los nombre de las clases no hacen mención al lenguaje subyacente que están representando (OWL, RDF, etc.), por lo tanto es independiente del lenguaje. Por ejemplo, la clase OntClass puede hacer referencia a una clase OWL o de RDF Schema.

Para representar las diferencias entre los distintos lenguajes, cada uno posee un *profile* que lista todo los constructores permitidos con los nombre de las clases y propiedades. Este profile esta 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 sea con OntModel como con Model esta almacenada en forma de tripletas RDF (sujeto, predicado y objeto).

Otro aspecto a tener en cuenta es cómo el framework trabaja la manera del polimorfismo en Java. Jena resuelve el tema de los distintos niveles polimórficos de RDF y la abstracción de clases en Java (OntClass, Restriction, DataTypeProperty, ObjectProperty, etc.) considerando a estas abstracciones como una vista del recurso. Es decir, existe una relación uno a muchos, un recurso puede ser representado por varias vistas.

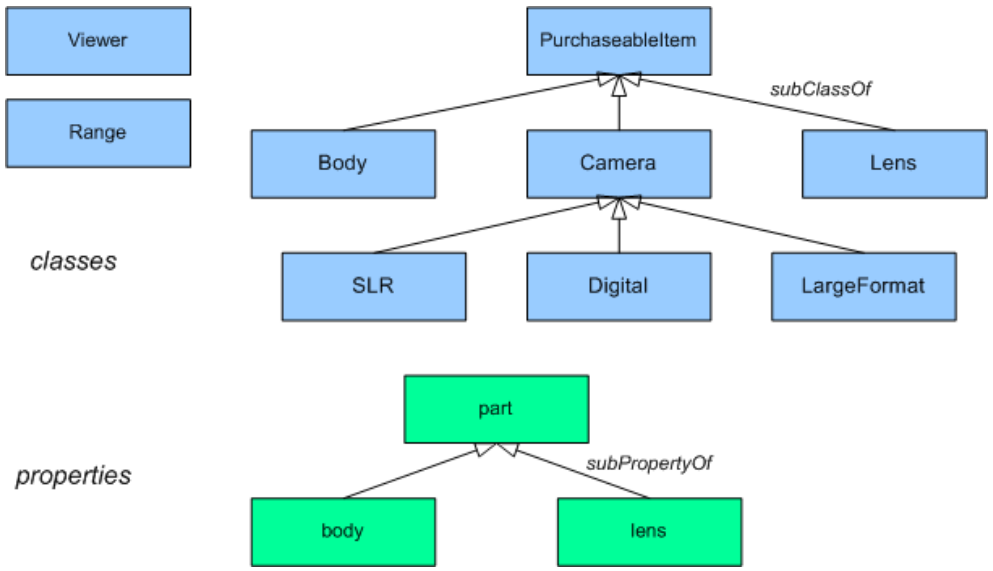

Figura 3.23. Herencia y polimorfismo en Jena

3.3.5.3 - 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).

Ejemplo:

```
public class CreateOntologyAction {

    private IDBConnection connection;
    private OntologyVO ontologyVO;

public CreateOntologyAction(IDBConnection connection, OntologyVO ontologyVO) {
        this.connection = connection;
        this.ontologyVO = ontologyVO;

    }

public OntModel execute() throws InternalErrorException {


    ModelMaker maker = ModelFactory.createModelRDBMaker(connection);

    ModelRDB base = (ModelRDB) maker.createModel(ontologyVO.getOntologyName(),
false);

   OntModelSpec(OntModelSpec.OWL_DL_MEM_RULE_INF);

   OntModelSpec spec = OntModelSpec.getDefaultSpec(ProfileRegistry.RDFS_LANG);

   spec.setImportModelMaker(maker);

   OntModel ontModel = ModelFactory.createOntologyModel(spec, base);

        if (ontologyVO.getImportFromFile()) {

            InputStream inputStream = null;

            try {

             inputStream = new  FileInputStream(ontologyVO.getImportSource());

             ontModel.read(inputStream, "");
            } catch (FileNotFoundException ex) {
                throw new InternalErrorException(ex);
            } finally {
                try {
                    inputStream.close();
                } catch (IOException ex) {
                    throw new InternalErrorException(ex);
                }
            }
      } else {

      ontModel.read(ontologyVO.getImportSource(),ontologyVO.getLanguage());
```

```
        }
          return ontModel;
      }
}
.
.
.
```

Código Fuente 3.3. Fragmento de la clase OntologyFacade.java

3.3.5.4 - 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:

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

Incluye también un mecanismo para incluir nuevos razonadores. Para hacer inferencias debemos crear un Modelo Inferido a partir de un razonador.

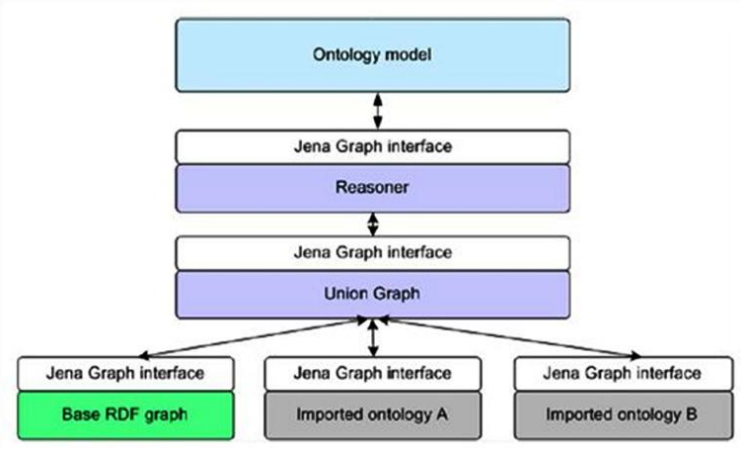

Figura 3.24. Inferencia en Jena

3.4 - Otras herramientas empleadas

3.4.1 – swREUSER

swREUSER (REUSER,2008) es una herramienta computer-aided software engineering (CASE) diseñada para permitir a los ingenieros de software el desarrollo de sistemas de información basados en el paradigma de la reutilización y utilizando la gestión del conocimiento apropiada. Por lo tanto, swREUSER incluya un completo conjunto de aditamentos para poder manejar software como si fuese un tipo más de representación de conocimiento.

Como herramientas CASE que soporta el paradigma de la Orientación a Objetos, swREUSER está completamente basada en el lenguaje UML (Unified Modeling Language), manteniendo una de las representaciones de su semántica más precisas del mercado.

Sin embargo, con el propósito de soportar gestión del conocimiento y reutilización, swREUSER permite llevar a cabo actividades de Ingeniería y Análisis de Dominios para recopilar una representación completa del conocimiento que la organización posee sobre un dominio dado. Esto es el núcleo de un completo repositorio que permite avanzadas técnicas de gestión dentro de la herramienta.

El repositorio gestionado por swREUSER (llamado REUSEServer) permite llevar a cabo actividades de reutilización institucional de alto nivel, incluyendo la reutilización de análisis, diseños, requisitos, riesgos… Por ello, el interfaz de recuperación proporcionado por esta familia de herramientas, dista en cierta medida de su visión más tradicional, permitiendo, por ejemplo, que un usuario cree un pequeño diagrama UML que será contrastado en el repositorio para devolver aquellos modelos que más se parecen a éste. Y todo esto, no basado en palabras claves ni en su descripción, sino basado en su contenido real.

SwREUSER cubre no solo las fases de Análisis y Diseño con UML, sino también otras etapas del ciclo de vida de desarrollo de software: gestión de requisitos, gestión

de riesgos, estimación de costes y recursos, conexión con la etapa de planificación, modelado mediante UML, control de los proyectos mediante diferentes métricas, generación de código, control de calidad… Además, se ha implementado un potente sistema de traza entre todos estos elementos.

Asimismo, la herramienta no ha olvidado la importancia de la Compartición del Conocimiento: modelos UML pueden ser compartidos, debatidos, comparados de forma automática y transformados utilizando un sistema completo de foros.

Teniendo en cuenta todo lo comentado, así como otras interesantísimas propiedades, swREUSER, junto con su metodología asociada ROM (Reuse Oriented Method) cubre la siguiente funcionalidad:

- Alta precisión semántica con UML.
- Sistema avanzado de Reutilización.
- Trabajo Colaborativo.
- Soporte a Requisitos.
- Gestión de Riesgos.
- Técnicas de Estimación de Proyectos.
- Patrones de Diseño.
- Comparador de Modelos UML.
- Generación de Código y Esquemas.
- Gestión de Casos de Prueba.
- Sistema de Foros Integrado.
- Conexión con www.umlmodels.org.
- Traza completa entre todos los elementos del ciclo de vida.

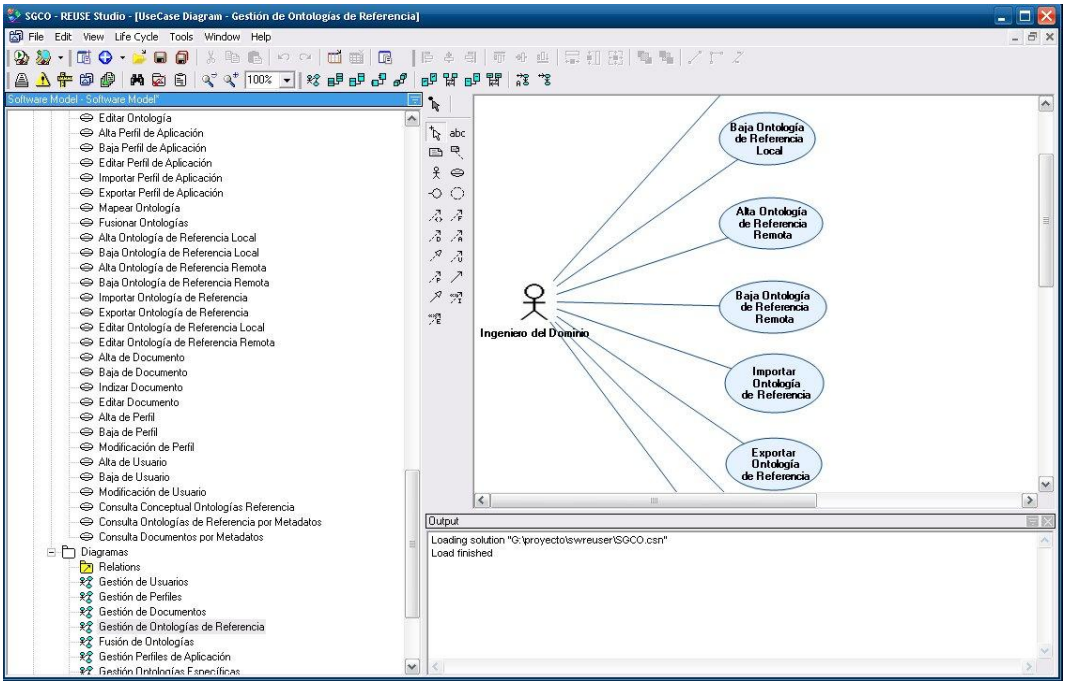

Figura 3.25. swREUSER

3.4.2 - Pgadmin III

PgAdmin III es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia Open Source. Está escrita en C++ usando la librería gráfica multiplataforma wxWidgets, lo 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.

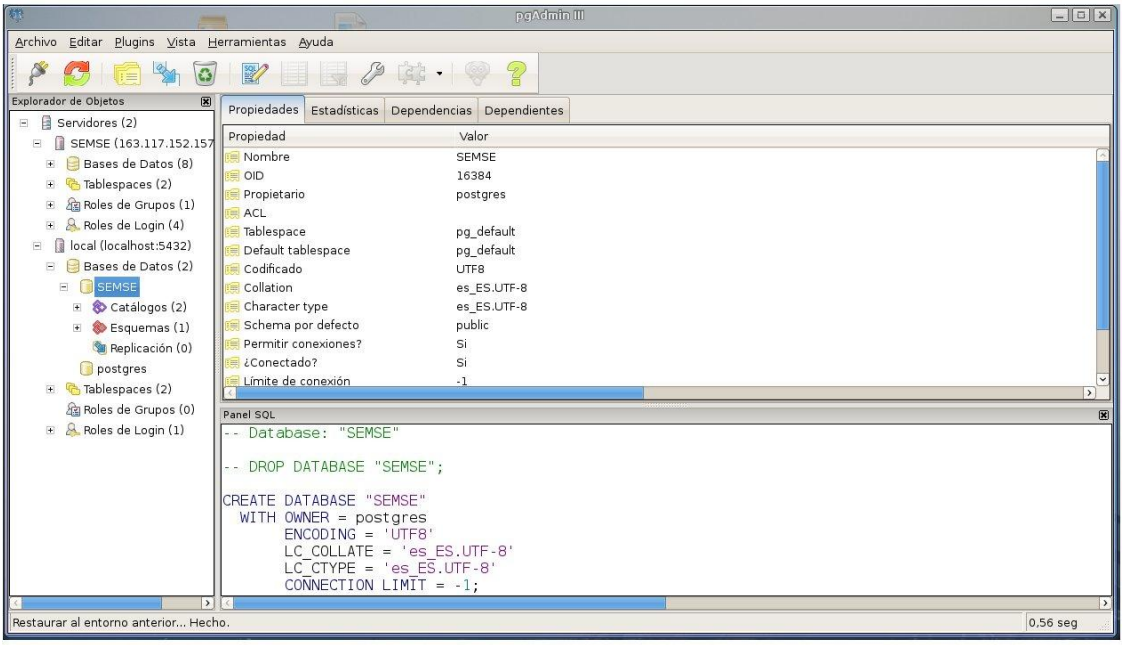

Figura 3.26. Pgadmin III

3.4.3 - Protégé

Protégé es un software libre, de código abierto que provee una plataforma respaldada por una gran comunidad de usuarios con un conjunto de herramientas para construir modelos de dominio y conocimiento de aplicaciones basadas en ontologías. En su núcleo, Protégé implementa un conjunto de conocimientos de modelado de estructuras y acciones que apoyan la creación, visualización y manipulación de ontologías en distintos formatos de representación.

Protégé se puede personalizar para ofrecer un soporte de fácil utilización en un dominio conocido, para la creación de modelos de conocimiento y la introducción de datos. Además, Protégé se puede ampliar por medio de un plug-in y una arquitectura basada en Java Application Programming Interface (API) para la construcción basada en el conocimiento de instrumentos y aplicaciones.

3.4.2.1 -Tipos principales de modelado de ontologías

Protégé soporta dos formas principales de modelado de ontologías:

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 marcos de un dominio basado en ontologías, personalizando los formularios de entrada de datos, y la entrada de los datos instanciados. 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 individuales de los conceptos que tienen valores específicos de sus propiedades.

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, siedo aprobado por el World Wide Web Consortium (W3C) para promover la visión de la Web Semántica.

Una ontología OWL puede incluir las descripciones de las clases, las propiedades y sus instancias. Habida cuenta de esta ontología, la semántica formal de OWL especifica cómo derivar sus consecuencias lógicas, es decir, hechos no literalmente presentes en la ontología, sino que implica la semántica. Estas vinculaciones se basarán en un documento único o múltiple documentos distribuidos, que se combinarán mediante mecanismos definidos en OWL.

El editor Protégé-OWL permite a los usuarios:

1. Cargar y salvar ontologías OWL y RDF.
2. Editar y visualizar clases, propiedades y reglas SWRL .
3. Definir la lógica de las clases mediante expresiones OWL.
4. Ejecutar razonadores como clasificadores lógicos de una clasificación.
5. Permitirá editar ontologías OWL individuales para marcado semántico.

3.4.2.2 - Características de Protégé

Protégé presenta las siguientes características.

- Requisito para la ejecución: Protégé 3.3.1 necesita JDK 1.5.
- Editor ProtégéFrames: Las principales funcionalidades que incluye son:

a) Amplio conjunto de elementos de interfaz de usuario que pueden ser personalizados para permitir a los usuarios modelar el conocimiento e introducir datos en formularios amigables de dominio.

b) Un plugin de arquitectura que se puede extender con elementos depersonalización diseñados, tales como componentes gráficos (por ejemplo, gráficos y tablas), media (por ejemplo, sonido, imágenes y vídeo), varios formatos de almacenamiento (por ejemplo, RDF, XML, HTML y componentes servidor o de servicio de bases de datos), y herramientas de soporte adicional (por ejemplo, para gestión de ontologías, visualización de ontologías, inferencia y razonamiento, etc.).

c) Una API basada en Java que permite a los plugins y otras aplicaciones acceder, usar y mostrar ontologías creadas con ProtégéFrames.

- Editor ProtégéOWL: Es una extensión de Protégé que soporta OWL. OWL es el lenguaje de ontologías estándar desarrollado más recientemente para promover la visión de Web Semántica. Este editor permite a los usuarios:

    a) Cargar y guardar ontologías OWL y RDF.

    b) Editar y visualizar clases, propiedades y reglas SWRL.

    c) Definir características de clases locales como expresiones OWL.

    d) Ejecutar razonadores como un clasificador de descripciones lógicas.

    e) Editar elementos OWL para marcado de Web Semántica.

- La flexibilidad de la arquitectura de ProtégéOWL hace más sencillo configurar y extender la herramienta. ProtégéOWL 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 servicios arbitrarios de Web Semántica.

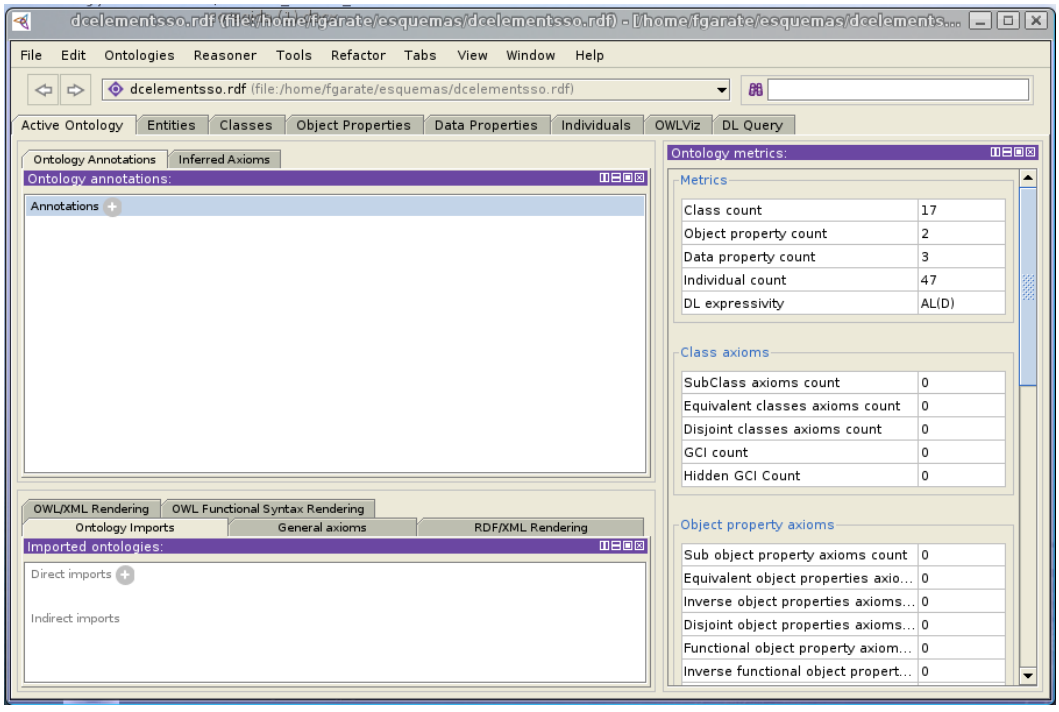

Figura 3.27. Protégé

Capítulo 4. Desarrollo del proyecto

El proyecto PFC, englobado dentro del proyecto SEMSE, está centrado en la implementación de una aplicación Web mediante la cual se pueda realizar la gestión y consulta de esquemas, tanto originales como cualificados de forma rápida y sencilla. Se proporciona así, una aplicación Web creada a partir de las herramientas más empleadas, potentes y novedosas en este tipo de desarrollos.

A continuación se presentarán las distintas fases llevadas a cabo en este proyecto, describiendo detalladamente el desarrollo y los productos finales de cada una de ellas.

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 intentará 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, resolviendo 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.

El objetivo último es que el sistema permita gestionar, reutilizar e interoperar esquemas, representados como esquemas semánticos, según la base ontológica definida para tal fin.

Además de lo anterior, el sistema deberá proporcionar una síntesis e integración de la información existente en las fuentes, una vez eliminadas redundancias sintácticas y semánticas, así como soportar consultas e inferencia.

Así pues, la aplicación y su entorno deberán contemplar 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 válido, 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 2.2.1. *"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 volumen de datos es impredecible por lo que se pensará en una solución que no nos ate a un Sistema Gestor de Base de Datos para, en caso de ser necesario, poder ampliar la funcionalidad a otros de mayor capacidad sin efectuar cambios en el código de la aplicación.

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 nos ate a un fabricante/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[48] para pasar a un diseño arquitectónico e implementación de modo iterativo-incremental[49].

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

1. **Complejidad:** Debido a la complejidad del proyecto y los recursos disponibles se ha preferido acometer su desarrollo de forma iterativa-incremental. De este modo, se aborda cada problema por separado, lo que permite disponer de entregables (artefactos) antes y corregir anomalías y desviaciones.

2. **Riesgo tecnológico**: existían riesgos derivados del desconocimiento de tecnologías existentes para la gestión de ontologías. 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. De esta fase surge una memoria que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos. Es importante señalar que en esta etapa se debe consensuar todo lo que se requiere del sistema.

[48] 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.

[49] 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.

- **Diseño del sistema** Se descompone y organiza el sistema en elementos que puedan elaborarse por separado. El resultado es la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.

- **Diseño del programa** Es la fase en donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario así como también los análisis necesarios para saber que herramientas usar en la etapa de codificación.

- **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 hacer que la programación sea un proceso mucho más rápido.

- **Pruebas** Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente antes de ser puesto en producción.

- **Implantación** El software obtenido se pone en producción. La implantación es la fase con más duración y con más cambios en el ciclo de elaboración de un proyecto.

4.1.2 - Especificación de requisitos de Usuario

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

Especificaciones informales

El objetivo principal de la aplicación será facilitar la gestión y manejo, de esquemas, con un ámbito heterogéneo, esto es, relativos a distintos dominios de conocimiento. La totalidad de sus funciones estarán relacionadas con el concepto de reutilizar la información de un dominio determinado. No será necesario que el usuario sea un experto en ontologías o en software de estas características para poder manejar con soltura la aplicación, es decir, la aplicación deberá potenciar la usabilidad, a través de interfaces intuitivos y de fácil manejo.

La aplicación deberá permitir compartir y gestionar representaciones ontológicas. El sistema propuesto permitirá realizar una representación ontológica de conocimiento expresada mediante distintos formalismos, inicialmente esquemas de metadatos, para posteriormente, relacionarla con ontologías de referencia, locales y remotas. Asimismo, el sistema permitirá que el usuario pueda crear sus propias conceptualizaciones, reutilizando el conocimiento ya incorporado en el sistema. La aplicación deberá proporcionar ayuda para manejar la aplicación.

4.1.3 - Diagrama de casos de uso inicial

Definidos los objetivos y requisitos de la aplicación 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 varios tipos de usuarios que pueden 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 consultar todos los esquemas y ontologías de la aplicación y subir sus propios esquemas originales.

- **Administrador:** Persona encargada de la gestión de usuarios en la aplicación, de la asignación de permisos y perfiles a éstos, y de la validación de los usuarios que pretendan trabajar con el sistema.

- **Experto responsable del dominio:** Persona con amplios conocimientos de un dominio en concreto. Los expertos dan de alta esquemas cualificados y editan tanto esquemas cualificados como originales.

- **Experto del dominio:** Es la persona que está al cargo de los expertos responsables del dominio. Además de sus atribuciones, son los encargados de validar los esquemas originales subidos por los usuarios y pueden eliminar tanto esquemas originales como cualificados.

- **Ingeniero responsable del dominio:** Persona encargada de dar de alta y gestionar las ontologías específicas.

- **Ingeniero del dominio:** Persona que además de las atribuciones del ingeniero responsable de dominio está encargada de dar de alta y gestionar la ontología de referencia, así como de poder eliminar las de referencia

- **Diseñador del dominio:** Usuario que abstrae en el sistema tanto a Ingenieros como a Expertos.

Esta jerarquía puede representarse de la siguiente forma:

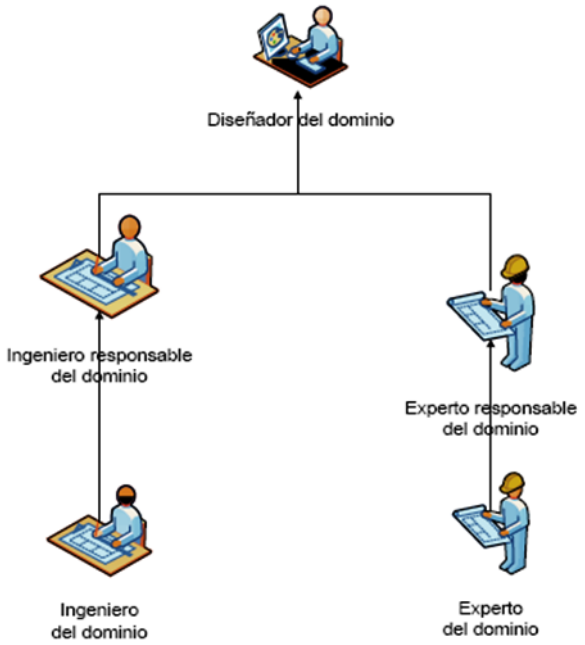

Figura 4.1. Jerarquía de roles del sistema

Los requisitos funcionales del sistema se muestran a continuación mediante diagramas de casos de uso. Al estar en la fase inicial, se muestra el diagrama de casos de uso para un usuario genérico, sin tener en cuenta el rol de dicho usuario dentro de la aplicación.

Figura 4.2. Diagrama de casos de uso

A continuación se ofrece una descripción de los casos de uso **Gestión de esquemas, Gestión de usuarios** y **Ayuda**, que aparecen que el diagrama anterior y es en los que está centrado este proyecto de final de carrera:

Gestión de esquemas: Permite la creación, edición, eliminación, consulta y búsqueda de esquemas, tanto originales como cualificados.

Gestión esquemas originales

- **Esquemas no validados.**
  - Buscar por nombre.
  - Buscar por metadatos.
  - Añadir un esquema original.
  - Eliminar esquema/as original/es.
  - Editar los metadatos un esquema original.
  - Mostrar los metadatos asociados al esquema original.
  - Ver el contenido de un esquema en otra pestaña (ventana) del navegador.
  - Descargar un esquema original.
  - Validar un esquema original.

- **Esquemas validados.**
  - Buscar por nombre.
  - Buscar por metadatos.
  - Añadir un esquema origina.
  - Eliminar esquema/as original/es.
  - Editar los metadatos un esquema original.
  - Mostrar los metadatos asociados al esquema original.

- Ver el contenido de un esquema en otra pestaña (ventana) del navegador.
- Descargar un esquema original.

.

Gestión esquemas cualificados

- Buscar por nombre.
- Buscar por metadatos.
- Añadir un esquema cualificado
- Eliminar esquema/as cualificado/os.
- Editar los metadatos un esquema cualificado.
- Mostrar los metadatos asociados al esquema cualificado.
- Ver el contenido de un esquema en otra pestaña (ventana) del navegador.
- Descargar un esquema cualificado.

Gestión de usuarios: Permite el alta, baja y edición de los distintos usuarios de la aplicación.

- Alta de nuevos usuarios.
- Baja de antiguo/os usuario/os.
- Editar los datos de usuarios registrados en el sistema.
- Validación de usuarios que se registran en la aplicación.

Ayuda: En cualquier momento de interacción con el sistema el usuario tiene la posibilidad de consultar la ayuda que ofrece la aplicación sobre los temas que puedan resultar más complicados en cuanto al manejo de ésta.

Ayuda esquemas originales

- Ayuda gestión de esquemas originales.
- Ayuda alta de un esquema original.
- Ayuda eliminar esquema/as originala/es.
- Ayuda editar los metadatos de un esquema original.
- Ayuda mostrar los metadatos de un esquema original.
- Ayuda validar un esquema original.
- Ayuda descargar un esquema original.

Ayuda esquemas cualificados

- Ayuda gestión de esquemas cualificados.
- Ayuda alta de un esquema cualificado.
- Ayuda eliminar esquema/as cualificado/os.
- Ayuda editar los metadatos de un esquema cualificado.
- Ayuda mostrar los metadatos de un esquema cualificado.
- Ayuda descargar un esquema cualificado.

4.2- Fase de análisis

En esta fase se presenta una visión general del proceso de análisis, paso previo a la implementación de la aplicación. Se presenta un diagrama de casos de uso mucho más completo, incluyendo también la jerarquía definitiva de usuarios.

4.2.1- Diagrama de casos de uso en la fase de análisis

En esta fase se decidió realizar una primera especificación de requisitos completa, para minimizar riesgos derivados de realizar especificaciones de requisitos incrementales tales como: aparición de nuevos requisitos, incompatibilidad de los nuevos requisitos con los existentes, etc., que pueden llevar al rediseño de partes ya desarrolladas. 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 en la Figura 4.3. *"Diagrama de casos de uso y jerarquía de usuarios"*.

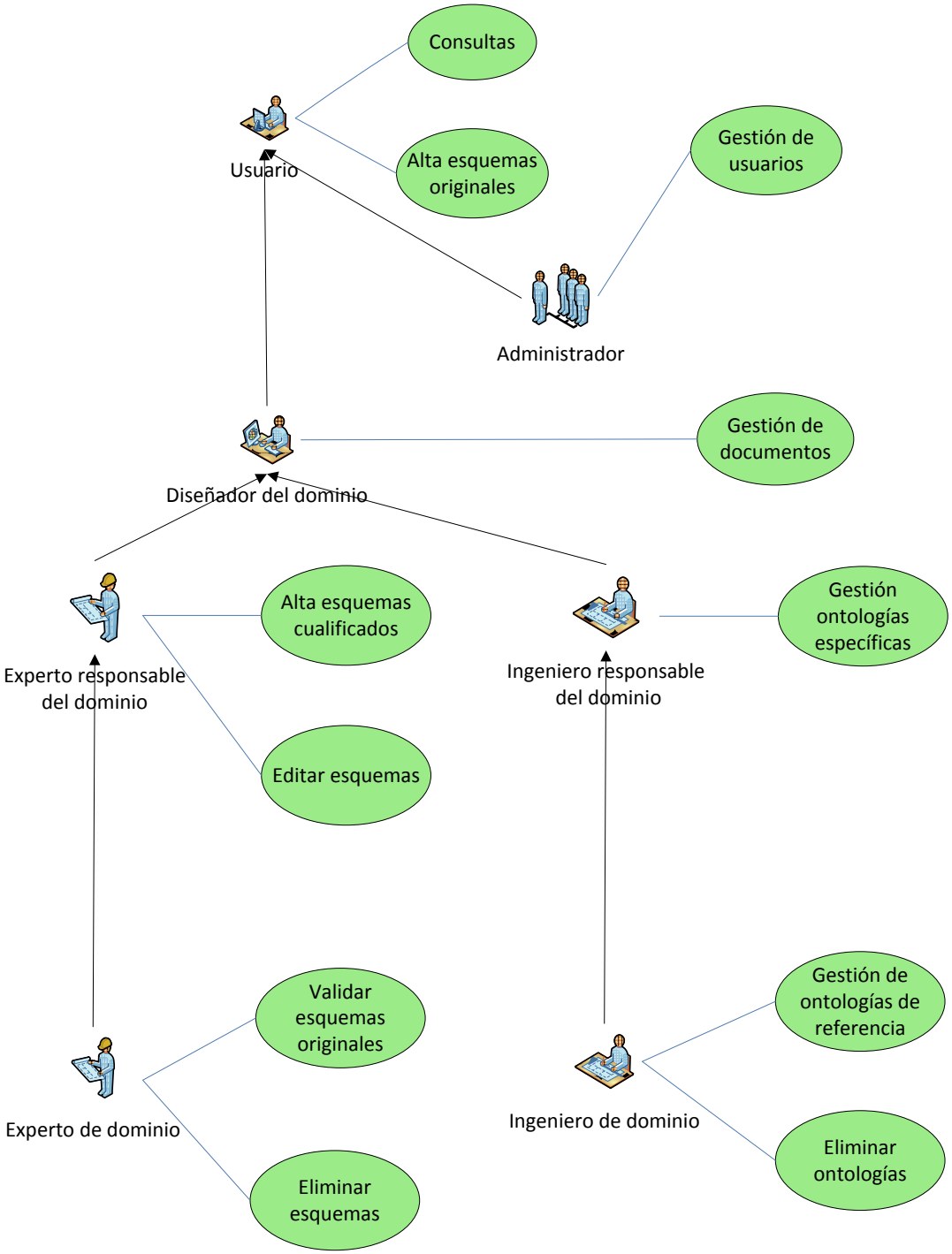

Figura 4.3. Diagrama de casos de uso y jerarquía de usuarios

A continuación, se ofrece una descripción más detallada del caso de uso ***Gestión de esquemas***. Para cada caso 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. 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.

Los casos de uso abstraen las funcionalidades generales del sistema, y cada uno de ellos incluye sus respectivos requisitos, que veremos en el siguiente punto.

A continuación se muestran los casos de uso identificados en el sistema:

CU-01: Consultas			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Descripción:	Caso de uso que abstrae el conjunto de consultas del sistema		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0101: Consulta Esquema Original por Nombre RUC-0102: Consulta Esquema Original por Metadatos RUC-0103: Consulta Esquema Cualificado por Nombre RUC-0104: Consulta Esquema Cualificado por Metadatos			

Tabla 4.1. CU-01: Consultas

CU-02: Gestión Esquemas Originales			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Descripción:	Caso de uso que abstrae los CUs relacionados con la gestión de esquemas originales		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0201: Añadir esquema original RUC-0202: Eliminar esquema original RUC-0203: Mostrar metadatos esquema original RUC-0204: Editar esquema original RUC-0205: Descargar esquema original RUC-0206: Abrir esquema original RUC-0207: Validar esquema original			

Tabla 4.2. CU-02: Gestión Esquemas Originales

CU-03: Gestión Esquemas Cualificados			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Descripción:	Caso de uso que abstrae los CUs relacionados con la gestión de esquemas cualificados		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0301: Añadir esquema cualificado RUC-0302: Eliminar esquema cualificado RUC-0303: Mostrar metadatos esquema cualificado RUC-0304: Editar esquema cualificado RUC-0305: Descargar esquema cualificadol RUC-0306: Abrir esquema cualificado			

Tabla 4.3. CU-03: Gestión Esquemas Cualificados

CU-04: Gestión de Usuarios			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Descripción:	Caso de uso que abstrae los CUs relacionados con la gestión de usuarios		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0401: Alta Nuevos Usuarios. RUC-0402: Baja Antiguos Usuarios. RUC-0403: Editar Usuarios. RUC-0404: Validación de usuarios.			

Tabla 4.4. CU-04: Gestión de usuarios

CU-05: Ayuda			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Descripción:	Caso de uso que abstrae los temas de ayuda del sistema		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0501: Ayuda Esquemas Originales RUC-0502: Ayuda Esquemas Cualificados			

Tabla 4.5. CU-05: Ayuda

4.2.2 - Especificación de requisitos de usuario en la fase de análisis

A continuación se enumeran los requisitos de usuario que la aplicación debe cumplir generados a partir de los casos de uso anteriores.

Los requisitos de usuario detallan la funcionalidad de cada uno de los casos de uso, se muestran a continuación agrupados por funcionalidad:

Consultas

RUC-0101: Consulta Esquema Original por Nombre			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Obtener los distintos esquemas originales cuyo nombre coincida con el introducido por el usuario como criterio de búsqueda.			
Escenario básico:	1. Introducir el nombre que desea buscar 2. Realizar la búsqueda entre los esquemas originales 3. Mostrar resultados		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-01: Consultas RUC-1101: Formato consultas			

Tabla 4.6. RUC-0101: Consulta Esquema Original por Nombre

RUC-0102: Consulta Esquema Original por Metadatos			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Obtener los distintos esquemas originales cuyos metadatos coincida con los introducidos por el usuario como criterio de búsqueda.			
Escenario básico:	1. Introducir los metadatos que desea buscar 2. Realizar la búsqueda entre los esquemas originales 3. Mostrar resultados		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-01: Consultas RUC-1101: Formato consultas			

Tabla 4.7. RUC-0102: Consulta Esquema Original por Metadatos

RUC-0103: Consulta Esquema Cualificado por Nombre			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Obtener los distintos esquemas cualificados cuyo nombre coincida con el introducido por el usuario como criterio de búsqueda.			
Escenario básico:	1. Introducir el nombre que desea buscar 2. Realizar la búsqueda entre los esquemas cualificados 3. Mostrar resultados		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-01:Consultas RUC-1101: Formato consultas			

Tabla 4.8. RUC-0103: Consulta Esquema Cualificado por Nombre

RUC-0104: Consulta Esquema Cualificado por Metadatos			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Obtener los distintos esquemas cualificados cuyos metadatos coincidan con los introducidos por el usuario como criterio de búsqueda.			
Escenario básico:	1. Introducir los metadatos que desea buscar 2. Realizar la búsqueda entre los esquemas cualificados 3. Mostrar resultados		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-01:Consultas RUC-1101: Formato consultas			

Tabla 4.9. RUC-0104: Consulta Esquema Cualificado por Nombre

Gestión Esquemas Originales

RUC-0201: Añadir Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Dar de alta o importar un esquema original en el sistema. El esquema original se podrá dar de alta a partir de fichero o URL.			
Escenario básico:	1. Introducir el nombre del esquema original 2. Introducir el nombre y el correo electrónico del autor 3. Seleccionar la fuente del esquema original: fichero o URI 4. Introducir los datos del esquema, dominio, versión, comentarios 5. Dar de alta el esquema		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1201: Metadatos Esquema. RUC-1301: Formalismos Soportados			

Tabla 4.10. RUC-0201: Añadir Esquema Original

RUC-0202: Eliminar Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Eliminar un esquema original del sistema			
Escenario básico:	1. Listar esquemas originales 2. Seleccionar esquema original 3. Borrar el esquema original 4. Guardar cambios		
Precondiciones:	Existe al menos un esquema original en el sistema		
Pos condiciones:	El esquema original queda eliminado del sistema		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1201: Metadatos Esquema.			

Tabla 4.11. RUC-0202: Eliminar Esquema Original

RUC-0203: Mostrar Metadatos Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Muestra los metadatos de un esquema original del sistema			
Escenario básico:	1. Listar esquemas originales 2. Seleccionar esquema original 3. Ver metadatos del esquema original 4. Guardar cambios		
Precondiciones:	Existe al menos un esquema original en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1201: Metadatos Esquema.			

Tabla 4.12. RUC-0203: Mostrar Metadatos Esquema Original

RUC-0204: Editar Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
media	Modificable	Usuario	1.0
Objetivo:			
Poder editar un esquema original del sistema			
Escenario básico:	1. Seleccionar un esquema original. 2. Editar los metadatos del esquema original. 3. Guardar cambios.		
Precondiciones:	Existe al menos un esquema original en el sistema		
Pos condiciones:	El esquema original queda almacenado con las modificaciones		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1201: Metadatos Esquema.			

Tabla 4.13. RUC-0204: Editar Esquema Original

RUC-0205: Descargar Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder descargar un esquema original almacenado en el sistema			
Escenario básico:	1. Seleccionar el esquema original. 2. Confirmar la operación.		
Precondiciones:	Existe al menos un esquema original en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-02: Gestión Esquemas Originales.			

Tabla 4.14. RUC-0205: Descargar Esquema Original

RUC-0206: Abrir Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder abrir en una pestaña del navegador un esquema original almacenado en el sistema			
Escenario básico:	1. Seleccionar el esquema original. 2. Confirmar la operación.		
Precondiciones:	Existe al menos un esquema original en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1301: Formalismos Soportados			

Tabla 4.15. RUC-0206: Abrir Esquema Original

RUC-0207: Validar Esquema Original			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder validar un esquema original sin validar almacenado en el sistema			
Escenario básico:	1. Seleccionar el esquema original sin validar. 2. Validar esquema original. 3. Guardar los cambios.		
Precondiciones:	Existe al menos un esquema original sin validar en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-02: Gestión Esquemas Originales. RUC-1301: Formalismos Soportados			

Tabla 4.16. RUC-0207: Validar Esquema Original

Gestión Esquemas Cualificados

RUC-0301: Añadir Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Dar de alta o importar un esquema cualificado en el sistema. El esquema cualificado se podrá dar de alta a partir de fichero o URL.			
Escenario básico:	1. Seleccionar esquema original asociado. 2. Introducir el nombre del esquema cualificado. 3. Introducir el nombre y el correo electrónico del autor. 4. Seleccionar la fuente del esquema cualificado: fichero o URI. 5. Introducir los datos del esquema, versión, comentarios. 6. Dar de alta el esquema		
Precondiciones:	Existe al menos un esquema original validado en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados. RUC-1201: Metadatos Esquema.			

Tabla 4.17. RUC-0301: Añadir Esquema Cualificado

RUC-0302: Eliminar Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Eliminar un esquema cualificado del sistema			
Escenario básico:	1. Listar esquemas cualificados 2. Seleccionar esquema cualificados 3. Borrar el esquema cualificados 4. Guardar cambios		
Precondiciones:	Existe al menos un esquema cualificado en el sistema		
Pos condiciones:	El esquema cualificado queda eliminado del sistema		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados. RUC-1201: Metadatos Esquema.			

Tabla 4.18. RUC-0302: Eliminar Esquema Cualificado

RUC-0303: Mostrar Metadatos Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Muestra los metadatos de un esquema cualificado del sistema			
Escenario básico:	1. Listar esquemas cualificados 2. Seleccionar esquema cualificado 3. Ver metadatos del esquema cualificado 4. Guardar cambios		
Precondiciones:	Existe al menos un esquema cualificado en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados. RUC-1201: Metadatos Esquema.			

Tabla 4.19. RUC-0303: Mostrar Metadatos Esquema Cualificado

RUC-0304: Editar Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
media	Modificable	Usuario	1.0
Objetivo:			
Poder editar un esquema cualificado del sistema			
Escenario básico:	1. Seleccionar un esquema cualificado. 2. Editar los metadatos del esquema cualificado. 3. Guardar cambios.		
Precondiciones:	Existe al menos un esquema cualificado en el sistema		
Pos condiciones:	El esquema cualificado queda almacenado con las modificaciones		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados. RUC-1201: Metadatos Esquema.			

Tabla 4.20. RUC-0304: Editar Esquema Cualificado

RUC-0305: Descargar Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder descargar un esquema cualificado almacenado en el sistema			
Escenario básico:	1. Seleccionar el esquema cualificado. 2. Confirmar la operación.		
Precondiciones:	Existe al menos un esquema cualificado en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados.			

Tabla 4.21. RUC-0305: Descargar Esquema Cualificado

RUC-0306: Abrir Esquema Cualificado			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder abrir en una pestaña del navegador un esquema cualificado almacenado en el sistema			
Escenario básico:	1. Seleccionar el esquema cualificado. 2. Confirmar la operación.		
Precondiciones:	Existe al menos un esquema cualificado en el sistema		
Pos condiciones:	--------		
Relacionado con:			
CU-03: Gestión Esquemas Cualificados. RUC-1301: Formalismos Soportados			

Tabla 4.22. RUC-0306: Exportar Esquema Cualificado

Gestión de usuarios

RUC-0401: Alta Nuevos Usuarios.			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Dar de alta a nuevos usuarios dentro del portal			
Escenario básico:	1. Introducir los datos de usuario. 2. Introducir la contraseña del usuario. 3. Confirmar la contraseña del usuario. 4. Asignar un rol al usuario. 5. Dar de alta al usuario.		
Precondiciones:	Existe al menos un usuario en el sistema con permisos para dar de alta a nuevos usuarios.		
Pos condiciones:	Hay un nuevo usuario en el sistema.		
Relacionado con:			
CU-04: Gestión Usuarios. RUC-1401: Datos Usuario.			

Tabla 4.23. RUC-0401: Alta Nuevos Usuarios

RUC-0402: Baja Antiguos Usuarios.			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Dar de baja a antiguos usuarios del portal			
Escenario básico:	1. Listar usuarios 2. Seleccionar el usuario 3. Borrar el usuario 4. Guardar cambios		
Precondiciones:	Existe al menos un usuario en el sistema con permisos para dar de baja a antiguos usuarios y al menos otro usuario al que se da de baja		
Pos condiciones:	El usuario queda eliminado del sistema.		
Relacionado con:			
CU-04: Gestión Usuarios. RUC-1401: Datos Usuario.			

Tabla 4.24. RUC-0402: Baja Antiguos Usuarios

RUC-0403: Editar Usuarios.			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder editar un usuario del sistema			
Escenario básico:	1. Seleccionar un usuario. 2. Editar los datos del usuario. 3. Guardar cambios.		
Precondiciones:	Existe al menos un usuario en el sistema con permisos para editar usuarios, y al menos otro usuario al que se va a editar.		
Pos condiciones:	El usuario queda almacenado con las modificaciones.		
Relacionado con:			
CU-04: Gestión Usuarios. RUC-1401: Datos Usuario.			

Tabla 4.25. RUC-0403: Editar Usuarios

RUC-0404: Validar Usuarios.			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Un usuario deberá autenticarse para acceder al sistema, mediante la introducción de login y password. Dependiendo del perfil del usuario podrá acceder a las acciones permitidas, en la definición de su perfil.			
Escenario básico:	1. Introducir login y password 2. Chequear los credenciales del usuario. 3. El usuario accederá al sistema con las funcionalidades concedidas a su perfil.		
Precondiciones:	Existe al menos un usuario en el sistema.		
Pos condiciones:	El usuario queda validado en el sistema.		
Relacionado con:			
CU-04: Gestión Usuarios. RUC-1401: Datos Usuario.			

Tabla 4.26. RUC-0404: Validar Usuarios

Ayuda

RUC-0501: Ayuda Esquemas Originales			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder consultar la ayuda relativa a los esquemas originales			
Escenario básico:	1. Ir a la página de Ayuda. 2. Seleccionar la ayuda sobre esquemas originales 3. Consultar el tema de ayuda		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-05: Ayuda.			

Tabla 4.27. RUC-0501: Ayuda Esquemas Originales

RUC-0502: Ayuda Esquemas Cualificados			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Poder consultar la ayuda relativa a los esquemas cualificados			
Escenario básico:	1. Ir a la página de Ayuda. 2. Seleccionar la ayuda sobre esquemas cualificados 3. Consultar el tema de ayuda		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
CU-05: Ayuda			

Tabla 4.28. RUC-0402: Ayuda Esquemas Cualificados

Otros requisitos

RUC-1101: Formato Consultas			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
El resultado de las consultas se visualizará en forma de listado.			
Escenario básico:	--------		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0101: Consulta Esquema Original por Nombre RUC-0102: Consulta Esquema Original por Metadatos RUC-0103: Consulta Esquema Cualificado por Nombre RUC-0104: Consulta Esquema Cualificado por Metadatos			

Tabla 4.29. RUC-1101: Formato Consultas

RUC-1201: Metadatos Esquema			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Alta	Baja	Usuario	1.0
Objetivo:			
Todo esquema registrado en el sistema llevará asociado un conjunto de metadatos como resumen del mismo. Los metadatos incluirán: nombre del esquema, nombre de autor, correo electrónico del autor, fecha de alta, enlace al esquema, versión, dominio y documentos asociados.			
Escenario básico:	--------		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0201: Añadir Esquema Original RUC-0202: Eliminar Esquema Original RUC-0203: Mostrar Metadatos Esquema Original RUC-0204: Editar Esquema Original RUC-0301: Añadir Esquema Cualificado RUC-0302: Eliminar Esquema Cualificado RUC-0203: Mostar Metadatos Esquema Cualificado RUC-0304: Editar Esquema Cualificado			

Tabla 4.30. RUC-1201: Metadatos Esquema

RUC-1301: Formalismos Soportados			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Media	modificable	Usuario	1.0
Objetivo:			
La aplicación debe permitir Importar y exportar un esquema original o cualificado en los formalismos definidos en el sistema.			
Escenario básico:	--------		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0201: Añadir Esquema Original RUC-0206: Abrir Esquema Original RUC-0207: Validar Esquema Original RUC-0301: Añadir Esquema Cualificado RUC-0306: Abrir Esquema Cualificado			

Tabla 4.31. RUC-1301: Formalismos Soportados

RUC-1401: Datos Usuario			
Prioridad	**Estabilidad**	**Fuente**	**Versión**
Media	modificable	Usuario	1.0
Objetivo:			
Todo usuario registrado en el sistema llevará asociado una serie de datos que lo identifican. Los datos serán: nombre, apellidos, correo electrónico, login, contraseña y perfil.			
Escenario básico:	--------		
Precondiciones:	--------		
Pos condiciones:	--------		
Relacionado con:			
RUC-0401: Alta Nuevo Usuario RUC-0402: Baja Antiguo Usuario RUC-0403: Editar Usuario RUC-0404: Validar Usuario			

Tabla 4.32. RUC-1401: Datos Usuario

4.3- Diseño arquitectónico

Esta fase del proyecto comprende la última versión de la arquitectura definida para el soporte de la aplicación. En esta sección se pretende dar una visión global de la descomposición en componentes del sistema, así como del despliegue de los mismos, haciendo uso de los estilos arquitectónicos definidos en el Capítulo 2 *"Estado del arte"*.

En el desarrollo del sistema se ha aplicado el patrón MVC descrito en el Apartado 2.2.3. *"Estilos y patrones de diseño"*. 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 JSF e ICEFaces que ofrece una manera sencilla y clara de generar páginas Web, mientras que para la capa de infraestructura se ha empleado Jena e Hibernate.

Entre la vista y el modelo se sitúa la capa de control. Es precisamente aquí donde se integran los frameworks utilizados: Spring y JSF, los cuales han sido descritos en el Apartado 3.3. *"Frameworks empleados"*.

Para la capa de persistencia 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 4.4. *"División en capas de la aplicación",* se muestra el particionado en capas de la aplicación y las tecnologías empleadas en cada una de ellas.

Figura 4.4. División en capas de la aplicación

4.3.1- Infraestructura Hardware

Como se indicó en el Capítulo 3 *"Herramientas para la elaboración del proyecto"* se parte de una infraestructura hardware compuesta de un PC (a modo de servidor) adquirido para este proyecto y que satisface las necesidades que se requieren para la elaboración del mismo.

Cabe destacar que dicho PC se encuentra ubicado en un entorno estable y con medidas de seguridad adecuadas, para garantizar la disponibilidad y la integridad del mismo.

4.3.2- Infraestructura Software

Las decisiones en cuanto a software se comentaron en el Capítulo 3 *"Herramientas para la elaboración del proyecto"* y se resumen en el siguiente listado:

- 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 Hibernate para la persistencia de los distintos metadatos e identificadores que definen los esquemas y las ontologías, así como el mecanismo para almacenar los distintos usuarios dentro de la aplicación.

- JSF se utiliza como marco para el desarrollo del sistema basado en un modelo arquitectónico Modelo-Vista-Controlador. Su elección se debe a que permite la programación de la interfaz a través de componentes basados en eventos proporcionando una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario, y manejar eventos, entre otras muchas características, la implementación concreta de JSF que se ha seleccionado ha sido ICEFaces cuya suite de componentes permite realizar interfaces semejantes a las de las aplicaciones de escritorio de forma rápida y sencilla.

- 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.

- 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 6.5 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 aplicaciones Web orientadas a objetos al estilo swing.

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 se vinculan con la capa de control a través del framework JSF, que controla las peticiones del usuario. De igual manera la capa de control está enlazada con todas las clases de acción. Dichas clases delegan la petición del usuario a la capa de negocio, que 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 de los frameworks de Jena e Hibernate. Desde estas clases, que conforma el modelo del dominio, se accede a la Base de Datos (PostgreSQL) para operaciones de inserción, borrado y consulta.

4.4- Diseño detallado

Este apartado trata de cómo se ha implementado la funcionalidad requerida, se analizarán los aspectos más importantes a la hora de integrar los distintos frameworks, así como una relación de las clases empleadas.

JSF permite el mapeo de las acciones que se invocan desde los backing beans de las páginas JSF, permitiendo la ejecución de los métodos asociados a ellas. JSF también define la navegación entre páginas mediante reglas de navegación. Para ello se ha configurado el fichero faces-config.xml de la siguiente manera:

```
<managed-bean>
    <managed-bean-name>GestionEsquemasCualificados</managed-bean-name>
    <managed-bean-class>semse.GestionEsquemasCualificados</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>

    <managed-property>
      <property-name>metadatosService</property-name>
      <value>#{metadatosService}</value>
    </managed-property>

    <managed-property>
      <property-name>ontologyFacade</property-name>
      <value>#{ontologyFacade}</value>
    </managed-property>

    <managed-property>
      <property-name>esquemaOriginalService</property-name>
      <value>#{esquemaOriginalService}</value>
    </managed-property>

    <managed-property>
      <property-name>esquemaCualificadoService</property-name>
      <value>#{esquemaCualificadoService}</value>
    </managed-property>
</managed-bean>
```

Código Fuente 4.1. Configuración de los backing bean en JSF

En este fragmento se observa cómo se configura el backing bean GestionEsquemasOriginales, cada una de las propiedades corresponden con variables definidas en el bean el de esta manera no es necesario instanciar el objeto, siendo Spring quien hace la inyección de dependencias.

```
<navigation-rule>
    <from-view-id>/GestionEsquemasCualificados.jsp</from-view-id>

    <navigation-case>
       <from-outcome>cancelar</from-outcome>
       <to-view-id>/GestionEsquemasOriginales.jsp</to-view-id>
       <redirect />
    </navigation-case>

    <navigation-case>
       <from-outcome>editar</from-outcome>
       <to-view-id>/EditarEsquemaCualificado.jsp</to-view-id>
       <redirect />
    </navigation-case>

    <navigation-case>
       <from-outcome>ver</from-outcome>
       <to-view-id>/DatosEsquemaCualificado.jsp</to-view-id>
    </navigation-case>

    <navigation-case>
       <from-outcome>alta</from-outcome>
       <to-view-id>/AltaEsquemasCualificados.jsp</to-view-id>
       <redirect />
    </navigation-case>
</navigation-rule>
```

Código Fuente 4.2. Configuración de la navegación en JSF

Netneans permite visualizar y editar gráficamente la navegación definida en el fichero faces-config.xml mediante la opción *PageFlow,* donde mediante un diagrama con las distintas páginas de la aplicación se puede definir cómo será la navegación. En la Figura 4.5. *"Diagrama de navegación definido en el fichero faces-config.xml"* se

muestra el diagrama correspondiente a las páginas involucradas en la gestión de esquemas.

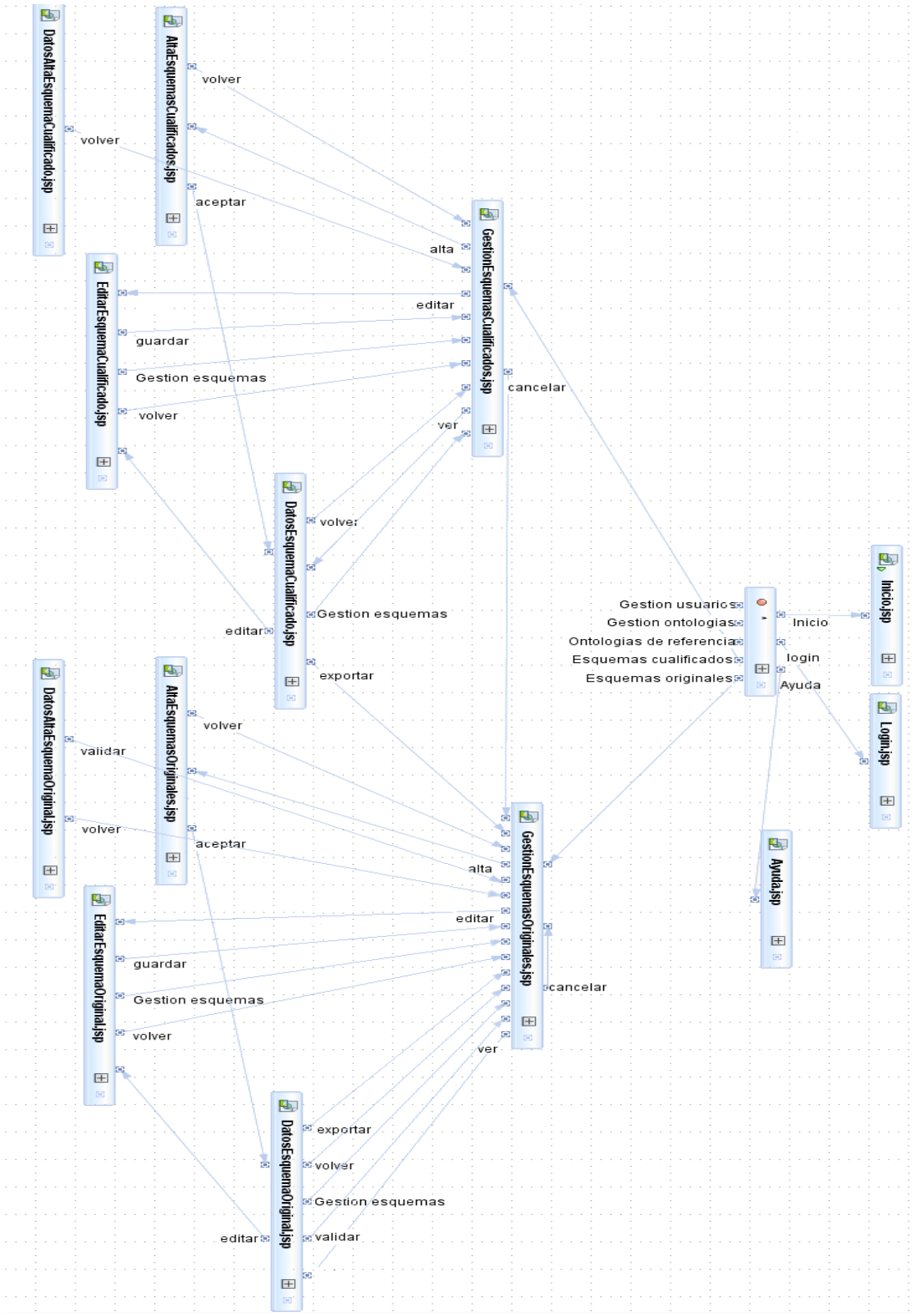

Figura 4.5. Diagrama de navegación definido en el fichero faces-config.xml

Mediante este diagrama se puede determinar el número de páginas de la aplicación y las opciones de navegación de una forma rápida y sencilla, permitiendo la creación de páginas, así como la modificación de las distintas reglas de navegación.

```
public String botonEditar() {

    metadatoSeleccionado = (Metadatos) metadatosSeleccionados.get(0);

    HttpSession session = null;

session=(HttpSession)FacesContext.getCurrentInstance().getExternalCont
ext().getSession(true);

    session.setAttribute("EditarMetadatos", metadatoSeleccionado);

    this.setDesactivarVerEditarExportar(true);

    this.setDesactivarEliminar(true);

    return "editar";

}
```

Código Fuente 4.3. Método botonEditar

En este fragmento se configuran las acciones que se realizan cuando se pulsa el botón editar en la página GestionEsquemasCualificados.jsp, en el momento en que se pulsa el botón editar de la página Web se ejecuta el código del método botonEditar y devuelve el valor *"editar",* JSF recoge este valor y direcciona a la página según indique el fichero faces-config.xml, en este caso, a la página EditarEsquemaCualificado.jsp.

4.4.1– Diagramas de clases

A partir de los casos de uso identificados en el Apartado 4.2.2. *"Especificación de requisitos de usuario en la fase de análisis"* se ha desarrollado el diagrama de clases, incluido en la Figura 4.6. "*Diagrama de clases de la aplicación"*.

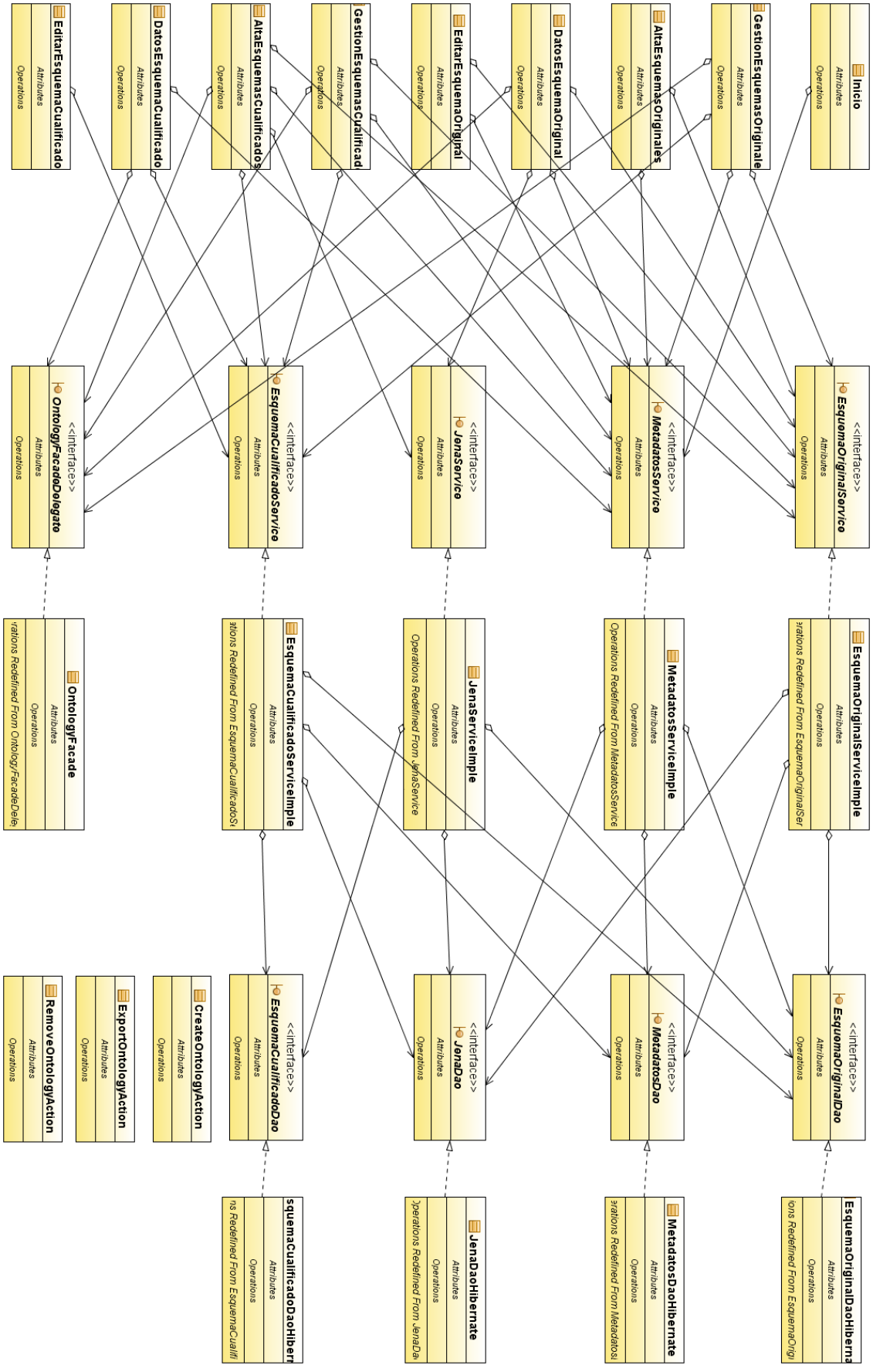

Figura 4.6. Diagrama de clases de la aplicación

A cada una de las páginas del portal le corresponde una clase denominada backing bean, en estos backing beans se controlan las distintas acciones que realiza el usuario, como pulsar el botón de alta de un esquema, en función de la acción se ejecuta un código que comprueba los parámetros y ejecuta el código de la acción, normalmente es necesario colaborar con otras clases que nos permiten acceder a la base de datos, comprobar que los datos son correctos, etc.

Los backing bean se relacionan con dos interfaces, los Services y los FacadeDelegate. Los primeros se utilizan para las relaciones con Hibernate y los segundos para las relaciones con Jena. Estos interfaces están implementados por las clases ServiceImple y Facade respectivamente, y a su vez están relacionadas con los interfaces Dao y clases DaoHibernate. Los interfaces Dao definen los métodos que se utilizarán para operar sobre la base de datos y son implementados por las clases DaoHibernate.

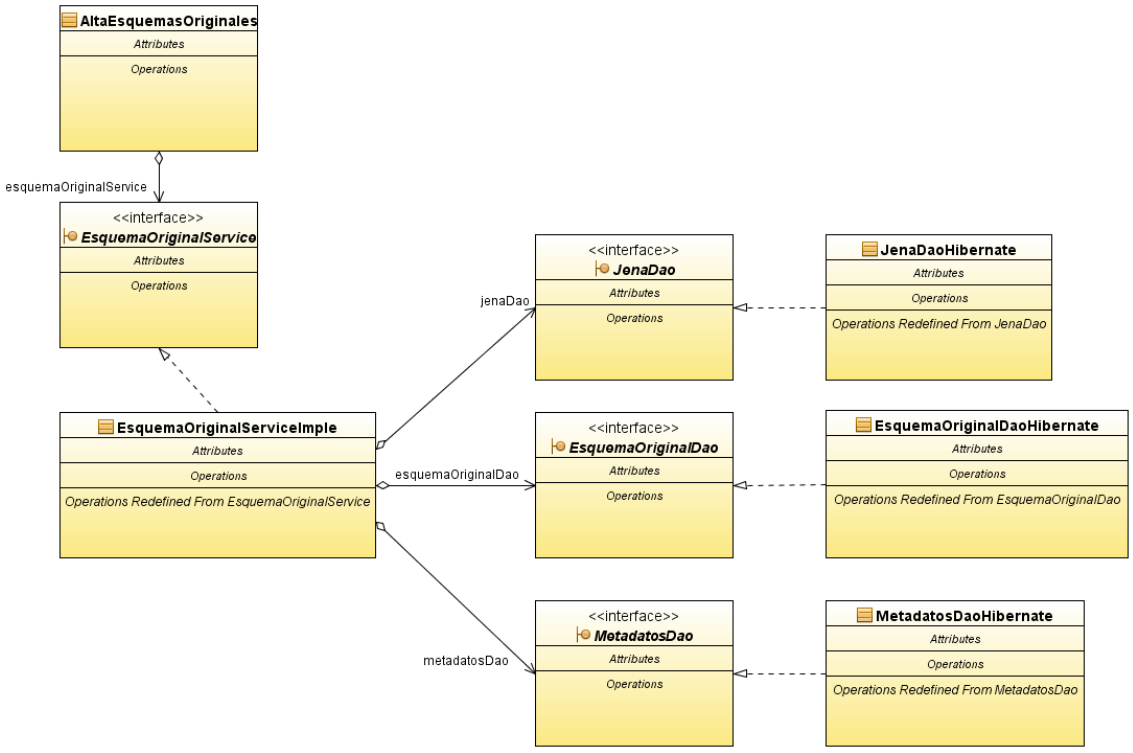

Figura 4.7. Diagrama de clases del caso de uso alta de esquema original

En primer lugar aparece el backing bean AltaEsquemaOriginales.java que define y controla las acciones que se realizan en la página AltaEsquemasOriginales.jsp. Cuando se pulsa el botón *"alta"* se ejecuta el código que se encuentra en el método *botonAlta* donde una vez comprobado que están correctamente introducidos los datos necesarios para proceder a dar de alta un esquema se invoca el método *agregarEsquemasOriginales* definido en el interfaz EsquemaOriginalService e implementado en la clase EsquemaOriginalServiceImple

```
try {

   esquemaOriginalService.agregarEsquemasOriginales(metadatos);

} catch (InstanceNotFoundException ex) {
Logger.getLogger(AltaEsquemasOriginales.class.getName()).log(Level.SEVERE,
null, ex);         }
```

Código Fuente 4.4. Fragmento de la clase AltaEsquemasOriginales.java

En este fragmento de código se invoca al método agregarEsquemasOriginales y pasándole como parámetro los metadatos que definen el esquema original.

```
public interface EsquemaOriginalService {

      public void agregarEsquemasOriginales(Metadatos metadatos)

      public void validarEsquemasOriginales(Metadatos metadatos)

      public Esquemacualificado seleccionarEsquemaCualificado(Esquemaoriginal esquema)

      public void eliminarEsquemasOriginales(Metadatos metadatosEliminar)

      public boolean buscaNombre(String nombre)

      public List<Metadatos> listarMetadatosEsquemas()

      public List<Metadatos> consultaPorMetadatos(Metadatos metaConsulta)

      public List<Metadatos> listarMetadatosEsquemasValidados()

      public List<Metadatos> listarMetadatosEsquemasNoValidaddos()

      public List<Metadatos> listarMetadatosEsquemasValidaddosSinCualificado()

      public void modificarMetadatosEsquemaOriginal(Metadatos metadatos)
```

```
	public boolean comprobarValidado(long id)

	public Esquemaoriginal buscarPorId(long id)

}
```

Código Fuente 4.5. Interfaz EsquemaOriginalSerevice

En el interfaz EsquemaOriginalService se definen los métodos que implementará la clase EsquemaOriginalServiceImple, en el siguiente fragmento de código se observa la implementación del método agregarEsquemasOriginales.

```
public void agregarEsquemasOriginales(Metadatos metadatos) throws
InstanceNotFoundException {
	metadatosDao.agregarMetadatos(metadatos);
	esquemaoriginal = new Esquemaoriginal();
	esquemaoriginal.setEsquemaid(metadatos.getTimestampid());
	esquemaoriginal.setEsquemaoriginalid(metadatos.getTimestampid());
	esquemaoriginal.setAprobada(false);
	esquemaOriginalDao.agregarEsquemaOriginal(esquemaoriginal);
}
```

Código Fuente 4.6. Método agregarEsquemasOriginales

Para poder dar de alta un esquema original es necesario emplear 2 nuevos interfaces MetadatosDao y EsquemaOriginalDao, así como las clases que los implementan, que son las que finalmente hacen los accesos a la base de datos mediante los métodos que proporciona Hibernate.

```
@SuppressWarnings("unchecked")
@Override
public void agregarEsquemaOriginal(Esquemaoriginal esquema) {
	getSession().save(esquema);
```

```
}
```

Código Fuente 4.7. Método agregarEsquemasOriginal

Al emplear Hibernate, tal y como se comento en el Apartado 3.3.4. *"Hibernate"* se tiene la posibilidad de modelar las distintas entidades como clases java. En la Figura 4.8. *"Diagrama de clases de Hibernate"* se muestra el diagrama de las clases relacionadas con Hibernate.

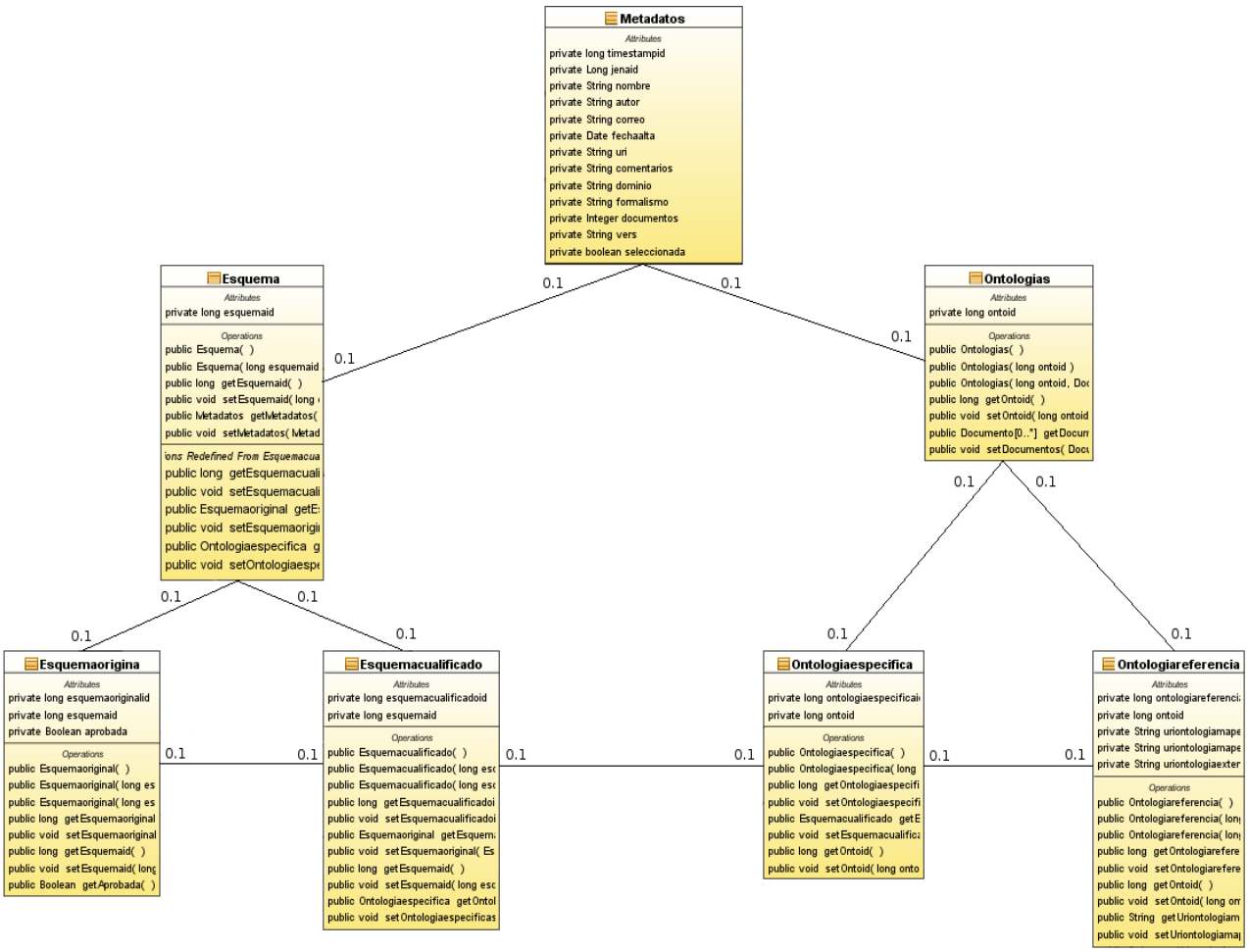

Figura 4.8. Diagrama de clases de Hibernate

En la clase Metadatos están definidos todos los atributos comunes tanto para los esquemas como para las ontologías, como pueden ser el nombre, el autor, la fecha de alta, la versión, etc. De la clase Esquema heredan las clases Esquemaoriginal y Esquemacualificado.

De esta forma cada esquema Esquemaoriginal y cada Esquemacualificado tendrá asociados metadatos, Los esquemas originales adicionalmente tendrán un campo que indicará si están validados o no.

Por otro lado, Spring, en la capa de negocio y como ya se describió en el Apartado 3.3.3.1. *"Estructura de Spring"*, permite inyectar dependencias en el momento en el que una clase es ejecutada. Para ello se emplea otro archivo de configuración denominado applicationContext.xml, de este modo, es Spring el encargado de la creación del objeto.

```
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderCon
figurer"
      p:location="/WEB-INF/jdbc.properties" />

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource"
      p:driverClassName="${jdbc.driverClassName}"
      p:url="${jdbc.url}"
      p:username="${jdbc.username}"
      p:password="${jdbc.password}" />

<bean id="ontologyFacade"
class="semse.model.service.ontologyfacade.plain.OntologyFacade">
        <property name="dataSource" ref="dataSource"/>
        <property name="DBType" value="${jdbc.dbType}"/>
</bean>
<!-- Hibernate Session Factory -->

<bean id="sessionFactory"
class="org.springframework.orm.Hibernate3.annotation.AnnotationSession
FactoryBean"
      p:dataSource-ref="dataSource"
      p:configLocation="classpath:Hibernate.cfg.xml"/>

<!-- Transaction manager for a single Hibernate SessionFactory. -->

<bean id="transactionManager"
class="org.springframework.orm.Hibernate3.HibernateTransactionManager"
      p:sessionFactory-ref="sessionFactory" />
```

```
<!-- Enable the configuration of transactional behavior based on
annotations. -->

<tx:annotation-driven transaction-manager="transactionManager" />

<!--  ============ Business Objects ============ -->
<!--  DAOs. -->
.
.
.
<bean id="metadatosDao"
class="semse.model.bussinessobject.metadatos.dao.MetadatosDaoHibernate
"
       p:sessionFactory-ref="sessionFactory" />

<bean id="esquemaOriginalDao"
class="semse.model.bussinessobject.esquemaoriginal.dao.EsquemaOriginal
DaoHibernate"
       p:sessionFactory-ref="sessionFactory" />

<bean id="esquemaCualificadoDao"
class="semse.model.bussinessobject.esquemaCualificado.dao.EsquemaCuali
ficadoDaoHibernate"
       p:sessionFactory-ref="sessionFactory" />

<!--Service layer. -->
.
.
.
<bean id="esquemaOriginalService"
class="semse.model.bussinessobject.esquemaOriginalService.EsquemaOrigi
nalServiceImple"
        p:metadatosDao-ref="metadatosDao"
        p:jenaDao-ref="jenaDao"
        p:esquemaOriginalDao-ref="esquemaOriginalDao"
        />

<bean id="esquemaCualificadoService"
class="semse.model.bussinessobject.esquemaCualificadoService.EsquemaCu
alificadoServiceImple"
        p:metadatosDao-ref="metadatosDao"
        p:jenaDao-ref="jenaDao"
        p:esquemaOriginalDao-ref="esquemaOriginalDao"
        p:esquemaCualificadoDao-ref="esquemaCualificadoDao"
        />
```

Código Fuente 4.8. Fragmento del fichero applicationContext.xml

Las propiedades especificadas en la definición del Bean son pasadas como atributos de la clase. En ningún momento la clase instancia el objeto por sí misma. Es decir, en este ejemplo cuando se definen los atributos del bean esquemaCualificadoService Spring realiza la inyección de dependencias de manera que en la clase no es necesario instanciar el objeto metadatosDao para poder utilizar los métodos que implementa.

```
@Transactional
public class EsquemaCualificadoServiceImple implements
EsquemaCualificadoService {

    private EsquemaCualificadoDao esquemaCualificadoDao;
    private EsquemaOriginalDao esquemaOriginalDao;
    private JenaDao jenaDao;
    private MetadatosDao metadatosDao;

@Transactional
@Override
public void eliminarEsquemasCualificados(Metadatos metadatosEliminar)
throws InstanceNotFoundException {

        getMetadatosDao().eliminarMetadatos(metadatosEliminar);

}

public MetadatosDao getMetadatosDao() {
        return metadatosDao;
}

public void setMetadatosDao(MetadatosDao metadatosDao) {
        this.metadatosDao = metadatosDao;
}
```

Código Fuente 4.9. Fragmento de la clase EsquemaCualificadoServiceImple.java

4.4.2– Gestión de la seguridad

Para gestionar la seguridad del portal se ha empleado Sprint Security, se trata de un subproyecto del framework Spring que proporciona una autenticación y autorización avanzada permitiendo gestionar completamente la seguridad de la aplicación, las principales ventajas que presenta la utilización de este framework y las cuales han decantado por su utilización son:

- Es capaz de gestionar seguridad en varios niveles: URLs que se solicitan al servidor, acceso a métodos y clases Java, y acceso a instancias concretas de las clases.

- Permite separar la lógica de las aplicaciones del control de la seguridad, utilizando filtros para las peticiones al servidor de aplicaciones o aspectos para la seguridad en clases y métodos.

- La configuración de la seguridad es portable de un servidor a otro, ya que se encuentra dentro del WAR o el EAR de la aplicación.

- Soporta muchos modelos de identificación de los usuarios (HTTP BASIC, HTTP Digest, basada en formulario, LDAP, OpenID, JAAS y muchos más). Además es posible ampliar estos mecanismos implementando clases que extiendan el modelo de Spring Security.

El fichero de configuración de Spring Security es applicationContext-security.xml y es que se ha editado para adaptarlo a las necesidades de seguridad del portal.

En primer lugar se definió el acceso para cada una de las páginas del portal, para ello se configuraron los interceptores de Spring Security, y establecieron los permisos adecuados para cada una de las páginas:

```
<http auto-config="true" access-denied-page="/Login.jsp">

        <intercept-url pattern='/Login.jsp' filters="none" />

       <intercept-url pattern="/GestionUsuarios.jsp"
access="ROLE_Administrador,ROLE_root"/>

        <intercept-url pattern="/AltaUsuario.jsp"
access="ROLE_Administrador,ROLE_root"/>

        <intercept-url pattern="/EditarEsquemaOriginal.jsp"
access="ROLE_ExpertoDominio,ROLE_ExpertoResponsableDominio,ROLE_root"/>

        <intercept-url pattern="/EditarEsquemaCualificado.jsp"
access="ROLE_ExpertoDominio,ROLE_ExpertoResponsableDominio,ROLE_root"/>

        <intercept-url pattern="/AltaEsquemasCualificados.jsp"
access="ROLE_ExpertoDominio,ROLE_ExpertoResponsableDominio,ROLE_root"/>

        <intercept-url pattern="/EditarOntologiaEspecifica.js"
access="ROLE_IngenieroDominio,ROLE_IngenieroResponsableDominio,ROLE_root"/>

        <intercept-url pattern="/EditarOntologiaReferencia.jsp"
access="ROLE_IngenieroDominio,ROLE_root"/>

        <intercept-url pattern="/AltaOntologiaReferencia.jsp"
access="ROLE_IngenieroDominio,ROLE_root"/>

        <intercept-url pattern="/AltaOntologiasEspecificas.jsp"
access="ROLE_IngenieroDominio,ROLE_IngenieroResponsableDominio,ROLE_root"/>

        <intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/>

        <form-login login-page="/Login.jsp" authentication-failure-
url="/Login.jsp" default-target-url="/Inicio.jsp"/>
        <logout invalidate-session="true" logout-success-url="/Inicio.jsp"/>
.
.
.
    </http>
```

Código Fuente 4.10. Fragmento del fichero applicationContext-security.xml

Las páginas tendrán una acceso restringido en función de una serie de roles, de manera que solo los usuarios validados con ese rol podrán acceder a ellas, las paginas que no tengan un interceptor definido podrán ser accedidas por cualquier usuario anónimo.

Spring Security aplicará la lista de interceptores en el orden en que aparece. Cuando encuentre un patrón adecuado, aplicará los permisos correspondientes y no continuará leyendo el resto de interceptores. Por tanto es importante colocar primero los filtros más específicos y después los más generales.

Además se indico que muestre una página de login cuando se intente acceder a una de las páginas restringidas y no se tengan los permisos oportunos, el tipo de autenticación que se va a utilizar, así como la página a la que se debe redirigir al hacer logout.

El siguiente aspecto a configurar es el AuthenticationProvider, que permite configurar los permisos de los usuarios en una base de datos a la que se accede por medio de un DataSource.

```
<authentication-provider>
        <password-encoder hash="md5"/>
        <jdbc-user-service data-source-ref="dataSource" />
</authentication-provider>
```

Código Fuente 4.11. Configuración del AuthenticationProvider

En la base de datos se creó la tabla users donde se encuentran los usuarios con su login y su password, necesarios para la validación y la tabla authorities donde se relaciona a cada usuario con su perfil.

Spring Security permite gestionar la seguridad a varios niveles de modo que es posible que algunos usuarios puedan acceder a una determinada página pero no a todos sus componentes, por ejemplo cualquier usuario puede ver los metadatos asociados a un esquema original, pero solo un experto de domino puede validarlo, de forma que el botón validar solo esta accesible a los usuario miembros de este perfil, y no aparece para los que no lo sean.

```
<ice:commandButton title="Validar" alt="Validar"
image="resources/validarVerde.png" id="validar"
action="#{DatosEsquemaOriginal.botonValidar}"
renderedOnUserRole="ROLE_root,ROLE_ExpertoDominio"  value="Validar"/>
```

Código Fuente 4.12. Configuración de la seguridad a varios niveles

Empleado Spring Security combinado con los atributos renderedOnUserRole y enableOnUserRole de IceFaces permite controlar individualmente la seguridad de cada uno de los elementos del portal.

4.5- Manual de usuario

4.5.1- Página de Inicio

Figura 4.9. Página de inicio

La página de inicio se divide en dos partes claramente diferenciadas, en la parte superior hay una descripción de los objetivos del proyecto SEMSE y en la parte inferior se listan los últimos esquemas introducidos en el sistema, los esquemas listados son enlazables, de manera que si se pulsa sobre ellos enlaza con la página que muestra sus metadatos asociados. Si se pulsa sobre "*ver todos*" se enlaza con la página de gestión de esquemas, originales o cualificados según corresponda.

Esta página presenta la estructura básica que presentarán el resto de páginas de la aplicación, una cabecera en la que aparece el logotipo de la Universidad Carlos III, el nombre del proyecto SEMSE y el enlace que permite validarse en la aplicación. En la parte inferior de la cabecera aparecerán las distintas secciones de la aplicación, "*Inicio*", "*Gestión de esquemas*", "*Gestión de ontologías*","*Gestión de usuarios*" y "*Ayuda*". En esta zona también se encuentra el enlace que permite cambiar el idioma de la aplicación, pudiendo seleccionar entre el español o el inglés.

Figura 4.10. Página de inicio traducida al inglés

4.5.2- Gestión de esquemas originales

Figura 4.11. Pantalla de gestión esquemas originales

Desde esta pantalla se pueden gestionar los esquemas originales. Si se dispone de los permisos adecuados (Experto de dominio o Experto responsable de dominio), se puede ver en la parte izquierda un selector que permite elegir entre esquemas validados, sin validar o todos. El resto de usuarios sólo podrán acceder a los esquemas validados.

El en panel de la zona izquierda se pueden realizar consultas por metadatos de esquemas originales dados de alta en el sistema. Este tipo de consulta podrá ser realizada por cualquier usuario. Los tipos de consulta que se contemplan son:

- **Consulta por nombre del esquema original**

- **Consulta por el nombre del autor que dio de alta el esquema original**
- **Consulta por la fecha en la que se dio de alta el esquema original**
- **Consulta por la uri donde se encuentra publicado el esquema original**
- **Consulta por el dominio del esquema original**

En la parte central aparecerá una tabla con los esquemas cualificados almacenados en el sistema o en caso de haber realizado alguna consulta, aparecerán únicamente los esquemas cualificados encontrados.

En esta pantalla también se tendrá acceso, mediante la selección de un esquema en la tabla anterior, a las siguientes acciones:

- **Alta de un esquema original:** puede ser realizado por el Experto responsable del dominio o por un Experto de dominio

- **Eliminación de uno o más esquemas originales:** puede ser realizado por un Experto de dominio

- **Edición de un esquema original:** puede ser realizado por el Experto responsable del dominio o por un Experto de dominio

- **Ver los datos de un esquema original:** puede ser realizado por todos los usuarios

Estos botones serán mostrados dependiendo del usuario validado, teniendo acceso solo a las acciones permitidas según su perfil.

4.5.3– Dar de alta un esquema original

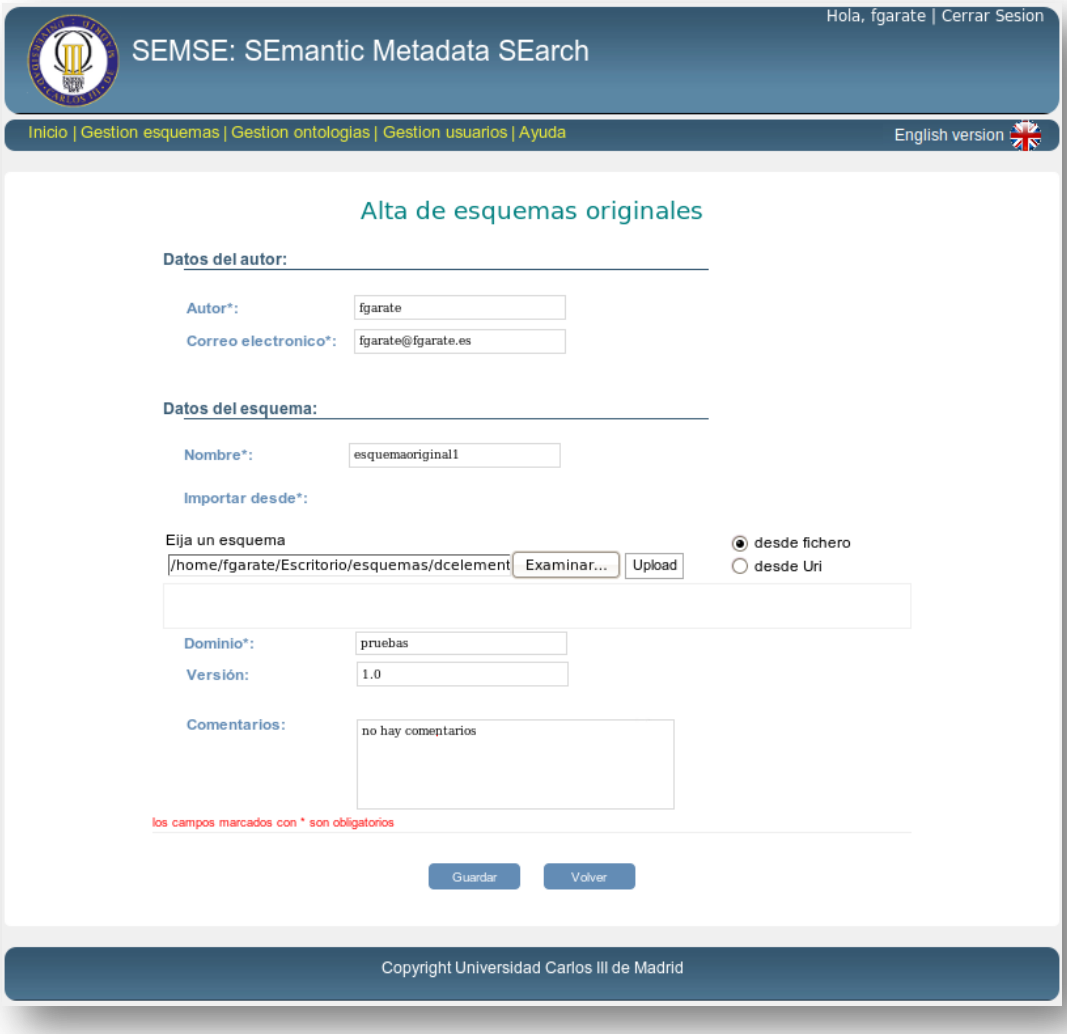

Figura 4.12. Pantalla de alta esquemas originales

El rol necesario de acceso a esta pantalla es el de **Experto de Dominio** o **Experto Responsable de Dominio.**

El alta de un Esquema Original se basa en cumplimentar los siguientes campos:

- **Nombre**: se tiene que definir un nombre para el nuevo esquema original. El nombre elegido debe ser único en la aplicación, en caso de no ser así se le notificará al usuario.

- **Autor**: se tiene que indicar el nombre de la persona que va a dar de alta el esquema original.

- **Correo electrónico**: se tiene que facilitar una dirección de correo para tener una forma de contacto con el usuario correspondiente.

- **Importar desde**: el experto responsable del dominio o un experto de dominio tiene que dar de alta el fichero que contiene el esquema original. Puede hacerlo de dos formas:

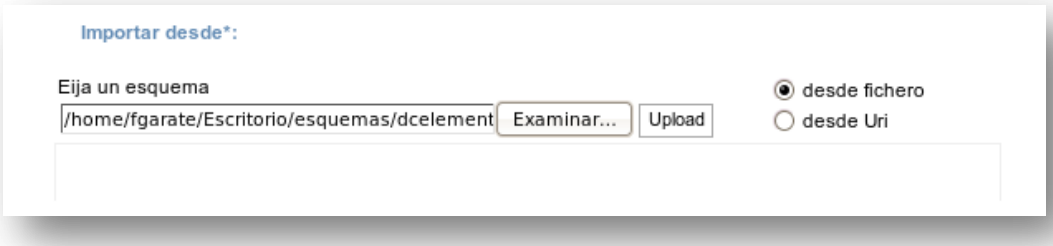

Figura 4.13. Cuadro de entrada para la importación de un archivo

1. **Desde fichero:** se puede acceder al directorio de ficheros pulsando el botón **Examinar**, y a continuación se selecciona el fichero correspondiente. Por último, el usuario tiene que pulsar el botón **upload** para subir el fichero.

2. **Desde uri:** se puede indicar la uri donde se encuentra publicado el esquema original. El usuario tiene que escribir la uri en el espacio dedicado y después tiene que pulsar el botón **upload** para subir el fichero.

- **Versión:** se puede indicar la versión de dicho esquema original para controlar los cambios realizados sobre el mismo. Por defecto, si el usuario no indica nada en este campo, se aplica el número de versión 1.0.

- **Comentarios:** se podrá añadir las anotaciones que sean necesarias.

Por último, el usuario tiene que pulsar el botón **alta** para terminar el proceso de alta del esquema original o puede volver a la página de gestión de esquemas originales pulsando la flecha verde. Si el usuario no cumplimenta algún campo obligatorio la aplicación lo indicará con un mensaje. Si todo es correcto, a continuación se pide la confirmación del alta del esquema original.

4.5.4- Eliminar un esquema original

Figura 4.14. Eliminar un esquema original

Un **Experto de dominio** puede eliminar uno o más esquemas originales seleccionándolos desde la tabla.

Para eliminar un esquema original que tiene asociado un esquema cualificado, es necesario eliminar previamente el esquema cualificado, de no hacerlo así aparecerá una ventana emergente notificándolo

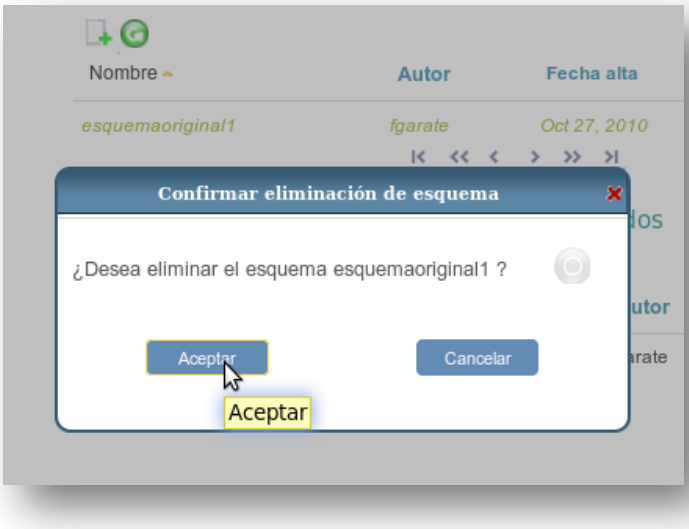

Figura 4.15. Pantalla emergente de confirmación al eliminar un esquema

A continuación, la aplicación solicita confirmar la eliminación de los esquemas originales seleccionados por el usuario. En caso contrario, se volverá a la pantalla gestión de esquemas originales

4.5.5- Editar un esquema original

Figura 4.16. Pantalla de edición de un esquema original

Un **experto responsable de dominio** o un **experto de dominio**, puede modificar los siguientes metadatos asociados a un esquema original:

- Autor, Correo, Fichero, Dominio, Versión, Comentarios

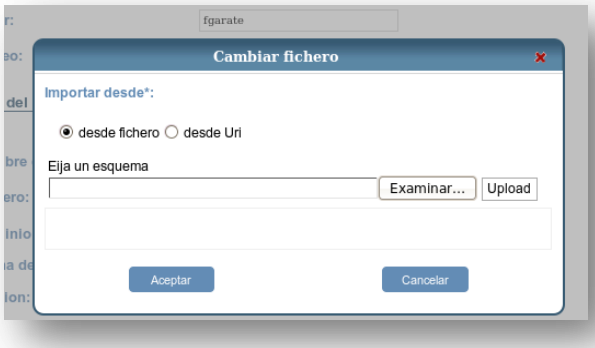

Figura 4.17. Pantalla emergente para cambiar el fichero

Si el usuario desea modificar el fichero asociado a un esquema original tiene que pulsar el botón **cambiar fichero**. Después se le pide al usuario que ingrese el nuevo fichero indicando la uri en la que se encuentra publicado el esquema original o seleccionando desde el árbol de directorios el correspondiente fichero.

Finalmente, el usuario tiene que pulsar el botón guardar para salvar los cambios realizados. En caso de confirmar, se modificarán dichos metadatos y se volverá a la pantalla **Gestión de Esquemas Originales**. Por el contrario, si el usuario pulsa el botón **volver**, la aplicación navegará a la pantalla **Gestión de Esquemas Originales**.

4.5.6- Mostrar los metadatos asociados a un esquema original

Figura 4.18. Pantalla con los metadatos de un esquema original

Todos los usuarios pueden consultar los metadatos asociados a un esquema original, como se aprecia en la imagen. Además puede descargar en su directorio local dicho esquema pulsando el botón **descargar esquema**. También puede visualizar el contenido del esquema pulsando el botón **visualizar esquema**.

En caso de ser **experto responsable de dominio** o **experto de domino**, se puede editar el esquema original pulsando el botón **editar**. En ese caso se accede a la página Editar esquema original, y así el usuario podrá modificar los metadatos. Habiendo finalizado la edición del esquema se volverá a la pantalla de gestión de esquemas originales.

También es posible comprobar si el esquema original esta validado, en caso de no estarlo, es posible validar los esquemas originales si se tiene permisos de **experto de domino.**

4.5.7- Visualizar un esquema original

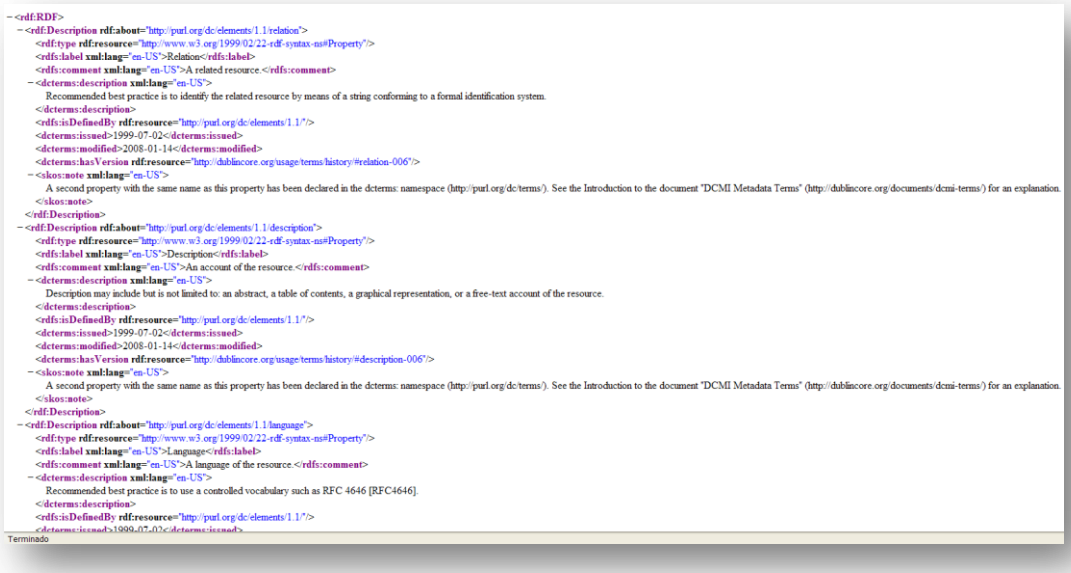

Figura 4.19. Esquema original abierto en una pestaña del navegador

Cualquier usuario puede visualizar el contenido de un esquema original validado mediante el navegador, pudiendo ser visto en una nueva ventana o pestaña del propio navegador.

4.5.8- Descargar un esquema original

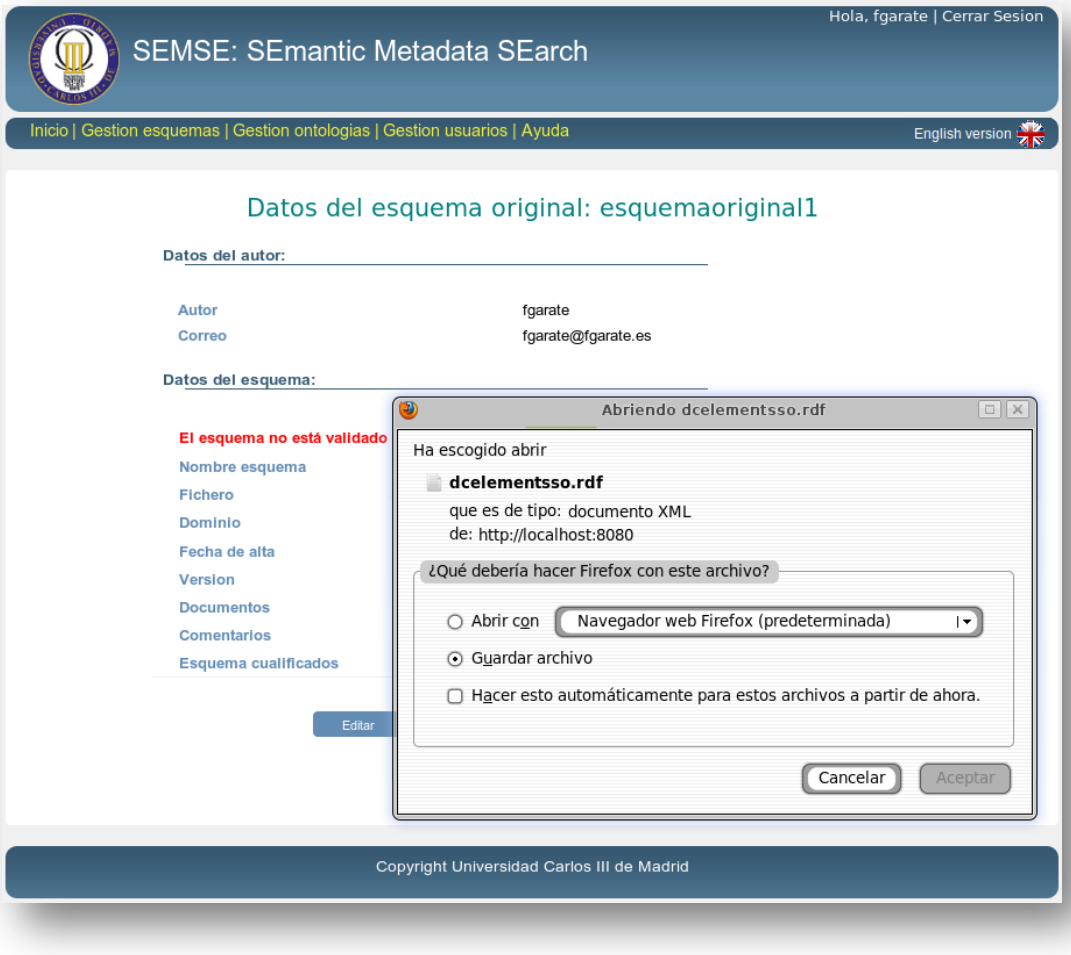

Figura 4.20. Pantalla de descarga de un esquema original

Un esquema original puede ser descargado por cualquier usuario. Esta acción es ejecutada por el propio navegador, dando la opción de **guardar** o abrir el fichero correspondiente del esquema cualificado con extensión rdf. Si el usuario desea cancelar dicha acción, tiene que pulsar cancelar en la ventana emergente abierta por el navegador.

4.5.9- Gestión de esquemas cualificados

Figura 4.21. Pantalla de gestión esquemas cualificados

En esta pantalla se puede ver en la parte de la izquierda un panel para realizar consultas por metadatos de esquemas cualificados dados de alta en el sistema. Este tipo de consulta podrá ser realizada por cualquier usuario. Los tipos de consulta que se contemplan son:

- **Consulta por nombre del esquema cualificado**
- **Consulta por el nombre del autor que dio de alta el esquema cualificado**
- **Consulta por la fecha en la que se dio de alta el esquema cualificado**
- **Consulta por la uri donde se encuentra publicado el esquema cualificado**

- **Consulta por el dominio del esquema cualificado**

En la parte central aparecerá una tabla con los esquemas cualificados almacenados en el sistema o en caso de haber realizado alguna consulta, aparecerán únicamente los esquemas cualificados encontrados.

En esta pantalla también se tendrá acceso, mediante la selección de un esquema en la tabla anterior, a las siguientes acciones:

- **Alta de un esquema cualificado:** puede ser realizado por el Experto responsable del dominio o por un Experto de dominio

- **Eliminación de uno o más esquemas cualificados:** puede ser realizado por un Experto de dominio

- **Edición de un esquema cualificado:** puede ser realizado por el Experto responsable del dominio o por un Experto de dominio

- **Ver los datos de un esquema cualificado:** puede ser realizado por todos los usuarios

Estos botones serán mostrados dependiendo del usuario validado, teniendo acceso solo a las acciones permitidas según su perfil.

4.5.10- Alta de un esquema cualificado

Figura 4.22. Pantalla de alta esquemas cualificados

El rol necesario de acceso a esta pantalla es el de **Experto de Dominio** o **Experto Responsable de Dominio.**

El alta de un Esquema Cualificado se basa en cumplimentar los siguientes campos:

- **Esquema original**: se tiene que indicar el esquema original que se asociará a este nuevo esquema cualificado. Para ello, se da una ayuda al usuario para autocompletar el nombre de dicho esquema original

- **Nombre**: se tiene que definir un nombre para el nuevo esquema cualificado. El nombre elegido debe ser único en la aplicación, en caso de no ser así se le notificará al usuario

- **Autor**: se tiene que indicar el nombre de la persona que va a dar de alta el esquema cualificado

- **Correo electrónico**: se tiene que facilitar una dirección de correo para tener una forma de contacto con el usuario correspondiente

- **Importar desde**: el experto responsable del dominio o un experto de dominio tiene que dar de alta el fichero que contiene el esquema cualificado. Puede hacerlo de dos formas:

    1. **Desde fichero:** se puede acceder al directorio de ficheros pulsando el botón **Examinar**, y a continuación se selecciona el fichero correspondiente. Por último, el usuario tiene que pulsar el botón **upload** para subir el fichero.

    2. **Desde uri:** se puede indicar la uri donde se encuentra publicado el esquema cualificado. El usuario tiene que escribir la uri en el espacio dedicado y después tiene que pulsar el botón **upload** para subir el fichero.

- **Versión:** se puede indicar la versión de dicho esquema cualificado para controlar los cambios realizados sobre el mismo. Por defecto, si el usuario no indica nada en este campo, se aplica el número de versión 1.0.

- **Comentarios:** se podrá añadir las anotaciones que sean necesarias.

Por último, el usuario tiene que pulsar el botón **alta** para terminar el proceso de alta del esquema cualificado o puede volver a la página de gestión de esquemas cualificados pulsando la flecha verde. Si el usuario no cumplimenta algún campo obligatorio, la aplicación lo indica como se observa en la imagen. Si todo es correcto, a continuación se pide la confirmación del alta del esquema cualificado.

4.5.11- Eliminar esquemas cualificados

Un Experto de dominio puede eliminar uno o más esquemas cualificados seleccionándolos desde la tabla.

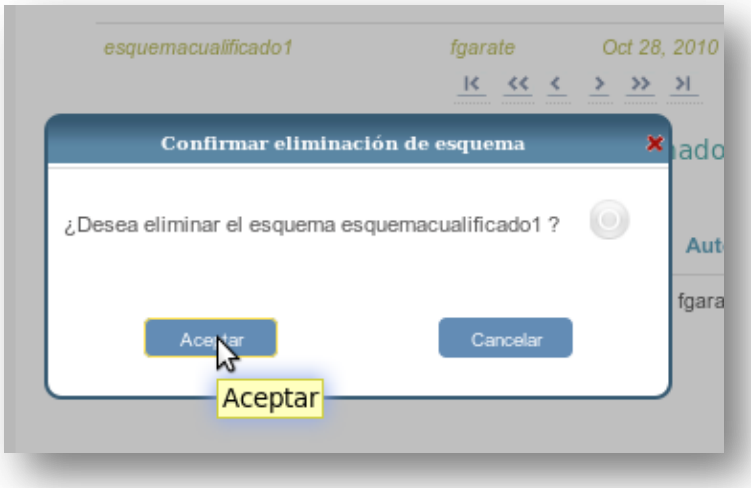

Figura 4.23. Pantalla emergente de confirmación al eliminar un esquema

Para eliminar un esquema cualificado que tiene asociado una ontología, es necesario eliminar previamente la ontología, de no hacerlo así aparecerá una ventana emergente notificándolo

A continuación, la aplicación solicita confirmar la eliminación de los esquemas cualificados seleccionados por el usuario. En caso contrario, se volverá a la pantalla gestión de esquemas cualificados

4.5.12- Editar un esquema cualificado

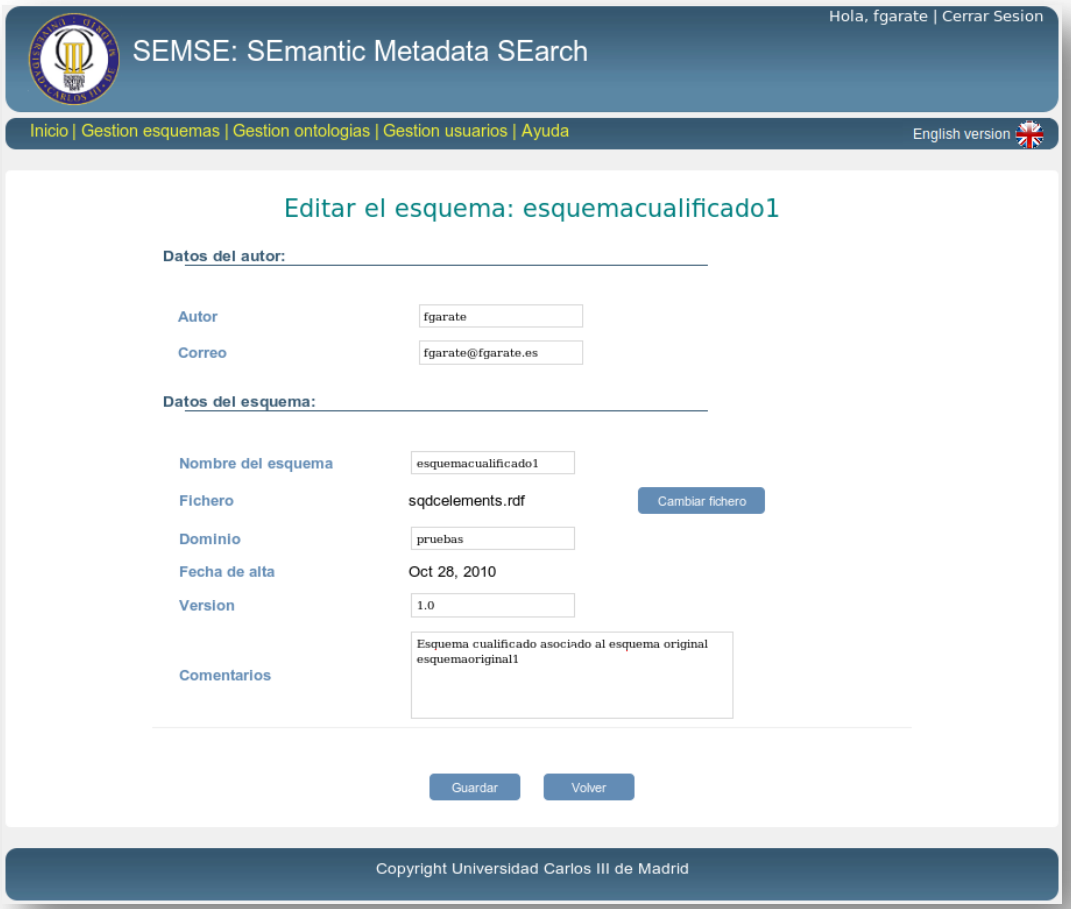

Figura 4.24. Pantalla de edición de un esquema cualificado

Un **experto responsable de dominio** o un **experto de dominio**, puede modificar los siguientes metadatos asociados a un esquema cualificado:

- Autor, Correo, Fichero, Dominio, Versión, Comentarios

Si el usuario desea modificar el fichero asociado a un esquema cualificado tiene que pulsar el botón **cambiar fichero**. Después se le pide al usuario que ingrese el nuevo fichero indicando la uri en la que se encuentra publicado el esquema cualificado o seleccionando desde el árbol de directorios el correspondiente fichero.

Finalmente, el usuario tiene que pulsar el botón guardar para salvar los cambios realizados. En caso de confirmar, se modificarán dichos metadatos y se volverá a la pantalla **Gestión de Esquemas Cualificados**. Por el contrario, si el usuario pulsa el botón **volver**, la aplicación navegará a la pantalla **Gestión de Esquemas Cualificados**.

4.5.13- Mostrar los metadatos asociados a un esquema cualificado

Figura 4.25. Pantalla con los metadatos asociados a un esquema cualificado

Cualquier tipo de usuario puede consultar los metadatos asociados a un esquema cualificado, como se aprecia en la imagen. Además puede descargar en su directorio local dicho esquema pulsando el botón **descargar esquema**. También puede visualizar el contenido del esquema pulsando el botón **visualizar esquema**.

Por último, en caso de ser **experto responsable de dominio** o **experto de domino**, se puede editar el esquema cualificado pulsando el botón **editar**. En ese caso se accede a la página Editar esquema cualificado, y así el usuario podrá modificar los metadatos. Habiendo finalizado la edición del esquema se volverá a la pantalla de gestión de esquemas cualificados. Por el contrario, si el usuario pulsa el botón **volver** de la página Editar esquema cualificado, se volverá a la página Ver esquema cualificado.

4.5.14- Descargar un esquema cualificado

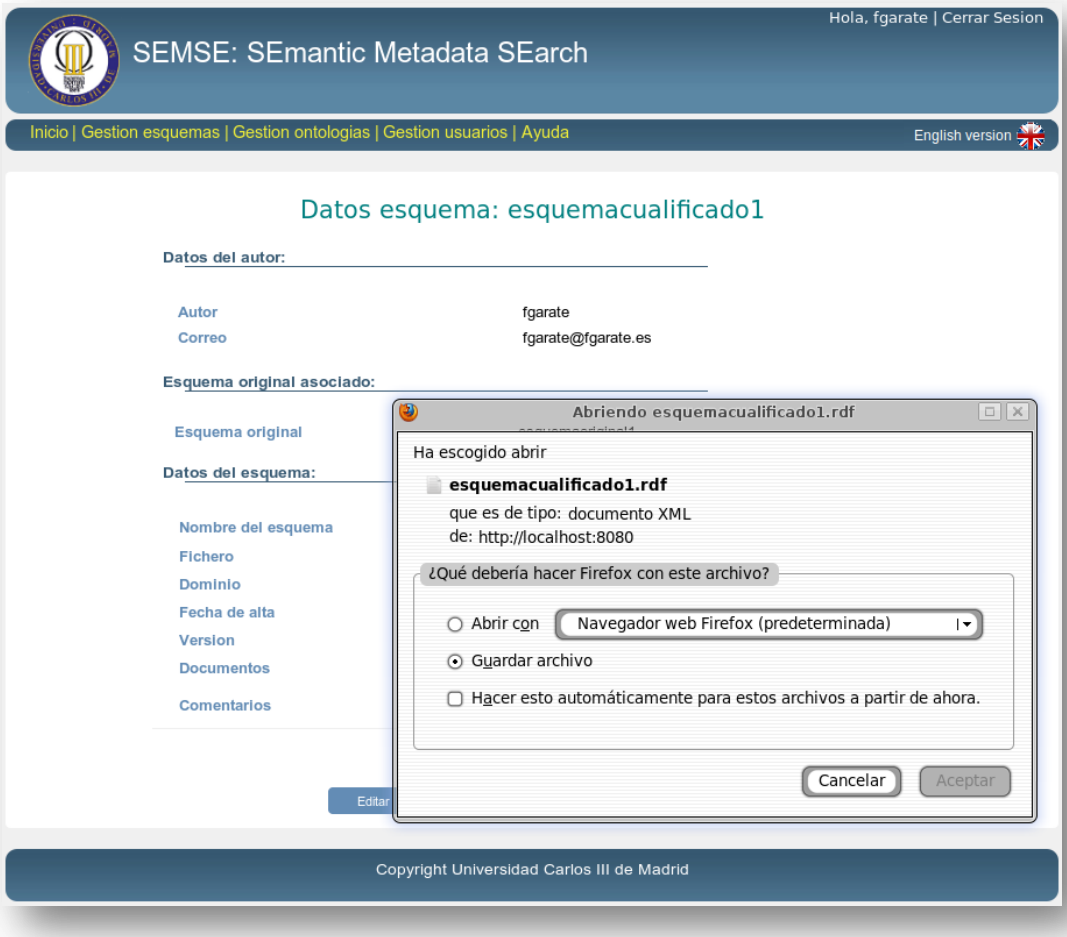

Figura 4.26. Pantalla de descarga del esquema cualificado

Un esquema cualificado puede ser descargado por cualquier usuario. Esta acción es ejecutada por el propio navegador, dando la opción de **guardar** o abrir el fichero correspondiente del esquema cualificado con extensión rdf. Si el usuario desea

cancelar dicha acción, tiene que pulsar cancelar en la ventana emergente abierta por el navegador.

4.5.15- Visualizar un esquema cualificado

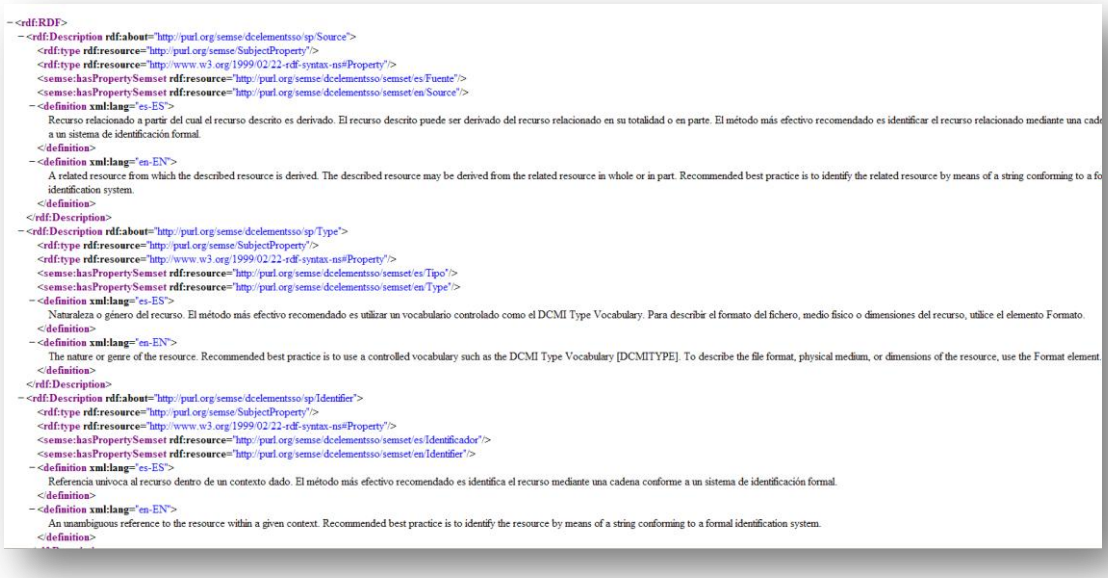

Figura 4.27. Esquema original abierto en una pestaña del navegador

Cualquier usuario puede visualizar el contenido de un esquema cualificado mediante el navegador, pudiendo ser visto en una nueva ventana o pestaña del propio navegador.

4.6- Resumen del proyecto

La planificación inicial que se planteó para este proyecto fue cumplida en la medida de lo posible. A grandes rasgos, dicha planificación no sufrió modificaciones drásticas en el tiempo estimado para cada tarea. Salvo pequeños reajustes, propios del desarrollo de cualquier trabajo, las líneas maestras trazadas al inicio fueron cumplidas rigurosamente.

Se presenta en este apartado un resumen de las tareas realizadas y su consecución en el tiempo. Para ello se ha utilizado la herramienta gráfica conocida como Diagrama de Gantt, que permite mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

4.6.1- Panificación final

A continuación se puede visualizar un diagrama de Gantt. Desde sus inicios, estos diagramas se han convertido en una herramienta básica en la gestión de proyectos de todo tipo, con la finalidad de representar las diferentes fases, tareas y actividades programadas como parte de un proyecto o para mostrar una línea de tiempo en las diferentes actividades haciendo el método más eficiente.

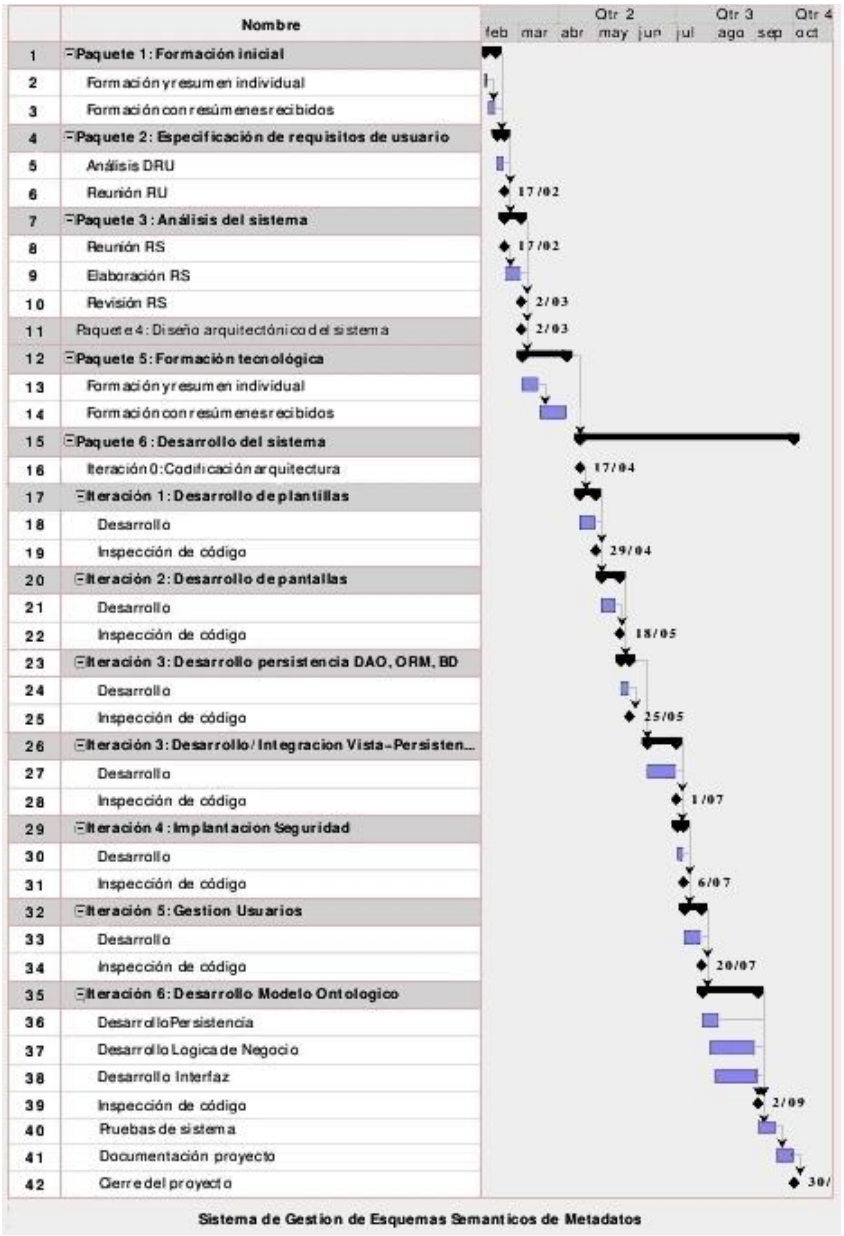

Figura 4.28. Planificación final del proyecto en Diagrama de Gantt

Capitulo 5. Conclusiones

La aplicación desarrollada supone un avance importante en la realización de los objetivos definidos en el proyecto SEMSE, facilitando la gestión y consulta de esquemas tanto originales como cualificados.

Para el desarrollo se han utilizado tecnologías, lenguajes de programación y frameworks que hoy en día representan la punta de lanza del desarrollo de aplicaciones Web. Java, el lenguaje de programación orientado a objetos más extendido, ha sido la base de este proyecto, girando todo el desarrollo alrededor de él.

Las tecnologías empleadas, y las arquitecturas software y hardware ya habían sido evaluadas previamente en anteriores proyectos, por lo que antes de definir los requisitos de la aplicación se realizaron una serie de estudios sobre los distintos conceptos y tecnologías involucrados en la aplicación y que en la medida de lo posible han sido incluidos en el presente proyecto fin de carrera.

Una vez definidos los requisitos de la aplicación se desarrolló el modelo de clases del sistema y el modelo de la base de datos, de forma que fue posible comenzar a desarrollar la aplicación, trabajando conjuntamente en todas las capas del modelo. En primer lugar se creó la vista en función de los diseños definidos en la fase de análisis, posteriormente se desarrollaron las acciones que manejan dicha vista, (listado de esquemas, los formularios de consulta, etc.) y finamente se implemento la parte relacionada con la persistencia de los datos, tanto en la relación con Hibernate como con Jena.

En el proceso de desarrollo se han encontrado las dificultades propias del empleo de tecnologías y frameworks que hasta la hora de la elaboración del proyecto desconocía. Al tratarse de un proyecto innovador ha habido un gran trabajo en la integración de las distintas tecnologías que ha provocado el tener que modificar en algunos momentos el plan de trabajo.

5.1- Futuras líneas de desarrollo

La ejecución de este proyecto es un punto inicial y necesario sobre el que se apoyarán futuros proyectos que desembocarán en la consecución de los objetivos propuestos en el proyecto SEMSE.

Por tanto, en posteriores trabajos se incluirá una ontología de referencia así como la posibilidad de gestionar y consultar distintas ontologías específicas generadas a partir de los esquemas cualificados incluidos en el portal. De esta forma, se podrá interrelacionar los elementos incluidos en los esquemas con los conceptos incluidos en la ontología de referencia, permitiendo realizar consultas conceptuales sobre el modelo generado.

Otras funcionalidades interesantes en la evolución del proyecto pueden ser la gestión de perfiles de aplicación y la gestión de documentos, los cuales serán obtenidos de internet mediante un crawler, y se podrán relacionar con las ontologías del sistema, de forma que será posible tener un control de los documentos generados por cada una de las ontologías almacenadas en el sistema.

Basándose en Jena como framework de gestión de ontologías y SPARQL como lenguaje de consultas, se podrán mejorar las consultas sobre los modelos gestionados.

Capítulo 6. Bibliografía

[ABIÁN, M.A. 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 febrero del 2009.

[ALARCÓN, R. 2007] Diseño orientado a objetos con UML. Editorial: Grupo Edidos.

[ALTOVA SEMANTICWORKS. 2009] Editor de Ontologías. Disponible en http://www.altova.com/products_ semanticworks.html, consultado en abril de 2010.

[APOLLO. 2009] Editor de Ontologías. Disponible en http://apollo.open.ac.uk/, consultado en abril 2010.

[BCN. 2007] Web 3.0: La inteligencia hecha Web. Disponible en http://www.bcn.cl/carpeta_temas_profundidad/temas_profundidad.2007-04 11.6955930382, consultado en febrero del 2010.

[BERNERS-LEE, T.; HENDLER, J. & LASSILA, O. 2001] The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American.

[BOOCH, G. 2006. UML] El lenguaje unificado de modelado: guía del usuario. Editorial: Adisson Wesley.

[CASTEJÓN, J.S. 2004] Arquitectura y diseño de sistemas Web modernos. Revista de Ingeniería Informática del CIIRM.

[COLLINS-SUSSMAN, B.; FITZPATRICK, B.W. & PILATO, C. M. 2008] Control de versiones con Subversion. Libro Online en HTML, PDF y XMLDocbook. Disponible en http://svnbook.red-bean.com/, consultado en abril del 2009.

[COE. 2009]. Editor de Ontologías. Disponible en http://cmap.ihmc.us/coe, consultado en abril del 2010.

[DAWSON, F; HOWES, t. 1998] RFC2426 - vCard MIME Directory Profile. Disponible en http://www.faqs.org/rfcs/rfc2426.html, consultado en febrero del 2010.

[DCMI 2009] Dublin core Metadata initiative. Disponible en http://dublincore.org/documents/dc-rdf/, consultado en febrero del 2009.

[DODDS, L. 2002] FOAF-a-Matic. Disponible en http://www.ldodds.com/foaf/foaf-a-matic.es.html, consultado en febrero del 2010.

[ECKEL,B. 2007] Think in Java 4rd Edition Revision 4.0. Editorial: Prentice Hall.

[FURINI, G. 2007] Java, su historia, ediciones, versiones y características como plataforma y lenguaje de programación. Disponible en http://www.clubdesarrolladores.com/artículos/, consultado en agosto del 2009.

[GAMMA, E.; HELM, R.; JOHNSON, R. & VLISSIDES, J. 1998] Design Patterns, Elements ofReusable Object-Oriented Software. Editorial: Adisson Wesley.

[GlASSFISH, 2008] Servidor de aplicaciones. Disponible en https://glassfish.dev.java.net/, consultado agosto del 2010.

[GONZÁLEZ, V. 2006] Metadatos y documentos XML/RDF para recuperación, Víctor García González. Disponible en http://www.denibol.com/metadatos_xml_rdf/?id=metadatos_dc, consultado en febrero del 2009.

[GRUBER, T.R. 1993] A Translation Approach to Portable Ontology Specifications. Editorial: Knowledge Acquisition Journal. Vol. 5 pp. 199-200.

[GUARINO, N. 1998] Formal Ontology and Information Systems. Editorial: IOS Press.

[HERNANDEZ, E, 2005] El lenguaje unificado de modelado (UML).

[HONRUBIA LÓPEZ, F. J. 2002] Introducción a las Ontologías. Escuela Universitaria de Albacete. Disponible en http://www.infoab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabaj--ontologias.pdf , consultado en febrero del 2009.

[JASPER, R. & USCHOLD, M. 1999] A Framework for Understanding and Classifying Ontology Applications. Editorial: CEUR Publications and University of Amsterdam. Vol.18.

[LAMARCA, M.J. 2009] Hipertexto: El nuevo concepto de documento en la cultura de la imagen. Disponible en http://www.hipertexto.info/documentos/metadatos.htm, consultado en febrero del 2009.

[LEANDRO, M. 2002] Descripción de Foaf. Disponible en http://www.f14Web.com.ar/inkel/2003/01/27/foaf.html, consultado en febrero del 2010.

[LINKFACTORY. 2009] Editor de Ontologías. Disponible en http://www.landcglobal.com/images/linkfactory.pdf, consultado en abril del 2010.

[LOZANO , A. 2001] Ontologías en la Web Semántica. I Jornadas de Ingeniería Web'01. Disponible en http://www.informandote.com/jornadasIngWEB/articulos/jiw02.pdf, consultado en febrero del 2009.

[MAEDCHE, A. 2002] Ontology Learning for the semantic Web. Editorial: Kluwer Academic Publishers.

[MALIK, A. 2003] XML, Ontologies, and the semantic Web, The second generation of the Web. Publicado por XML Journal. Disponible en http://goliath.ecnext.com/coms2/summary_0199-2476390_ITM, consultado en febrero del 2009.

[METADATA, 2003] An Introduction to Metadata. Disponible en http://www.library.uq.edu.au/iad/ctmeta4.html, consultado en febrero del 2010.

[MENENDEZ, E.M. 1999] Un modelo de metadatos flexible para las bibliotecas digitales del próximo milenio. Disponible http://www.bib.uc3m.es/~mendez/publicaciones/7jc99/rdf.htm, consultado en febrero del 2009.

[MUSEN, M. 2009] Editor de Ontologías. Disponible en http://protege.stanford.edu/ consultado en abril de 2010.

[NECHES, R.; FIKES, R.E.; FININ, T.; GRUBER, T.R.; SENATOR, T. & SWARTOUT, W.R. 1991] Enabling technology for knowledge sharing. AI Magazine. Vol.12 (3) p.p. 36-56.

[NEON TOOLKIT. 2009]. Editor de Ontologías. Disponible en http://www.neon-toolkit.org/, consultado en abril del 2010.

[NETBEANS 6.5, 2008] NetBeans IDE 6.5 Release Candidate Information. Disponible en http://www.netbeans.org/community/releases/65/, consultado en septiembre del 2009.

[OGDEN, C. K. & RICHARDS, I. A. 1923] Meaning of meaning. Editorial: New York:Harcourt & Brace.

[OILED. 2009] Editor de Ontologías. Disponible en http://oiled.man.ac.uk, consultado en abril del 2010.

[ONTOSTUDIO. 2009] Editor de Ontologías. Disponible en http://www.ontoprise.de/content/e1171/e1249/index_eng.html y en http://www.ontoprise.de/content/e1171/e1249/e1672/e1673/OntoStudio_2.0_docu_eng.pdf, consultado en Noviembre de 2007.

[OWL. 2004] OWL Web Ontology Language Overview. Disponible en http://www.w3.org/TR/owl-features/, consultado en febrero del 2010.

[PARADA, R.A. 2009] DOAC: Description of a Career. Disponible en http://ramonantonio.net/doac/, consultado en febrero del 2010.

[PATRONES, 2008] Core J2EE Patterns. Disponible en http://java.sun.com/blueprints/corej2eepatterns/Patterns/, consultado en abril del 2009.

[POSTGRESQL 8.3, 2008] PostgreSQL 8.3 Press Kit. Disponible en http://www.postgresql.org/about/press/, consultado en septiembre del 2009.

[PROYECTO DOAP. 2010] Doap Project. Disponible http://trac.usefulinc.com/doap, consultado en febrero del 2010.

[PROYECTO FOAT. 2009] Foaf Project. Disponible en http://www.foafproject.org/original-intro, consultado febrero del 2010.

[RICHERO, A; BRAVO, H. 207] La Web 3.0 añade significado. Disponible en http://www.crdasesores.com/_Contenido/noticias/PDF/0711_la_Web.pdf, consultado en febrero del 2009.

[SERVIDOR WEB, 2009] Servidor Web - Wikipedia, la enciclopedia libre. Disponible en http://es.wikipedia.org/wiki/Framework, consultado en agosto del 2009.

[STUDER, R.; BENJAMINS, V. R. & FENSEL, D. 1998] Knowledge Engineering: Principles and methods. Editorial: Data & Knowledge Engineering. Vol. 25, num. 1-2, pp. 161-197.

[TERZIEV, I. ; KIRYKOV,A. & MANOV, D. 2005] PROTON Guidance. D1.8.1 Base Upper-level Ontology (BULO) Guidance. Disponible en http://proton.semanticWeb.org/D1_8_1.pdf, consultado en enero de 2010.

[UBUNTU, 2008] About Ubuntu. Disponible en http://www.ubuntulinux.org/, consultado en agosto del 2009.

[UML 2.0, 2008] The Current Official UML Specification. Disponible en http://www.uml.org/#UML2.0, consultado en abril del 2009.

[VARGAS,A.2008] Introduccion al Servidor de Aplicaciones OpenSource de Sun Microsystems. Disponible en http://osum.sun.com/group/itparral/forum/topics/2181626:Topic:521610, consultado en agosto del 2009.

[W3C 22. 1999] Especificación del modelo y la sintaxis RDF. Disponible en http://www.sidar.org/recur/desdi/traduc/es/rdf/rdfesp.htm, consultado en febrero del 2009.

[W3C 27. 2000] Especificación del esquema RDF. Disponible en http://www.sidar.org/recur/desdi/traduc/es/rdf/rdfsch.htm, consultado en febrero del 2009