UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMATICA DE GESTIÓN

PROYECTO FIN DE CARRERA

Estudio tecnológico y diseño arquitectónico de un Sistema de Gestión de Esquemas Semánticos basados en Ontologías

AUTOR: TATIANA FERNÁNDEZ-TOSTADO CANOREA

TUTOR: VICENTE PALACIOS MADRID

26 de Febrero de 2009

Agradecimientos

Quiero agradecer este trabajo a toda la gente que ha estado apoyándome incondicionalmente estos últimos años y que no ha dudado de mí en ningún momento.

En primer lugar, agradezco a mi tutor Vicente su ayuda y apoyo durante el desarrollo del proyecto.

En segundo lugar, a mis padres por todo el esfuerzo que han hecho para que yo pueda llegar a este momento y a mi hermano Sergio, que sin ser muy consciente de ello, también ha aportado su granito de arena.

También quiero hacer una mención especial para una persona también muy especial, David, por aguantarme todo este tiempo y espero que mucho más y haberme apoyado tanto en los buenos como en los malos momentos.

Tampoco quiero olvidarme del resto de mi familia (especialmente mi abuela Josefa) y de mis amigos que no han dejado de animarme a tirar para adelante y a sobreponerme a las adversidades.

Por último quiero dedicar este proyecto a mi hermano Iván: *"porque allá donde estés siempre estarás conmigo"*.

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 proyecto SEMSE propone cualificar semánticamente esquemas de metadatos mediante un sistema informático que permita hacer uso de la semántica incluida en ontologías distribuidas vía web. El presente trabajo se centra, tras el análisis de la problemática existente en la Web actual y la identificación de elementos que pudieran solventar esta situación, en el estudio de las mejores tecnologías para la infraestructura del sistema informático propuesto, así como en el diseño de una arquitectura óptima para el sistema informático de soporte para el proyecto SEMSE.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Originalmente, la Web se diseñó para su utilización por personas en vez de para aplicaciones. Este planteamiento ha limitado las posibilidades de las aplicaciones Web, tales como la recuperación de información, los sistemas pregunta-respuesta o el razonamiento mediante inferencias. Las 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 son una parte importante de la Web semántica. En la Web semántica, las ontologías 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.

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

El proyecto SEMSE (SEMSE, 2008) tiene como objetivo el desarrollo de un sistema de gestión de conocimiento basado en ontologías. Concretamente, proporciona

representaciones ontológicas de esquemas de metadatos consistentes en dotar a 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.

Englobado dentro del proyecto SEMSE y dentro de la fase de desarrollo del sistema informático que soportará al proyecto, este PFC aborda primeramente la necesidad de realizar un estudio de las herramientas necesarias y de las tecnologías punteras en el ámbito de la informática y la gestión de ontologías. Una vez analizadas, se seleccionarán aquellas que permitan maximizar la flexibilidad, modularidad y escalabilidad del sistema a desarrollar. Finalmente, se diseñará una arquitectura que maximice las características anteriores cubriendo el equivalente a la primera fase de diseño del sistema.

1.1. Motivación

Como parte del proyecto SEMSE éste será el que marque los requisitos y necesidades que serán objeto de estudio en este PFC. Se pueden resumir en:

Necesidades tecnológicas

El sistema será accesible vía Web, lo más seguro posible y estará basado en

herramientas de software libre, tanto para minimizar al extremo el coste en software, como para ofrecer una solución independiente de la plataforma.

Necesidades de diseño

El sistema tendrá un diseño amigable, sencillo y fácil de utilizar. Deberá proporcionar flexibilidad, ser ampliable y adaptable fácilmente, además de tener un bajo acoplamiento y una alta cohesión.

Derivada de las necesidades comentadas anteriormente surgió la idea de este PFC que, como parte del proyecto SEMSE, pretende cubrir dichas necesidades de la forma más óptima posible y servir como punto de partida para la implementación del sistema informático. Para ello, el PFC realizará un estudio de las distintas tecnologías que intervendrán en el desarrollo del sistema, así como la elección de su infraestructura.

En resumen, la motivación de este PFC proviene de la necesidad de minimizar los riesgos derivados de la falta de experiencia en el uso de repositorios semánticos (ontologías), así como de otras tecnologías necesarias para el desarrollo del sistema informático.

1.2. Objetivos

En un estudio como el presente, habría que diferenciar entre dos tipos de objetivos, los que persigue el proyecto SEMSE y los que se persiguen con la realización de este PFC. Dado que el PFC está englobado dentro del proyecto SEMSE y que sus objetivos provienen de los requisitos y las capacidades de él, aquí solo se comentarán los objetivos de este trabajo. Los principales objetivos perseguidos son los descritos a continuación:

- **Evaluar las tecnologías necesarias para el desarrollo óptimo del sistema**. Realizar un estudio de las tecnologías más punteras existentes en la actualidad y seleccionar aquellas que proporcionen la mejor opción para el desarrollo del sistema.

- **Establecer el lenguaje de programación y el entorno de desarrollo.** Seleccionar el lenguaje de programación a utilizar, así como el entorno de trabajo necesario para desarrollar el sistema.
- **Diseñar la arquitectura software del sistema.** Determinar los componentes que mejor satisfagan las necesidades del sistema, maximizando cualidades como adaptibilidad, seguridad, facilidad de uso y extensibilidad.
- **Diseñar la arquitectura hardware del sistema.** Determinar el despliegue de componentes en máquinas o nodos durante la fase de desarrollo.
- **Probar la arquitectura diseñada.** Comprobar que el diseño arquitectónico elegido permite realizar con éxito el desarrollo del sistema. Para ello se realizará una prueba de concepto consistente en la implementación de una funcionalidad diferencial de la aplicación.

1.3. 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 "*Estado del arte"* se abordará 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 tres "*Herramientas para la elaboración del proyecto"* se describe el conjunto de herramientas que se utilizarán para la elaboración del proyecto, tanto en su desarrollo, como en el despliegue de la aplicación.

El capítulo cuatro *"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 cinco *"Conclusiones"* se abordan las conclusiones del proyecto y se presentan las líneas futuras de ampliación y mejora.

Seguidamente vendrá la *"Bibliografía"* con las referencias bibliográficas.

Capítulo 2

Estado del Arte

Tal y como se planteaba en la introducción de este trabajo, el PFC consiste en el análisis y elección de la infraestructura y las tecnologías necesarias para el desarrollo de un sistema informático, accesible vía Web, que permita la reutilización de esquemas y que sea comprensible por el usuario, además de tener un interfaz amigable y fácil de usar.

En este capítulo se realizará la presentación de las tecnologías. Para ello se parte 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 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.

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

Finalmente, se presenta un análisis de las principales tecnologías que se han tenido en cuenta para realizar la implementación del sistema informático cuyo desarrollo se completará en futuros trabajos.

2.1. Web Semántica

Según (LOZANO, 2001), actualmente, 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 navegadores de web realizan la búsqueda de información, con más o menos fortuna, mediante palabras clave que aparecerán en el código HTML 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.

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 debemos 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.

Es indudable que 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 la visión de la web semántica para 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. ¿Qué es la Web Semántica?

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

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 World Wide Web (WWW). Tiene como visión la búsqueda de una Web más inteligente en la que se pueda conseguir una comunicación efectiva entre ordenadores y centra sus esfuerzos principalmente en la búsqueda de descripciones enriquecidas semánticamente para los datos en la Web. En este sentido, se promueven descripciones que incluyan no solo las estructuras de datos, sino también las relaciones existentes con otros conceptos, las restricciones, reglas que permitan realizar inferencia, etc. Así mismo se promueve la definición y reutilización de vocabularios u ontologías de conceptos que faciliten el procesamiento por parte de las máquinas.

Tal y como aparece reflejado en (MALIK, 2003), está previsto que la Web Semántica sea un lugar donde los datos puedan ser compartidos y procesados tanto por herramientas de manera automatizada como por la gente. La clave subyace en la automatización y la integración de los procesos a través de lenguajes legibles por máquinas. En orden a 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 computadores sean mucho más útiles, la Web Semántica extiende la Web actual con conocimiento formalizado y datos que son procesados por ordenadores. Para ser capaz de buscar y procesar información relativa a alguna materia de interés, los programas necesitan información que haya sido modelada de una forma coherente. Una ontología modela todas las entidades y relaciones en un dominio. La ontología es necesaria para la representación del conocimiento.

La clave de las ontologías es que pueden ser compartidas y por lo tanto incrementan en eficiencia e interoperabilidad. Sin embargo, se puede dar el caso en el que dos organizaciones distintas usen dos nombres diferentes para identificar el mismo concepto, es decir, las ontologías sean distintas. En tales casos, la habilidad para asociar los términos de una y otra (mapping o mapeado) es crucial para mantener las ventajas de la Web Semántica.

Agentes

La potencia real de la Web Semántica se conoce cuando agentes capaces de manejar contenido semántico son usados para recoger y procesar información Web e intercambiar los resultados con otros agentes. Herramientas como el intercambio de pruebas o la firma digital asegurarán que los resultados intercambiados entre agentes sean válidos y se pueda confiar en ellos.

2.1.2. El intercambio de conocimientos en la Web Semántica

La idea es que los datos puedan ser utilizados y "comprendidos" por los ordenadores sin necesidad de supervisión humana, de forma que los agentes web puedan ser diseñados para tratar la información situada en las páginas web de manera semiautomática.

Se trata de convertir la información en conocimiento, referenciando datos dentro de las páginas web a metadatos con un esquema común consensuado sobre algún dominio. Los metadatos no sólo especificarán el esquema de datos que debe aparecer en cada instancia, sino que además podrán tener información adicional de cómo hacer deducciones con ellos, es decir, axiomas que podrán aplicarse en los diferentes dominios que trate el conocimiento almacenado (LOZANO, 2001).

Con ello, se mejorará la búsqueda de información y se potenciará el desarrollo de aplicaciones de comercio electrónico, ya que las anotaciones de información seguirán un esquema común, y los buscadores web compartirán con las anotaciones web los mismos esquemas. Empresas que traten con clientes y proveedores, podrán intercambiar sus datos de productos siguiendo estos esquemas comunes consensuados.

Los agentes web no sólo encontrarán la información de forma precisa, si no que podrán realizar inferencias automáticamente buscando información relacionada con la que se encuentra situada en las páginas, y con los requerimientos de la consulta indicada por el usuario.

Para que esto pueda llevarse a cabo, se necesita que el conocimiento de la web esté representado de forma que sea legible por los ordenadores, esté consensuado, y sea reutilizable. Las ontologías proporcionan la vía para representar este conocimiento.

2.1.2.1. De metadatos a ontologías

Los metadatos son datos acerca de datos, y denotan cualquier tipo de conocimiento que puede usarse para conseguir información sobre la estructura y el

contenido de una colección de documentos. Por ejemplo, estableciendo una analogía con una videoteca, los datos serian las cintas de video o dvd´s, mientras que los metadatos serían la información contenida en las fichas, es decir, el título del libro, director, protagonistas, etc. Una de las facetas más importantes en el entorno de los metadatos es la interoperatividad entre diferentes entidades que no tienen porque compartir el mismo conocimiento y tecnología. Los metadatos describen el contenido, calidad, condición, y otras características de los datos. Describen el quién, qué, cuándo, dónde, porqué, y el cómo sobre un conjunto de datos.

Podemos resumir las aportaciones de los metadatos en dos aspectos:

- Información descriptiva sobre un objeto o recurso

DATOS	METADATOS
Tatiana Fernández	Nombre
Calle La Coma S/N	Dirección
Madrid	Ciudad
Madrid	Provincia

Tabla 2.1. Descripción de un recurso

- Permiten el etiquetado o catalogado.

Pero los metadatos sólo van a estructurar los contenidos, por lo que se necesita algo que permita estructurar la semántica de un recurso, ese algo se denomina ontología, figura 2.1. La principal motivación de las ontologías es la de que ellas van a permitir compartir y reutilizar bases de conocimiento de manera computacional.

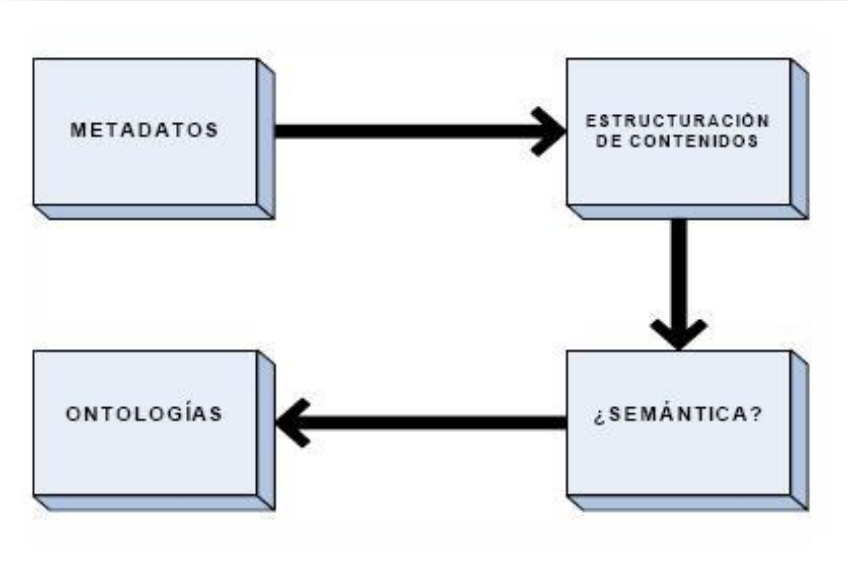

Figura 2.1. De metadatos a ontologías

El uso de ontologías proporciona una forma de representar y compartir conocimiento haciendo uso de un vocabulario común. Mediante esta representación permiten utilizar un formato de intercambio de este conocimiento, y a su vez brindan la posibilidad de ampliar, integrar otras ontologías o reutilizarlas en la aplicación de otros dominios. Separan el conocimiento del dominio del conocimiento operacional, haciendo independientes las técnicas y algoritmos, así como hacen que nuestras suposiciones sobre el dominio se hagan explícitas, lo que facilita replantearse éstas y ayuda a que otros puedan entender su descripción. Por último, permiten analizar el conocimiento del dominio utilizando métodos formales (HONRUBIA, 2002).

Pero, ¿Qué se quiere decir con semántica? Para responder a esta pregunta pensemos en lo que ocurre cuando leemos un texto y encontramos en él símbolos. Estos símbolos serán interpretados (su significado) por nosotros con respecto a un modelo mental. Es decir, nosotros poseemos el significado (semántica) de alguna parte de/del mundo en nuestras mentes. Cualquier otro individuo realizará su interpretación según su propio modelo mental que obviamente no tiene porque ser común al resto de individuos. Por tanto podemos asegurar que no hay un conocimiento en estos documentos sino alguien o algo que interprete su semántica. Si nosotros deseamos

diseminar conocimiento embebido en los documentos, necesitaremos, al menos parcialmente, automatizar el proceso de interpretación semántica. Necesitamos describir y representar computacionalmente una porción de nuestros modelos mentales sobre dominios específicos. Las ontologías son una herramienta fundamental para lograr este objetivo. Las ontologías intentan limitar las posibles interpretaciones a un solo modelo mental, el que nosotros queremos expresar. Ningún otro modelo como puedan ser las taxonomías, bases de datos etc. es capaz de hacerlo. Los computadores de esta forma suplantarán a los humanos en la interpretación de los diferentes vocabularios, mediante un proceso de inferencia que se asemejará al proceso de interpretación o razonamiento humano (DACONTA, 2003).

Derivado de lo dicho anteriormente se concluye que un lenguaje de representación del conocimiento debe incluir:

- Sintaxis del lenguaje. Describe la configuración que puede constituir sentencias.
- Semántica. Determina los hechos y significado basado en las sentencias.

2.1.3. 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 2.2, define la interacción entre símbolos o palabras, pensamientos y cosas del mundo real.

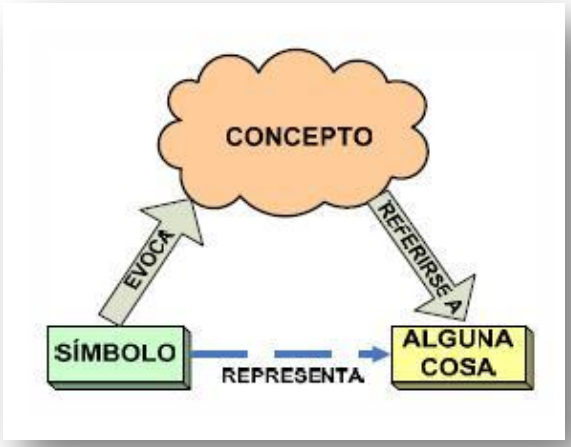

Figura 2.2. Triángulo del Significado

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. Por ejemplo, utilizaremos los términos o símbolos "Tatiana" + "Fernández" (sintaxis: símbolos) para dar un sentido o evocar al concepto <Tatiana Fernández> y este concepto referenciará o denotará (mediante su semántica: significado) a un referente en el mundo real o posible (aspecto pragmático: uso). 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:

> "u*na 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.

2.1.3.1. 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).

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.

2.2. Diseño de Aplicaciones Web

De un tiempo a esta parte (CASTEJÓN, 2004), la rápida expansión de Internet ha supuesto una transformación en las necesidades de información de las 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 quiere desarrollar está claramente orientada al usuario se intentará 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. Para comprender la importancia de este elemento, sólo tendríamos que recordar cuál fue el último portal Web visitado que nos sorprendió positivamente, ya fuera por su presentación o eficacia.

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, el sitio debe verificarse en las últimas dos versiones de Internet Explorer y Firefox en sistemas Microsoft Windows y Apple Mac, última versión de Firefox y Konqueror en Linux y en un navegador de sólo texto tipo Lynx.

2.2.2. Usabilidad de un sitio Web

Actualmente, para un desarrollador de software es necesario conocer los principios básicos de la arquitectura de la información, la usabilidad y la ingeniería de la interfaz de usuario. Como arquitecto de la información, el desarrollador deberá organizar los patrones inherentes en los datos, clarificando lo complejo y creará la estructura o mapa de la información que permitirá a otras personas encontrar sus caminos personales al conocimiento. Deberá, además, clarificar la misión y visión del sitio, equilibrando las necesidades de la organización y de los potenciales usuarios. Determinará, por último, como acomodará el sitio el cambio y crecimiento a lo largo del tiempo.

En la medida que aumenta el tamaño y complejidad de los sitios y aplicaciones Web, los desarrolladores afrontan mayores retos en la creación de sitios eficaces y funcionales, y la usabilidad se convierte en una característica clave de esos sitios.

La facilidad de aprendizaje y utilización de un sistema de información es lo que se denomina usabilidad de un sistema. La usabilidad necesita jugar un papel importante desde el principio de cada proyecto y estar al mismo nivel que las demás características tradicionales de un software de calidad.

2.2.2.1. Evaluación heurística de Jakob Nielsen

Nielsen (NIELSEN, 1994) presenta diez principios genéricos para el diseño de interfaces de usuario usables. Las denomina heurísticas porque están más en la intención de definir reglas generales que guías específicas de usabilidad. Para el desarrollo de la aplicación se seguirán, en el grado en que sea posible, estos diez principios básicos de la usabilidad.

Visibilidad del estado del sistema

El sistema debe mantener siempre informado al usuario sobre lo que está sucediendo, mediante la realimentación apropiada en un tiempo razonable.

Coherencia entre sistema y mundo real

El sistema debe hablar el lenguaje del usuario, con palabras, frase y conceptos familiares, más que en términos orientados al sistema. Se deben seguir las convenciones del mundo real, mostrando la información en un orden lógico y natural.

Control de usuario y libertad

Los usuarios a menudo seleccionan funciones del sistema por error y necesitarán una indicación clara de “salida de emergencia” para abandonar el estado no deseado sin tener que atravesar un diálogo extenso. Se deben soportar las operaciones deshacer y rehacer.

Consistencia y estándares

Los usuarios no deberían preguntarse si diferentes palabras, situaciones o acciones significan lo mismo. Se deben seguir las convenciones de plataforma.

Prevención de errores

Incluso mejor que buenos mensajes de error es un cuidado diseño que prevenga la ocurrencia de errores al primer intento.

Reconocimiento en lugar de recuerdo

Se deben hacer visibles objetos, acciones y opciones. El usuario no debe tener que recordar información de una parte del diálogo a otra. Las instrucciones de uso del sistema deben ser visibles y fácilmente recuperables donde sea apropiado.

Flexibilidad y eficiencia de uso

Los aceleradores, que pasan desapercibidos para el usuario novato, pueden acelerar la interacción para el usuario experto, de manera que el sistema puede atender tanto a los usuarios con experiencia como a los que no la tienen. Permitirán que el usuario adapte acciones frecuentes.

Diseño estético y minimalista

Los cuadros de diálogo no deberán contener información que sea irrelevante o que se utilice rara vez. Cada unidad de información adicional compite con las unidades relevantes y disminuye su visibilidad relativa.

Ayudar a que los usuarios reconozcan, diagnostiquen y solucionen errores

Los mensajes de error deben tener un lenguaje simple (sin código), indicar precisamente el problema y sugerir constructivamente una solución.

Ayuda y documentación

A pesar de que es mejor si el sistema se puede utilizar sin documentación, puede resultar necesario proporcionar ayuda y documentación. Esta información deberá encontrarse fácilmente, centrarse en la tarea del usuario, enumerar los pasos concretos que se han de dar y no ser demasiado grande.

2.3. Desarrollo de software

En esta sección se explicará qué es un proceso de desarrollo software, se mostrarán sus actividades y su organización en modelos de ciclo de vida. Se describirá

brevemente el lenguaje de modelado unificado y, por último, se presentarán los estilos y patrones de diseño que se utilizarán en este proyecto.

2.3.1. Proceso de desarrollo de software

La idea de aplicar el concepto de proceso al desarrollo software proviene del campo de la fabricación, donde los procesos (pasos en la fabricación) están definidos y se controlan de manera continua. Este enfoque se puede aplicar al mundo del software para la gestión del proceso a nivel de proyecto y para mejorar las capacidades de los grupos de desarrollo. Según (CUEVAS, 2003), el proceso software establece el marco de trabajo tanto técnico como de gestión para poder aplicar los métodos, herramientas y personas a la tarea de desarrollo de software. La definición del proceso identifica los roles y las tareas específicas y establece medidas para el control de ejecución de cada paso.

Se denomina definición del proceso software a la descripción del proceso que se sigue para elaborar el software. La definición adecuada del proceso permitirá a una organización dedicada al desarrollo de software asegurar que cada elemento de trabajo se asigna apropiadamente y que se conoce su estado en cada momento. A su vez, esta definición oficial del proceso en la organización, permitirá incorporar nuevos métodos que lo mejoren.

Un proceso definido permite que cada nuevo proyecto sea construido en base a la propia experiencia y la de sus predecesores. Los problemas recogidos a medida que se utiliza el proceso permiten identificar sus causas y corregirlos. De este modo, tanto el proceso, como su definición y la infraestructura de soporte evolucionarán con la experiencia.

En la figura 2.3 se muestra el proceso de desarrollo donde se han incorporado las actividades de gestión: gestión de requisitos, planificación, gestión de la configuración y gestión de calidad.

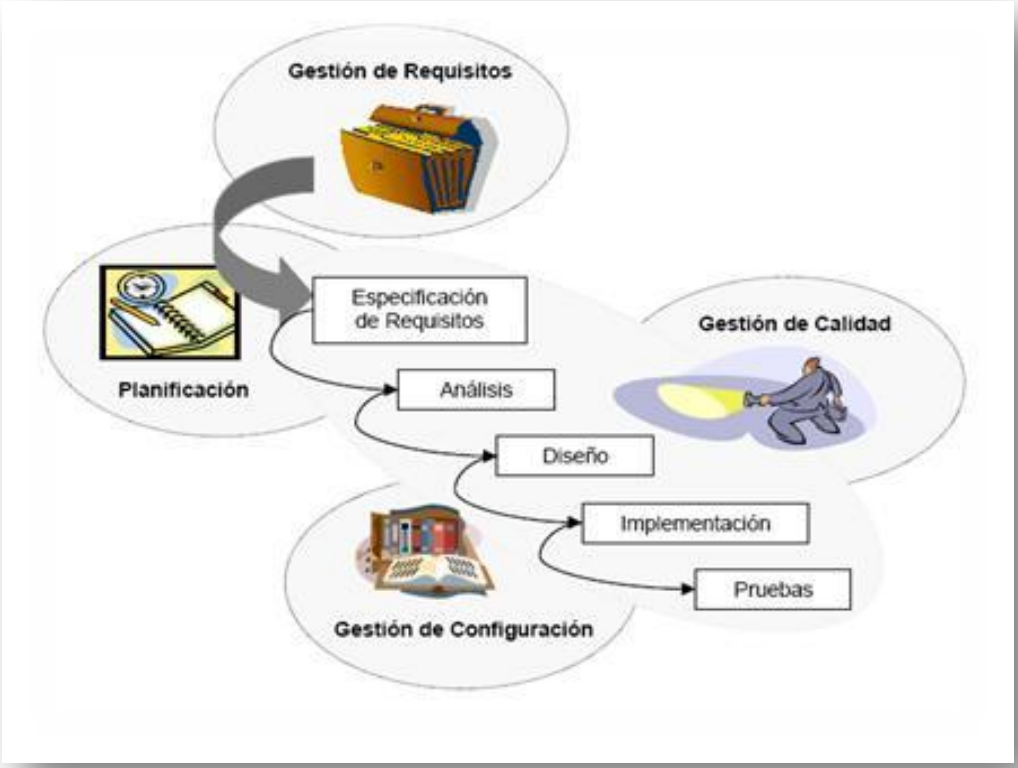

Figura 2.3. Representación de un proceso de desarrollo software

2.3.1.1. Actividades de gestión del Proceso Software

Las actividades de gestión están orientadas a controlar el desarrollo del proceso software y a corregir las desviaciones con respecto a los parámetros de calidad establecidos. Así, son tareas que se desarrollan a lo largo del ciclo de vida del software.

Gestión de requisitos

La gestión de requisitos, una vez obtenida la especificación de requisitos, se ocupa de fijar el marco de referencia, de carácter contractual, que determinará el desarrollo. Incluirá todas las actividades de control que mantengan la integridad y exactitud del acuerdo sobre los requisitos a medida que progrese el proyecto. Para ello, llevará a cabo el tratamiento y control de las actualizaciones y cambios a los mismos garantizando su trazabilidad en el ciclo de vida.

Gestión de configuración

La gestión de configuración del software es el medio que permite conocer, en todo momento, qué componentes y versiones, tanto de un producto como de sus elementos, son las correctas. De forma más concreta, la gestión de configuración es el proceso de identificar y definir los elementos de la configuración[1] para:

- Controlar la liberación de versiones y cambios durante todo el ciclo de vida.
- Registrar e informar de su estado y de las peticiones de cambios.
- Verificar la corrección y acabado de los elementos.

Gestión de calidad

Una vez establecidos los objetivos de calidad de un proyecto software y especificados aquellos procesos operativos y recursos necesarios para satisfacer dichos objetivos, la gestión de calidad será el conjunto de actividades diseñadas para evaluar el proceso por el que se desarrollan los productos. Deberá garantizar que el sistema, componente o proceso cumple los requisitos especificados y cubre las necesidades o expectativas del cliente o usuario.

Planificación

La planificación del proyecto consistirá en establecer la estructura temporal de las fases, actividades y tareas del proyecto, en función de los recursos de que se disponga. El seguimiento y control de evolución determinarán la necesidad de nuevas planificaciones que puedan alterar la fecha de finalización de proyecto.

2.3.2. Ciclo de vida del software

El ciclo de vida software es el periodo que comienza cuando un producto software es concebido y termina cuando deja de estar disponible (Cuevas, 2003). El ciclo de vida se divide normalmente en fases que estructuran y organizan las etapas de concepción, desarrollo y mantenimiento. Un modelo de ciclo de vida es la descripción de las distintas formas de desarrollo de un proyecto, es decir, la orientación que debe

[1] Elemento de configuración es un conjunto software, y, opcionalmente hardware, tratado como una unidad por la gestión de configuración

seguirse para obtener a partir de los requerimientos del cliente, sistemas que puedan ser utilizados por dicho cliente.

Las funciones principales de un ciclo de vida software son:

- Determinar el orden de las fases y procesos involucrados en el desarrollo del software y su evolución incluyendo la explotación y el mantenimiento.
- Establecer los criterios de transición para pasar de una fase a la siguiente (productos intermedios). Todo ello incluye los criterios para verificar la terminación de la fase actual y los criterios para seleccionar e iniciar la fase siguiente.

El modelo de ciclo de vida se escoge en la planificación inicial del proyecto. Cada proyecto se estructura en actividades, que serán previamente planificadas y agrupadas en fases. Además, los productos deben ser terminados y entregados, de acuerdo al plan, y funcionar correctamente. Todo ciclo de vida debe incluir las fases incluidas en la tabla 2.2.

FASE	DEFINICIÓN	ENTREGABLE
UR	Definición de Requerimientos de Usuario	**URD**
SR	Definición de Requerimientos del Software	**SRD**
AD	Diseño de la Arquitectura	**ADD**
DD	Diseño Detallado y codificación	**DDD, SUM, CODE**
TR	Transferencia del software	**STD**
OM	Operación y mantenimiento	**PHD**

Tabla 2.2. Fases de desarrollo del proyecto

Las primeras cuatro fases finalizan con una revisión. Estas fases han de aparecer siempre, independientemente del tamaño, tipo, software empleado, o equipo de desarrollo, ya sea propio o externo. El ciclo de vida del software, siguiendo la notación

del estándar PSS-05 de la ESA[2] (ESA, 2008), empieza realmente con la entrega del Documento de Requisitos de Usuario (URD, User Requirements Document) a los desarrolladores. Por lo tanto, la revisión del URD es la primera actividad del ciclo de vida. Junto con esta revisión, ha de producirse la entrega de un plan de gestión de proyecto, incluyendo una planificación y una estimación de costes para el proyecto. Los entregables de cada fase deben de ser revisados y aprobados antes de proceder con la siguiente fase. Hay seis hitos que indican el progreso del ciclo de vida:

- Aprobación del Documento de Requisitos de Usuario (User Requirements Document: URD).
- Aprobación del Documento de Requisitos de Software (Software Requirements Document: SRD).
- Aprobación del Documento de Diseño Arquitectónico (Architectural Design Document: ADD).
- Aprobación del Documento de Diseño Detallado (Detailed Design Document: DDD), Manual de Usuario del Software (Software User Manual: SUM) y Código, así como confirmación de disponibilidad para las pruebas de aceptación provisional.
- Entrega del Documento de Transferencia del Software (Software Transfer Document: STD) y confirmación de aceptación provisional.
- Entrega del Documento Histórico del Proyecto (Project History Document: PHD) y confirmación de aceptación final.

A continuación se describen las fases del ciclo de vida del software.

Definición de requisitos del usuario

En esta fase se trata de plantear el problema que deberá ser solucionado mediante un sistema software. Para ello, se realizarán entrevistas con el cliente, de forma que se obtienen los requisitos que debe tener el sistema para resolver sus necesidades.

[2] El documento ESA PSS-05-0 describe los estándares de software que deben ser aplicados a todas las entregas de software. Fue desarrollado por la Agencia Espacial Europea (ESA).

Definición de requisitos del software

Esta es la fase de análisis del problema. A partir de las necesidades que han originado la puesta en marcha del proyecto, se trata de comprender *qué es* lo que tiene que realizar el sistema en su conjunto. Se analizará el entorno en el que se desenvolverá el sistema y la manera en que éste interactuará con él.

Diseño de la arquitectura

Esta fase corresponde al planteamiento de la solución. El sistema se descompone en un conjunto de subsistemas, de manera que se simplifica el sistema global. Se definen las relaciones que tienen cada uno de los subsistemas con los demás.

Diseño detallado y codificación

En esta fase se refina el diseño realizado en la fase anterior. Además se realiza la implementación del sistema, se valida y se verifica, de manera que el sistema realizado cumpla con los requisitos definidos en las fases iníciales.

Transferencia del software

En esta fase, el sistema es transferido a su entorno de explotación. Puede haber un periodo de enseñanza para que el usuario conozca la manera de utilizar el sistema.

Operación y mantenimiento

Esta suele ser la fase más larga del ciclo de vida. Transcurre durante el uso del sistema en su entorno de explotación. Además tendrán lugar actuaciones de mantenimiento y mejora del sistema.

El proyecto se basa en un ciclo de vida "híbrido" entre el modelo de ciclo de vida en cascada y el modelo de ciclo de vida iterativo-incremental. A continuación se describirán estos dos ciclos de vida por su relación con este trabajo.

2.3.2.1. Ciclo de vida en cascada

El ciclo de vida en cascada es el ciclo más simple. Históricamente, fue el primer ciclo en aparecer. 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 (Cuevas, 2003).

Este ciclo de vida se ha utilizado para realizar un primer enfoque de los requisitos y análisis del proyecto a nivel global. En la Figura 2.4 se esquematiza este ciclo de vida.

Figura 2.4. Ciclo de vida en cascada

2.3.2.2. Ciclo de vida iterativo- incremental

Este ciclo de vida comienza determinando la especificación de requisitos y realizando un análisis del sistema, para ello se recogen las necesidades del usuario y se realiza el diseño del sistema y la planificación de la construcción. 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. En otras palabras en el ciclo de vida iterativo-incremental se va creando el sistema software añadiendo componentes funcionales al sistema.

En la ejecución de este proyecto primeramente se realizó una especificación de requisitos de usuario y un análisis completo basado en el ciclo de vida en cascada. Posteriormente se ha realizado el diseño arquitectónico de forma iterativa/incremental (añadiendo componentes uno a uno) y, finalmente, se realizará

el diseño detallado e implementación también de forma iterativa/incremental. En la Figura 2.5 se muestra el ciclo de vida iterativo-incremental.

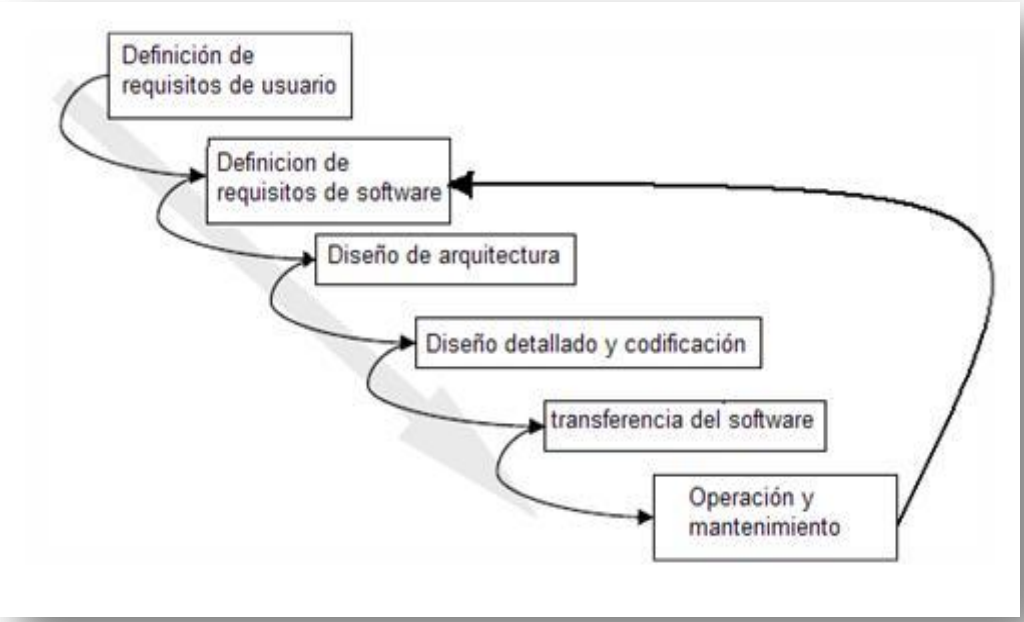

Figura 2.5. Ciclo de vida iterativo-incremental

2.3.3. UML

El lenguaje de modelado unificado UML (Unified Modeling Language) es un lenguaje estándar con el que es posible modelar todos los componentes del proceso de desarrollo de aplicaciones. Como se comenta en (BOOCH, 2006) UML es el lenguaje más conocido y utilizado en la actualidad y está respaldado por el OMG (Object Management Group). Para entender que es UML basta con analizar por separado cada una de las palabras que lo componen:

- **Lenguaje:** el UML es, precisamente, un lenguaje. Lo que implica que éste cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.
- **Modelado:** el UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.

- **Unificado**: unifica varias técnicas de modelado en una única.

Las raíces del UML provienen de tres métodos distintos: el método de Grady Booch, la Técnica de Modelado de Objetos de James Rumbaugh y "Objectory" de Ivar Jacobson. En UML, los procesos de desarrollo son diferentes según los distintos dominios de trabajo. UML recomienda utilizar los procesos que otras metodologías tengan definidos. 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.

El lenguaje UML es una técnica de modelado de objetos, y como tal, supone una abstracción de un sistema para llegar a construirlo en términos concretos. El modelado no es más que la construcción de un modelo a partir de una especificación. Un modelo es una abstracción de la realidad que se quiere representar, el cual se elabora para comprender la realidad antes de construir el sistema. El modelo omite detalles que no resultan esenciales para la comprensión del original, por lo que facilita la comprensión. Los modelos, además, permiten comprobar más fácilmente los sistemas que se modelan y detectar los posibles errores.

Para representar la realidad, se utilizan diversos diagramas que ofrecen distintos puntos de vista de esta realidad. Un diagrama es una representación gráfica de una colección de elementos del modelo. A continuación se comentan los diferentes diagramas que ofrece UML 1.5, que es la versión utilizada para el desarrollo del proyecto.

2.3.3.1. Diagramas estáticos o estructurales

Los diagramas estáticos permiten visualizar, especificar, construir y documentar los aspectos estáticos de un sistema. Los diagramas estáticos hacen referencia a la descripción de los componentes del sistema, así como de sus relaciones.

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.

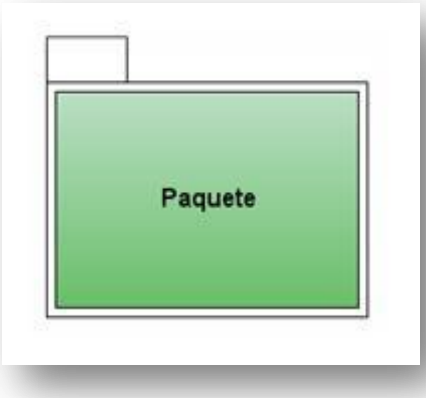

Figura 2.6. Figura de un paquete

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

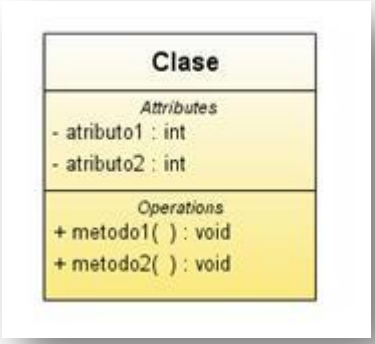

Figura 2.7. Figura de una clase

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

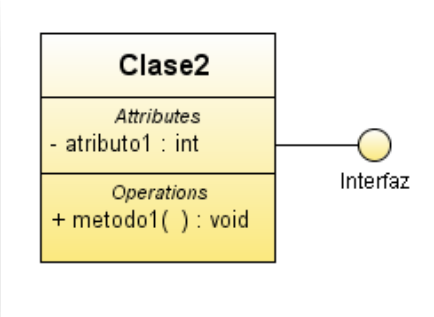

Figura 2.8. Figura de una interfaz

En estos diagramas se muestra, además, las relaciones estructurales existentes entre los elementos descritos anteriormente. Un ejemplo de un diagrama de clases completo, podría ser el de la Figura 2.9.

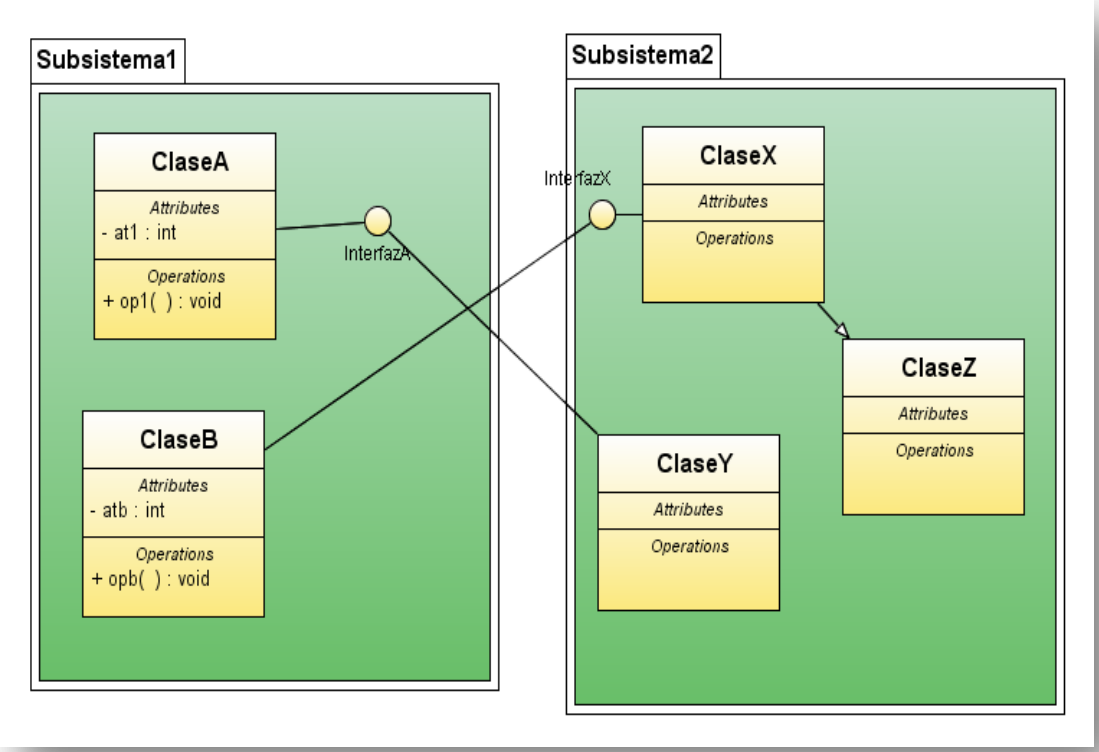

Figura 2.9. Ejemplo de diagrama de clases

Diagrama de objetos

Un diagrama de objetos se puede considerar un caso especial de un diagrama de clases. Los diagramas de objetos usan un subconjunto de elementos de un diagrama de clase para enfatizar la relación entre las instancias de las clases en algún punto en el tiempo. En otras palabras, en este diagrama se modelan las instancias de las clases del diagrama de clases. Muestra los objetos y sus relaciones, pero en un momento concreto del sistema. Estos diagramas contienen objetos y enlaces. En los diagramas de objetos también se pueden incorporar clases, para mostrar la clase de la que es un objeto representado.

El diagrama de la figura 2.10 muestra un diagrama objeto con su intercalación de clase definida, e ilustra la forma en la que un diagrama objeto se puede usar para probar las multiplicidades de tareas en los diagramas de clase.

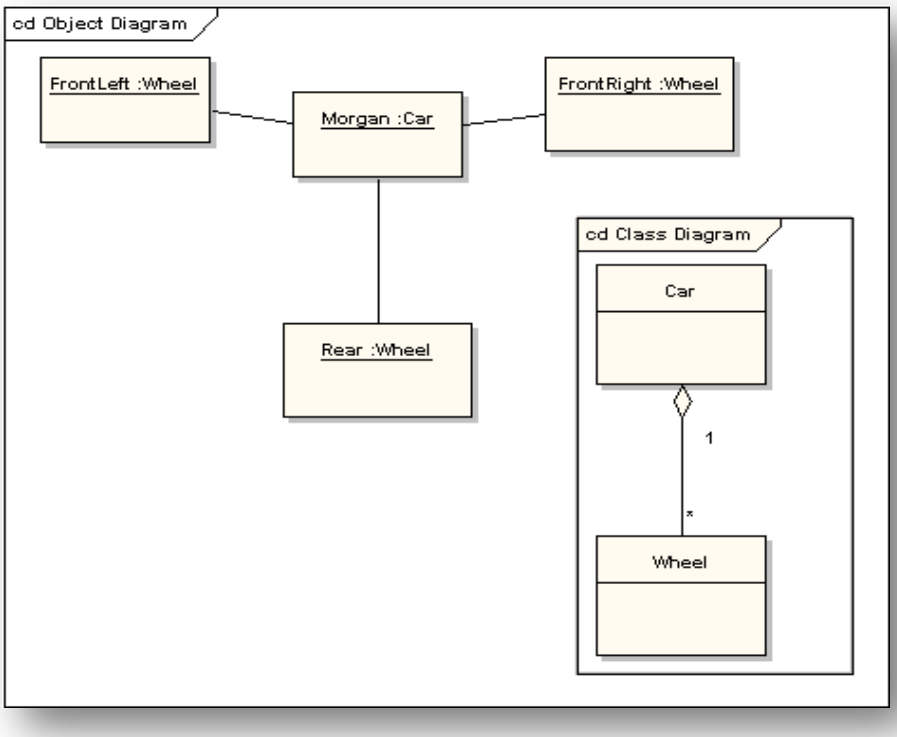

Figura 2.10. Ejemplo de diagrama de objetos

Diagramas de implementación

A diferencia del resto de diagramas, que expresan la lógica del sistema, estos diagramas modelan el sistema desde un punto de vista físico. Se entiende como físico también, los ficheros ejecutables y cualquier otro tipo de archivo que intervenga en el sistema.

Los dos tipos de este diagrama, componentes y despliegue, se comentan a continuación.

Diagramas de componentes

Muestra la dependencia entre los distintos componentes de software, incluyendo componentes de código fuente, binario y ejecutable. Un

componente es una pieza del sistema que se está representando. En la figura 2.11 se muestra un ejemplo de este diagrama.

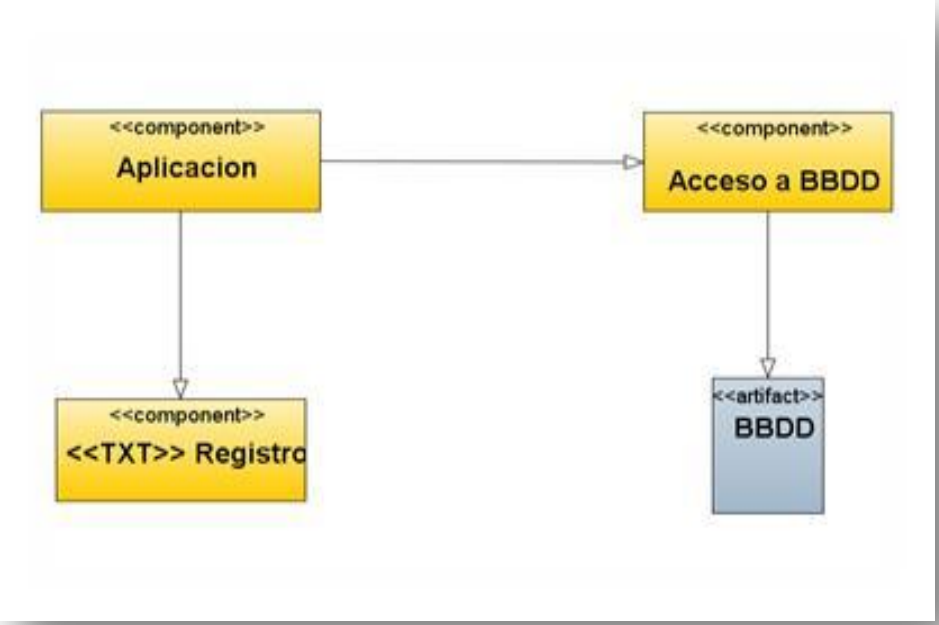

Figura 2.11. Ejemplo de diagrama de componentes

Diagramas de despliegue

Muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. En este tipo de diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de los nodos y objetos que se encuentran a su vez dentro de los componentes. Un **nodo** es un objeto físico en tiempo de ejecución, es decir, una máquina que se compone habitualmente de, por lo menos, memoria y capacidad de procesamiento, a su vez puede estar formada por otros componentes.

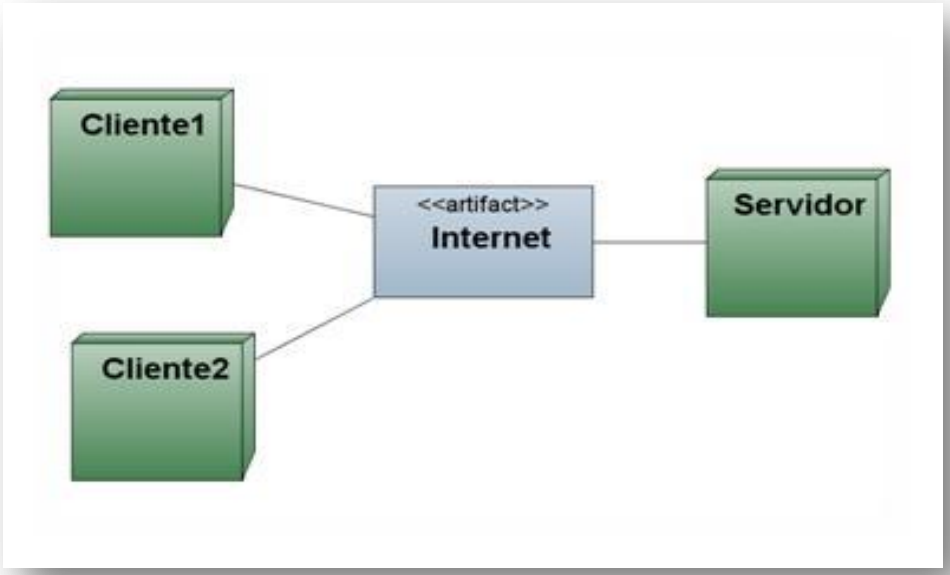

Figura 2.12. Ejemplo diagrama de despliegue

2.3.3.2. Diagramas dinámicos o de comportamiento

Los diagramas dinámicos se emplean para visualizar, especificar, construir y documentar los aspectos dinámicos de un sistema. Los diagramas dinámicos hacen referencia al comportamiento del sistema en tiempo de ejecución. Complementan a los diagramas estáticos.

Diagramas de casos 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 2.13 se muestra un ejemplo.

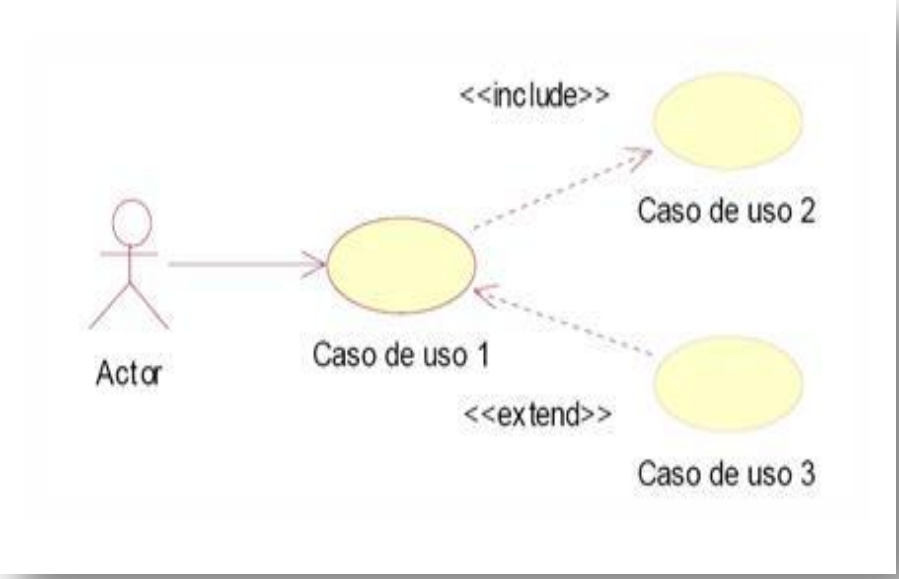

Figura 2.13. Ejemplo diagrama de casos de uso

Diagramas de estados

Representan la secuencia de los posibles estados por los que un objeto, o una interacción entre objetos, transcurre durante su tiempo de vida, en respuesta a los eventos recibidos. Modela el sistema como una máquina de estados.

Un **estado** en UML, se puede definir como la situación en que se encuentra un objeto o una interacción cuando satisface una condición, desarrolla alguna acción o se encuentra esperando un evento. Cuando un objeto o una interacción pasa de un

estado a otro, debido a un evento, se dice que ha sufrido una **transición**. Puede suceder, que con la aparición de un evento, el estado de origen y el de destino sea el mismo, teniendo así, una transición reflexiva.

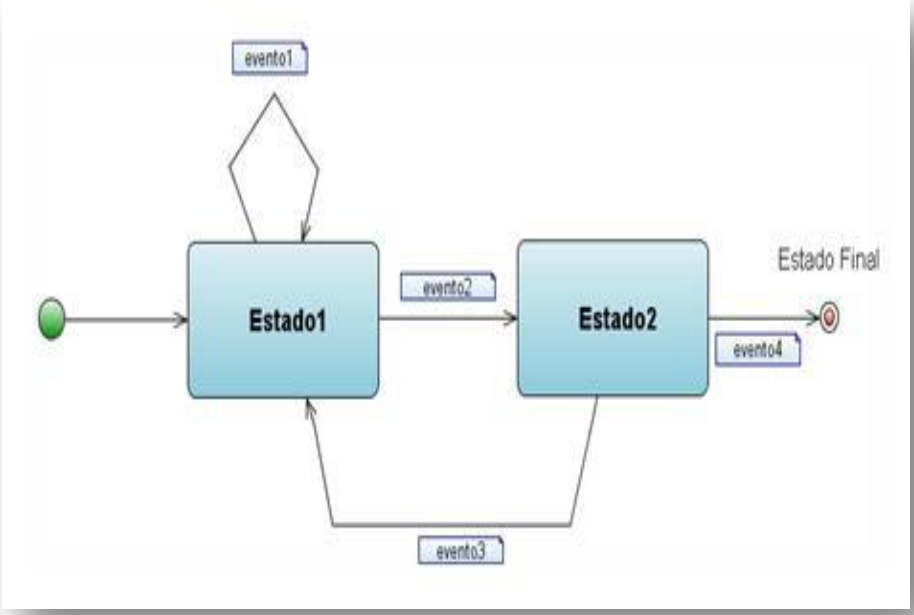

Figura 2.14. Ejemplo diagrama de estados

Diagrama de actividades

Son similares a los diagramas de flujo tradicionales. En realidad, se corresponden con un caso especial de los diagramas de estado, donde los estados son estados de acción, y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Es decir, estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o dicho de otro modo, un paso en la ejecución de lo que será un procedimiento.

Generalmente, estos diagramas están asociados a la implementación de un caso de uso, o de un método. Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema.

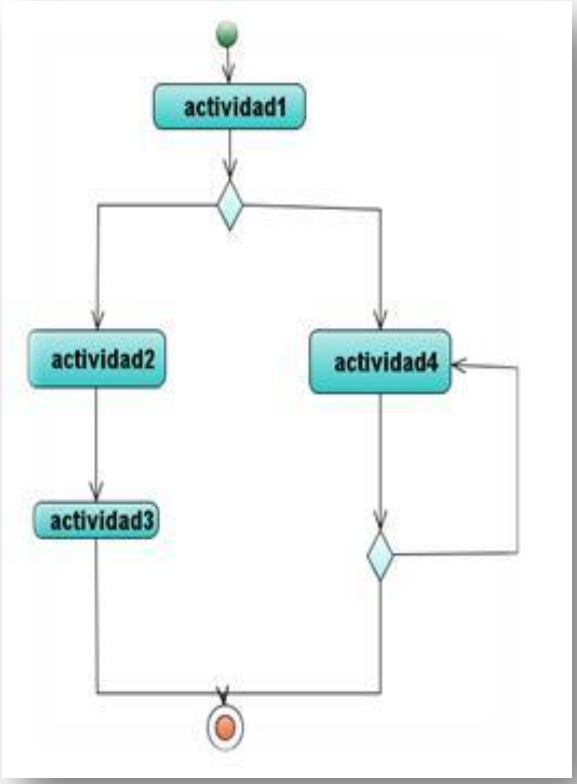

Figura 2.15. Ejemplo diagrama de actividades

Diagramas de interacción

Estos diagramas muestran las relaciones de interacción que tienen los objetos entre sí. Tienen cierto carácter dinámico, ya que representan las interacciones a lo largo del tiempo.

Los dos tipos de estos diagramas, secuencia y colaboración, son equivalentes. La diferencia está en que en el diagrama de secuencia se pone más énfasis en el transcurso del tiempo, mientras que en el diagrama de colaboración se enfatiza más las relaciones de interacción.

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, se puede crear o puede ser destruido durante la ejecución de la interacción. 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, por parte de un objeto origen, de una operación de 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.

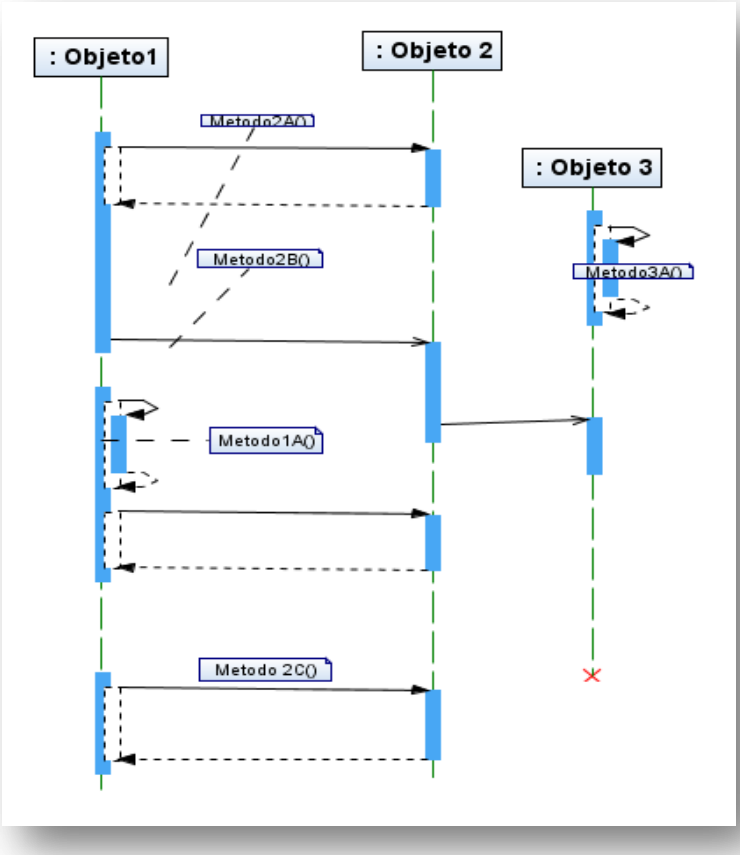

Figura 2.16. Ejemplo diagrama de secuencia

Diagrama de colaboración

Muestran la interacción entre varios objetos y los enlaces que existen entre ellos. Representa las interacciones entre objetos organizadas alrededor de los

objetos y sus vinculaciones. A diferencia de un diagrama de secuencia, un diagrama de colaboraciones muestra las relaciones entre los objetos, no la secuencia en el tiempo en que se producen los mensajes. Los componentes de los diagramas de colaboración son los objetos, enlaces y mensajes. Un **enlace** es una instancia de una asociación que conecta dos objetos de un diagrama de colaboración. El enlace puede ser reflexivo si conecta a un elemento consigo mismo. La existencia de un enlace entre dos objetos indica que puede existir un intercambio de mensajes entre los objetos conectados. Los diagramas de interacción muestran el flujo de mensajes entre elementos del modelo. El flujo de mensajes representa el envío de un mensaje desde un objeto a otro, si entre ellos existe un enlace.

Los mensajes que se envían entre objetos pueden ser de distintos tipos según como se produzcan en el tiempo. Los tipos de mensajes básicos son, al igual que en los diagramas de secuencia, simples, síncronos y asíncronos.

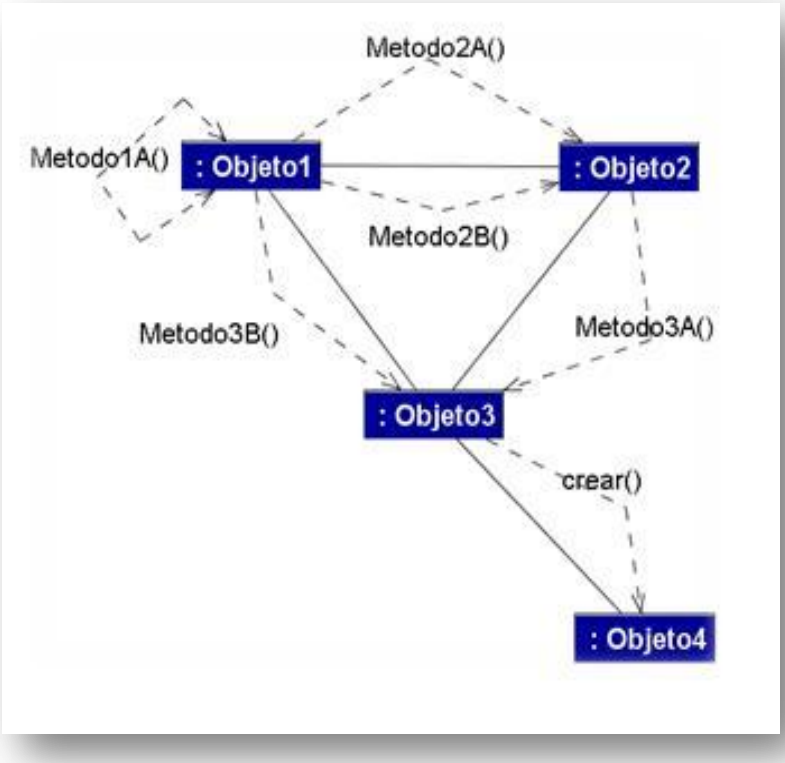

Figura 2.17. Ejemplo diagrama de colaboración

2.3.3.3. UML 2

UML 2 es la mayor revisión que se le ha hecho a UML desde la versión 1.0. El modelo conceptual ha sido reestructurado completamente y nuevos diagramas han sido incorporados. Los diagramas tradicionales también han sido mejorados (UML 2.0, 2008).

En UML 2 el diagrama de colaboración se ha transformado en el Diagrama de Comunicación. Adicionalmente se incorporan los siguientes diagramas:

- **Diagrama de Estructura Compuesta**: se emplea para visualizar de manera gráfica las partes que definen la estructura interna de un clasificador. Cuando se utiliza en el marco de una clase, este diagrama permite elaborar un diagrama de clases donde se muestran los diferentes atributos (partes) y las clases, a partir de las cuales se definen los atributos, indicando principalmente las asociaciones de agregación o de composición de la clase a la que se le elabora el diagrama.
- **Diagrama General de Interacción**: se emplea fundamentalmente para representar las interacciones, a través de diagramas o fragmentos de diagramas de secuencias, entre los actores y el sistema como una gran caja negra, y de diagramas de actividades en los que aparecen dichos fragmentos.
- **Diagramas de Tiempos**: empleados para mostrar las interacciones donde el propósito fundamental consiste en razonar sobre la ocurrencia de eventos en el tiempo que provocan el cambio de estados de un elemento estructural (clase, componente, etc.).
- **Diagrama de Comunicación**: equivalente al diagrama de colaboración de UML 1.x. Permite especificar interacciones entre objetos que conforman la estructura interna de un clasificador.

2.3.4. 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 mil ocasiones sin repetir la forma dos veces (GAMMA, 1998). Del mismo modo que los novelistas y escritores raramente idean el argumento de su obra desde cero, y siguen patrones (novela romántica, trama policíaca ó héroes desterrados), 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 y componentes, o de la relación entre ellos, o bien, desacoplan componentes relacionados. Otros ayudan a implementar ciertos aspectos de diseño en un lenguaje de programación específico.

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

2.3.4.1. 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 (BUSCHMANN, 1996). A continuación se describirán los patrones MVC y Layers (capas).

MVC

MVC (Model View Contoller o modelo Vista Controlador) 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.

Este patrón es muy popular y ha sido portado a una gran cantidad de entornos y frameworks como pueden ser Spring, Struts, JSF, etc. El MVC es un patrón ampliamente utilizado en múltiples plataformas y lenguajes. Algunos de sus principales beneficios son:

- Menor acoplamiento
  - Desacopla las vistas de los modelos
  - Desacopla 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
- 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

- Una vista para cada dispositivo que puede varias según sus capacidades
- Una vista para la Web y otra para aplicaciones de escritorio

➢ Más claridad de diseño

➢ Facilita el mantenimiento

➢ Mayor escalabilidad

Actualmente existen dos tipos de patrón MVC: MVC Modelo 1 y MVC Modelo 2.

MVC Modelo 1

En el MVC Modelo 1 las páginas JSP 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), y se transforma el modelo para ser desplegado una vez más. Esta arquitectura se puede ver en la figura 2.18.

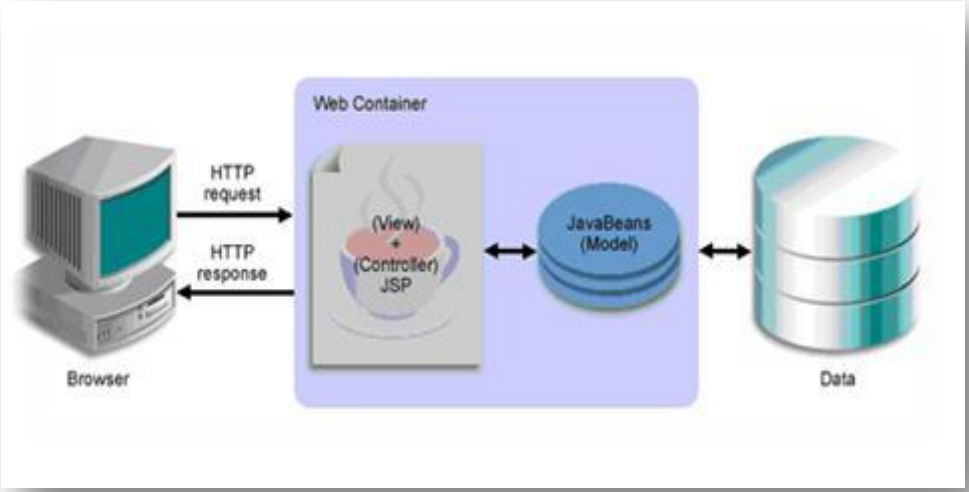

Figura 2.18. Arquitectura MVC Modelo 1

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. En la figura 2.19 se muestra esta arquitectura MVC que se utiliza para aplicaciones web.

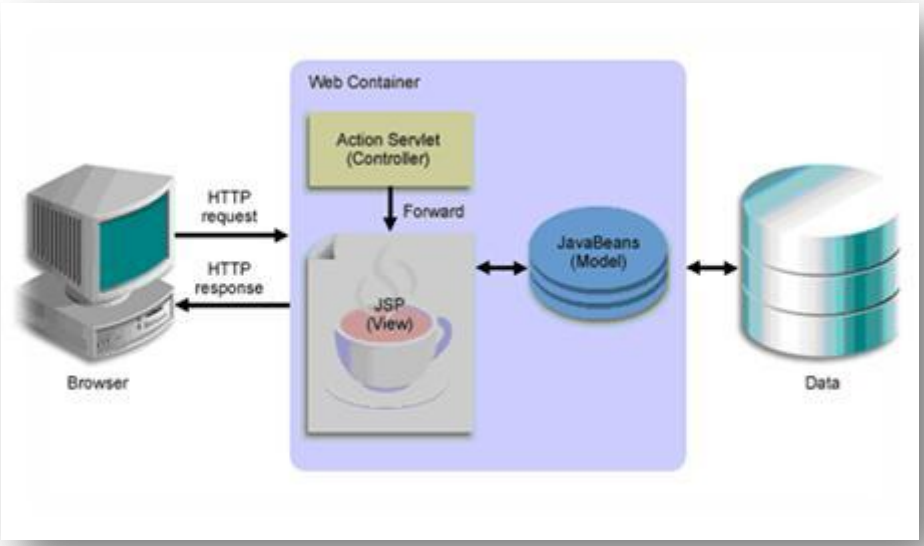

Figura 2.19. Arquitectura MVC Modelo 2

Layers

El patrón layers (capas) 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. En la Figura 2.20 se muestra la estructura descrita.

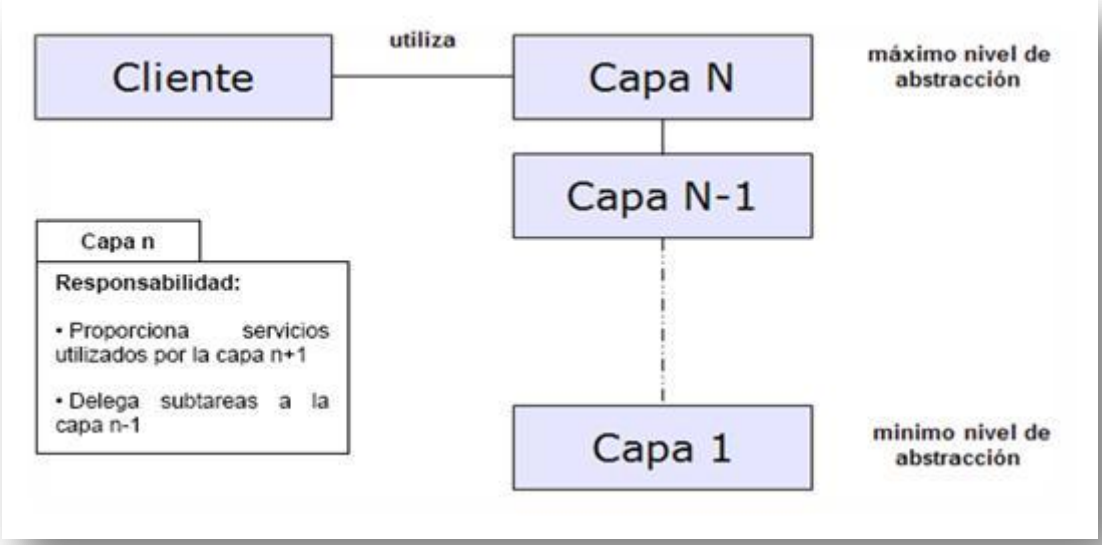

Figura 2.20. Estructura del patrón capas

Examinando las capas individuales en mayor detalle se observará que son entidades complejas que constan de diferentes componentes. En la Figura 2.21 se muestra una posible arquitectura, en la que los componentes de cada capa utilizan los servicios proporcionados por los componentes de la capa inferior.

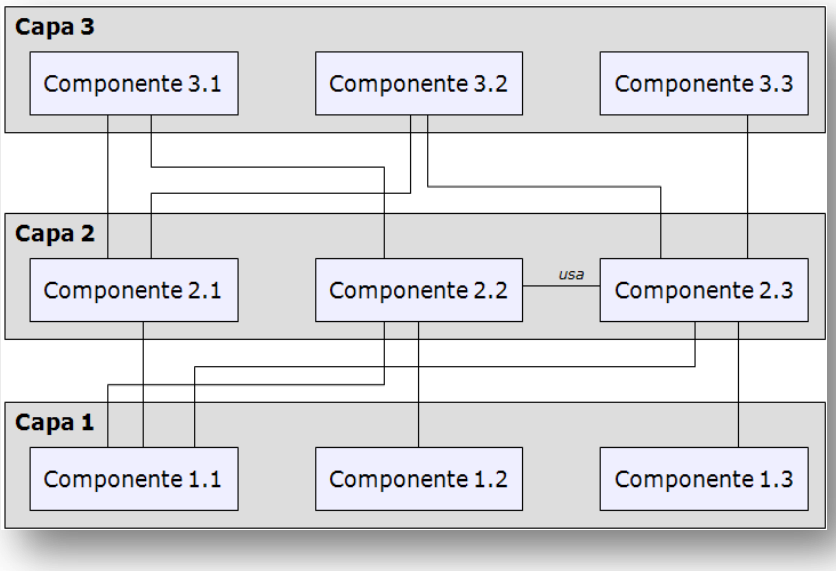

Figura 2.21. Composición y comunicación de capas

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

- Definir el criterio de abstracción para agrupar las tareas en capas. Por ejemplo, la distancia desde el hardware puede dirigir las capas bajas y la complejidad conceptual las altas.
- Determinar el número de niveles de abstracción de acuerdo al criterio anterior. Cada nivel corresponderá a una capa del patrón. Debe alcanzarse un punto de equilibrio entre la sobrecarga que impone un número elevado de capas y la complejidad que puede provocar utilizar pocas.
- Identificar las capas y asignar tareas a cada una de ellas. La tarea de más alto nivel comprenderá la funcionalidad del sistema tal como la percibe el cliente. Las demás capas servirán de apoyo a capas superiores.
- Especificar los servicios. El principio más importante en la implementación es que las capas se encuentren estrictamente separadas, y por tanto, ningún componente deberá extenderse a más de una capa.
- Refinar la estructura iterando sobre los puntos anteriores hasta evolucionar a una arquitectura natural y estable.

- Especificar una interfaz para cada capa. Si la capa n debe ser una caja negra para la capa n+1, deberá diseñarse una interfaz que ofrezca todos los servicios de la capa n.
- Estructurar cada capa individualmente. En función de su complejidad puede ser necesario separar en componentes la capa.
- Especificar la comunicación entre capas adyacentes. El mecanismo más utilizado para la comunicación entre capas es el modelo solicitud respuesta.
- Desacoplar las capas adyacentes mediante el uso de interfaces.
- Diseñar una estrategia para el manejo de errores. Como regla general, los errores se deben tratar en la capa más baja posible. Hacia capas superiores sólo deben propagarse errores generales que tengan sentido en el nivel de abstracción.

2.3.4.2. Patrones de diseño

Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software. Estos patrones resuelven un problema de diseño general en un contexto particular (BUSCHMANN, 1996). A continuación se comentarán los patrones de diseño que se utilizarán 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.

En la figura 2.22 se puede ver la estructura de este patrón donde un cliente recibe o envía un Value Object desde o hacia un objeto de negocio, serializándose y deserializándose cuando el objeto de negocio es remoto.

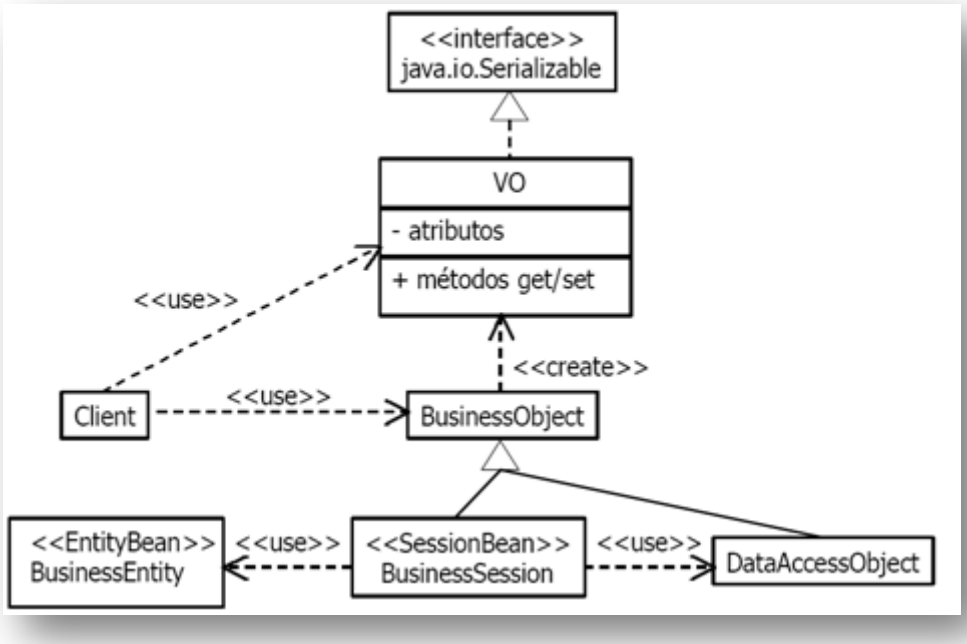

Figura 2.22. Estructura del Patrón Value Object

IoC

El patrón de diseño IoC (*Inversion of Control o* Inversión del 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[3] 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 podemos destacar:

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

Factory

Factory Method define una interfaz para crear objetos, pero deja que sean las subclases quienes decidan qué clases instanciar. Permite que una clase delegue en sus subclases la creación de objetos, de esta forma la clase derivada toma la decisión sobre qué clase instanciar y cómo instanciarla.

El patrón Factory Method permite escribir aplicaciones que son más flexibles respecto de los tipos a utilizar difiriendo la creación de las instancias en el sistema a subclases que pueden ser extendidas a medida que evoluciona el sistema. Permite

[3] Un ejemplo de contenedor de inversión es Spring.

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, que se muestra en la figura 2.23, 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.

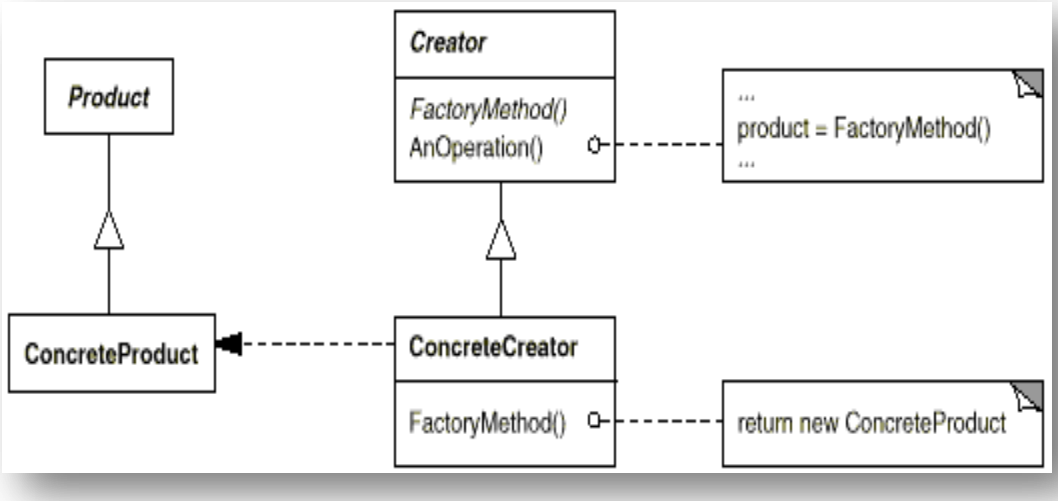

Figura 2.23. Estructura del Patrón Factory

Mediante esta estructura el Creador se basa en sus subclases para definir el Factory Method (método de fabricación) 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 (método de fabricación) 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. En la figura 2.24 se muestra el propósito del patrón Facade.

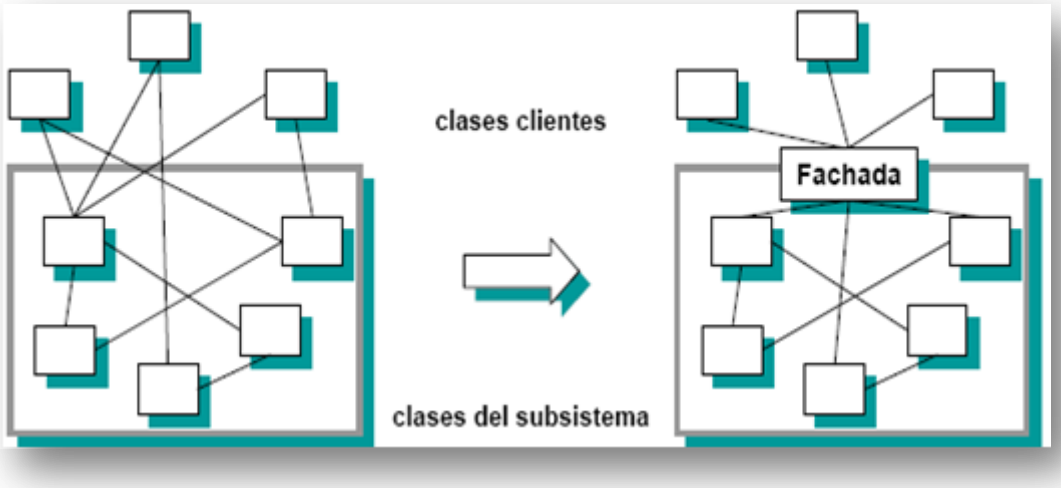

Figura 2.24. Propósito del Patrón Facade

La estructura del patrón Facade, que se muestra en la figura 2.25, está formada por los siguientes elementos:

- Facade (fachada)
  - 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).

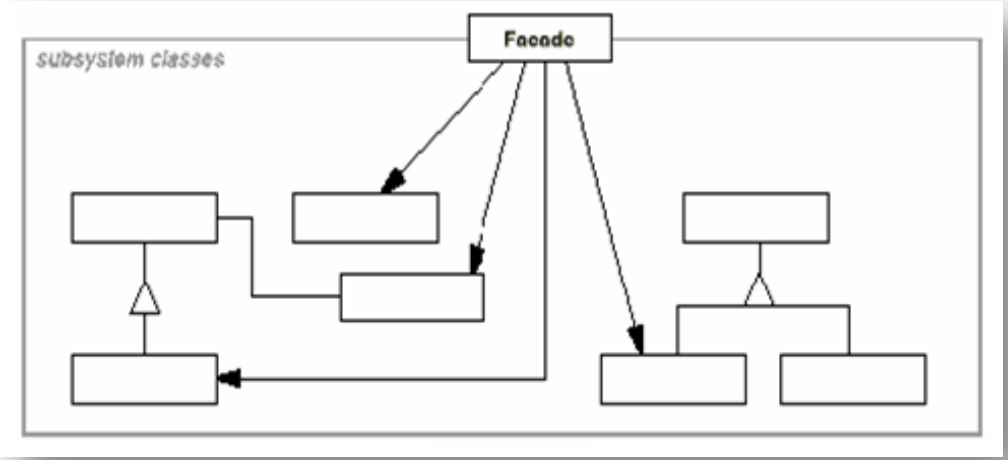

Figura 2.25. Estructura del Patrón Facade

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.

BussinesDelegate

El patrón Business Delegate (PATRONES, 2008) se utiliza para reducir el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio. Business Delegate oculta los detalles de la implementación del servicio de negocio. De esta forma es fácil manejar los cambios porque están centralizados en un sólo lugar, el Business Delegate.

Este patrón actúa como una abstracción de negocio del lado del cliente. La figura 2.26 muestra el diagrama de clases que representa al patrón Business Delegate. El cliente solicita al *BusinessDelegate* que le proporcione acceso al servicio de negocio subyacente. El *BusinessDelegate* utiliza un *LookupService* para localizar el componente *BusinessService* requerido.

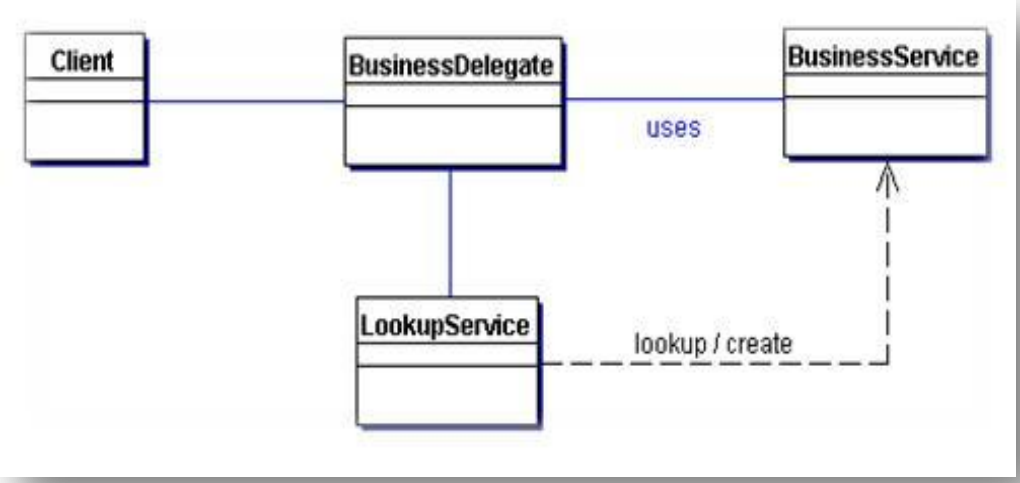

Figura 2.26. Diagrama de clases del Patrón Business Delegate

Este patrón aporta las siguientes ventajas:

- Traduce las Excepciones del Servicio de Negocio: Business Delegate es el responsable de traducir cualquier excepción de red o relacionada con la infraestructura en excepciones de negocio, aislando a los clientes del conocimiento de las especificaciones de la implementación.
- Implementa Recuperación de Fallos y Sincronización de Threads: Cuando Business Delegate encuentra un fallo en el servicio de negocio, puede imple-

mentar características de recuperación automática sin exponer el problema al cliente. Si la recuperación tiene éxito, el cliente no necesita saber nada sobre el fallo. Si el intento de recuperación no tiene éxito, entonces el Business Delegate necesita informar al cliente del fallo. Además, los métodos del Business Delegate podrían estar sincronizados, si fuera necesario.

- Impacto en el Rendimiento: Business Delegate podría proporcionar servicio de caché (y un mejor rendimiento) a la capa de presentación para las peticiones de servicios comunes.

Action

El patrón action, también conocido como command, especifica una forma simple de separar la ejecución de un acción/comando del entorno que generó dicha acción/ comando. Este patrón presenta una forma sencilla y versátil de implementar un sistema basado en acciones/comandos que permita su extensibilidad y mantenimiento. El patrón action se suele aplicar cuando se quiere:

- Facilitar la parametrización de las acciones a realizar.
- Independizar el momento de petición del de ejecución.
- Implementar CallBacks. El parámetro de una orden puede ser otra orden a ejecutar.
- Soportar o permitir el “deshacer”.
- Desarrollar sistemas utilizando órdenes de alto nivel que se construyen con operaciones sencillas (primitivas).

La estructura del patrón action se muestra en la figura 2.27. El cliente crea un objeto “ConcreteCommand” y especifica su receptor. Un objeto “invoker” almacena el objeto “ConcreteCommand”. El invocador envía una petición llamando a “Execute” sobre la orden. El objeto “ConcreteCommand”, invoca operaciones de su receptor para llevar a cabo la petición.

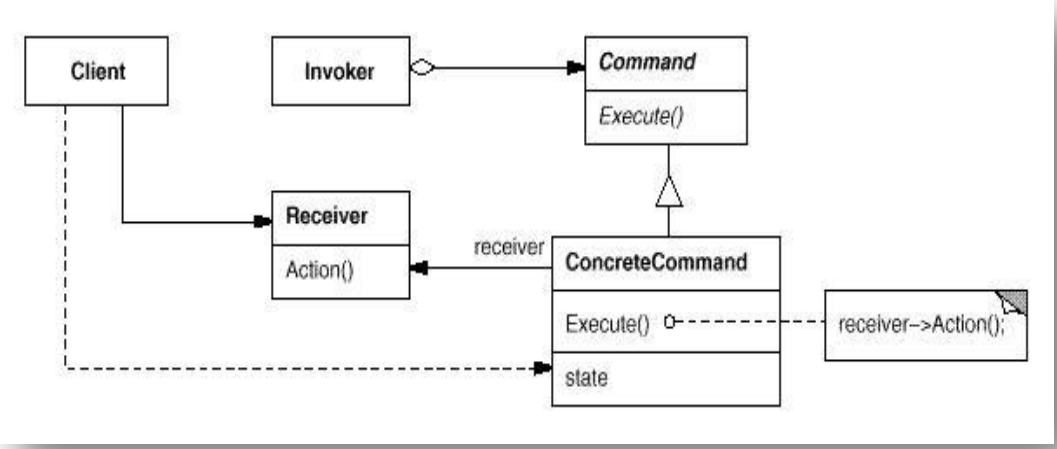

Figura 2.27. Estructura del patrón Action

Reflection

El patrón Reflection modela un mecanismo que permite cambiar la estructura y el comportamiento de un sistema de forma dinámica. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones. Posee dos niveles: Metadatos (permite al software autodescubrirse) y Lógica de aplicación. El comportamiento se define y puede variar en tiempo de ejecución, adaptabilidad, pero pueden resultar propagación de cambios inadvertidos por cambios en los metadatos.

Este patrón permite mejorar la adaptabilidad de los sistemas, los cambios en los requisitos, cambios en el marco tecnológico, etc.

2.4. Análisis de tecnologías

En esta sección se realizará el análisis de tecnologías en base a los patrones arquitectónicos utilizados, *MVC y layers (capas)*, que fueron descritos en el apartado *2.3.4 Estilos y Patrones de diseño*. En primer lugar se analizarán los distintos frameworks de desarrollo Web que proporcionan MVC y que cubrirían varias capas. Posteriormente se mostrará el análisis de tecnologías en las diferentes capas que componen el proyecto. Y finalmente se realizará una breve introducción del lenguaje

de programación elegido, Java (en el C*apítulo 3. Herramientas para la elaboración del proyecto,* se hablara más en profundidad sobre él).

2.4.1. Frameworks de desarrollo Web

El término framework o marco de trabajo se ha popularizado en los últimos años dentro del ambiente del desarrollo de software. Es común encontrar dicho término en diversas circunstancias: leyendo un libro sobre un lenguaje de programación, buscando información de interés en internet sobre una nueva tecnología Web, etc.

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. Es importante no confundir el término con una librería. La diferencia fundamental entre ambos es que una librería contiene funciones o rutinas que una aplicación puede invocar, mientras que un framework proporciona componentes genéricos y cooperantes entre sí que una aplicación puede extender para proporcionar un conjunto particular de funciones. (FRAMEWORK, 2008).

A continuación se comentan algunos framework de desarrollo web.

2.4.1.1. Struts

Figura 2.28. Struts Logo

Struts es un framework utilizado para la construcción de aplicaciones web basadas en servlet y JSP. Fue creado por Craig R. McClanahan. Struts se desarrollaba como parte del proyecto Jakarta de la *Apache Software Foundation,* pero actualmente es un proyecto independiente conocido como Apache Struts (CAVANESS, 2005).

Su carácter de software libre y su compatibilidad con todas las plataformas en que Java está disponible, lo convierte en una herramienta altamente disponible y se ha

convertido en el estándar de hecho para aplicaciones web en Java. Entre las características de Struts se puede mencionar:

- Basado en el patrón MVC2
- Tecnologías J2EE
- Soporta diferentes implementaciones de modelo (JavaBeans, EJB, etc.)
- Soporta diferentes implementaciones de presentación (JSP, XML / XSLT[4], JSF)
- Estable y maduro
- Configuración del control centralizada
- Componente Tiles para soluciones de plantillas (templates) y de layout (esquemas). Tiles puede ser utilizado para crear componentes de vista reutilizables y organizar layouts
- Interrelaciones entre acciones y página u otras acciones se especifican por tablas XML[5] en lugar de codificarlas en los programas o páginas
- Componentes de aplicación, que son el mecanismo para compartir información bidireccionalmente entre el usuario de la aplicación y las acciones del modelo
- Librerías de entidades para facilitar la mayoría de las operaciones que generalmente realizan las páginas JSP
- Struts contiene herramientas para la validación de campos de plantillas bajo varios esquemas que van desde validaciones locales en la página (en javaScript) hasta las validaciones de fondo hechas a nivel de las acciones

Actualmente se ha lanzado una nueva versión, Struts 2. Su objetivo es el mismo que el de Struts 1, brindar soporte para el desarrollo de aplicaciones Web bajo el

[4] XSLT o Transformaciones XSL es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad.

[5] eXtensible Markup Language ó Lenguaje extensible de marcas. XML es un conjunto de reglas que sirven para definir etiquetas semánticas para describir la estructura de los documentos de texto.

patrón MVC. La filosofía de trabajo se basa en tres conceptos: construir, desplegar y mantener. A continuación se detallan algunas de las novedades que aporta Struts 2 y que se comentan en (CARPI, 2008).

Diseño mejorado

Todas las clases de Struts 2 están basadas en interfaces. Las interfaces del núcleo son independientes de la arquitectura HTTP[6].

Valores por defecto

La mayoría de los elementos de configuración poseen un valor por defecto que puede ser "*setteado*" (y no hay que preocuparse por configurarlo nuevamente).

Resultados mejorados

A diferencia de los antiguos "*ActionForward*" de Struts 1, los resultados de Struts 2 son tipificados permitiendo preparar el "*response*" en los casos que sea necesario.

Tags mejorados

Con los nuevos tags (etiquetas) de Struts 2 es posible crear páginas consistentes con menor escritura de código.

Soporte para Ajax

Ajax permite a las aplicaciones interactuar con el servidor mediante requerimientos asincrónicos.

Inicio rápido

Muchos cambios pueden ser realizados dinámicamente sin ser necesario el reinicio del contenedor web.

Test de las acciones

Las acciones de Struts 2 son independientes de la arquitectura HTTP con lo cual pueden ser fácilmente testeadas mediante test de unidad sin la necesidad de recurrir a objetos que simulen el comportamiento de los objetos reales.

[6] Protocolo de Transferencia de Hipertexto (HTTP), especifica cómo se comunican el navegador y el servidor entre ellos.

Integración con Spring

Struts 2 es "Spring-aware". En otras palabras las acciones pueden ser creadas como beans de Spring.

Plugins

Las extensiones de Struts 2 pueden ser agregadas simplemente copiando un "*jar*[7]". No es necesaria una configuración manual.

Formularios

Ahora es posible usar cualquier JavaBeans contenido en una acción o escribir directamente sobre los atributos de una acción. Por otro lado, Struts 2 convierte automáticamente los valores cargados en el formulario web a los tipos de valores que posea el JavaBeans referenciado.

Acciones POJO[8]

Cualquier clase puede ser usada como una acción. Ya no es necesario implementar una interfaz o subclasificar alguna clase del framework.

2.4.1.2. J2EE

J2EE (Java Platform 2 Enterprise Edition) o Java EE (Java Platform Enterprise Edition) como se conoce actualmente y como lo llamaremos a partir de ahora, es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La especificación original J2EE fue desarrollada por Sun Microsystems.

Figura 2.29. J2EE Logo

[7] acrónimo para Java Application Resource, un jar es un archivo en el cual se distribuyen librerías (clases Java) para que puedan ser usadas por otras aplicaciones.

[8] 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.

Java EE permite al desarrollador crear una aplicación empresarial portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores (JENDROCK, 2008). Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel. Java EE se compone de contenedores y componentes.

Un contenedor es una interfaz entre un componente y la funcionalidad de bajo nivel que soporta el componente. Permiten el desarrollo concentrado en resolver el problema del negocio y no en los requerimientos no funcionales como por ejemplo manejo transaccional, multiprocesos y *pooling* de recursos. Proporciona los siguientes servicios:

- Modelo de seguridad: acceso de componentes web o EJB únicamente por parte de usuarios autorizados
- Modelo transaccional: relación entre los métodos que participan en una transacción para garantizar su atomicidad
- Servicio de *lookup* utilizando JNDI (*Java Naming and Directory Interface*): interface unificada para el nombramiento de recursos y el servicio de directorio, disponible a todos los componentes de aplicación
- Modelo de conexión remota: comunicación entre clientes y EJB, garantizando transparencia a la localización
- Servicios no configurables: ciclo de vida de los *servlets*, *pooling* de conexión a la base de datos, persistencia de datos

Existen diferentes tipos de contenedores: servidor Java EE, contenedor EJB, contenedor web, contenedor de aplicaciones cliente y contenedores de applets.

Un componente Java EE es un componente que se ensambla en una aplicación Java EE con sus correspondientes clases y archivos y que se comunica con otros componentes. Los componentes están escritos en Java y se ejecutan igual que cualquier programa escrito en Java. Existen diferentes componentes:

- Aplicaciones cliente y applets: se ejecutan en el cliente
- JSP, JSF y servlets: componentes web que se ejecutan en el servidor
- EJB: componentes de negocio que se ejecutan en el servidor

Las definiciones de Bill Shannon y Mark Hapner explican de forma sencilla la relación entre contenedores y componentes:

> *"The container is the car,*
> *The component is the driver"*
> Bill Shannon. (J2EE Lead Architect)
> JavaOne 1999.

> *"The container is the platform,*
> *The Component is your application"*
> Mark Hapner. (J2EE Platform Architect)
> JavaOne 1999.

Actualmente está disponible Java EE 5 que está definido por una *especificación* del Java Community Process bajo el JSR 244. Esta versión aporta las siguientes características:

- Su objetivo es proporcionar un poderoso conjunto de APIs a la vez que se reduce el tiempo de desarrollo, la complejidad de la aplicación y se mejora el rendimiento de las aplicaciones
- Se usan anotaciones que sustituyen a los descriptores de despliegue. Con las anotaciones, la especificación de la información se pone directamente en el código junto al elemento del programa que afecta
- Inyección de dependencias
- Nueva API de persistencia de Java (JPA)
    - Es un mapeador objeto/relacional al estilo de Hibernate
    - El desarrollador puede anotar las clases persistentes, llamadas entities (entidades), para que la implementación del API de Persistencia sepa cómo mapear las instancias a la BD

- No se accede directamente a las clases persistentes, entities, sino que se persisten a través de un interface llamado EntityManager

Por su importancia dentro de la plataforma Java EE se comentarán con más detalle los componentes EJB.

EJB

Los Enterprise JavaBeans (EJB) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales JEE. Los EJB se ejecutan en un contenedor de EJB dentro del servidor de aplicaciones. El contenedor de EJB proporciona servicios añadidos como transacciones o seguridad. Un EJB consta de tres elementos:

- Una interfaz Home, utilizada para gestionar el ciclo de vida de los EJB (contiene los métodos create, locate y remove).
- Una interfaz Remote, utilizada para definir los métodos de negocio.
- Implementación del EJB.

Los EJB proporcionan un componente del lado del servidor que encapsula la lógica de negocio de una aplicación. Existen tres tipos de EJB: EJB de entidad[9], EJB de Sesión y EJB dirigido por mensajes.

EJB de Entidad (Entity EJBs)

El objetivo de los EJB de Entidad es encapsular los objetos del lado del servidor que almacena los datos. Los EJB de entidad presentan la característica fundamental de la persistencia que puede ser gestionada de dos formas:

- Persistencia gestionada por el contenedor (CMP): el contenedor se encarga de almacenar y recuperar los datos del objeto de entidad mediante el mapeo de una tabla de la base de datos.
- Persistencia gestionada por el bean (BMP): el propio objeto entidad se encarga, mediante una base de datos u otro mecanismo, de almacenar y

[9] En la documentación de java para JEE 5 de donde se ha obtenido la información, los EJB de Entidad (*Entity EJB)* desaparecen ya que son remplazados por JPA (Java Persistence Api), aún así aquí se comentaran.

recuperar los datos a los que se refiere, por lo cual, la responsabilidad de implementar los mecanismos de persistencia es del programador.

EJB de Sesión (Session EJBs)

Los EJB de sesión gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos:

- Con estado (*stateful*). Los beans de sesión con estado son objetos distribuidos que poseen un estado. El estado no es persistente, pero el acceso al bean se limita a un solo cliente.
- Sin estado (*stateless*). Los beans de sesión sin estado son objetos distribuidos que carecen de estado asociado permitiendo por tanto que se los acceda concurrentemente. No se garantiza que los contenidos de las variables de instancia se conserven entre llamadas al método.

EJBs dirigidos por mensajes (Message-driven EJBs)

Los EJB dirigidos por mensajes son los únicos beans con funcionamiento asíncrono. Usando el *Java Messaging System* (JMS), se suscriben a un tema (*topic*) o a una cola (*queue*) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

Actualmente en Java EE 5 la versión disponible es EJB 3.0 cuyo objetivo ha sido simplificar las tareas del desarrollador de aplicaciones empresariales. Algunos cambios que incluye esta nueva versión son:

- Uso de anotaciones
- Inyección de dependencias: Podemos ahora acceder a otros EJB o recursos simplemente indicándole al contenedor que inyecte las referencias que necesitamos, sin tener que usar JNDI
- Un modelo más sencillo para la implementación de interfaces (fachadas): las interfaces local y remota son opcionales, puede

declararse la una o la otra ó ambas ya no se obliga a tener las dos, y se reduce y simplifica el código con las anotaciones

- Por compatibilidad, incluye las API del modelo anterior (EJB 2.1)

2.4.1.3. Spring

Figura 2.30. Spring Logo

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). Desde 2003 es un proyecto de un pequeño equipo de desarrolladores, publicado en Sourceforge.

Spring no intenta "reinventar la rueda" sino 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*[10] 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[11] y un único JDBC DataSource. En contraste con el JTA simple o EJB CMT[12], el soporte de transacciones de *Spring* no está atado a entornos J2EE.

- Una capa de abstracción JDBC[13] que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores, y reduce considerablemente la cantidad de código necesario.

- Integración con *Hibernate*, Java Data Object (JDO) e iBatis SQL Maps en términos de soporte a implementaciones DAO[14] y estrategias con transacciones. Especial soporte a *Hibernate* añadiendo convenientes características de *IoC*, y solucionando muchos de los comunes problemas de integración de *Hibernate*. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO.

- Funcionalidad AOP[15], totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa. Con Spring se puede tener gestión de transacciones declarativa sin EJB, incluso sin JTA, si se utiliza una única base de datos en un contenedor Web sin soporte JTA.

[10] 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.
[11] Java Transaction API o API para transacciones en Java
[12] Enterprise Java Beans Container-Managed Transactions: EJB con transaccionalidad manejada por el container
[13] Java Database Connectivity o Conectividad a la Base de Datos de Java
[14] Data Access Object o Objeto de Acceso a Datos
[15] Aspect Oriented Programming o Programación Orientada a Objetos

- 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:

- 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), Spring Security y mock testing que se comentarán a continuación.

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 "inversión", ya que invierte la acción de llamada a alguna librería externa (JOHNSON, 2007).

Inyección de dependencia

Es una forma de IoC que está basada en constructores de Java en vez de usar interfaces específicas del framework. Con este principio en lugar de que el código de la aplicación utilice el API del framework para resolver las dependencias como: parámetros de configuración y objetos colaborativos, las clases de la aplicación exponen o muestran sus dependencias a través de métodos o constructores que el framework puede llamar con el valor apropiado en tiempo de ejecución, basado en la configuración.

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 a los programadores modularizar:

- 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.

AOP se puede utilizar para: persistencia, manejo de transacciones, seguridad, logging y debugging.

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 *advice*[16] 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.

Spring implementa AOP utilizando *proxies*[17] dinámicos. Además se integra transparentemente con los BeanFactory[18] existentes.

Persistencia (ORM)

En lugar de que Spring proponga su propio modulo ORM (Object-Relational Mapping) para los usuarios que no se sienten confiados en utilizar simplemente JDBC, propone un modulo que soporta los frameworks más populares del mercado, entre ellos: Hibernate, TopLink, iBatis. Todo esto se puede utilizar en conjunto con las transacciones estándar del framework. Alguna de las ventajas que brinda Spring al combinarse con alguna herramienta ORM son:

- Manejo de sesión: Spring hace de una forma más eficiente, sencilla y segura la forma en que se manejan las sesiones de cualquier herramienta ORM que se quiera utilizar.
- Manejo de recursos: se puede manejar la localización y configuración de los SessionFactories de Hibernate o las fuente de datos de JDBC por ejemplo.

[16] Es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad.
[17] Interceptor entre la clase y el aspecto (aspect).
[18] Implementación del patrón Factory, permite instanciar objetos de forma muy rápida y fácil. Puede crear muchos tipos diferentes de beans.

- Manejo de transacciones integrado: se puede utilizar una plantilla de Spring para las diferentes transacciones ORM.
- Envolver excepciones: con esta opción se pueden envolver todas las excepciones para evitar las molestas declaraciones y los catch en cada segmento de código necesarios.
- Evitar limitarse a un solo producto: si se desea migrar o actualizar a otra versión de un ORM distinto o del mismo, Spring trata de no crear una dependencia entre la herramienta ORM, el mismo Spring y el código de la aplicación, para qué cuando sea necesario migrar a un nuevo ORM no sea necesario realizar tantos cambios.
- Facilidad de prueba: Spring trata de crear pequeños pedazos que se pueden aislar y probar por separado, ya sean sesiones o una fuente de datos (datasource).

Spring Security

Spring Security, antes conocido como Acegi Security, es un marco de seguridad que nos proporciona la posibilidad de crear una capa de seguridad declarativa en nuestras aplicaciones basadas en Spring. Spring Security va más allá de la especificación JEE5, proporcionando un marco más completo, y lo que es más importante, totalmente independiente del entorno en el que se despliega la aplicación. Como se comenta en (SPRING SECURITY, 2007) las condiciones de seguridad viajan en el artefacto a desplegar, lo que hace más fácil tanto el desarrollo como la migración o el mantenimiento de nuestro sistema. Spring Security aporta una solución completa para los dos principales requisitos de seguridad, la autentificación y la autorización, y lo hace tanto al nivel de peticiones web, como a la hora de las llamadas a métodos. Para conseguir sus misiones Spring Security hace uso de las posibilidades de dependencia inyectada de Spring, las capacidades de AOP para la seguridad en la llamada a los métodos, y el uso de los filtros para la seguridad en las peticiones web, usa los mejor de Spring y lo mejor de JEE5.

Algunos de los sistemas de identificación para los que Spring Security proporciona de serie la posibilidad de usar son: HTTP BASIC, HTTP Digest, HTTP X.509, LDAP,

Formularios Html, Base de datos, OpenID, JA-SIG Central Authentication, Computer Associates Siteminder, JAAS, Soluciones para contenedores (Jboss, tomcat, jetty), Java Open Source Single Sign On, Autentificación anónima.

Spring Security es una librería en continua evolución, en cada versión se añaden nuevas funcionalidades, y lo que parecía todavía más difícil, cada vez es más simple configurarla. La última versión de Spring Security es la 2.0.x.

Mock Testing

Las pruebas son uno de los principios clave detrás del framework Spring, es decir, la capacidad para poner a prueba cada uno de los componentes independientemente de su naturaleza. Tradicionalmente realizar las pruebas de los componentes fuera del contenedor, incluso las pruebas en los contenedores, era una tarea difícil. En este sentido, Spring es una mejora importante ya que simplifica las pruebas unitarias utilizando objetos mock y JUnit (BEGOLI, 2005).

Mediante los objetos mock se puede testear de forma aislada una clase sin preocuparse por sus dependencias. Más aún, sin preocuparse por si realmente funcionan estas dependencias. Usando objetos mock podemos asegurar un "entorno perfecto y a medida", haciendo que este entorno responda como se necesita y así si el test de la clase falla, será por un problema en esta misma clase y no en sus dependencias ya que, por hipótesis, el entorno era ideal.

2.4.2. Tecnologias para la capa de presentación

La misión de la capa de presentación, también conocida como interfaz gráfica, es aislar y facilitar al usuario su interacción con los datos del sistema y con las distintas operaciones que sobre ellos se realizan. Recogerá las peticiones del usuario, las trasladará hasta la capa de negocio, validará que los datos introducidos son correctos de acuerdo a unas reglas predefinidas y trasladará hasta el usuario los datos proporcionados por las otras capas de la aplicación. Y hará lo posible por qué esta información se muestre de una manera correcta, funcional y bonita, para que la interacción del usuario sea lo más amena posible.

2.4.2.1. JSP

Java Server Pages (JSP) proporciona una manera rápida y sencilla de crear páginas web que muestran contenido generado dinámicamente. Fue desarrollado por la compañía Sun Microsystems (JENDROCK, 2008).

Figura 2.31. JSP Logo

JSP es una nueva manera de hacer servlets. Las JSPs funcionan como aplicaciones del lado del servidor. Son conjuntos de etiquetas y scriptlets en lenguaje Java que se integran dentro del propio HTML[19] y que el servidor de aplicaciones compila para dar como resultado un servlet y de la ejecución de éste, el código HTML que recibe el navegador cliente.

Las páginas JSP separan la lógica de la página de su diseño y pantalla. En otras palabras, JSP generalmente se usa para crear la vista de la aplicación (utilizando el concepto de Modelo-Vista-Controlador), restringiendo el uso a la presentación de datos o a la captación de los datos (formularios), dejando que los servlets (controlador) sean los que interactúan con las bases de datos (modelo) y recogen la información para servírsela a las JSP's.

A día de hoy está disponible JSP 2.1, algunas de las características que incluye esta versión son (DELISLE, 2006):

- Integración entre JSP 2.1 y JSF 1.2
- Un nuevo lenguaje de expresión unificado (EL unificado), con lo cual podemos mezclar tags de JSTL con componentes de JSF
- Inyección de recursos mediante anotaciones, de forma que se puede inyectar a las páginas recursos gestionados por el contenedor en lugar de tener que realizar búsquedas JNDI

[19] *Lenguaje de Marcado de Hipertexto* (HTML), usado para definir la estructura y contenido de documentos de hipertexto.

La librería JSTL y el lenguaje de expresión unificado tienen un papel importante en las JSP por ese motivo se comentarán a continuación.

JSP Standard Tag Library (JSTL)

JSTL son un conjunto de etiquetas que engloban una cantidad de tareas que se vienen realizando en las JSP como:

- iteraciones y sentencias condicionales
- manejo de ficheros XML
- acceso a bases de datos
- formateo de cadenas e internacionalización

La idea de JSTL es crear un conjunto de etiquetas que simplifiquen aún más la tarea de programar JSPs. Actualmente está la versión JSTL 1.2 que se ajusta al lenguaje de expresión unificado para permitir la integración de etiquetas JSTL con componentes JSF.

Lenguaje de Expresión unificado

La característica más importante que incorpora JSP 2.1 es el lenguaje de expresión unificado que consiste en la unión del lenguaje de expresión ofrecido por JSP 2.0 y el lenguaje de expresión creado para las JSFs.

Mientras que las JSPs tienen un ciclo de vida solicitud/respuesta simple las JSFs disponen de un ciclo de vida multifase. Estas diferencias en los ciclos de vida de las páginas provocaban conflictos al mezclar contenido JSP con etiquetas JSF. Para resolver estos problemas se desarrolló el lenguaje de expresión unificado que permite realizar las siguientes tareas:

- Leer de forma dinámica datos almacenados en JavaBeans, diversas estructuras de datos y objetos implícitos
- Escribir de forma dinámica en JavaBeans
- Invocar métodos públicos
- Realizar operaciones aritméticas dinámicamente

2.4.2.2. JavaScript

Figura 2.32. JavaScript Logo

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. 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[20].

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.

[20] cada uno de los programas, aplicaciones o trozos de código creados con el lenguaje de programación JavaScript

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

De especial importancia son los componentes Dojo, Prototype y Scriptaculous que se comentan a continuación.

Dojo

Dojo es un conjunto libre de herramientas ("toolkit") de JavaScript que permite desarrollar aplicaciones web profesionales de forma sencilla y más rápida (DOJO, 2008). Dojo proporciona un sólido conjunto de herramientas para la manipulación del DOM[21], animaciones, Ajax, Drag&Drop, widget, etc. Se compone de tres capas:

- Dojo Core: es el resultado de años de evolución y perfeccionamiento. Construido por expertos en JavaScript, Dojo une lo que realmente se necesita en un único paquete. Proporciona una base sólida para desarrollar con confianza aplicaciones de alta calidad.
- Dijit: Incluye todos los widget (componentes reutilizables) que permiten la interacción con el usuario. Dijit viene incluido con su propio tema, "*tundra*", que trae un diseño y un esquema de color común a todos los widgets. También se puede modificar fácilmente un tema existente o incluso crear tu propio tema.
- Dojox: Actúa como una incubadora de nuevas ideas, un banco de pruebas para la experimentación de adiciones a la herramienta Dojo, así como un repositorio para las versiones más estable y maduro.

La última versión es Dojo 1.2.

[21] *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. El DOM define la manera en que objetos y elementos se relacionan entre sí en el navegador y en el documento para evitar incompatibilidades entre los distintos navegadores. El responsable del DOM es el consorcio W3C (*World Wide Web Consortium*).

Prototype

Prototype (PROTOTYPE, 2008) es un framework escrito en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y se está convirtiendo rápidamente en el código de elección para los desarrolladores de aplicaciones web de todo el mundo. Prototype es la creación de Sam Stephenson y un equipo central de desarrolladores de software libre. Algunos de los componentes básicos de Prototype son:

- Funciones $: funciones sencillas de ayuda para las tareas comunes de scripting.
- Nuevos métodos de Strings y Números: se han añadido numerosos métodos útiles al construir objetos String como, por ejemplo, *"camelize"*. También añade un método para Numerar, "*times".*
- Nuevos métodos de objeto: añade a la función objeto dos métodos: *"bind"* y *"bindAsEventListener"*. Éstos son usados para asociar una función a un objeto particular de modo que la palabra clave apunte a ese objeto.
- Manejo de formularios: Los objetos "*Form"* y "*Field"* prevén un número de funciones simples pero convenientes para trabajar con formularios y campos «input», así como el código que soporta la puesta en práctica de Ajax con Prototype.
- objeto "*Hash"* y "*Enumerable"*: "*Enumerable*", es uno de los pilares básicos del Prototype y una de las formas más sencillas de mejorar la productividad cuando se desarrollan aplicaciones JavaScript. "*Hash"* añade una serie de métodos que permiten enumerar las claves y valores, iterar sobre pares de clave/valor, fusionar dos hash juntos, etc.
- Funciones de DOM: Prototype tiene algunos objetos, *"Element"*, "*Insertion"*, *"Observer"* y "*Position",* que permiten distintas formas de manipular el DOM.

La última versión es Prototype 1.6.

Scriptaculous

Scriptaculous (ó Script.aculo.us) es una librería Javascript que permite el uso de controles AJAX, drag & drop y otros efectos visuales en una página web. Scriptaculous está basada en prototype y fue creada por Thomas Fuchs, aunque actualmente recibe contribuciones de numerosos programadores, ya que la librería es software libre (SCRIPTACULOUS, 2008). La librería está dividida en varios módulos:

- Efectos: permite añadir de forma muy sencilla efectos especiales a cualquier elemento de la página. La librería incluye una serie de efectos básicos y otros efectos complejos construidos con la combinación de esos efectos básicos. Entre los efectos prediseñados se encuentran el parpadeo, movimiento rápido, aparecer/desaparecer, aumentar/disminuir de tamaño, desplegarse, etc.
- Controles: define varios controles que se pueden añadir directamente a cualquier aplicación web. Los tres controles que forman este módulo son:
  - Arrastrar y soltar: permite definir los elementos que se pueden arrastrar y las zonas en las que se pueden soltar elementos.
  - Autocompletar: permite definir un cuadro de texto en el que los valores que se escriben se autocompletan con ayuda del servidor.
  - Editor de contenidos: permite modificar los contenidos de cualquier página web añadiendo un sencillo editor AJAX en cada elemento.
- Utilidades: la utilidad principal que incluye se llama *"builder"*, que se utiliza para crear fácilmente nodos y fragmentos complejos de DOM.

La última versión es Scriptaculous 1.8.

2.4.2.3. JSF

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 2.33. JSF Logo

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.

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 que estamos acostumbrados en 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* ni nada.

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 definitivas cuentas, 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 (BALL, 2006):

- 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

2.4.2.4. 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.

2.4.3. Tecnologías para la capa de negocio

La capa de negocio es el corazón de la aplicación. El objetivo de esta capa es que toda la lógica de negocio de la aplicación esté bien localizada. Aquí es donde debe realizarse todas las operaciones que verdaderamente dan valor a la aplicación. Esta capa recibe peticiones de la capa de presentación, procesa la lógica de negocio basada en las peticiones, y envía las respuestas tras el proceso.

Para la implementación de esta capa se barajaron spring y ejb (Componente de J2EE) que ya se han comentado anteriormente.

2.4.4. Tecnologías para la capa de infraestructura

La capa de infraestructura es adyacente a todas las demás. Esta capa comprende todos aquellos servicios susceptibles de ser requeridos desde cualquiera de las capas lógicas del sistema.

2.4.4.1. Hibernate

Figura 2.34. Hibernate Logo

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. Como se describe en (CARO, 2008) Hibernate dispone de distintos módulos que se comentan a continuación.

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.

2.4.4.2. Sesame

Sesame (SESAME, 2008) es un framework para Java de software libre con soporte para el almacenamiento y la consulta de datos RDF.

Originalmente desarrollado por Aduna, ahora es mantenido en cooperación con NLnet Foundation y un grupo de desarrolladores voluntarios.

Actualmente está disponible Sesame 2.x. Esta nueva versión es incompatible con versiones anteriores e incorpora una gran cantidad de mejoras en el API, así como en el framework en general. Algunas de las nuevas características que incluye son:

Figura 2.35. Sesame Logo

- Precisa de la versión 5 de Java o posteriores
- Da soporte a información contextual que permite seguir el rastro de unidades de datos RDF
- Permite la gestión de transacciones y “rollbacks”
- Incorpora un protocolo HTTP “RESTful” (transferencia de estado representacional) que da soporte al protocolo SPARQL y al formato XML de resultados de consultas SPARQL
- Soporta el lenguaje de consultas SPARQL

Los componentes más importantes de Sesame se comentan a continuación.

Repository (Repositorio)

El repository (repositorio) es una API que ofrece un gran número de procedimientos para el manejo de datos RDF. El objetivo principal de esta API es hacer la vida de los desarrolladores de aplicaciones tan fácil como sea posible. Ofrece diversos métodos para cargar ficheros de datos, hacer consultas, extraer y manipular los datos.

RIO

RIO significa "RDF I/O". Se trata de un conjunto de analizadores y escritores de RDF que han sido diseñados con la velocidad y el cumplimiento de los estándares como las principales preocupaciones, Actualmente apoya la lectura y la escritura de RDF/XML y N-Triples, y la escritura de N3. RIO es parte de Sesame pero también puede ser utilizado como una herramienta independiente y se puede descargar por separado (RIO, 2008).

SeRQL

SeRQL (Sesame RDF Query Languaje) es un lenguaje de consulta en RDF que es similar a SPARQL, pero con otra sintaxis. SeRQL fue desarrollado originalmente como una alternativa mejor para los lenguajes de consulta RQL y RDQL. Una gran cantidad de características de SeRQL se encuentran en SPARQL y a cambio SeRQL ha adoptado algunas de las características de SPARQL.

La última revisión de SeRQL es la revisión 3.0 que modifica el lenguaje SeRQL para ser como SPARQL, adoptando su semántica y operadores.

2.4.4.3. Jena

Figura 2.36. Jena logo

Jena (JENA FRAMEWORK, 2008) es un framework de software libre para el desarrollo de aplicaciones Java relacionadas con la web semántica. Jena fue desarrollado por *HP Labs.*

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 y es independiente del lenguaje. Actualmente está disponible Jena 2 que incluye los siguientes componentes:

- API para RDF (Resource Description Framework)
- API para OWL (Ontology Web Languaje)
- Lectura y escritura RDF en formato RDF/XML, N3 y N-Triples
- Motor de consultas SPARQL
- Almacenamiento en memoria y almacenamiento persistente

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

API para RDF

Permite crear y manipular modelos RDF desde una aplicación Java. 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. Cada sentencia tiene tres partes:
  - 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 URI[22] de un recurso
- Consultar un modelo: se podrá buscar información del modelo y realizar consultas avanzadas.
- Operaciones sobre modelos: unión, intersección y diferencia.

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 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 como 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.

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

[22] El *Identificador de Recurso Uniforme* (URI) es un sistema universal para referenciar recursos en la Web como páginas web.

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).

2.4.5. Tecnologías para la capa de persistencia

La capa de persistencia sirve como puente entre la capa de negocio y la base de datos. Esta capa es la encargada de hacer que la información almacenada en los objetos de negocio, con todos los cambios realizados en cada operación, sea guardada de una manera segura y efectiva tanto en el caso de las inserciones como de las actualizaciones. También proporciona los medios adecuados para acceder a esta información de una manera rápida y con distintos criterios de búsqueda, facilitando, de igual manera, el análisis de los datos por las distintas formas en que estos sean requeridos.

2.4.5.1. MySQL

MySQL (MYSQL, 2008) es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Fue desarrollado por MySQL AB que ha sido comprado recientemente por Sun MicroSystems.

Figura 2.37. MySql Logo

MySQL es software bajo licencia dual. Por un lado se puede usar como software libre y por otro lado si se quiere usar en una aplicación comercial permite comprar la licencia para ello. Algunas de las principales características de MySQL son:

- Velocidad y robustez
- Escrito en C y C++, probado con GCC 2.7.2.1. Usa GNU y Autoconf para potabilidad
- Clientes C, C++, JAVA, Perl, TCL
- Multiproceso, es decir puede usar varias CPU si éstas están disponibles
- Puede trabajar en distintas plataformas y sistemas operativos
- Sistema de contraseñas y privilegios muy flexible y seguro. Todas las palabras de paso viajan cifradas en la red
- Registros de longitud fija y variable
- Utilidad para chequear, optimizar y reparar tablas
- Los clientes usan TCP o UNIX Socket para conectarse al servidor
- El servidor soporta mensajes de error en distintas lenguas
- Soporta triggers, claves ajenas y vistas

Actualmente la última versión disponible es MySQL 5.1. Esta versión incluye características como:

- Particiones de índices y tablas que permiten consultas más rápidas
- Backup en línea para todos los motores de almacenamiento

- Dos nuevas formas de replicación: híbrida y basada en filas
- Restricciones a nivel de columna
- Planificación de eventos
- Funciones XML

Existe una nueva versión, MySQL 6.0, que aún se encuentra en la fase de desarrollo experimental. Esta versión aparte de dar soporte a un sistema de vistas, triggers, funciones de usuario y demás opciones incluye como gran novedad su propio motor de almacenamiento. Éste, bautizado como *Falcon*, ha sido diseñado como la nueva alternativa a los ya clásicos InnoDB y BerkeleyDB y estará enfocado a las nuevas tecnologías que compondrán los sistemas Web 2.0.

2.4.5.2. PostgreSQL

Figura 2.38. PostgreSql Logo

PostgreSQL (POSTGRESQL, 2008) es un sistema de gestión de bases de datos objeto/relacional de software libre. Es el trabajo colectivo de cientos de desarrolladores, basándose en veintiún años de desarrollo que comenzaron en la Universidad de California en Berkeley.

PostgreSQL funciona en todos los principales sistemas operativos, incluyendo Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows. Soporta la mayoría de tipos de datos de SQL92 y SQL99, incluyendo INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL y TIMESTAMP. PostgreSQL ejecuta procedimientos almacenados en más de una docena de lenguajes de programación, como Java, Perl, Python, Ruby, Tcl, C / C + +, y su propio PL / pgsql, que es similar al de Oracle PL / SQL. Algunas de las características de PostgreSQL son:

- Alta concurrencia: mediante un sistema denominado MVCC (Acceso Concurrente Multiversión) PostgreSQL permite que mientras un proceso

escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.

- Claves ajenas
- Disparadores (triggers)
- Vistas
- Integridad transaccional
- Herencia de tablas
- Tipos de datos y operaciones geométricas

A día de hoy la última versión disponible de PostgreSQL es la 8.3. Esta versión incluye una cantidad récord de características nuevas y mejoradas, que van a aumentar los beneficios para diseñadores de aplicaciones, administradores de bases de datos y usuarios (POSTGRESQL 8.3, 2008).

PostgreSQL 8.3 entrega mayor consistencia en el rendimiento que versiones anteriores, asegurando que cada usuario pueda obtener el mismo alto nivel de rendimiento para todas las transacciones, tanto en horas pico como fuera de ellas. Las mejoras de rendimiento más importantes incluyen:

- Heap Organized Tuples (HOT), que eliminan hasta un 75% de la sobrecarga de mantenimiento en tablas frecuentemente actualizadas.
- Checkpoints extendidos y autoafinamiento del escritor en segundo plano, que reducen el impacto de los checkpoints en los tiempos de respuesta.
- Opción de confirmación (commit) asincrónico de transacciones, que permite tiempos de respuesta más breves para algunas transacciones.

PostgreSQL 8.3 es la primera base de datos de software libre en implementar Recorrido Sincronizado, que reduce el uso de E/S en aplicaciones de minería de datos. Además incluye nuevas características para desarrolladores de aplicaciones, como:

- Soporte SQL/XML de acuerdo al estándar ANSI, incluyendo exportación en formato XML

- Búsqueda en texto: la herramienta avanzada de búsqueda en texto, TSearch2, ha sido incorporada en la distribución central, con mejor manejo y nuevos diccionarios e idiomas
- Soporte de autentificación GSSAPI y SSPI
- Nuevos tipos de datos: UUID, ENUM y arreglos de tipos compuestos

2.4.6. Introducción a Java

Java (FURINI, 2007) es una plataforma virtual de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y diferentes ordenadores. La plataforma Java consta de las siguientes partes:

Figura 2.39. Java Logo

- El lenguaje de programación Java
- La maquina virtual de Java (Java Virtual Machine)
- El API Java, una biblioteca estándar para el lenguaje

Los programas en Java generalmente son compilados a un lenguaje intermedio llamado bytecode, que luego son interpretados por una máquina virtual (JVM).

Existen varias Ediciones de Java, cada una de ellas diseñada para cierto ambiente en particular. Estas ediciones son:

- Java Standard Edition (Java SE)
- Java Micro Edition (Java ME)
- Java Enterprise Edition (Java EE)
- Java Card

Java Standard Edition (Java SE)

Java Platform, Standard Edition o Java SE (conocido anteriormente hasta la versión 5.0 como Plataforma Java 2, Standard Edition o J2SE) es una colección de APIs del lenguaje de programación Java utiles para muchos programas de la Plataforma Java.

Java Standard Edition es la edición que se emplea en ordenadores personales. Se le conoce también como Java Desktop (escritorio) y es la versión que se tiene que instalar para poder programar en Java aunque los programas estén destinados para alguna de las otras ediciones.

Java Enterprise Edition (Java EE)

Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4) incluye todas las clases de J2SE, además de algunas más que son útiles para programas que se ejecutan en servidores sobre workstations.

Java EE es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (JavaMail), aplicaciones Web(JSP y Servlets),etc. Esto permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

La plataforma Java EE está definida por una especificación. Similar a otras especificaciones del Java Commubity Process, Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE*; no obstante sin un estándar de ISO o ECMA.

Java Micro Edition (Java ME)

La plataforma Java 2, Micro Edition, o Java ME (anteriormente J2ME) es la edición que se emplea en dispositivos móviles. Es una versión recortada del Java SE con ciertas extensiones enfocadas a las necesidades particulares de esos tipos de dispositivos. La plataforma Java ME es una colección de APIs en Java orientadas a productos de consumo como PDAs, teléfonos móviles o electrodomésticos.

Java ME se ha convertido en una buena opción para crear juegos en teléfonos móviles debido a que se puede emular en un PC durante la fase de desarrollo y luego subirlos fácilmente al teléfono. Al utilizar tecnologías Java el desarrollo de aplicaciones o videojuegos con estas APIs resulta bastante económico de portar a otros dispositivos.

Java Card

Java Card es la versión de Java enfocada a aplicaciones que se ejecutan en tarjetas de crédito con chip. Es una versión muy recortada de Java. Una Java Card es una tarjeta capaz de ejecutar mini-aplicaciones Java. En este tipo de tarjetas el sistema operativo es una pequeña máquina virtual Java (JVM) y en ellas se pueden cargar dinámicamente aplicaciones desarrolladas específicamente para este entorno.

Capítulo 3

Herramientas para la elaboración del proyecto

El presente capítulo tiene como objetivo el análisis y elección de la infraestructura, la elección de las tecnologías más apropiadas derivadas del estudio realizado en el capitulo anterior y la elección de las herramientas que se utilizarán para el desarrollo del sistema informático de soporte al proyecto SEMSE.

El capítulo se ha dividido en tres secciones: en primer lugar la sección que abarca desde la infraestructura del servidor (instalación y elección del sistema operativo) hasta el sistema gestor de bases de datos, el servidor web, el servidor de aplicaciones y el repositorio semántico seleccionados. A continuación se tratan las herramientas que se utilizarán para el desarrollo de la aplicación, las tecnologías involucradas y el sistema de control de versiones. Por último se habla de las herramientas que darán servicios añadidos y que están fuera de las otras dos secciones.

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. 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 - disco duro)

3.1.1. Sistema Operativo

La elección del sistema operativo a veces se fundamenta en motivos económicos, otras veces en acuerdos que tienen los grupos o las empresas con distribuidores. Generalmente la elección debe hacerse acorde a la especificación de requisitos del proyecto en cuestión, obviando motivaciones económicas, sociales o filosóficas. Se eligió GNU/Linux como sistema operativo del servidor por las siguientes razones (ESCARTÍN, 2005):

- Es más 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.

- Es más rápido: Al tener una plataforma más estable, se favorece la utilización de aplicaciones de todo tipo de aplicaciones. 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.

- Es más económico: Ya que requiere menor mantenimiento. Los servidores Windows son más costosos debido a que es necesaria una frecuente atención y monitoreo 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[23], motivo por el que son distribuidas gratuitamente. No requieren supervisión constante ni pagos de mantenimiento para obtener Service Packs, que no son más que parches de seguridad para aplicaciones mal diseñadas.

Dentro del sistema operativo GNU/Linux, hay una amplia variedad de distribuciones, que no son más que colecciones de software y maneras de

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

empaquetado del mismo, pero que todas ellas corren encima de un Kernel Linux. En lo que se refiere a servidores tenemos las siguientes distribuciones más usuales:

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

La distribución elegida ha sido Ubuntu en su versión estable a fecha de instalación de este sistema operativo. Ubuntu (UBUNTU, 2008) está basado en Debian, pero el planteamiento está inspirado en los principios de la corriente ubuntu, un movimiento humanista encabezado por el obispo Desmond Tutu, premio Nobel de la Paz en 1984. Económicamente el proyecto se sostiene con aportaciones de la empresa Canonical Ltd. del millonario sudafricano Mark Shuttleworth y está mantenido por una incipiente comunidad que no para de crecer.

Figura 3.1. Ubuntu Logo

El proyecto nació por iniciativa de algunos programadores de los proyectos Debian, Gnome y Arch que se encontraban decepcionados con la manera de operar del proyecto Debian. La versión estable era utilizada solo por una minoría debido a la poca o nula vigencia que poseía en términos de la tecnología Linux actual. Tras varios meses de trabajo y un breve periodo de pruebas, la primera versión de Ubuntu fue lanzada en octubre del 2004.

La distribución Ubuntu, totalmente 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. A diferencia de la mayoría de las distribuciones, que vienen con una enorme cantidad de software que puede o no ser de utilidad, la lista de paquetes de Ubuntu se ha reducido para incluir solo aplicaciones importantes y de alta calidad.

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, 2008) es un programa que implementa el protocolo HTTP (hypertext transfer protocol). Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas web o páginas HTML (hypertext markup language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de sonidos.

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

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

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

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

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

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

3.1.3. Servidor de aplicaciones

Para poder utilizar tanto Servlets como JSP en nuestro servidor web, debemos, por norma general, complementar éste con un servidor de Servlets/JSP (comúnmente denominado contenedor de Servlets). Como se ha indicado anteriormente se ha elegido el servidor web Tomcat (TOMCAT, 2008) incluido en NetBeans.

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat es un servidor web con soporte de servlets y JSPs. Es mantenido y desarrollado por una comunidad de programadores voluntarios y miembros de la Apache Software Foundation. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software Licence*[24].

Figura 3.2. Tomcat Logo

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual de java.

En el sistema a implementar, Tomcat estará en la versión 6, cuyas características fundamentales son:

- Implementado a partir de las especificaciones Servlets 2.5 y JSP 2.1.
- Soporte para *Unified Expression Language* 2.1.
- Diseñado para funcionar en Java SE 5.0 y posteriores.
- Soporte para Comet[25] a través de la interfaz CometProcessor.

[24] La licencia Apache requiere la conservación del aviso de copyright y permite el uso y distribución del código fuente para software libre y software propietario.

3.1.3.1. Servlets

Un Servlet (CRAWFORD, 1998) es un programa java que se carga dinámicamente en el servidor de aplicaciones para ofrecer funcionalidades que el servidor web no puede dar por sí mismo.

Los servlets toman el lugar que llevan ocupando durante muchos años los scripts CGI[26]. Los servlets se ejecutan en la máquina virtual de Java del servidor, con lo que podemos crear aplicaciones web seguras y portables. Por supuesto, al operar a nivel del servidor, nuestros clientes no tendrán que instalar nada en sus ordenadores, solo necesitarán un navegador web.

Los Servlets se ejecutan en hebras separadas en el servidor, con lo que conseguimos escalabilidad y eficiencia. Al ejecutarse dentro del servidor web, puede interactuar completamente con el servidor (siempre dentro del contexto de la máquina virtual) y podemos acceder a recursos como bases de datos, otros servlets que esten en el servidor. . .

Los servlets ofrecen las siguientes ventajas frente a los tradicionales CGI's:

- Portabilidad. Dado que están escritos en java y conforme a las API's desarrolladas y ampliamente aceptadas, los servlets pueden funcionar en diversos sistemas operativos. Puedes desarrollar tus servlets en windows con un servidor de aplicaciones y despues desplegarlos en un sistema Linux con otro servidor de aplicaciones distinto que funcionará sin realizar cambio alguno. *Escribir una vez y ejecutar en cualquier sitio.*

---

[25] Es una técnica de programación Web muy similar a AJAX, que utiliza XMLHttpRequest, se utiliza para la entrega de datos entre cliente servidor a través del protocolo HTTP, y la entrega de datos se hace sin que el cliente lo haya solicitado. Comet también es conocido como *server push*, *HTTP push*, *HTTP streaming*, *Pushlets*, *Reverse Ajax*, y otros.

[26] Common Gateway Interface (en castellano Interfaz Común de Pasarela, abreviado CGI). CGI es un estándar que permite a los servidores Web interactuar con aplicaciones externas de forma que las páginas no tengan que ser estáticas. Sin embargo, aunque las aplicaciones CGI eran muy buenas en sus cometidos, tenian serias limitaciones. El mayor de los problemas es que por cada petición del navegador se crea un nuevo proceso que consume muchos recursos. Una vez ejecutado el script CGI el proceso ha de ser finalizado por el SO. Este constante arrancar y parar era altamente ineficiente.

- Potencia. Los servlets pueden aprovechar el potencial de las API's de Java, entre ellas las de redes (networking), acceso a URLs, multihebra, manipulación de imágenes, compresión de datos, conexión a bases de datos, internacionalización, invocación de métodos remotos (RMI), CORBA[27]. . .

- Eficiencia y estabilidad. La invocación de servlets es muy eficiente. Una vez que un servlet ha sido cargado, éste permanece en la memoria del servidor como una instancia de un objeto. Si después se utiliza ese servlet de nuevo, ya está cargado en la memoria y responderá a las peticiones que se soliciten inmediatamente. Se puede aprovechar esta característica para mantener conexiones a bases de datos, agilizando este trámite.

- Seguridad. Los servlets requieren de buenas prácticas de programación y dado que están escritos en java heredan la seguridad del lenguaje Java. Además el API de los servlets se ha diseñado para que sea segura y no un agujero de seguridad en potencia. Hace uso del *garbaje collector* o recolector de memoria, con lo que se evitan problemas con punteros o desbordamientos de buffer; También tienen un control de excepciones cerrado que no permite la salida fuera del contexto de la aplicación ni compromete la disponibilidad del servidor.

- Elegancia. El código de los servlets es limpio, orientado a objetos y simple. Una razón para esta simplicidad es la API de los Servlets en sí, que incluye métodos y clases para manejar muchas de las rutinas y operaciones que se requieren de los servlets, como las cookies, el control de las sesiones. . .

- Integración. Se integran completamente con el servidor en el que residen. Se puede cooperar con el servidor en formas que los CGI's no pueden. Por ejemplo, se pueden escribir trazas, chequear autorizaciones, efectuar mapeos MIME[28], añadir usuarios a los servidores de bases de datos. . .

[27] Common Object Request Broker Architecture - arquitectura común de intermediarios en peticiones a objetos

[28] Multipurpose Internet Mail Extensions, Extensiones de Correo Internet Multipropósito

- Extensibilidad y Flexibilidad. La API de los servlets ha sido diseñada para ser extensible. A día de hoy, la API incluye clases optimizadas para servidores HTTP, pero es posible que hagan otras optimizaciones para otros protocolos como FTP, NTP. . . La flexibilidad viene dada por la manera en la que se interactúa con el código HTML. Puede incrustarse código de Servlet dentro del código HTML, usar los servlets como aplicaciones intermediarias, o una sintaxis curiosamente similar a las ASP (Active Server Pages) de Microsoft.

3.1.4. Sistema Gestor de Base de Datos

Una Base de Datos 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.

Como Sistema Gestor de Base de Datos se eligió PostreSQL, un servidor de Base de Datos Relacional Orientado a Objetos de software libre, publicado bajo la licencia BSD[29]:

- Servidor: en el sentido de que es una implementación cliente/servidor que consta de un servidor y diferentes clientes (programas/librerías). Podemos agregar, acceder, y procesar datos grabados en una base de datos.
- Sistema Gestor de Bases de Datos Relacional Orientado a Objetos (ORDBMS): reúne las ventajas de los dos modelos. En su faceta relacional es capaz de soportar aplicaciones para grandes transacciones de datos y puede usar lenguajes de interrogación estándar como el SQL. Por otra parte, como sistema orientado a objetos es capaz de gestionar relaciones muy complejas de los componentes y al mismo tiempo adaptarse más fácilmente a nuevos tipos y formatos de datos. También proporcionan los medios para ampliar las posibilidades de una base de datos a funciones y tipos de datos definidos por los usuarios.

[29] *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.

3.1.4.1. PostgreSQL vs otras Bases de Datos

PostgreSQL posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta (en algunos benchmarks se dice que ha llegado a soportar el triple de carga de lo que soporta MySQL).

Además implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en los que MySQL no podría.

También tiene la capacidad de comprobar la integridad referencial, así como la de almacenar procedimientos en la propia base de datos, equiparándolo con los gestores de bases de datos de alto nivel, como puede ser Oracle.

Por estos motivos, PostgreSQL se puede considerar a la altura de los principales sistemas gestores de bases de datos comerciales y presentarse como un importante competidor entre las tecnologías disponibles para el diseño de sistemas.

3.1.5. Repositorio semántico

Un repositorio semántico es un software de soporte o middleware[30] que permite el almacenamiento y gestión de ontologías. La posibilidad de gestión de la persistencia de ontologías es lo que ha llevado a denominar a estos sistemas como Repositorios Semánticos.

Se ha elegido Jena como repositorio semántico. La elección de Jena se ha basado en la compatibilidad con las últimas versiones de Java, en la facilidad de uso, en el

[30] Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

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 8.3. 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.

3.2. Entorno de trabajo

Haciendo uso de la propiedad de Java de poder programar en un sitio y ejecutarlo en cualquier otro sin preocuparte por nada más, se utilizará un entorno de trabajo bajo el sistema operativo GNU/Linux. Por supuesto, todas las herramientas que se utilizarán son software libre.

Se trabajará en varios ordenadores haciendo uso de una herramienta de control de versiones en alza, de la que hablaremos más adelante. El despliegue de la aplicación en producción se limitará a utilizar la herramienta de despliegue de que dispone Tomcat en el servidor proporcionado para el proyecto.

3.2.1. Java 6

Java, por su estabilidad y características, será el lenguaje de programación seleccionado para el sistema. A fecha de realización de este estudio se contaba con la versión 6 de Java. A continuación se enumeran las características más interesantes de este lenguaje (NAUGHTON, 1997).

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 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 se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

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 el servidor.

Arquitectura neutral

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, 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 recompilación. 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.

Por tanto, lo que es verdaderamente dependiente del sistema es la Máquina Virtual Java (JVM) y las librerías fundamentales, que también nos permitirían acceder directamente al hardware de la máquina.

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.

Interpretado

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. Las clases se van cargando conforme van siendo necesitadas. 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. Pueden sustituirse las clases por prototipos, sin que eso implique una recompilación de las clases cuando se introduzcan las definitivas. 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, pues:

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

Seguro

El lenguaje Java ha sido desarrollado para ejecutar aplicaciones distribuidas por redes, por lo que la seguridad es uno de los puntos más importantes para evitar la carga y ejecución de programas que puedan poner en peligro nuestro sistema.

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

distinto del servidor en el cuál el applet está publicado o el hecho de que un applet es ciego a la organización de memoria de la máquina local.

Dinámico

La naturaleza portable e interpretada de Java proporcionan un sistema dinámico y dinámicamente extensible. Las clases son enlazadas en el momento en que son requeridas, no antes de la ejecución del programa y pueden ser cargadas de la red. Todo el código nuevo es verificado antes de ser ejecutado. La carga dinámica permite evitar tener que recompilar todos los programas que contengan clases hijas cuando se modifica una clase padre. También permite actualizar un programa simplemente cambiando algunas de sus clases, sin tener que recompilarlo al completo, siempre que las nuevas clases mantengan la misma interfaz.

Soporte para Multithread

Al ser multihilado (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. Capacidades de Java

Las características antes comentadas de Java, le hacen ser un lenguaje potente y versátil, que permite al programador desarrollar una gran diversidad de software:

- **Applets:** programas elementales incluidos en páginas HTML a través de una

etiqueta especial y que se despliegan en el browser (navegador) tras cargarse la página.

- **Aplicaciones:** programas escritos en Java y que se ejecutan de forma independiente de los navegadores. Esto se realiza llamando a los intérpretes de Java con el programa como opción.
- **Métodos nativos:** Métodos que se declaran en una determinada clase Java pero son implementados en C.
- **Manipuladores de contenido:** Programas que se cargan en el navegador y que a través de la red acceden a un fichero interpretando su contenido de una determinada forma.
- **Programación de Servidores:** Java permite crear programas multihebra que den servicio a una serie de clientes, soportando así aplicaciones sobre arquitecturas de tipo cliente-servidor.
- **Soporte para acceso a bases de datos JDBC:** Java dispone de un interfaz propio para conectarse a bases de datos que es independiente del SGBD en sí mismo, ya que se comunica con este a través de drivers.

Este último punto es muy importante porque aísla del sistema gestor de bases de datos (SGBD) que se utilice en el servidor. Con esto se consigue que, siempre que la estructura de la base de datos se respete, se pueda cambiar de servidor a uno que satisfaga las necesidades que surjan. Seguidamente se explicara un poco más en profundidad JDBC.

JDBC

JDBC (Java Database Connectivity) es la API estándar de acceso a Bases de Datos con Java, y se incluye con el Kit de Desarrollo de Java (JDK) a partir de la versión 1.1. Sun optó por crear una nueva API, en lugar de utilizar APIs ya existentes, como ODBC[31], con la intención de obviar los problemas que presenta el uso desde Java de estas APIs, que suelen ser de muy bajo nivel y utilizar características no soportadas directamente

[31] Open DataBase Connectivity

por Java, como punteros, etc. Aunque el nivel de abstracción al que trabaja JDBC es alto en comparación, por ejemplo, con ODBC, la intención de Sun es que sea la base de partida para crear librerías de más alto nivel, en las que incluso el hecho de que se use SQL para acceder a la información sea invisible.

Para trabajar con JDBC es necesario tener controladores (drivers) que permitan acceder a las distintas Bases de Datos: cada vez hay más controladores nativos JDBC. Sin embargo, ODBC es hoy en día la API más popular para acceso a Bases de Datos: Sun admite este hecho, por lo que, en colaboración con Intersolv (uno de principales proveedores de drivers ODBC) ha diseñado un puente que permite utilizar la API de JDBC en combinación con controladores ODBC.

Un último detalle: algunos fabricantes, como Microsoft, ofrecen sus propias APIs, en lugar de JDBC, como RDO[32], ADO[33], etc. Aunque estas APIs pueden ser muy eficientes, y perfectamente utilizables con Java, su uso requiere tener muy claras las consecuencias, sobre todo la pérdida de portabilidad, factor decisivo en la mayor parte de los casos para escoger Java en lugar de otro lenguaje.

Así pues, acceder a bases de datos con java se limita a crear una instancia del driver que se necesita (en el caso de este proyecto PostgreSQL), crear una conexión al servidor de bases de datos, identificarse con el login y palabra de paso, preparar una consulta y recorrer el resultado utilizando las clases que JDBC proporciona para ello.

JDBC proporciona también un control de excepciones SQL (como *warnings,* comprobaciones de integridad, truncado de datos) que facilitan enormemente el control de errores con la base de datos.

3.2.1.2. Novedades de Java 6

Los cambios más importantes introducidos en Java SE 6 son:

- Incluye un nuevo marco de trabajo y APIs que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript.

[32] Remote Data Objects, anticuada
[33] ACtiveX Data Objects, más usada actualmente

- Incluye el motor Rhino, de Mozilla, una implementación de Javascript en Java.
- Incluye un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, como JAX-WS 2.0, JAXB 2.0, STAX y JAXP.
- Mejoras en la interfaz gráfica y en el rendimiento.

3.2.2. NetBeans

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

Figura 3.3. Netbeans Logo

Netbeans (NETBEANS, 2008) es un IDE[34], desarrollado en Java, que surgió como un proyecto de estudiantes de la república Checa, y que se llamaba originalmente "Xelfi" por el año 1996. 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. Tras unas negociaciones, la empresa había nacido.

[34] Del inglés Integrated Development Environment, entorno integrado de desarrollo

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[35]. 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. El éxito estaba garantizado.

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 barriendo a sus directos competidores como Eclipse.

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

3.2.2.1. La Plataforma

En todo el proceso de gestación y de desarrollo de esta herramienta han pasado cosas interesantes: los desarrolladores (usuarios) empezaron a desarrollar aplicaciones utilizando la herramienta Netbeans y plugins[36] propios desarrollados por ellos mismos. El hecho de la existencia de estos plugins fue lo que le ha dado esta cuota de mercado.

Aprovechando esta situación, se trabajó para ofrecer una infraestructura modular de la aplicación, convirtiéndose así en una herramienta de escritorio utilizable para más de un propósito. Así, actualmente soporta otras tecnologías como C/C++, Modelado de Datos UML, Acceso a bases de datos, despliegue de aplicaciones web, entorno de desarrollo de aplicaciones para móviles. . .

3.2.2.2. NetBeans hoy en día

Como todo proyecto open source, Netbeans no es un producto sino un proceso, como un ente que tiene entidad propia o vida. Se tomó su tiempo para encontrar la línea por la que seguir y acertar con las decisiones que se tomaban, pero se consiguió, y gracias a su cualidad de ser software libre, siempre hay desarrolladores interesados en colaborar en su desarrollo y hacer sus colaboraciones únicas en él.

Gracias a estos desarrolladores, se ha conseguido un producto de calidad, ampliamente utilizado, y con una comunidad de colaboradores y de usuarios que están haciendo que Netbeans crezca a una velocidad y calidad que nunca podrían haber imaginado.

3.2.2.3. NetBeans 6.5

Para la realización de este proyecto se utilizará NetBeans 6.5 RC2[37]. NetBeans funciona en las principales arquitecturas y sistemas operativos (Windows, Linux, Solaris, MacOS), es sencillo de instalar, y proporciona a los desarrolladores todas las

[36] Aplicaciones que no son herramientas de desarrollo en sí, sino que complementan y dan más funcionalidad a lo ya existente.

[37] Release candidate 2: La segunda versión candidata, si no se encuentra ningún problema serio esta será la última versión antes de la versión definitiva.

herramientas necesarias para crear aplicaciones orientadas a escritorios, empresariales, web y móviles (NETBEANS 6.5, 2008).

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

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, aunque la versión final de ese lenguaje todavía no esté disponible y sería anunciada en diciembre.

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, 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.

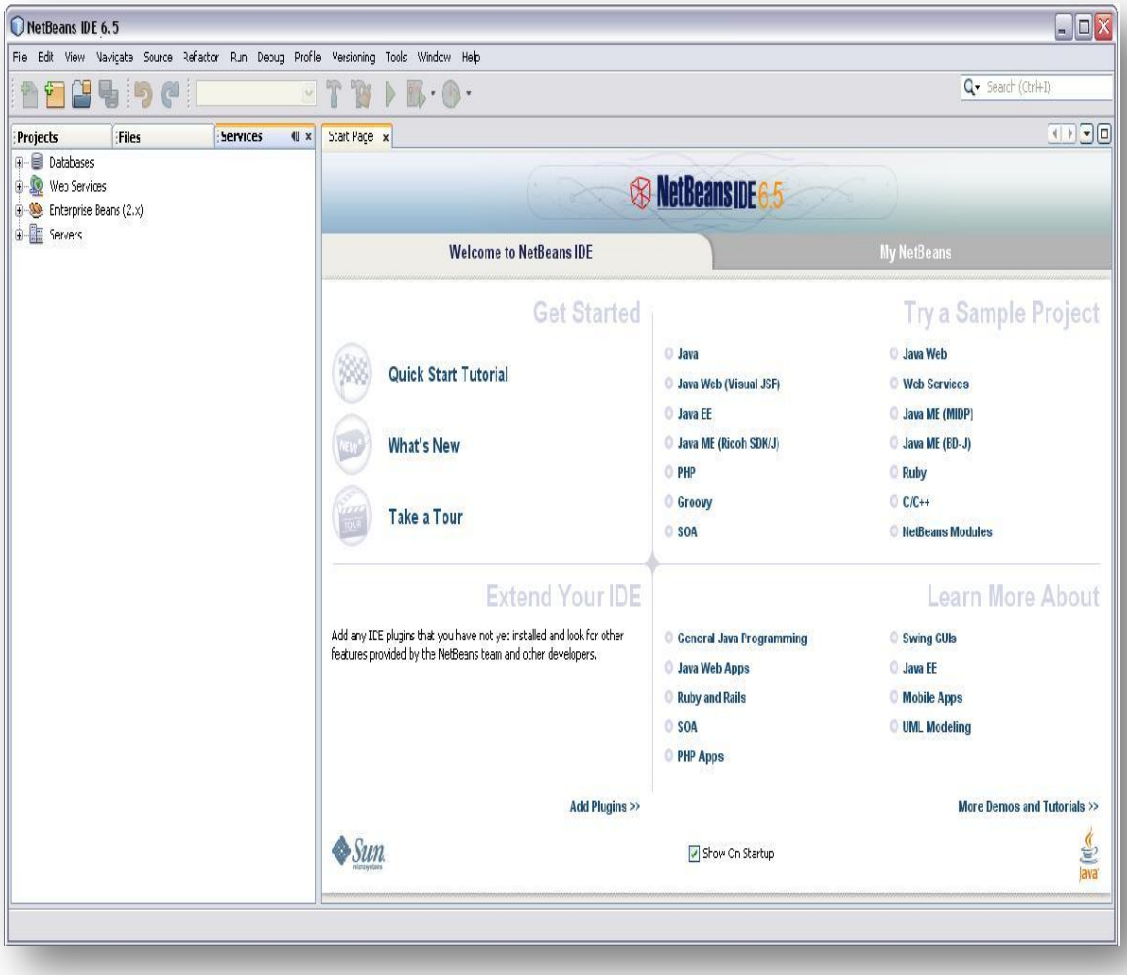

Figura 3.4. NetBeans 6.5

3.2.3. Frameworks utilizados

En este apartado se comentarán los frameworks seleccionados y su función en el desarrollo de la aplicación.

3.2.3.1. JSF

JSF es un framework que implementa el patrón de arquitectura MVC (Modelo-Vista-Controlador) en Java. El patrón de arquitectura MVC define la arquitectura del software separando los datos de una aplicación (modelo), la interfaz (vista) y la lógica de control (controlador) en tres componentes distintos.

El comportamiento básico de este tipo de aplicaciones desarrolladas bajo MVC es el siguiente. El navegador genera una petición provocada, generalmente, por el

usuario (por ejemplo, al hacer clic en un botón o al enviar un formulario) que es atendida por el controlador (Servlet). El controlador se encarga de invocar la acción (action) correspondiente. El action instanciará y/o utilizará los objetos de negocio para realizar la tarea correspondiente. JSF sigue el paradigma JavaBean: si es necesario acceder a un dato, es necesario proveer a la clase de los métodos get y set correspondientes. Finalmente generará un resultado, según el cual el controlador, nuevamente, derivará la generación de una o varias vistas (JSF), las cuales podrán acceder a los objetos disponibles en ámbito de request.

La ventaja de este framework es la división clara de lo que es la gestión del workflow, la lógica de negocio y la generación de la interfaz.

El modelo comprende todos los Objetos de Negocio donde se implementa la lógica de negocio (el "cómo lo hacemos") y donde se debe soportar todos los requisitos funcionales del Sistema sin mezclarlo con partes correspondientes al workflow (el "qué hacemos") que corresponden al controlador.

El controlador comprende la funcionalidad involucrada desde que un usuario genera un estímulo (click en un link, envío de un formulario, etc.) hasta que se genera la interfaz de respuesta. Entre medias, llamará a los objetos de negocio del modelo para que resuelvan funcionalidad propia de la lógica de negocio y según el resultado de la misma ejecutará la JSF que deba generar la interfaz resultante.

Para la implementación de la aplicación se utilizara visual JSF que aporta todas las ventajas de JSF y además permite el desarrollo de la aplicación de forma más sencilla al poder crear las páginas con componentes en la modalidad drag & drop.

3.2.3.2. Spring

Spring es un framework de código abierto para el desarrollo de aplicaciones en Java que implementa el concepto de inversión de control (IoC) mediante la 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 (al menos en el caso de Spring) y así se evita tener que andar extendiendo clases o tumbar 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.

4. Cada property o constructor argument debe poder ser convertido de cualquier formato especificado al tipo actual de ese property o constructor argument. Por defecto, Spring puede convertir cualquier valor declarado como tipo String a cualquier tipo básico (int, long, boolean…).

Ejemplo:

applicationContext.xml

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

ontologyFacade.java

```
package semse.model.service.ontologyfacade.plain;
public class OntologyFacade implements OntologyFacadeDelegate {
  private DataSource dataSource;
  private String DBType;
  public OntologyFacade() throws InternalErrorException {
  }
  public OntModel createOntology(OntologyVO ontologyVO) throws InternalErrorException {
    try {
      IDBConnection connection = new DBConnection(getDataSource().getConnection(),
     getDBType());
      CreateOntologyAction action = new CreateOntologyAction(connection, ontologyVO);
      return action.execute();
    } catch (SQLException ex) {
      throw new InternalErrorException(ex);
    }
  }
)
```

Como se puede observar, 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.

3.2.4. Subversion

Para el desarrollo de este proyecto se utilizará un sistema de control de versiones, Subversión, que facilita el desarrollo colaborativo de aplicaciones. De esta forma el desarrollo se podrá llevar a cabo desde distintas ubicaciones y además facilitará poder volver atrás en el desarrollo cuando sea necesario.

Subversion (o SVN) es un sistema de control de versiones relativamente nuevo diseñado para ser el sucesor de CVS[38]. Los diseñadores se marcaron el objetivo de mejorar CVS de dos modos: creando un sistema de código fuente abierto con un diseño y apariencia similares a CVS, e intentando corregir los defectos más notables de CVS. A pesar de que el resultado no representa necesariamente la siguiente gran evolución en diseño de los sistemas de control de versiones, Subversion es potente, fácil de usar y flexible.

3.2.4.1. ¿Qué es Subversion?

Subversion (COLLINS-SUSSMAN, 2008) es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque 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. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo”.

[38] El Concurrent Versions System (CVS), también conocido como Concurrent Version System o Concurrent Versioning System, es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren. CVS se ha hecho popular en el mundo del software libre. Sus desarrolladores difunden el sistema bajo la licencia GPL

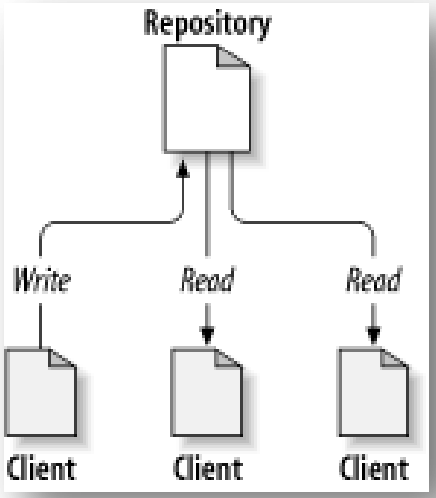

Figura 3.5. Estructura de Subversion

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

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. Sin embargo, Subversion no es uno de estos sistemas. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para un desarrollador, esos ficheros pueden ser código fuente, para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá.

3.2.4.2. Características de Subversion

Al discutir acerca de las características que Subversion aporta al mundo del control de versiones, a menudo es útil hablar de ellas en términos de cómo han mejorado sobre el diseño de CVS. Vamos a ver un listado de esas mejoras:

- **Versionado de directorios:** CVS solamente lleva el historial de ficheros individuales, pero Subversion implementa un sistema de ficheros versionado "virtual" que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.

- **Verdadero historial de versiones:** Dado que CVS está limitado al versionado de ficheros, operaciones como copiar y renombrar, las cuales pueden ocurrir sobre ficheros, pero que realmente son cambios al contenido del directorio en el que se encuentran, no son soportadas por CVS. Adicionalmente, en CVS no puede reemplazar un fichero versionado con algo nuevo que lleve el mismo nombre sin que el nuevo elemento herede el historial del fichero antiguo, que quizás sea completamente distinto al anterior. Con Subversion, se puede añadir, borrar, copiar, y renombrar ficheros y directorios. Cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo.

- **Envíos atómicos:** Una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.

- **Versionado de metadatos:** Cada fichero y directorio tiene un conjunto de propiedades, claves y sus valores, asociado a él. Se puede crear y almacenar cualquier par arbitrario de clave/valor que desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

- **Elección de las capas de red:** Subversion tiene una noción abstracta del acceso al repositorio, facilitando a las personas implementar nuevos mecanismos de red. Subversion puede conectarse al servidor HTTP Apache como un módulo de extensión. Ésto proporciona a Subversion una gran ventaja en estabilidad e interoperabilidad, y acceso instantáneo a las características existentes que ofrece este servidor-autenticación, autorización, compresión de la conexión, etc. También tiene disponible un servidor de Subversion independiente, y más ligero. Este servidor habla un protocolo propio, el cual puede ser encaminado fácilmente a través de un túnel SSH (Secure SHell).

- **Manipulación consistente de datos:** Subversion expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (legibles para computadoras). Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red. Ramificación y etiquetado eficientes. El coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Subversion crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante.

- **Hackability:** Subversion no tiene un equipaje histórico; está implementado como una colección de bibliotecas compartidas en C con APIs bien definidas. Esto hace a Subversion extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes.

A día de hoy la última versión de Subversion es la 1.5. Para la implementación del proyecto se utilizará el soporte a SVN integrado en NetBeans que permite trabajar con CVS o Subversion (en este caso) automáticamente.

3.3. Otras herramientas

3.3.1. Tomcat Manager y Tomcat Admin

Como se comenta en (RODRIGUEZ, 2007) el servidor de aplicaciones Tomcat incorpora dos herramientas para su administración y para la gestión de las aplicaciones que despliega.

En primer lugar está el **Gestor de Aplicaciones Tomcat** o como se le conoce por su nombre inglés, "*Tomcat Manager*". Permite desplegar aplicaciones, eliminar aplicaciones, conocer el estado de la memoria y las sesiones que están ejecutándose en cada una de las aplicaciones.

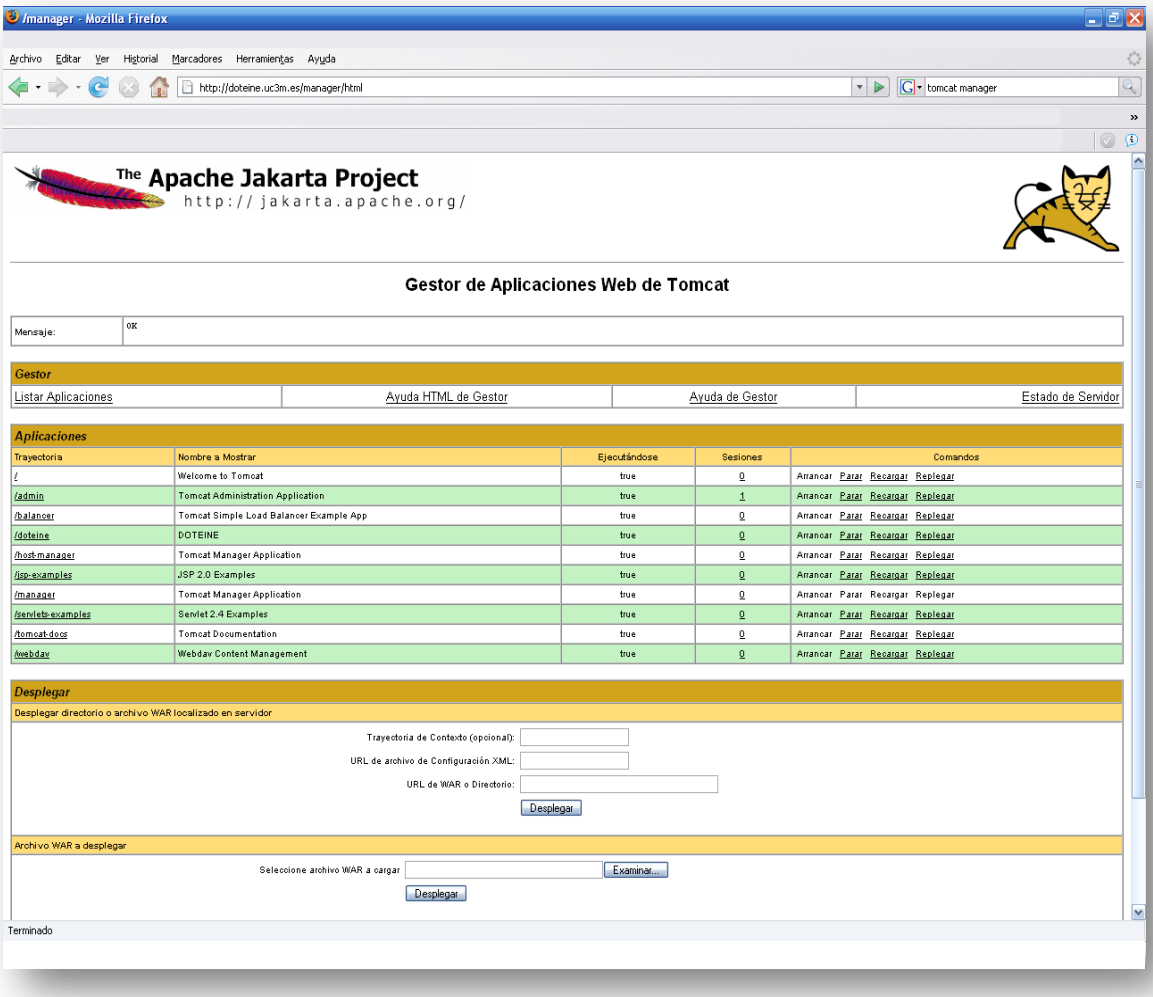

Figura 3.6. Gestor de Aplicaciones Tomcat

Y en segundo lugar, como herramienta más avanzada, está la **Administración del Servidor Tomcat** o en su nombre inglés: "*Tomcat Web Server Administration Tool*". Esta herramienta permite modificar la configuración de conectores, nombres JNDI[39], aplicaciones, usuarios. . .

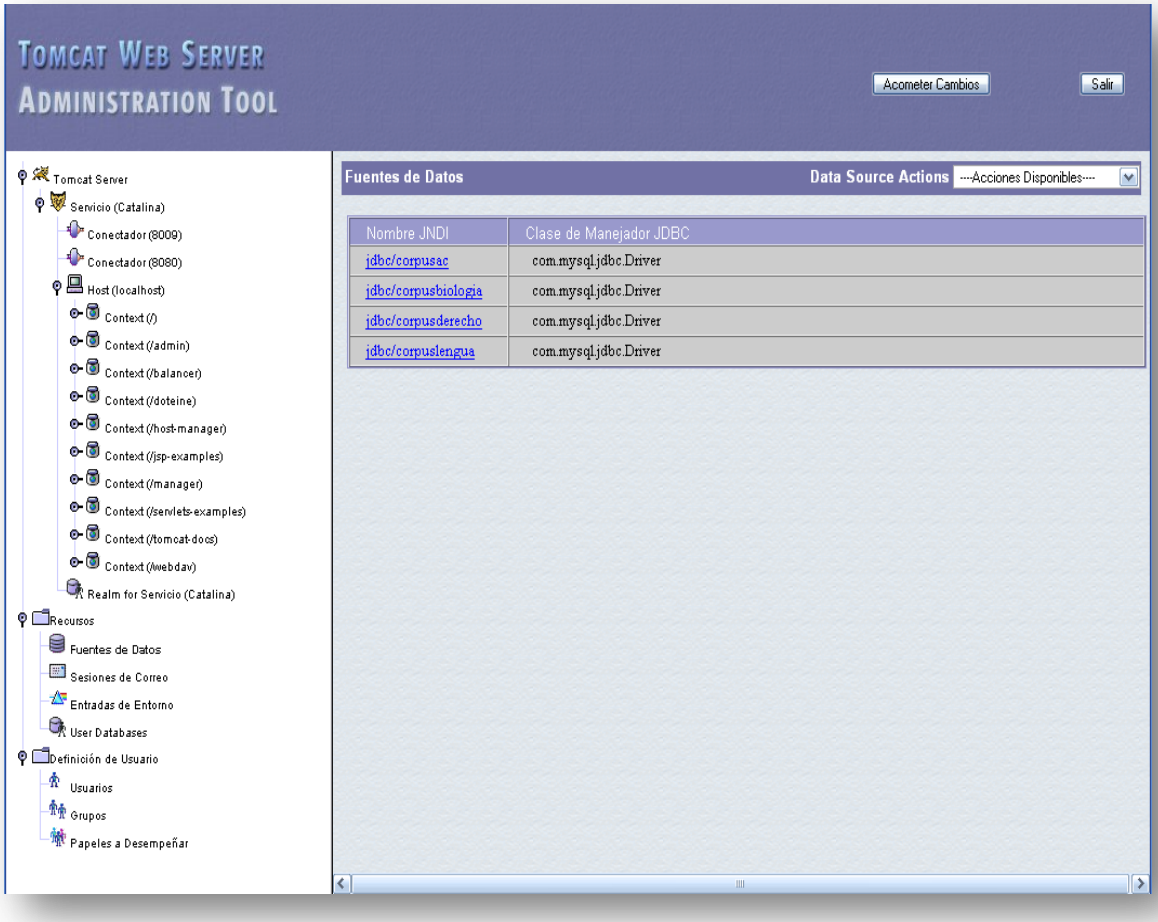

Figura 3.7. Administración del servidor Tomcat

3.3.2. PgAdmin III

Para poder administrar bases de datos a veces es necesario disponer de un programa que dé esta funcionalidad de una manera sencilla para usuarios poco expertos en el tema, es el caso de pgAdmin III.

[39] Interfaz de Nombrado y Directorio Java (Java Naming and Directory Interface)

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 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, etc. La conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets (en plataformas *nix), y puede encriptarse mediante SSL para mayor seguridad.

PgAdmin III está desarrollado por una comunidad de expertos en PostgreSQL de todo el mundo y está disponible en una docena de idiomas. Es software libre liberado bajo la licencia Artistic[40]. La versión actual de pgAdmin es 1.8.4. Se puede encontrar más información o descargar desde (PGADMIN, 2008).

[40] Esta licencia permite descargar el programa gratuitamente de la red y que su código sea utilizado en otros programas, bajo la condición que estos programas también sean publicados con una licencia del mismo tipo.

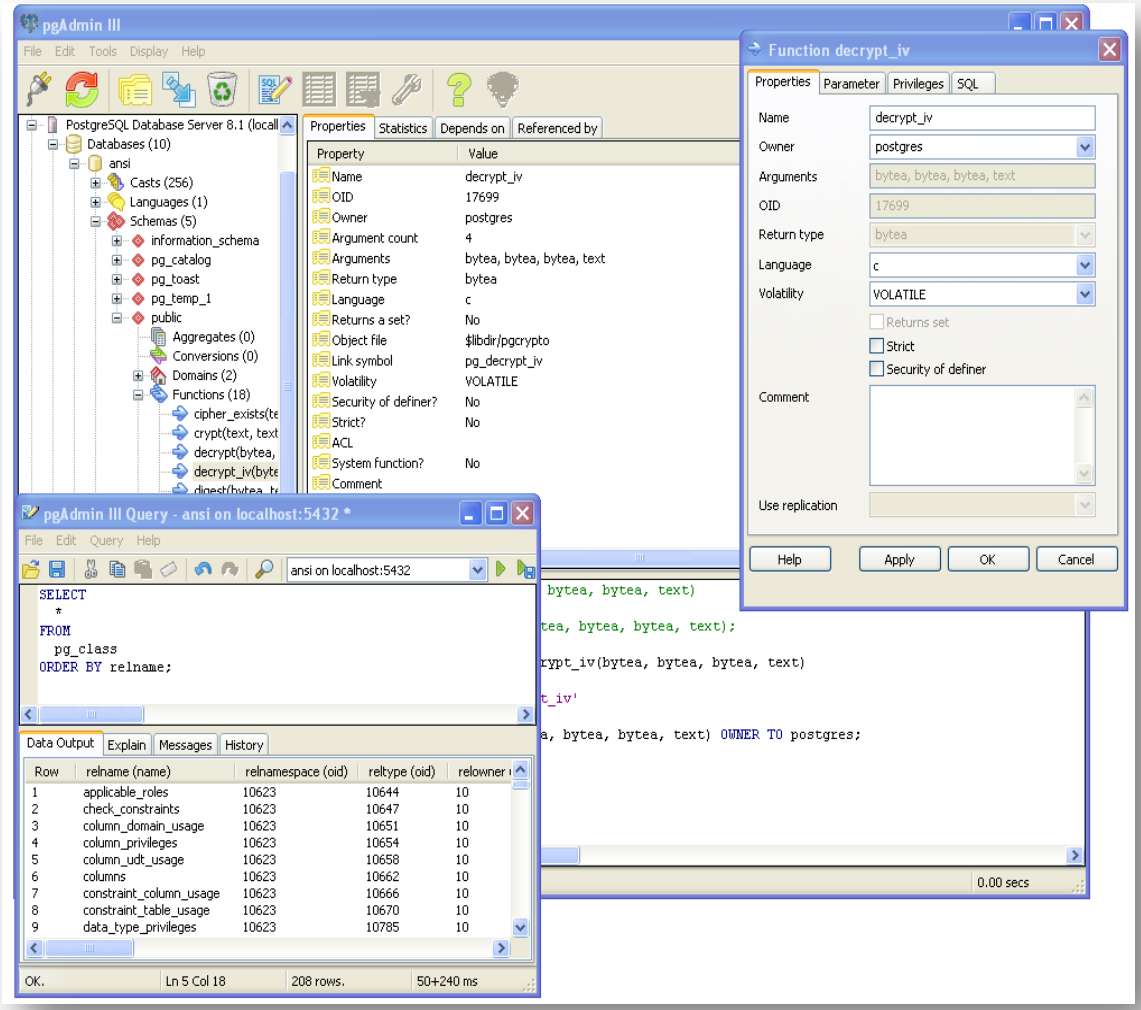

Figura 3.8. pgAdmin III

Capítulo 4

Desarrollo del Proyecto

Este proyecto es un estudio que permitirá organizar y poner en marcha la infraestructura de la aplicación web para su posterior implementación.

Seguidamente se comentarán 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 empezar a diseñar la aplicación informática y un primer análisis y especificación de requisitos.

Desde un punto de vista general, la aplicación intentará potenciar el uso de estructuras reusables de conocimiento (ontologías) para su aplicación en el trabajo diario de los interesados en la materia, resolviendo los problemas planteados para la trascripción, registro, recuperación y explotación de fuentes documentales mediante técnicas de la 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 ontologías, especificadas como esquemas semánticos, a partir de los conceptos registrados en recursos semánticos compartidos y teniendo en cuenta toda la propuesta teórica presentada anteriormente, definida para tal fin. Además de lo anterior, el sistema deberá proporcionar una síntesis e integración de la

información existente en las fuentes, eliminando redundancias y realizando inferencias.

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 *2.2.2. Usabilidad de un sitio web* 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 una plataforma.

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 para pasar a un diseño arquitectónico e implementación de modo iterativo-incremental. Estos modelos de ciclo de vida se explicaron en el apartado *2.3.2. Ciclo de vida del software*.

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.

4.1.2. Especificación de requisitos de usuario

En este punto se pretendió obtener una idea general de los requisitos funcionales (funcionalidad), que debía 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 ontologías

y documentos relacionados con la ingeniería del 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. Además, el sistema dispondrá de un indizador que permitirá incorporar nuevos documentos relacionados con el tema sobre el que quiera profundizar el usuario. También, permitirá realizar búsquedas por conceptos y recuperar documentos en los que se traten dichos conceptos. Finalmente, proporcionará 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 contenido.

La aplicación Web será utilizada por usuarios que podrán tener los siguientes roles que se definen a continuación:

- **Usuario:** Usuario de la aplicación cuya función está limitada, básicamente, a consultas.

- **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 del dominio:** Persona con amplios conocimientos de un dominio en concreto. Los expertos representan y validan los conceptos de las ontologías relacionadas con su dominio.
- **Experto responsable del dominio:** Es la persona que está al cargo de los expertos del dominio. Los conceptos más importantes sobre el dominio son validados por este experto. Es quien decide en caso de no existir consenso entre los expertos del dominio.
- **Ingeniero del dominio:** El encargado de gestionar y mantener la integridad de la ontología de referencia.
- **Ingeniero responsable del dominio:** Es un ingeniero en informática, destacado en el dominio. Ayuda a los expertos del dominio en sus tareas. Posee un mayor conocimiento del dominio. Su función se basa en coordinar y actuar como puente de comunicación entre ingenieros del dominio y expertos del dominio.
- **Diseñador del dominio:** Usuario que abstrae en el sistema tanto a Ingenieros como a Expertos.

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.

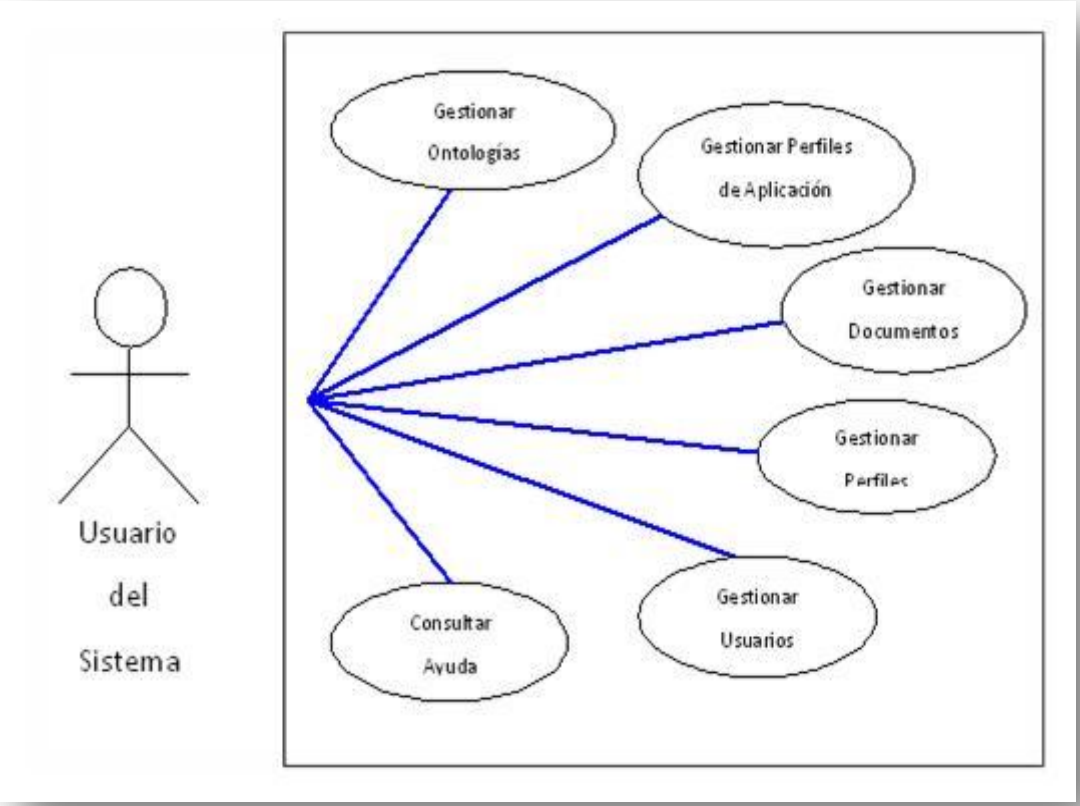

Figura 4.1. Diagrama inicial de casos de uso

A continuación se ofrece una descripción de los casos de uso mostrados en el diagrama anterior:

- **Gestión de ontologías.** Es un aspecto fundamental en la aplicación, que debe permitir:
  - Crear una nueva ontología.
  - Editar datos de una ontología existente.
  - Eliminar una ontología.
  - Importación y exportación de ontologías a formato RDF, OWL y esquemas XML.
  - Consultar ontologías existentes asociadas al concepto de búsqueda.
  - Mapear ontologías específicas contra ontologías genéricas.

- **Gestión de perfiles de aplicación.** Esta funcionalidad se divide en las siguientes operaciones:
    - Crear un perfil de aplicación.
    - Eliminar un perfil de aplicación.
    - Editar los datos de un perfil de aplicación.
    - Importar y exportar un perfil de aplicación.
- **Gestión de documentos.** La gestión de documentos permite:
    - Consultar documentos en los que aparece un concepto determinado, o bien que estén clasificados en un dominio o en los que aparezcan un conjunto específico de metadatos.
    - Crear un documento nuevo.
    - Eliminar un documento.
    - Modificar los campos asociados a un documento.
    - Indizar un documento al crearlo.
- **Gestión de perfiles.** El administrador de la aplicación debe poder añadir o eliminar perfiles de usuario en la aplicación, así como modificar cualquier información asociada a un perfil determinado.
- **Gestión de usuarios.** También es el administrador quien se encarga de:
    - Dar de alta a nuevos usuarios.
    - Dar de baja a antiguos usuarios.
    - Modificar datos de usuarios registrados en el sistema.
    - Validar a usuarios que se registran en la aplicación.

- **Consultar ayuda.** En cualquier momento de interacción con el sistema, se debe tener la posibilidad de consultar la ayuda que ofrece la aplicación sobre los temas que puedan resultar más complicados, en cuanto al manejo de ésta.

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. Como es natural, los requisitos expuestos en la Fase Inicial distan considerablemente de los que se presentan a continuación. De igual manera, 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 evitar 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 4.2.

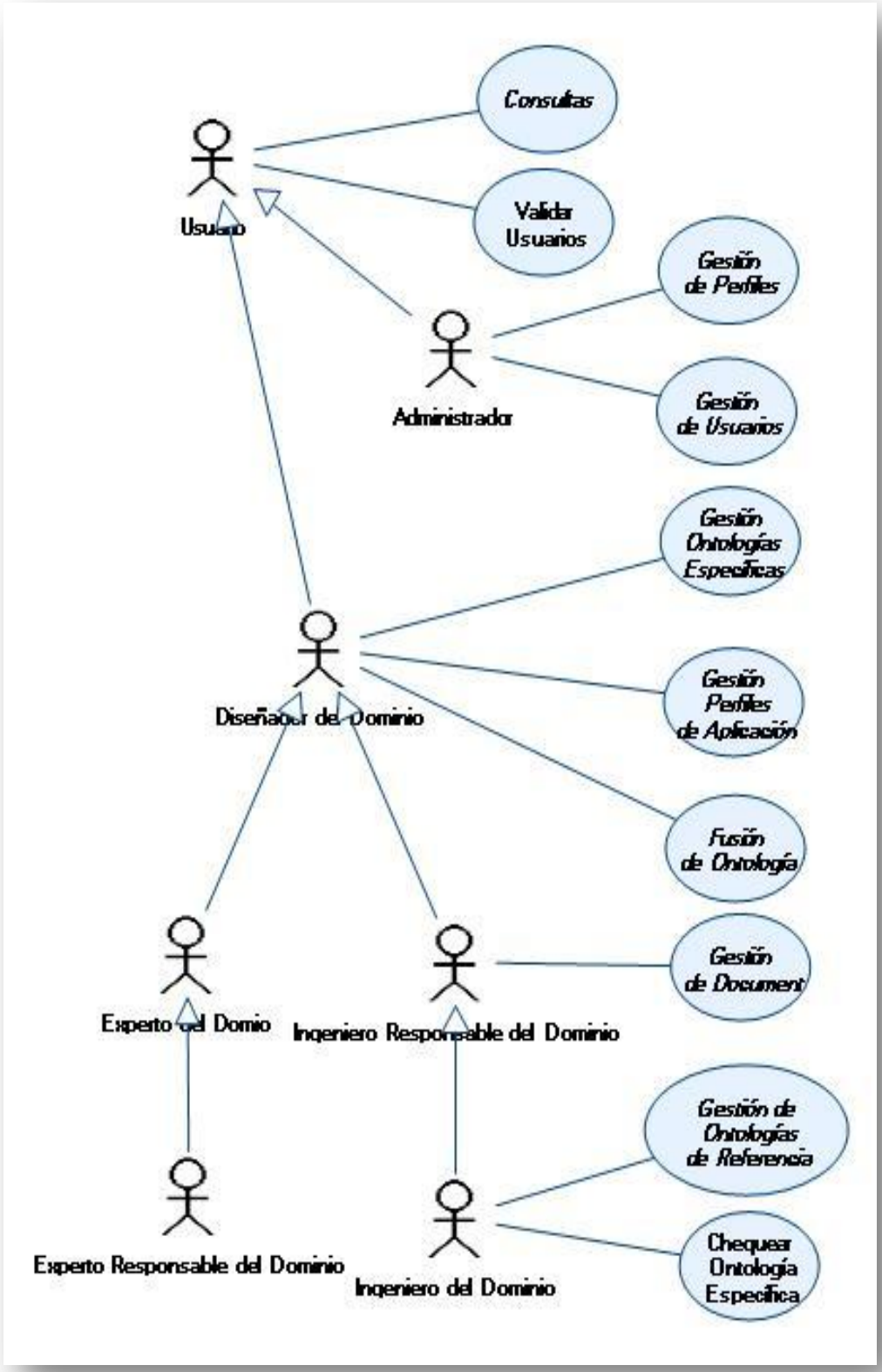

Figura 4.2. Diagrama general de casos de uso

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

Dichos casos de uso se han agrupado por usuario, es decir, por el rol que requiere dicho servicio o funcionalidad. Para facilitar su lectura e interpretación, se han incluido diagramas de casos de uso que representan los requisitos descritos textualmente.

4.2.1.1. Usuario

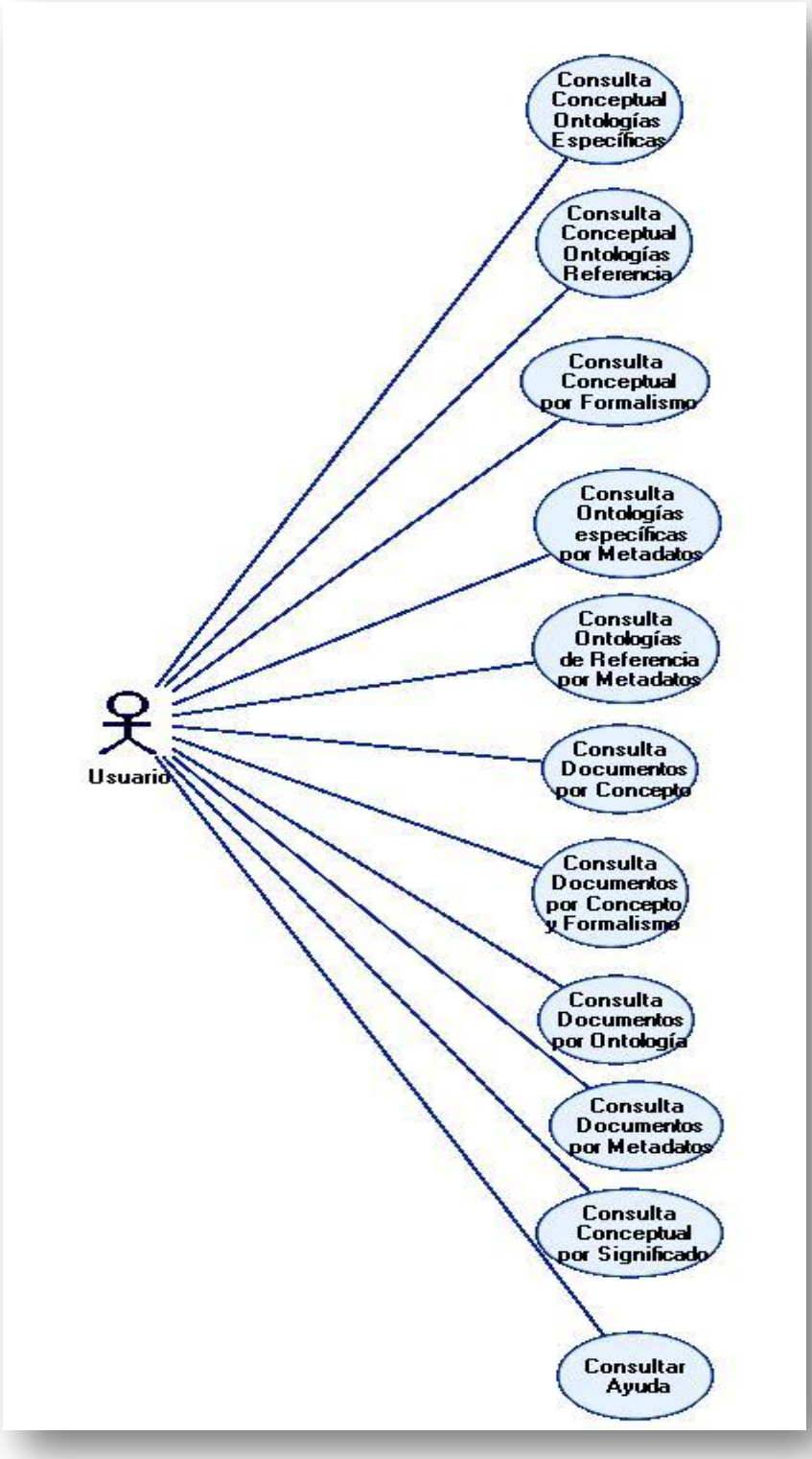

Figura 4.3. Diagrama de Consultas

Nombre	Consulta Conceptual Ontologías Específicas
Identificador	CU1
Actores	Usuario
Objetivo	Obtener la representación del concepto que busca el usuario en las distintas ontologías específicas. Se mostrará la definición del concepto, las relaciones con los conceptos padre e hijos y conceptos relacionados o sinónimos, además de la referencia a cada ontología especifica en la que se haya encontrado
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir concepto 2. Realizar la búsqueda 3. Mostrar resultados
Prioridad	Alta

Tabla 4.1. CU1: Consulta Conceptual Ontologías Específicas

Nombre	Consulta Conceptual Ontologías Referencia
Identificador	CU2
Actores	Usuario
Objetivo	Obtener la representación del concepto que busca el usuario en las distintas ontologías de Referencia. Se mostrará la definición del concepto, las relaciones con los conceptos padre e hijos y conceptos relacionados o sinónimos, además de la referencia a cada ontología de Referencia en la que se haya encontrado
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir concepto 2. Realizar la búsqueda 3. Mostrar resultados
Prioridad	Alta

Tabla 4.2. CU2: Consulta Conceptual Ontologías Específicas

Nombre	Consulta Conceptual por Formalismo
Identificador	CU3
Actores	Usuario
Objetivo	Obtener la representación del concepto que busca el usuario en las distintas ontologías especificas y que además se encuentran en el formalismo especificado
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir concepto y formalismo de búsqueda 2. Realizar la búsqueda 3. Mostrar resultados
Prioridad	Baja

Tabla 4.3. CU3: Consulta Conceptual por Formalismo

Nombre	Consulta Ontologías Especificas por Metadatos
Identificador	CU4
Actores	Usuario
Objetivo	Obtener las distintas ontologías especificas cuyos valores de metadatos coinciden con los introducidos por el usuario como criterio de búsqueda
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir metadatos por los que se desea buscar 2. Realizar la búsqueda 3. Mostrar resultados
Prioridad	Alta

Tabla 4.4. CU4: Consulta Ontologías Específicas por Metadatos

Nombre	Consulta Ontologías de Referencia por Metadatos
Identificador	CU5
Actores	Usuario

Objetivo	Obtener las distintas ontologías de referencia cuyos valores de metadatos coinciden con los introducidos por el usuario como criterio de búsqueda
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir metadatos por los que se desea buscar 2. Realizar la búsqueda entre las ontologías de referencia 3. Mostrar resultados
Prioridad	Alta

Tabla 4.5. CU5: Consulta Ontologías de Referencia por Metadatos

Nombre	**Consulta Documentos por Concepto**
Identificador	CU6
Actores	Usuario
Objetivo	Poder realizar búsquedas en el sistema por concepto, que nos permitan encontrar documentos en los que esté dicho concepto con la acepción adecuada. El resultado incluirá la siguiente información: • Nombre ontología (enlace a la ontología) • Fecha • Versión • Veces que ha sido usada
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir concepto de búsqueda 2. Realizar la búsqueda 3. Mostrar resultados relacionados con el concepto 4. Elegir una definición de las que el sistema encuentra 5. Se realiza la búsqueda en todos los documentos indizados según el concepto por el que se encuentra
Prioridad	Alta

Tabla 4.6. CU6: Consulta Documentos por Concepto

Nombre	Consulta Documentos por Concepto y Formalismo
Identificador	CU7
Actores	Usuario
Objetivo	Obtener documentos en los que aparezca el concepto con la acepción adecuada y que además, se encuentren en el formalismo especificado
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir concepto y formalismo de búsqueda 2. Realizar la búsqueda 3. Mostrar resultados 4. Se elige una definición de las que el sistema encuentra 5. Se realiza la búsqueda en todos los documentos indizados según el concepto por el que se consulta
Prioridad	Baja

Tabla 4.7. CU7: Consulta Documentos por Concepto y Formalismo

Nombre	Consulta Documentos por Ontología
Identificador	CU8
Actores	Usuarios
Objetivo	Obtener los documentos asociados a una ontología
Precondiciones	
Postcondiciones	
Escenario básico	1. Introducir dominio de búsqueda 2. Realizar la búsqueda 3. Mostrar Resultados 4. Se selecciona la ontología de la cual se quiere extraer la información 5. Se devuelven todos los documentos asociados a la ontología en cuestión
Prioridad	Alta

Tabla 4.8. CU8: Consulta Documentos por Concepto y Formalismo

Nombre	Consulta Documentos por Metadatos
Identificador	CU9
Actores	Usuario
Objetivo	Obtener la representación del concepto que busca el usuario en las distintas ontologías específicas. Se mostrará la definición del concepto, las relaciones con los conceptos padre e hijos y conceptos relacionados o sinónimos, además de a referencia a cada ontología especifica en la que se haya encontrado
Precondiciones	
Postcondiciones	
Escenario básico	1. Seleccionar los metadatos 2. Introducir los valores de búsqueda 3. Buscar ontologías según los valores de los metadatos 4. Buscar los documentos asociados a las ontologías encontradas 5. Mostrar resultados
Prioridad	Media

Tabla 4.9. CU9: Consulta Documentos por Metadatos

Nombre	Consulta Conceptual por Significado
Identificador	CU10
Actores	Usuario
Objetivo	Obtener documentos en los que aparezca el concepto con la acepción adecuada y los significados asociados al mismo. Se puede seleccionar uno de los significados y obtener todas las representaciones en las distintas ontologías específicas asociadas a ese significado. En una consulta por significado, se muestra el resultado además de otra información relacionada con el entorno del concepto: • Sinónimos • Antónimos • Padre del concepto • Hijos del concepto
Precondiciones	

Postcondiciones	
Escenario básico	1. Introducir concepto y significado de búsqueda 2. Realizar la búsqueda 3. Mostrar Resultados
Prioridad	Alta

Tabla 4.10. CU10: Consulta Conceptual por Significado

Nombre	**Consultar Ayuda**
Identificador	CU11
Actores	
Objetivo	Poder resolver las dudas que le surgen a un usuario en cualquier momento, durante la interacción con el sistema. Proporcionar un mecanismo de ayuda que pueda servir como guía en derminadas operaciones que se puedan realizar con la aplicación
Precondiciones	
Postcondiciones	
Escenario básico	1. Seleccionar la sección de ayuda del sistema 2. Consultar el área sobre el que desee obtener información
Prioridad	Alta

Tabla 4.11. CU11: Consultar Ayuda

Nombre	**Validar Usuario**
Identificador	CU12
Actores	Usuario
Objetivo	Validar a un usuario registrado en el sistema según el/los perfiles que tenga asignados
Precondiciones	
Postcondiciones	Usuario validado
Escenario básico	1. Introducir credencias 2. Realizar la validación
Prioridad	Alta

Tabla 4.12. CU12: Validar Usuario

4.2.1.2. Diseñador del dominio

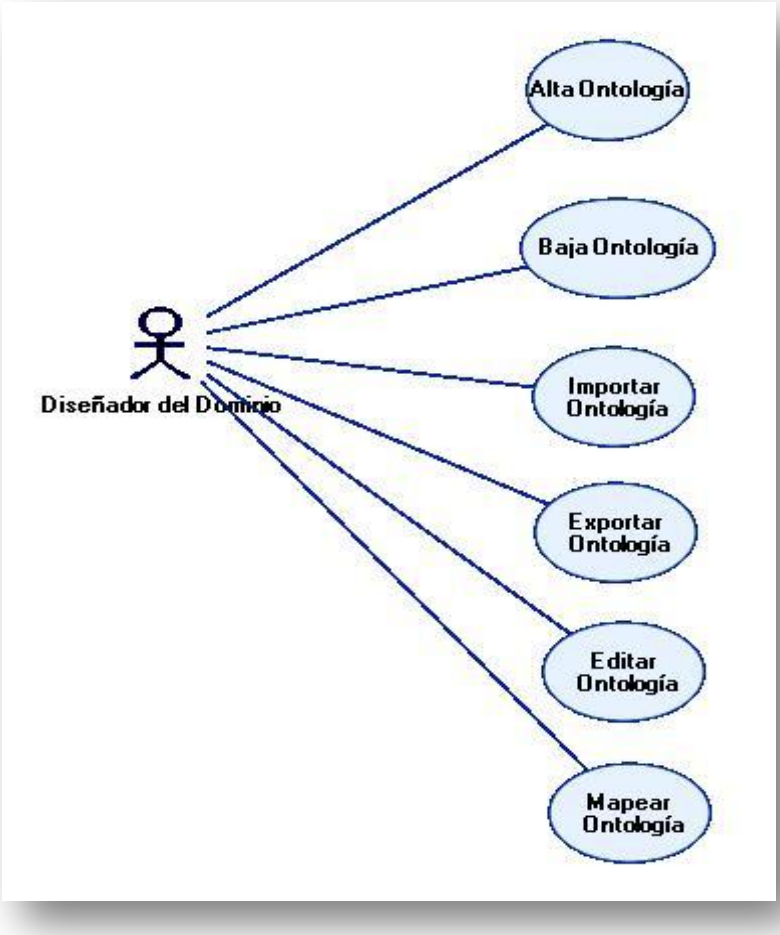

Figura 4.4. Diagrama de Gestión de ontologías específicas

Nombre	Alta Ontología
Identificador	CU13
Actores	Diseñador del dominio
Objetivo	Dar de alta una ontología especifica en el sistema
Precondiciones	
Postcondiciones	La ontología queda almacenada en el sistema
Escenario básico	1. Registrar la ontología 2. Guardar cambios
Prioridad	Alta

Tabla 4.13. CU13: Alta Ontología

Nombre	Baja Ontología
Identificador	CU14
Actores	Diseñador del dominio
Objetivo	Dar de baja una ontología especifica en el sistema
Precondiciones	Existe, al menos, una ontología dada de alta
Postcondiciones	La ontología se elimina del sistema
Escenario básico	1. Buscar ontología 2. Seleccionar ontología 3. Borrar ontología 4. Guardar cambios
Prioridad	Alta

Tabla 4.14. CU14: Baja Ontología

Nombre	Importar Ontología
Identificador	CU15
Actores	Diseñador del dominio
Objetivo	Poder importar una ontología especifica en el sistema
Precondiciones	La ontología ha sido dada de alto
Postcondiciones	La ontología se crea y se importa al sistema
Escenario básico	1. Buscar ontología 2. Seleccionar la fuente de la ontología a importar, por ejemplo desde fichero 3. Incluir metadatos de la nueva ontología creada
Prioridad	Alta

Tabla 4.15. CU15: Importar Ontología

Nombre	Exportar Ontología
Identificador	CU16
Actores	Diseñador del dominio
Objetivo	Poder exportar una ontología especifica del sistema
Precondiciones	Existe, al menos, una ontología dada de alta
Postcondiciones	
Escenario básico	1. Buscar ontología

	2. Seleccionar la ontología a exportar 3. Seleccionar el formato de exportación y ubicación, por ejemplo, "fichero"
Prioridad	Alta

Tabla 4.16. CU16: Exportar Ontología

Nombre	**Editar Ontología**
Identificador	CU17
Actores	Diseñador del dominio
Objetivo	Poder editar una ontología especifica del sistema
Precondiciones	Existe, al menos, una ontología dada de alta
Postcondiciones	La ontología queda almacenada con las modificaciones realizadas
Escenario básico	1. Seleccionar una ontología 2. Editar la ontología 3. Guardar los cambios
Prioridad	Alta

Tabla 4.17. CU17: Editar Ontología

Nombre	**Mapear Ontología**
Identificador	CU18
Actores	Diseñador del dominio
Objetivo	Relacionar los conceptos de la ontología especifica con los de la ontología de referencia. Cuando se trata de una ontología especifica, los mapeos se almacenan en atributos de los conceptos de la ontología y son editables, como cualquier otro.
Precondiciones	Existe, al menos, una ontología especifica y una ontología de referencia
Postcondiciones	La ontología especifica se mapea en el sistema
Escenario básico	1. Seleccionar la ontología a mapear 2. Realizar el proceso de mapeo
Prioridad	Alta

Tabla 4.18. CU18: Mapear Ontología

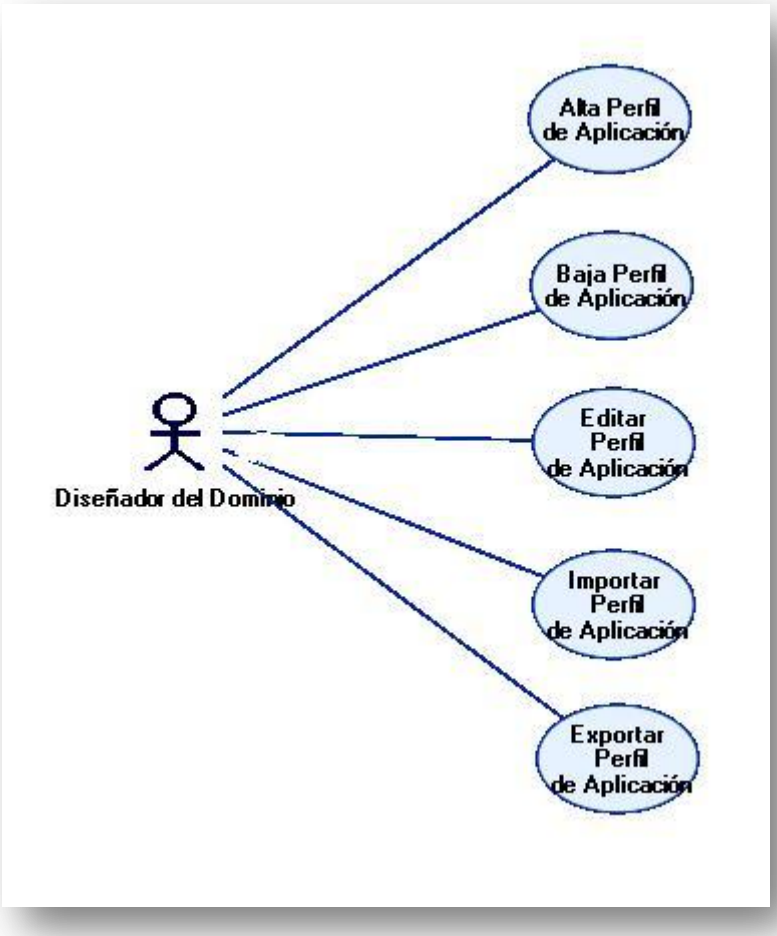

Figura 4.5. Diagrama de Gestión de Perfiles de Aplicación

Nombre	Alta Perfil de Aplicación
Identificador	CU19
Actores	Diseñador del dominio
Objetivo	Dar de alta un perfil de aplicación en el sistema
Precondiciones	
Postcondiciones	El perfil de aplicación queda almacenado en el sistema
Escenario básico	1. Registrar el nuevo perfil de aplicación 2. Guardar cambios
Prioridad	Media

Tabla 4.19. CU19: Alta Perfil de Aplicación

Nombre	Baja Perfil de Aplicación
Identificador	CU20
Actores	Diseñador del dominio
Objetivo	Dar de baja un perfil de aplicación en el sistema
Precondiciones	Existe, al menos, un perfil de aplicación dado de alta
Postcondiciones	El perfil de aplicación se elimina del sistema
Escenario básico	1. Buscar perfil de aplicación 2. Borrar perfil de aplicación 3. Guardar cambios
Prioridad	Media

Tabla 4.20. CU20: Baja Perfil de Aplicación

Nombre	Editar Perfil de Aplicación
Identificador	CU21
Actores	Diseñador del dominio
Objetivo	Poder modificar los datos referentes a un determinado perfil de aplicación
Precondiciones	Existe, al menos, un perfil de aplicación dado de alta
Postcondiciones	El perfil de aplicación queda actualizado con los nuevos datos
Escenario básico	1. Buscar perfil de aplicación 2. Realizar cambios sobre dicho perfil 3. Guardar cambios
Prioridad	Media

Tabla 4.21. CU21: Editar Perfil de Aplicación

Nombre	Importar Perfil de Aplicación
Identificador	CU22
Actores	Diseñador del dominio
Objetivo	Poder importar un perfil de aplicación al sistema
Precondiciones	El perfil de aplicación ha si sido dado de alta
Postcondiciones	El perfil se crea y se importa al sistema
Escenario básico	1. Buscar perfil de aplicación

	2. Seleccionar fichero importación 3. Importar un perfil de aplicación
Prioridad	Media

Tabla 4.22. CU22: Importar Perfil de Aplicación

Nombre	**Exportar Perfil de Aplicación**
Identificador	CU23
Actores	Diseñador del dominio
Objetivo	Poder exportar un perfil de aplicación del sistema
Precondiciones	Existe, al menos, un perfil de aplicación
Postcondiciones	
Escenario básico	1. Buscar perfil de aplicación 2. Seleccionar el perfil a exportar 3. Seleccionar el formato de exportación y ubicación, por ejemplo, “fichero”
Prioridad	Media

Tabla 4.23. CU23: Exporta Perfil de Aplicación

Nombre	**Fusionar Ontologías**
Identificador	CU24
Actores	Diseñador del dominio
Objetivo	Poder unir varias ontologías especificas en una sola a través de la referencia
Precondiciones	Deben existir, al menos, dos ontologías para poder realizar la fusión
Postcondiciones	De la fusión surge una nueva ontología en el sistema
Escenario básico	1. Seleccionar las ontologías a fusionar 2. Realizar el proceso de equivalencia de conceptos 3. Definir la información sobre la nueva ontología agregada (metadatos)
Prioridad	Baja

Tabla 4.24. CU24: Fusionar Ontologías

4.2.1.3. Ingeniero del Dominio

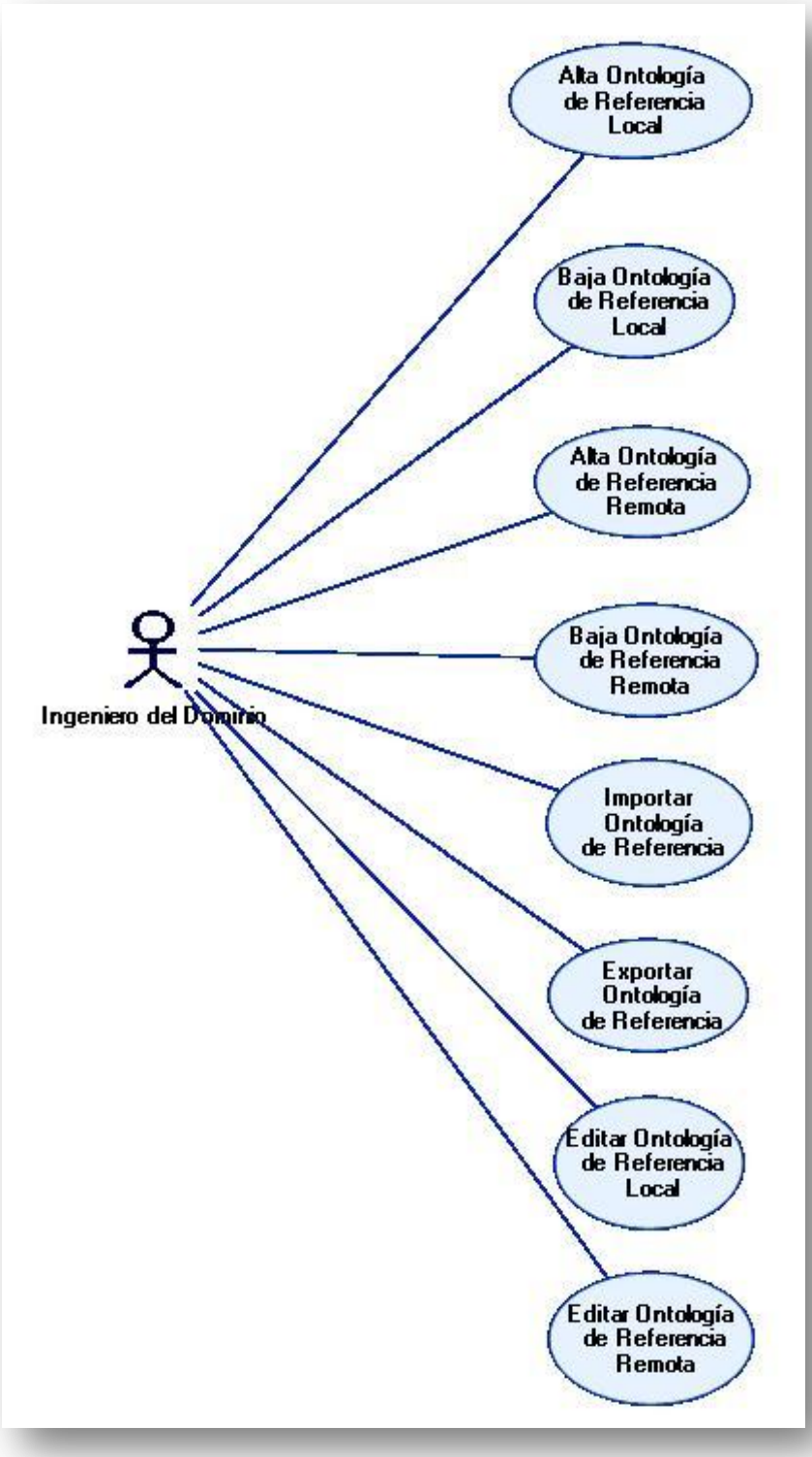

Figura 4.6. Diagrama de Gestión de Ontologías de Referencia

Nombre	Alta Ontología de Referencia Local
Identificador	CU25
Actores	Ingeniero del dominio
Objetivo	Poder dar de alta una ontología de referencia local al sistema
Precondiciones	
Postcondiciones	Se almacena una nueva ontología de referencia local al sistema
Escenario básico	1. Registrar la ontología de referencia 2. Importar la ontología de referencia 3. Guardar cambios
Prioridad	Alta

Tabla 4.25. CU25: Alta Ontología de Referencia Local

Nombre	Baja Ontología de Referencia Local
Identificador	CU26
Actores	Ingeniero del dominio
Objetivo	Poder dar de baja una ontología de referencia local al sistema
Precondiciones	Existe, al menos, una ontología de referencia local
Postcondiciones	Se elimina una ontología de referencia local. Además, como consecuencia de este borrado, las referencias a conceptos de la ontología desaparecen en cascada, es decir, al mismo tiempo que se elimina del sistema
Escenario básico	1. Seleccionar la ontología a eliminar 2. Borrar la ontología seleccionada 3. Guardar cambios
Prioridad	Alta

Tabla 4.26. CU26: Baja Ontología de Referencia Local

Nombre	Alta Ontología de Referencia Remota
Identificador	CU27
Actores	Ingeniero del dominio
Objetivo	Poder dar de alta una ontología de referencia remota
Precondiciones	

Postcondiciones	Se almacena un enlace a la nueva ontología de referencia remota
Escenario básico	1. Registrar la URL en la que se encuentra la ontología remota 2. Guardar cambios
Prioridad	Alta

Tabla 4.27. CU27: Alta Ontología de Referencia Remota

Nombre	**Baja Ontología de Referencia Remota**
Identificador	CU28
Actores	Ingeniero del dominio
Objetivo	Poder dar de baja una ontología de referencia remota
Precondiciones	Existe, al menos, una ontología de referencia remota
Postcondiciones	Se elimina una ontología de referencia remota, es decir, la URL que hace de enlace a la ontología de referencia. Además, como consecuencia de este borrado, las referencias a conceptos de la ontología desaparecen en cascada, es decir, al mismo tiempo que se elimina del sistema
Escenario básico	1. Seleccionar la URL de la ontología a eliminar 2. Borrar el enlace a la ontología 3. Guardar cambios
Prioridad	Alta

Tabla 4.28. CU28: Baja Ontología de Referencia Remota

Nombre	**Importar Ontología de Referencia**
Identificador	CU29
Actores	Ingeniero del dominio
Objetivo	Poder importar una ontología de referencia en el sistema
Precondiciones	La ontología de referencia se ha dado de alta como ontología local
Postcondiciones	Se almacena una nueva ontología de referencia en el sistema
Escenario básico	1. Buscar ontología de referencia 2. Seleccionar la fuente de la ontología a importar, por ejemplo desde “fichero”

	3. Incluir metadatos de la nueva ontología creada
Prioridad	Alta

Tabla 4.29. CU29: Importar Ontología de Referencia

Nombre	**Exportar Ontología de Referencia**
Identificador	CU30
Actores	Ingeniero del dominio
Objetivo	Poder exportar una ontología de referencia del sistema
Precondiciones	Existe, al menos, una ontología de referencia local
Postcondiciones	
Escenario básico	1. Buscar ontología 2. Seleccionar la ontología a exportar 3. Seleccionar el formato de exportación y ubicación, por ejemplo, "fichero"
Prioridad	Alta

Tabla 4.30. CU30: Exportar Ontología de Referencia

Nombre	**Editar Ontología de Referencia Local**
Identificador	CU31
Actores	Ingeniero del dominio
Objetivo	Poder editar cualquier elemento de la ontología de referencia del sistema. El ingeniero del dominio debe ser capaz de editar una ontología de referencia local, pudiendo modificar campos como los siguientes: • Clases • Subclases • Propiedades • Relaciones entre clases • Comentarios • Dominios • Condiciones booleanas • Etc…

Precondiciones	Existe, al menos, una ontología de referencia local
Postcondiciones	La ontología queda almacenada con las modificaciones realizadas
Escenario básico	1. Seleccionar la ontología de referencia 2. Editar elementos de la ontología 3. Guardar cambios
Prioridad	Alta

Tabla 4.31. CU31: Editar Ontología de Referencia Local

Nombre	**Editar Ontología de Referencia Remota**
Identificador	CU32
Actores	Ingeniero del dominio
Objetivo	Poder cambiar la URL que referencia a la ontología remota
Precondiciones	Existe, al menos, una ontología de referencia remota
Postcondiciones	La URL queda modificada
Escenario básico	1. Seleccionar la ontología de referencia 2. Realizar cambios en la URL 3. Guardar cambios
Prioridad	Alta

Tabla 4.32. CU32: Editar Ontología de Referencia Remota

Nombre	**Chequear Ontología Específica**
Identificador	CU33
Actores	Ingeniero del dominio
Objetivo	Permitir la comprobación de los conceptos y sus descripciones incluidos en una ontología especifica. En caso de ser necesario, deberá permitirse la edición de dicha ontología. La comprobación debe permitir consultar los valores de los atributos, incluyendo los significados asociados a la ontología de referencia.
Precondiciones	Existe, al menos, una ontología especifica
Postcondiciones	
Escenario básico	1. Seleccionar la ontología especifica

	2. Realizar la comprobación: a. Consultar valores de atributos b. Consultar significados asociados 3. Chequear si "casan" las referencias con la ontología de referencia 4. Editar la ontología local, en su caso
Prioridad	Alta

Tabla 4.33. CU33: Chequear Ontología Específica

4.2.1.4. Ingeniero Responsable del Dominio

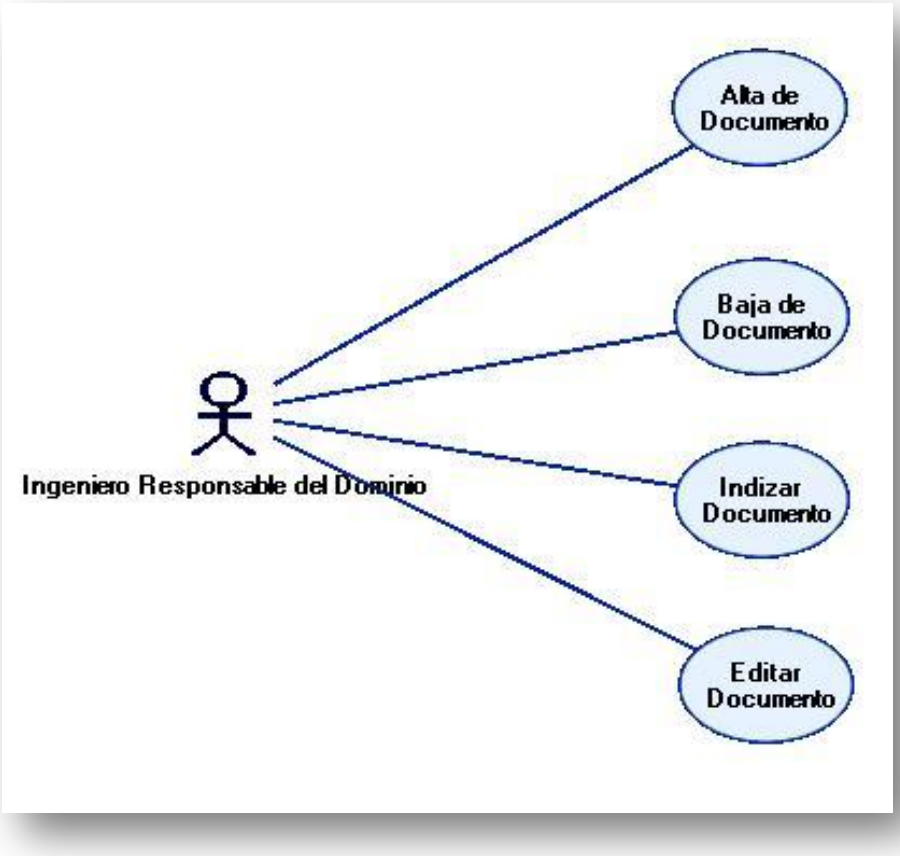

Figura 4.7. Diagrama de Gestión de Documentos

Nombre	Alta de Documento
Identificador	CU34
Actores	Ingeniero responsable del dominio
Objetivo	Poder dar de alta nuevos documentos
Precondiciones	
Postcondiciones	Se generarán las instancias correspondientes al documento
Escenario básico	1. Seleccionar el documento 2. Seleccionar la ontología que lo describe 3. Indizar el documento 4. Mostrar resultado indización y confirmar 5. Generar las instancias del documento indizado 6. Guardar cambios
Prioridad	Media

Tabla 4.34. CU34: Alta de Documento

Nombre	Baja de Documento
Identificador	CU35
Actores	Ingeniero responsable del dominio
Objetivo	Poder eliminar documentos, borrando al mismo tiempo las referencias entre el documento y los términos a los que estuviera asociado
Precondiciones	Existe, al menos, un documento dado de alta
Postcondiciones	Las instancias derivadas del documento son eliminadas
Escenario básico	1. Seleccionar el documento deseado 2. Eliminar las instancias asociadas al documento 3. Eliminar datos del documento 4. Guardar cambios
Prioridad	Media

Tabla 4.35. CU35: Baja de Documento

Nombre	Indizar Documento
Identificador	CU36
Actores	Ingeniero responsable del dominio
Objetivo	Generar un fichero con el resultado de la indización del documento
Precondiciones	
Postcondiciones	Se generará un fichero con el resultado de la indización
Escenario básico	1. Seleccionar el documento 2. Seleccionar la ontología que lo describe 3. Realizar la operación de indización 4. Guardar cambios a fichero
Prioridad	Media

Tabla 4.36. CU36: Indizar Documento

Nombre	Editar Documento
Identificador	CU37
Actores	Ingeniero responsable del dominio
Objetivo	La modificación de documentos se realizaría como una baja y la posterior creación del documento actualizado una vez indizado
Precondiciones	Existe, al menos, un documento dado de alta
Postcondiciones	El documento queda modificado
Escenario básico	1. Seleccionar el documento a eliminar 2. Eliminar el documento 3. Alta de documento 4. Guardar cambios
Prioridad	Media

Tabla 4.37. CU37: Editar Documento

4.2.1.5. Administrador

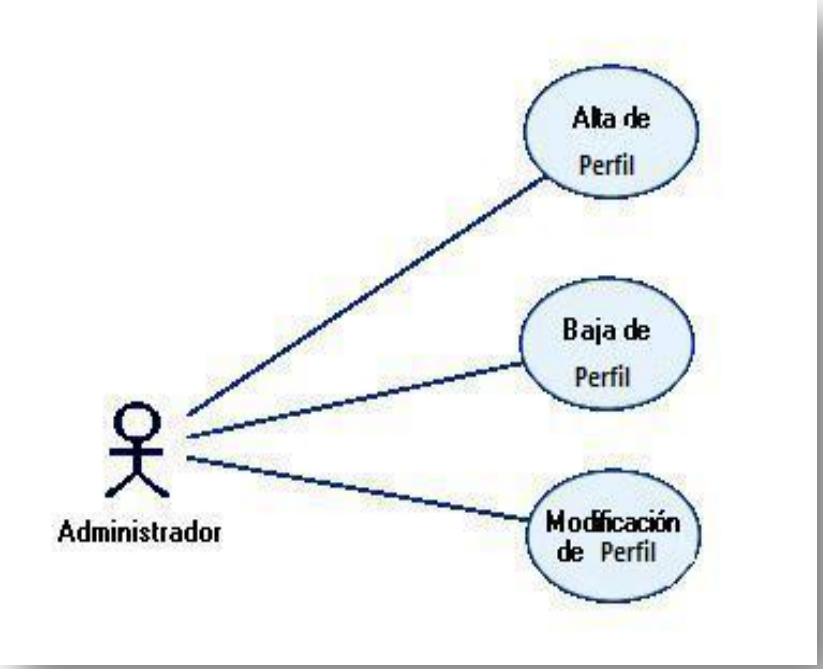

Figura 4.8. Diagrama de Gestión de Perfiles

Nombre	Alta de Perfil
Identificador	CU38
Actores	Administrador
Objetivo	Agregar un perfil de usuario al sistema
Precondiciones	
Postcondiciones	El perfil es almacenado en el sistema
Escenario básico	1. Introducir nombre de perfil 2. Asignar permisos al perfil 3. Guardar cambios
Prioridad	Alta

Tabla 4.38. CU38: Alta de Perfil

Nombre	Baja de Perfil
Identificador	CU39
Actores	Administrador
Objetivo	Eliminar un perfil de usuario del sistema
Precondiciones	
Postcondiciones	El perfil es eliminado del sistema
Escenario básico	1. Mostar los perfiles existentes 2. Seleccionar el perfil a eliminar 3. Seleccionar opción eliminación 4. Confirmar operación
Prioridad	Alta

Tabla 4.39. CU39: Baja de Perfil

Nombre	Modificación de Perfil
Identificador	CU40
Actores	Administrador
Objetivo	Modificar un perfil de usuario del sistema
Precondiciones	
Postcondiciones	El perfil es modificado en el sistema
Escenario básico	1. Mostar los perfiles existentes 2. Seleccionar el perfil a modificar 3. Mostrar los permisos, indicando cuales están asociados al perfil 4. Seleccionar los permisos a agregar/eliminar 5. Seleccionar opción de modificación 6. Confirmar operación
Prioridad	Alta

Tabla 4.40. CU40: Modificación de Perfil

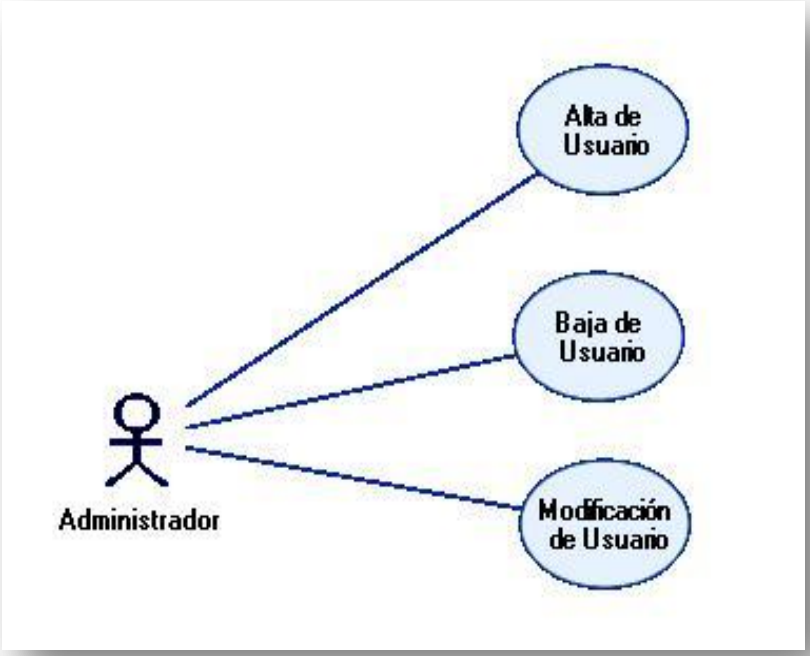

Figura 4.9. Diagrama de gestión de Usuarios

Nombre	Alta de Usuario
Identificador	CU41
Actores	Administrador
Objetivo	Alta de un usuario en el sistema, incluyendo la asignación de perfil
Precondiciones	El identificador del usuario no existe en el sistema
Postcondiciones	El usuario es dado de alta en el sistema
Escenario básico	1. Introducir los datos del usuario 2. Mostrar los perfiles disponibles 3. Seleccionar un perfil 4. Seleccionar opción de alta 5. Confirmar operación
Prioridad	Alta

Tabla 4.41. CU41: Alta de Usuario

Nombre	Baja de Usuario
Identificador	CU42
Actores	Administrador
Objetivo	Baja de un usuario en el sistema
Precondiciones	El identificador del usuario existe en el sistema
Postcondiciones	El usuario es dado de baja del sistema
Escenario básico	1. Introducir el identificador del usuario 2. Seleccionar opción de baja 3. Confirmar operación
Prioridad	Alta

Tabla 4.42. CU42: Baja de Usuario

Nombre	Modificación de Usuario
Identificador	CU43
Actores	Administrador
Objetivo	Modificación de un usuario del sistema
Precondiciones	El identificador del usuario existe en el sistema
Postcondiciones	El usuario es modificado en el sistema
Escenario básico	1. Introducir el identificar del usuario 2. Mostrar los datos del usuario incluyendo el perfil asignado a. Modificar los datos del usuario b. Seleccionar un nuevo perfil 3. Confirmar operación
Prioridad	Alta

Tabla 4.43. CU43: Modificación de Usuario

4.3. Diseño arquitectónico

Esta fase del proyecto comprende la última versión de la arquitectura definida para el soporte de la aplicación. Se pretende, en esta sección, 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 aplicará el patrón MVC descrito en el *apartado 2.3.4. Estilos y patrones de diseño* para la descomposición de la aplicación. De este modo, los componentes desarrollados 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 que ofrece una manera sencilla y clara de generar páginas Web, mientras que para la capa de infraestructura se usará Jena.

Entre la vista y el modelo se sitúa la capa del controlador. Es precisamente aquí donde se integran los frameworks utilizados: Spring y JSF. Ambos han sido descritos en los apartados *2.4. Análisis de Tecnologías* y *3.2.3. Frameworks utilizados*.

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 configura el fichero de configuración faces-config.xml de la siguiente manera:

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

De esta forma estando en la vista GestionOntologias.jsp mediante la acción alta se muestra la vista AltaOntologia.jsp.

Por otro lado, Spring, en la capa de negocio, permite inyectar dependencias en el momento en el que una clase es ejecutada. Para ello se utilizará otro archivo de configuración denominado applicationContext.xml, de tal manera que no es necesario crear un objeto sino que Spring crea el objeto por nosotros.

Por último, para la capa de persistencia se utilizará la tecnología PostgreSQL, que se ajusta al volumen de datos y tráfico que se prevé para esta aplicación.

A continuación, en la figura 4.10, se muestra el particionamiento de las capas de la aplicación y las tecnologías elegidas para cada capa.

Figura 4.10. Arquitectura en capas

4.3.1. Infraestructura Hardware

Como ya 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.

En el estado actual del proyecto, desarrollo, se utiliza una infraestructura con una única máquina (PC), cuando se llegue a la fase de despliegue se decidirá la arquitectura óptima para la fase de producción.

4.3.2. Infraestructura Software

Las decisiones en cuanto a software, que se comentaron también en el *Capítulo 3. Herramientas para la elaboración del proyecto,* 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 Tomcat 6, como servidor Web y contenedor de los componentes ejecutables vía Web (Servlets), así como el gestor de Bases de Datos PostgreSQL.

- Se eligió Jena para el repositorio semántico destinado a la gestión de la persistencia de las ontologías, a través de las peticiones que reciben de los Servlets. Con este framework se independiza la gestión de los repositorios semánticos y su persistencia del resto de la aplicación. Su elección se ha basado en la compatibilidad con las últimas versiones de Java y en su facilidad de uso principalmente.

- JSF se utilizará 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.

- 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, aparte de otros beneficios expuestos anteriormente.

- 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 porque 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 de Tomcat, 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 las ontologías a través del framework Jena. Desde estas clases, que conforman el modelo del dominio, se accede a la Base de Datos (PostgreSQL) para operaciones de inserción, borrado y consulta haciendo uso de Jena.

A continuación, en la figura 4.11, se muestra la arquitectura del sistema con las tecnologías seleccionadas.

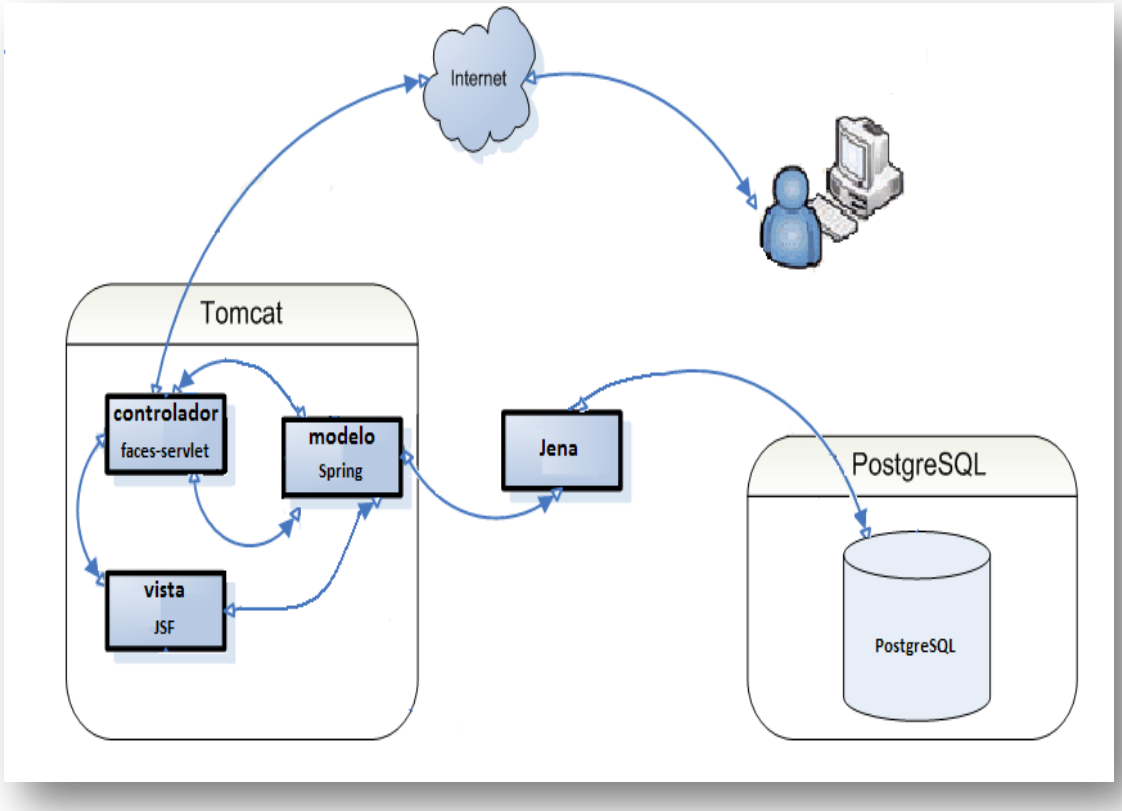

Figura 4.11. Arquitectura del sistema

4.4. Prueba de concepto

En este apartado se probará la validez de la arquitectura diseñada mediante la implementación de una pequeña parte de la funcionalidad del sistema. Como ejemplo se detallará la implementación del caso de uso “Alta de Ontología”.

En primer lugar, se mostrará la pantalla que permite dar de alta una ontología y el código de las clases involucradas en la implementación que permite realizar el alta.

AltaOntologia

Captura de pantalla de la página jsp asociada a la implementación de la página jsf correspondiente a Alta de Ontología.

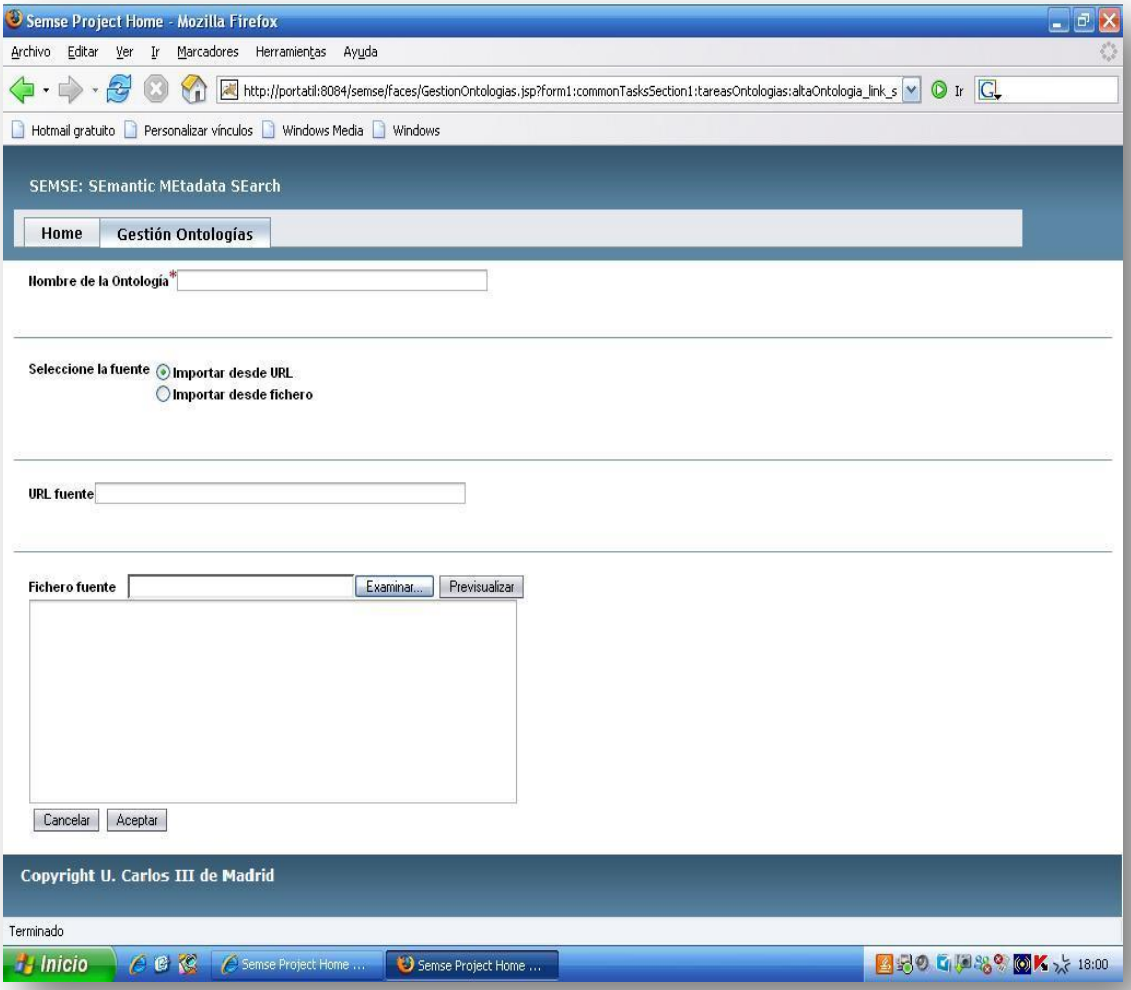

Figura 4.12. Alta de Ontología

OntologyBean.java

Managed Bean que alberga la información del BakingBean de la página AltaOntologia. Es utilizado para generar el VO y su código es similar al "OntologyVO".

OntologyVO.java

Value Object que alberga los datos de la ontología en el modelo.

```
package semse.model.bussinessobject.ontology.vo;

import java.io.Serializable;
```

```
public class OntologyVO implements Serializable {
   private String ontologyName = "";
   private String ontologyLanguage = "";
   private String reasoner = "";
   private String exportFileName = "";
   private String language = "";
   private Boolean exportAll = false;
   private String importSource = "";
   private Boolean importFromFile = false;

   public OntologyVO() {
   }
   public String getOntologyName() {
      return ontologyName;
   }
   public void setOntologyName(String ontologyName) {
      this.ontologyName = ontologyName;
   }
   public String getOntologyLanguage() {
      return ontologyLanguage;
   }
   public void setOntologyLanguage(String ontologyLanguage) {
      this.ontologyLanguage = ontologyLanguage;
   }
   public String getReasoner() {
      return reasoner;
   }
   public void setReasoner(String reasoner) {
      this.reasoner = reasoner;
   }
   public String getExportFileName() {
      return exportFileName;
   }
```

```
    public void setExportFileName(String exportFileName) {
        this.exportFileName = exportFileName;
    }
    public String getLanguage() {
        return language;
    }
    public void setLanguage(String language) {
        this.language = language;
    }
   public Boolean getExportAll() {
        return exportAll;
    }
   public void setExportAll(Boolean exportAll) {
        this.exportAll = exportAll;
    }
    public String getImportSource() {
        return importSource;
    }
    public void setImportSource(String importSource) {
        this.importSource = importSource;
    }
    public Boolean getImportFromFile() {
        return importFromFile;
    }
    public void setImportFromFile(Boolean importFromFile) {
        this.importFromFile = importFromFile;
    }
}
```

OntologyBuilder.java

Clase que mapea el OntologyBean y el OntologyVO.

```
package semse.view.builder;

import org.apache.commons.beanutils.BeanUtils;
import semse.model.bussinessobject.ontology.vo.OntologyVO;
import semse.model.util.exception.InternalErrorException;
import semse.view.bean.OntologyBean;

public class OntologyBuilder {
  public static void populateOntologyBean(OntologyBean ontologyBean, OntologyVO
 ontologyVO) throws  InternalErrorException {
        try {
              BeanUtils.copyProperties(ontologyBean, ontologyVO);
        } catch (Exception e) {
              throw new InternalErrorException(e);
        }
  }
  public static OntologyBean createOntologyBean(OntologyVO ontologyVO) throws
 InternalErrorException {
        OntologyBean ontologyBean = new OntologyBean();
        try {
              BeanUtils.copyProperties(ontologyBean, ontologyVO);
        } catch (Exception e) {
              throw new InternalErrorException(e);
        }
        return ontologyBean;
  }
  public static OntologyVO createOntologyVO(OntologyBean ontologyBean) throws
 InternalErrorException {
        OntologyVO ontologyVO = new OntologyVO();
        try {
              BeanUtils.copyProperties(ontologyVO, ontologyBean);
```

```
            } catch (Exception e) {
                    throw new InternalErrorException(e);
            }
            return ontologyVO;
    }
}
```

OntologyFacadeDelegate.java

Fachada con los casos de uso relacionados con la gestión de ontologías.

```
package semse.model.service.ontologyfacade.delegate;

import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import semse.model.bussinessobject.ontology.vo.OntologyVO;
import semse.model.util.exception.InstanceNotFoundException;
import semse.model.util.exception.InternalErrorException;

public interface OntologyFacadeDelegate {
    public OntModel createOntology(OntologyVO ontologyVO)
            throws InternalErrorException;
    public void removeOntology(OntologyVO ontologyVO)
            throws InternalErrorException;
    public void exportOntology(OntologyVO ontologyVO)
            throws InternalErrorException;
    public void editOntology()
            throws InstanceNotFoundException, InternalErrorException;
    public void mapOntology()
            throws InstanceNotFoundException, InternalErrorException;
    public ExtendedIterator getModelNames() throws InternalErrorException;
}
```

ontologyFacade.java

Implementación de la fachada en función de acciones.

```
package semse.model.service.ontologyfacade.plain;

import com.hp.hpl.jena.db.DBConnection;
import com.hp.hpl.jena.db.IDBConnection;
import com.hp.hpl.jena.db.RDFRDBException;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import java.sql.SQLException;
import javax.sql.DataSource;
import semse.model.bussinessobject.ontology.vo.OntologyVO;
import semse.model.service.ontologyfacade.delegate.OntologyFacadeDelegate;
import semse.model.service.ontologyfacade.plain.actions.CreateOntologyAction;
import semse.model.service.ontologyfacade.plain.actions.ExportOntologyAction;
import semse.model.service.ontologyfacade.plain.actions.RemoveOntologyAction;
import semse.model.util.exception.InstanceNotFoundException;
import semse.model.util.exception.InternalErrorException;

public class OntologyFacade implements OntologyFacadeDelegate {
   private DataSource dataSource;
   private String DBType;
   public OntologyFacade() throws InternalErrorException {
   }
   public OntModel createOntology(OntologyVO ontologyVO) throws InternalErrorException
  {
      try {
         IDBConnection connection = new DBConnection(getDataSource().getConnection(),
        getDBType());
         CreateOntologyAction action = new CreateOntologyAction(connection, ontologyVO);
         return action.execute();
      } catch (SQLException ex) {
```

```
            throw new InternalErrorException(ex);
        }
    }
    .
    .
    .
    .
    .
    .
    public ExtendedIterator getModelNames() throws InternalErrorException {
        try {
            IDBConnection connection = new DBConnection(getDataSource().getConnection(),
            getDBType());
            return connection.getAllModelNames();
        } catch (SQLException ex) {
            throw new InternalErrorException(ex);
        } catch (RDFRDBException rDFRDBException) {
            throw new InternalErrorException(rDFRDBException);
        }
    }
    public DataSource getDataSource() {
        return dataSource;
    }
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
    public String getDBType() {
        return DBType;
    }
    public void setDBType(String DBType) {
        this.DBType = DBType;
    }
}
```

CreateOntologyAction.java

Acción que encapsula la creación de un modelo persistente en el sistema.

Crea la ontología a partir de los datos incluidos en el Value Object.

```
package semse.model.service.ontologyfacade.plain.actions;

import com.hp.hpl.jena.db.IDBConnection;
import com.hp.hpl.jena.db.ModelRDB;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.ontology.ProfileRegistry;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import semse.model.bussinessobject.ontology.vo.OntologyVO;
import semse.model.util.exception.InternalErrorException;

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 spec = OntModelSpec.getDefaultSpec(ProfileRegistry.OWL_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, null);
          } 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;
    }
}
```

A continuación se explicará el funcionamiento del caso de uso “Alta de Ontología”. La página jsf “Alta Ontología” mediante la información de su BackingBean genera el OntologyBean que será utilizado posteriormente para generar el VO a través de la clase OntologyBuilder. Posteriormente se accederá a la interfaz OntologyFacadeDelegate. La clase que realiza la interfaz OntologyFacadeDelegate es proporcionada por el framework Spring. Para ello se configura el fichero faces-config para la configuración del atributo ‘ontologyFacade’:

<managed-bean>

<managed-bean-name>AltaOntologia</managed-bean-name>

<managed-bean-class>semse.AltaOntologia</managed-bean-class>

```
        <managed-bean-scope>request</managed-bean-scope>
        <managed-property>
          <property-name>ontologyFacade</property-name>
          <value>#{ontologyFacade}</value>
        </managed-property>
        <managed-property>
          <property-name>ontologyBean</property-name>
          <value>#{OntologyBean}</value>
        </managed-property>
    </managed-bean>
```

Y el fichero applicationContext para la configuración de la clase que realiza la interfaz:

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

Esta clase delegará en la acción correspondiente que, mediante el framework Jena y su paquete db, permitirá la conexión a la Base de Datos PostgreSQL y realizará el alta de ontología.

Finalmente se mostrará el diagrama que resume las clases involucradas en la implementación del caso de uso “Alta de Ontologia”.

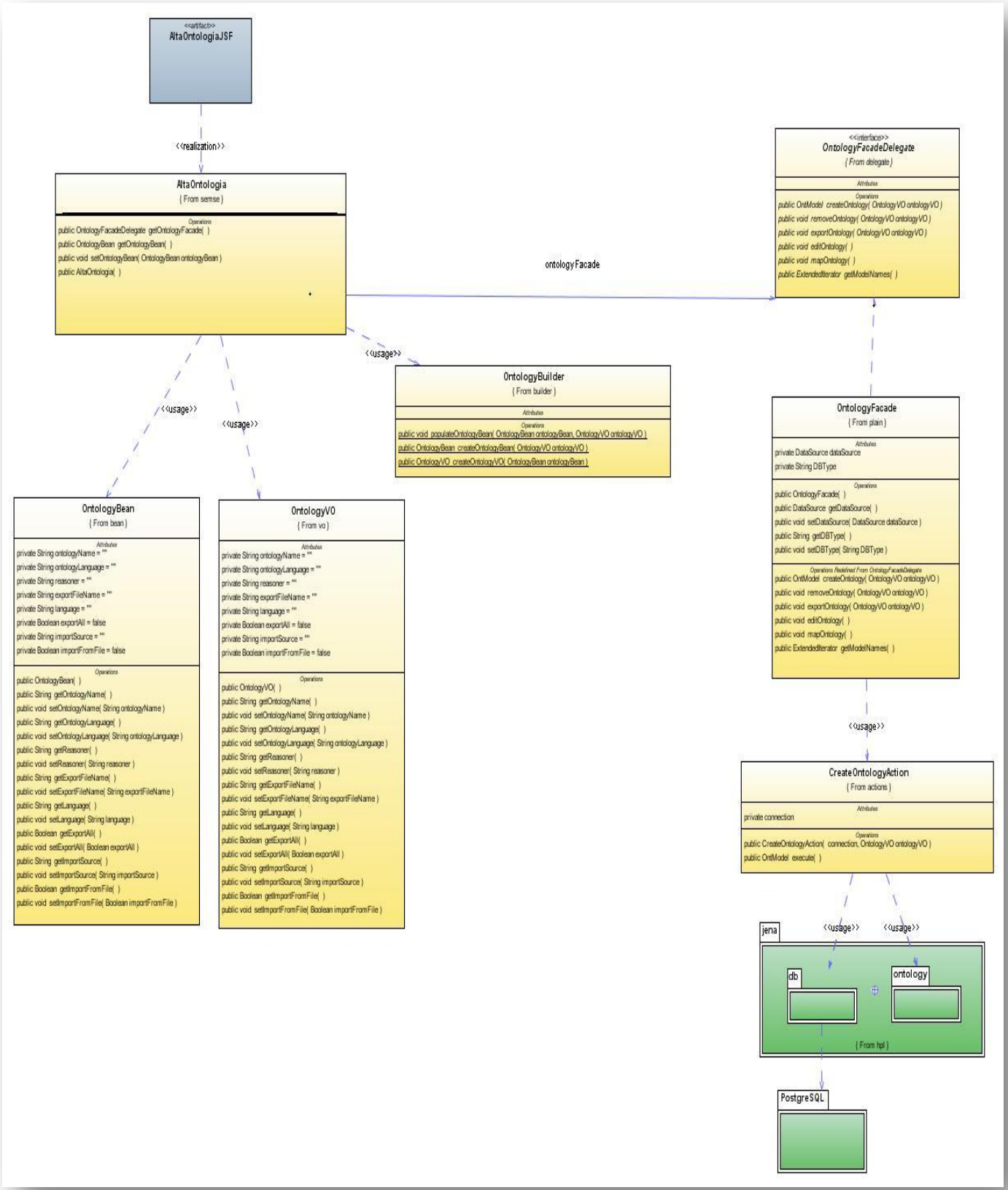

Figura 4.13. Diagrama de clases "Alta de Ontología"

Capítulo 5

Conclusiones

En este apartado se recogen las conclusiones resultantes de la confrontación entre lo deseado y lo conseguido. Para establecer el grado de éxito de este proyecto se recordarán los objetivos que se especificaron al inicio y se verá el grado de cumplimiento de cada uno:

- **Evaluar las tecnologías necesarias para el desarrollo óptimo del sistema**. Realizar un estudio de las tecnologías más punteras existentes en la actualidad y seleccionar aquellas que proporcionen la mejor opción para el desarrollo del sistema.
    - Se han evaluado los frameworks y tecnologías más utilizados a día de hoy y se ha realizado la selección de aquellos que permitirán llevar acabo con éxito la implementación del sistema. En el *apartado 2.4. Análisis de Tecnologías* se han resumido las tecnologías evaluadas, incluyéndose una descripción de las mismas.
- **Establecer el lenguaje de programación y el entorno de desarrollo.** Seleccionar el lenguaje de programación a utilizar, así como el entorno de trabajo necesario para desarrollar el sistema.
    - Se ha seleccionado Java 6 como lenguaje de programación y el entorno descrito en el *apartado 3.2. Entorno de trabajo* por su carácter libre e independiente de la plataforma.

- **Diseñar la arquitectura software del sistema.** Determinar los componentes que mejor satisfagan las necesidades del sistema, maximizando cualidades como adaptibilidad, seguridad, facilidad de uso y extensibilidad.
    - Se ha diseñado una arquitectura software capaz de satisfacer las necesidades del sistema mediante componentes de carácter libre. El diseño propuesto, mediante la aplicación de estilos arquitectónicos como Modelo 2 y Capas, y la aplicación de patrones de diseño como value object, IoC, factory o facade, maximiza la adaptabilidad (p.e. mediante el uso de la inversión de control es sencillo utilizar implementaciones alternativas a los beans declarados), la seguridad (al aprovecharse las facilidades proporcionadas por el framework Spring Security), la facilidad de uso (al haberse utilizado JSF y su capacidad para el desarrollo de interfaces amigables e intuitivas), extensibilidad (facilitada, p.e. al haberse utilizado patrones como action o el mismo framework de Spring).
- **Diseñar la arquitectura hardware del sistema.** Determinar el despliegue de componentes en máquinas o nodos durante la fase de desarrollo.
    - Se ha diseñado una arquitectura hardware que permite el despliegue de la infraestructura de desarrollo así como de los componentes que se desarrollen en la siguiente fase: implementación.
- **Probar la arquitectura diseñada.** Comprobar que el diseño arquitectónico elegido permite realizar con éxito el desarrollo del sistema. Para ello se realizará una prueba de concepto consistente en la implementación de una funcionalidad diferencial de la aplicación.
    - Se ha realizado una implementación de una parte diferencial de la funcionalidad del sistema para probar la validez de la arquitectura diseñada. Así, se ha podido comprobar cómo la arquitectura proporciona una diferenciación de roles en las clases implicadas,

potencia la independencia entre capas y facilita la realización de futuras ampliaciones/modificaciones sobre la funcionalidad. Como aspecto menos positivo podría destacarse la complejidad que implica el uso de la arquitectura propuesta frente a otras posibles alternativas más sencillas.

El grado de cumplimiento de estos objetivos puede considerarse satisfactorio puesto que con la realización de este trabajo se ha conseguido alcanzar el propósito general que se perseguía: diseñar una arquitectura óptima que permita la implementación del sistema informático, así como establecer el entorno de desarrollo necesario para llevar acabo dicha implementación.

Además, a través de este estudio se ha podido comprobar que el software libre es una buena apuesta para desarrollar grandes aplicaciones utilizando herramientas no comerciales y seguras, gracias a que, al tratarse de código abierto, los propios usuarios solucionan y notifican fallos sin coste con una respuesta más rápida a la ofertada por ciertos servicios de soporte.

5.1. Futuras líneas de desarrollo

Como primera propuesta para futuros trabajos estaría la finalización del desarrollo del sistema informático cuya arquitectura se plantea en este proyecto. El sistema permitiría la experimentación y refinamiento de los requisitos en su puesta en producción y ya está siendo desarrollo en otros PFC´s.

Otro aspecto importante a considerar sería la ampliación de la arquitectura hardware para que se adaptara a los volúmenes de datos y número de usuarios que se van a manejar en la puesta en producción del sistema informático. En esta propuesta se diseñó una arquitectura limitada a los recursos disponibles durante la fase de desarrollo.

También seria recomendable evaluar periódicamente el entorno de desarrollo para poder incorporar las mejoras provenientes de nuevas versiones.

Finalmente destacar que, al hacer uso de tecnologías punteras en el ámbito de la informática y la gestión de ontologías, de igual forma sería recomendable realizar evaluaciones periódicas del estado del arte en dichos ámbitos, similares a la realizada en este trabajo.

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 enero del 2009.

BALL, J. & BURNS, E. 2006. *Web Tier to Go With Java EE 5: Summary of New Features in JavaServer Faces 1.2 Technology*. Disponible en http://java.sun.com/javaee//reference/docs/index.html, consultado en enero del 2009.

BEGOLI, E. 2005. *Simplify Unit Testing for Spring Web Components*. Disponible en http://www.devx.com/Java/Article/30067, consultado en enero del 2009.

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.

BLANCO, M. 2008. *Almacenamiento persistente de ontologías con Jena y MySQL.* Disponible en http://marcosblanco.blogspot.com/2008/04/almacenamiento-persistente-de-ontologas.html, consultado en enero del 2009.

BOOCH, G. 2006. UML: el lenguaje unificado de modelado: guía del usuario. Editorial: Adisson Wesley.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H. & SOMMERLAD, P. 1996. *Pattern-Oriented Software Architecture.* Editorial: Wiley & Sons.

CARO, F. 2008. *Seminario Hibernate*. Disponible en http://paradigmatecnologico.pbwiki.com/, consultado en enero del 2009.

CARPI, F. 2008. *Struts 1 vs Struts 2*. Disponible en http://www.lifia.info.unlp.edu.ar/ es/difusion.htm, consultado en enero del 2009.

CASTEJÓN, J.S. 2004. *Arquitectura y diseño de sistemas web modernos*. Revista de Ingeniería Informática del CIIRM.

CAVANESS, C. 2005. *What is Struts*. Disponible en http://www.onjava.com/pub/a/onjava/2005/11/02/what-is-struts.html, consultado en enero del 2009.

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 enero del 2009.

CRAWFORD, W. & HUNTER, J. 1998. *JAVA TM Servlet Programming*. Editorial: O'Reilly.

CUEVAS AGUSTÍN, G. 2003. *Gestión del proceso software*. Editorial: Centro de Estudios Ramón Areces, S.A.

DELISLE, P. & BALL, J. 2006. *What´s New in JSP Tecnhology 2.1 (Web Tier to Go With Java EE 5: Summary of New Features in JSP 2.1 Technology)*. Disponible en http://java.sun.com/products/jsp/, consultado en enero del 2009.

DOJO, 2008. *Documentation*. Disponible en http://dojotoolkit.org/docs, consultado en noviembre del 2008.

EGUÍLUZ PÉREZ, J. 2008. *Introducción a JavaScript*. Disponible en http://www.librosweb.es/javascript/, consultado en enero del 2009.

ESA, 2008. *Estándar ESA*. Disponilble en http://www.fabricadesoftware.cl/fabrica_documentos.php, consultado en enero del 2009.

ESCARTÍN VIGO, J. A. 2005. *Servidor Linux para conexiones seguras de una LAN a Internet*. Universitat Politècnica de Catalunya. Facultat d'Informàtica de Barcelona. Disponible en https://upcommons.upc.edu/pfc/handle/2099.1/3451, consultado en enero del 2009.

FRAMEWORK, 2008. *Framework - Wikipedia, la enciclopedia libre*. Disponible en http://es.wikipedia.org/wiki/Framework, consultado en enero del 2009.

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 enero del 2009.

GAMMA, E.; HELM, R.; JOHNSON, R. & VLISSIDES, J. 1998. *Design Patterns, Elements ofReusable Object-Oriented Software*. Editorial: Adisson Wesley.

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.

HIBERNATE, 2008. *Relational Persistence for Java and .Net*. Disponible en http://www.hibernate.org/, consultado en enero del 2009.

HONRUBIA LÓPEZ, F. J. 2002. *Introducción a las Ontologías*. Escuela Universitaria de Albacete. Disponible en http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Ontologias.pdf, consultado en enero del 2009.

JASPER, R. & USCHOLD, M. 1999. *A Framework for Understanding and Classifying Ontology Applications*. Editorial: CEUR Publications and University of Amsterdam. Vol. 18.

JENA, 2008. *The Jena Ontology API*. Disponible en http://jena.sourceforge.net/documentation.html, consultado en enero del 2009.

JENA FRAMEWORK, 2008. *A Semantic Web Framework for Java.* Disponible en http://jena.sourceforge.net/, consultado en enero del 2009.

JENDROCK, E.; BALL, J. & OTHERS. 2008. *The Java EE 5 tutorial*. Disponible en http://java.sun.com/javaee/5/docs/tutorial/doc/, consultado en enero del 2009.

JOHNSON, R. 2002. *Expert One-on-One J2EE Design and Development*. Editorial: Wrox Press.

JOHNSON, R. 2007. *Introduction to Spring Framework 2.5*. Disponible en http://www.theserverside.com/tt/articles/article.tss?l=IntrotoSpring25, consultado en enero del 2009.

LOZANO TELLO, 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 enero del 2009.

MAEDCHE, A. 2002 (University of Karlsruhe, Germany). *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 enero del 2009.

MICHAEL C. DACONTA, LEO, J. OBRST, & KEVIN T. SMITH, 2003. The semantic Web. A guide to the future of XML, Web services, and knowledge management. Editorial: Wiley.

MySQL, 2008. *MySQL 5.1 Reference Manual*. Disponible en http://dev.mysql.com/doc/, consultado en enero del 2009.

NAUGHTON, P. & SCHILDT, H. 1997. *Java. Manual de referencia*. Editorial: McGraw-Hill.

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.

NETBEANS, 2008. *NetBeans History*. Disponible en http://www.netbeans.org/about/history.html, consultado en enero del 2009.

NETBEANS 6.5, 2008. *NetBeans IDE 6.5 Release Candidate Information*. Disponible en http://www.netbeans.org/community/releases/65/, consultado en enero del 2009.

NIELSEN, J. 1994. *Usability Inspection Methods*. Editorial: Wiley.

OGDEN, C. K. & RICHARDS, I. A. 1923. Meaning of meaning. Editorial: New York: Harcourt & Brace.

PATRONES, 2008. *Core J2EE Patterns* .Disponible en http://java.sun.com/blueprints/corej2eepatterns/Patterns/, consultado en enero del 2009.

PGADMIN, 2008. *pgAdmin PostgreSQL tools*. Disponible en http://www.pgadmin.org/, consultado en enero del 2009.

POSTGRESQL, 2008. *About*. Disponible en http://www.postgresql.org/about/, consultado en enero del 2009.

POSTGRESQL 8.3, 2008. *PostgreSQL 8.3 Press Kit*. Disponible en http://www.postgresql.org/about/press/, consultado en enero del 2009.

PROTOTYPE, 2008. *Prototype API Documentation*. Disponible en http://www.prototypejs.org/, consultado en enero del 2009.

RIO, 2008. *Rio*. Disponible en http://www.openrdf.org/about.jsp, consultado en enero del 2009.

RODRIGUEZ BULEO, J.J. 2007. *PFC- Sistema de Gestión de Repositorios de Metadatos Educativos del proyecto DOTEINE.*

SCRIPTACULOUS, 2008. *Docs & Wiki.* Disponible en http://script.aculo.us/, consultado en enero del 2009.

SEMSE, 2008. *SEMSE: SEmantic Metadata Search*. Disponible en http://163.117.147.101:9000/SEMSE/index.php, consultado en enero del 2009.

SERVIDOR WEB, 2008. *Servidor Web - Wikipedia, la enciclopedia libre*. Disponible en http://es.wikipedia.org/wiki/Framework, consultado en enero del 2009.

SESAME, 2008. *Sesame 2 User Documentation*. Disponible en http://www.openrdf.org/documentation.jsp, consultado en enero del 2009.

SPRING, 2008. *About Spring*. Disponible en http://www.springframework.org/about, consultado en enero del 2009..

SPRING SECURITY, 2007. *Spring Security Reference Documentation 2.0.x*. Disponible en http://static.springframework.org/spring-security/site/reference.html, consultado en enero 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.

TOMCAT, 2008. *Apache Tomcat*. Disponible en http://tomcat.apache.org, consultado en enero del 2009.

UBUNTU, 2008. *About Ubuntu*. Disponible en http://www.ubuntulinux.org/, consultado en enero del 2009.

UML 2.0, 2008. The Current Official UML Specification. Disponible en http://www.uml.org/#UML2.0, consultado en enero del 2009.

WINSTON, P. 2006. *Understanding Netbeans Visual JSF Web Application.* Disponible en http://www.winstonprakash.com/articles/javaserverfacesnetbeans/understanding_visual_jsf_app.html, consultado en enero del 2009.

WALLS, C. 2008. *Spring in Action*. Editorial: Manning.