UNIVERSIDAD CATÓLICA ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Desarrollo de una ontología y un buscador utilizando tecnologías de Web Semántica

TRABAJO ESPECIAL DE GRADO
presentado ante la
UNIVERSIDAD CATOLICA ANDRES BELLO
como parte de los requisitos para optar al título de
INGENIERO EN INFORMÁTICA

REALIZADO POR **Israel Fermín Montilla**

PROFESOR GUIA **Ph.D: Wilmer Pereira**

FECHA Caracas, septiembre de 2011

DEDICATORIA

A mi abuelo, se que donde estés, estás orgulloso de mi...

AGRADECIMIENTOS

Me gustaría agradecer, en primer lugar, a ese ser superior al que algunos llaman Dios, y otros, simplemente, no sabemos cómo llamarlo. A mi familia, que siempre me ha apoyado en mis metas y mis locuras. A mi papá y a mi mamá por estar siempre pendientes de mi formación académica y personal. A Giselle por todo su cariño y su apoyo incondicional, por siempre creer en mi y por hacer que mi último semestre fuera inolvidable y el mejor de toda la carrera, tú haces que todo valga la pena. A mi tutor, el Prof. Wilmer Pereira, y a mi amigo el Ing. Carlos Pérez, por su apoyo incondicional durante la realización de este trabajo a nivel práctico y a nivel conceptual. Sin ustedes, nada de esto habría sido posible.

También me gustaría agradecer a quienes me acompañaron semestre a semestre, mis amigos Gerardo Barcia, Jonathan Trujillo, Khaterine Castellano, Viviana Trujillo, Ronald Oribio y Freddy Florenca. A los profesores Carlos Barroeta, Lúcia Cardoso, Rodolfo Campos y Darío León, sin sus enseñanzas me sentiría inseguro trabajando en la calle en cualquier área de la carrera. Al Prof. Ricardo Casanova por brindarme su amistad durante los últimos semestres y a mis alumnos de preparaduría, especialmente a Marcos Ackerman, Aileen Posadas, Héctor Sam y Juan Francisco Perozo, fue más lo que ustedes me dejaron a mi que lo que yo dejé en ustedes.

A todos quienes, de alguna u otra manera me ayudaron, Muchas Gracias.

Israel Fermín Montilla

SINOPSIS

Toda la paja en una página.

Introducción

Copiar y pegar lo mismo de la propuesta

CAPÍTULO 1

Problema

1.1. Planteamiento del Problema

La World Wide Web (WWW), ha cambiado radicalmente la manera como las personas se comunican entre sí, la forma como la información se distribuye y como se diseminan los mensajes y los modelos de negocio de muchas empresas alrededor del mundo (Antoniou y Van Harmelen, 2003). Basta con revisar las páginas web de las grandes empresas a nivel nacional e internacional, como Polar, Apple Computer o Microsoft para darse cuenta de que ya no se enfocan sólo en tener presencia en radio y televisión, sino también en las distintas redes sociales que han surgido a través de la Web 2.0.

Ciertamente Internet, desde su aparición en 1975, ha evolucionado, ya los sitios web no son páginas con texto estático e imágenes en las que sólo un subconjunto de los usuarios es capaz de publicar contenido, mientras que otros sólo leen y reciben información. Ahora se cuenta con páginas y aplicaciones web complejas, en las que todos los usuarios están en capacidad de generar y publicar contenido. Según Pardo y Cobo (2007), esto se conoce como

Conocimiento Colectivo, de esta manera, con cada vez más usuarios publicando contenido en la web, la cantidad de información ha crecido exponencialmente, ya para 2009, el estimado de usuarios de Internet en América Latina y el Caribe era de 175,8 millones de personas aproximadamente, según estadísticas de *Éxito Explorador*. Lo que implica un enorme volumen y tráfico de datos en la WWW pues todos esos usuarios intercambian, publican y consultan información permanentemente.

Todos estos usuarios poseen necesidades de información esperando a ser satisfechas. La información para satisfacer a todos y cada uno de los usuarios, está disponible en línea, pero resulta difícil acceder a ella si no se sabe dónde se encuentra, es decir, si no se conoce su URL. Es por ello que fueron creados los buscadores como Yahoo, Google, Altavista y Bing, por mencionar algunos de los más populares todos estos funcionan buscando las palabras claves que proporciona el usuario en los documentos que ha indexado, es decir, son buscadores basados en palabras clave. De esta manera, si el usuario introduce, por ejemplo, "Internet", el buscador retornará todos los documentos conocidos que contienen esa palabra, bien sea en el título o en el cuerpo del texto. Pero cuando la búsqueda involucra más de una palabra, las cosas pueden complicarse un poco, por ejemplo: si el usuario introduce "Internet de Venezuela" los buscadores basados en palabras clave omiten los conectores pues aparecen en todos los documentos y sitios web, por lo que la búsqueda sería "Internet Venezuela" y el resultado de la búsqueda contendría todos los recursos en los que aparezcan ambas palabras o alguna de las dos.

Pero qué ocurre cuando el usuario desea realizar búsquedas más especializadas, por ejemplo "Aerolíneas que viajan a Valera", probablemente el buscador nos de lo que estamos buscando, pero requiere un trabajo adicional de búsqueda por parte del usuario pues los resultados obtenidos contendrían páginas de "Aerolíneas", páginas que contienen la palabra "viajan", y noticias sobre "Valera", localidad del Estado Trujillo, o sobre personas con el apellido "Valera". Esta resulta ser la principal desventaja de los buscadores actuales basados en palabras clave o "key words", cuando se desea hacer una búsqueda basada en un tópico específico o enmarcada en un contexto dado, los resultados de la búsqueda no son muy cer-

canos a lo que el usuario desea buscar pues los buscadores no son capaces de interpretar el significado detrás de las palabras clave que componen la consulta realizada.

Casos como el anteriormente expuesto se ven a diario entre los estudiantes de carreras afines a las Ciencias de la Computación, como lo es la Ingeniería Informática. Muchas veces se trata de buscar información que aparece en la web bajo otro nombre o, por ejemplo, al buscar conocimientos complejos como "cálculo de rutas óptimas", se consiguen resultados muy avanzados para el nivel de experiencia que se tiene en esa área, y no se desglosa los resultados en tópicos que son necesarios para dominar el objeto de la búsqueda, por ejemplo, resultados sobre el manejo de matrices de adyacencia y resultados sobre algoritmos ya existentes que tienen dicha utilidad como Dijkstra y Bellman-Ford. De esta manera, el estudiante puede tener una guía para poder atacar el tópico de su interés.

Por todo lo dicho con anterioridad, se plantea el desarrollo de una aplicación con capacidad semántica que sirva como herramienta de consulta en materia de Ciencias de la Computación e Ingeniería Informática.

Resulta de interés resaltar que todo lo relativo a la Web Semántica y sus estándares y tecnologías, se encuentra actualmente en definición y en proceso de desarrollo, por ello, se debe tener presente que la Web Semántica se encuentra en un nivel experimental, de igual manera que el desarrollo de este trabajo.

1.2. Objetivo General

- Desarrollar una ontología y un prototipo de buscador sobre dicha ontología utilizando tecnologías y estándares de Web Semántica.

1.3. Objetivos Específicos

1. Construir una Base de Conocimientos.
2. Evaluar de manera cualitativa distintos Motores de Inferencia y seleccionar el que mejor se adapte a las necesidades del proyecto.
3. Definir las reglas de inferencia a ser aplicadas sobre el vocabulario creado.
4. Construir una infraestructura básica de búsqueda para realizar las pruebas necesarias sobre la ontología.
5. Desarrollar las interfaces necesarias para realizar consultas y mostrar los resultados.
6. Implementar un procedimiento semiautomático para la evolución de la ontología.

1.4. Alcance

Desarrollar una ontología y un buscador con capacidad semántica para realizar búsquedas utilizando ontología desarrollada.

1.5. Limitaciones

- La ontología se realizará únicamente sobre tópicos referentes al área de Ciencias de la Computación e Ingeniería en Informática.

- Debido a la enorme cantidad de temas existentes dentro del área de Ciencias de la Computación e Ingeniería Informática, se cubrirá el dominio de temas en extensión.
- En general, se profundizará a tres (3) niveles en todos los temas cubiertos.
- Dado que el vocabulario a ser utilizado no es estándar sino que es producto del presente trabajo, la realización de búsquedas masivas en Internet no será un tema pertinente a este trabajo.

1.6. Justificación

En la última década (2000 – 2010), según estadísticas de *Éxito Explorador*, la cantidad de usuarios de Internet a nivel mundial se ha incrementado en un 446 %, esto da una idea acerca de la importancia que tiene la World Wide Web como herramienta de comunicación, búsqueda e intercambio de información entre las personas en la actualidad. La WWW, ha cambiado la manera como las personas interactúan entre si, extendiendo la forma también como se presta servicio de educación, mediante plataformas de e-learning. La web ha evolucionado y se ha vuelto tan confiable que en la última década la tendencia de muchas grandes empresas ha sido "subir todo a la nube", es decir, aprovechar plataformas de Cloud Computing para tener sus servicios accesibles en cualquier momento y en cualquier parte del mundo.

Dada la importancia de la WWW a nivel mundial, día a día se ven iniciativas para hacer más placentera la experiencia de los usuarios en la red, así como nuevas tecnologías y estándares para optimizar y mejorar su funcionamiento, al punto que gracias a tecnologías como el Really Simple Syndication (RSS), no es necesario visitar constantemente un sitio web para leer sus actualizaciones u obtener el último episodio de nuestro Podcast favorito. Todo esto no sólo le hace la vida más fácil al usuario, sino que optimiza el funcionamiento de la web.

La Web Semántica es una iniciativa del World Wide Web Consorcium (W3C) que pretende, según lo establecido por el W3C, "extender la web, dotándola con un mayor significado para que cualquier usuario pueda conseguir respuesta a sus preguntas en Internet de manera rápida y sencilla". La Web Semántica viene a resolver el problema de acceso a la información en internet, ya que al añadirle semántica a la Internet, se puede buscar, transferir y compartir información de una manera más sencilla, permitiéndole al usuario, delegar estas tareas en software de manera confiable.

Por todo lo anteriormente mencionado, cualquier proyecto o investigación destinado al mejoramiento y optimización de la WWW y, más aún, dentro del campo de la Web Semántica, tiene pertinencia en el presente, ya que es un tema novedoso y que es objeto de in-

vestigación en muchas Universidades importantes alrededor del mundo. Desde el año 2001, cuando Tim Berners-Lee escribió un artículo para la revista Scientific American describiendo las posibilidades de la Web Semántica, muchos grupos de investigadores han tomado este tema como tópico de investigación. Actualmente, en Venezuela, se llevan a cabo numerosas investigaciones en este campo, siendo la Universidad Simón Bolívar (USB) la que presenta mayor actividad, teniendo un grupo de investigadores dedicados a este tema, entre los que destacan la Prof. María Esther Vidal y la Prof. Soraya Abad-Mota, ambas especialistas y ponentes internacionales en Web Semántica.

En la Universidad Católica Andrés Bello (UCAB), no existe ninguna investigación en esta área, lo que da aún más pertinencia a un Trabajo de Grado en esta área, pues permitiría a la UCAB incursionar en este campo, contribuyendo al desarrollo de una tecnología de software nueva y a nivel internacional y abriría las puertas a una nueva línea de investigación dentro de la Escuela de Ingeniería Informática de la UCAB, permitiendo la realización de nuevos trabajos más avanzados acerca de este tema.

Para este proyecto, no sólo será necesario implementar estándares ya creados pues, como se explicará más adelante, casi la totalidad de los estándares y tecnologías que construyen la Web Semántica, aún cuando ya están siendo utilizados, se encuentran en desarrollo todavía. Será necesario también elaborar un procedimiento de traducción de la consulta del usuario a lenguaje de consultas sobre RDF (SPARQL), este vendría siendo, si se quiere, el elemento de mayor dificultad para lograr el buen funcionamiento y desempeño del prototipo de buscador que se desea desarrollar. Además, el desarrollo de un proceso de inclusión de nuevos conceptos a la ontología añade un factor más de dificultad que hace que sean necesarias las habilidades y conocimientos de un Ingeniero en Informática para resolver problemas que no sólo tienen que ver con programación o codificación de algoritmos (que es una de las capacidades del Ingeniero en Informática), sino con la definición y construcción de los mismos para resolver un problema computacional específico de manera eficaz y eficiente.

Por otra parte, el desarrollo de una ontología acerca en materia de las Ciencias de la Computación e Ingeniería Informática, permitiría organizar el conocimiento disponible en

esos campos y, además, disponer de un repositorio dónde consultar dicho conocimiento, con el valor agregado de que las búsquedas se realizarían con sentido semántico, lo que da una mayor seguridad al usuario de que los resultados del buscador tienen coherencia con la consulta realizada y la información deseada. El producto de este trabajo, podría implantarse como una herramienta de consulta electrónica para los estudiantes de carreras afines a la Ingeniería Informática en la UCAB. Posteriormente, la herramienta, podría ser extendida a otros temas mediante otros trabajos similares e incluso, podría portarse a otras carreras o a otras universidades.

CAPÍTULO 2

Marco Teórico

Para poder adentrarnos en el tema de la Web Semántica, resulta necesario estudiar y comprender los antecedentes y los eventos que poco a poco han ido llevando a la evolución de la Internet a la herramienta que tenemos hoy día y que, poco a poco también, irán llevando la Web actual, la Web 2.0, a su evolución natural: la Web 3.0, es decir, la Web Semántica.

Si actualmente nos encontramos en la llamada Web 2.0, tuvo que, en algún momento, existir una Web 1.0. Esta primera versión, por así llamarla, sólo contaba con portales que únicamente exponían contenido. La Web 1.0 estaba destinada a la publicación de contenidos corporativos, no daba la posibilidad de participación abierta a los usuarios, no existían espacios para la publicación de contenido por parte de los usuarios y, estos usuarios, eran importantes en tanto fueran consumidores, estas, según Ricardo Casanova (2010), eran algunas de las características principales de la Web 1.0 que, además, resultan ser grandes limitaciones. En la Web 1.0, únicamente personas especializadas eran capaces de crear contenido, por ello, sólo las grandes empresas podían disponer de un espacio en la red, el resto de los usuarios únicamente podían recibir (consumir) el mensaje o el contenido que era publicado, sin la

posibilidad de participar en la generación y actualización del mismo.

El término Web 2.0 aparece a mediados del año 2004, y fue creciendo paulatinamente hasta convertirse en portada de los principales seminarios y congresos en la navidad de 2006. Según O'Reilly (citado por Pardo y Cobo, 2007), principal promotor de la Web 2.0, algunos de sus principios básicos son: La web como plataforma, el aprovechamiento de la Inteligencia Colectiva, la gestión de Bases de Datos como competencia básica, el software no limitado a un solo dispositivo y brindar experiencias enriquecedoras al usuario. Todo esto quiere decir que ahora la interacción de los usuarios es bidireccional, aunque sigue siendo un subgrupo técnico de estos usuarios el que crea los portales, la gran diferencia es que ahora todos los usuarios son capaces de generar contenido. El usuario ya no es un simple consumidor, sino que pasa a ser consumidor-generador de contenido en la web. Pasamos de una red pasiva, de páginas estáticas, a una red activa de páginas dinámicas que interactúan con bases de datos para almacenar y actualizar su contenido que, a su vez, es generado por los usuarios que hacen vida dentro del portal.

La evolución es un factor constante en todas las áreas y tecnologías de las Ciencias de la Computación y la Ingeniería Informática, Internet no es la excepción a esta regla, por lo que es fácil deducir que la Web 2.0 no es más que el estado actual y transitorio de esta tecnología que, eventualmente, evolucionará a una Web 3.0, cuyas características están siendo definidas por el W3C bajo el marco de la Web Semántica.

2.1. La Web Semántica

El W3C (S/F) define a la Web Semántica como "una Web extendida, dotada de mayor significado en la que cualquier usuario de Internet, podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a que la información está mejor definida". Al dotar a la Web de un mayor significado y, por consiguiente, de mayor semántica, es posible obtener solución a problemas comunes de búsqueda de información, gracias a la implementación de

una infraestructura y lenguajes comunes de búsqueda, destinados a resolver dichos problemas, realizando dichas búsquedas tomando en cuenta el contexto y el significado real de la consulta.

Una vez definida la Web Semántica a nivel conceptual, conviene examinar brevemente cómo funciona. Supongamos que la Web tiene la capacidad de construir una base de conocimiento sobre las preferencias de los usuarios y que, a través de una combinación entre su conocimiento y la información disponible en la Internet, sea capaz de atender de forma exacta las demandas de información por parte de los usuarios, por ejemplo, reserva de hoteles, vuelos, médicos o libros.

Si esto ocurriese en la vida real, el usuario, al intentar encontrar "todos los vuelos a Praga para mañana por la mañana", obtendría resultados exactos sobre su búsqueda. Desafortunadamente la realidad es otra, los buscadores actuales mostrarían resultados acerca de "Praga" como localidad turística, noticias de sucesos ocurridos en "Praga", quizás páginas de periódicos locales, foros sobre "Praga", en resumen, sitios que contienen las palabras que conforman la consulta hecha por el usuario. Estos resultados son inexactos y, por sí solos, no satisfacen las necesidades de información del usuario, es necesario un segundo filtro, examinar uno a uno los resultados y extraer manualmente la información que resulte interesante. Por otro lado, un buscador con capacidad semántica, mostraría información más exacta a lo que se desea obtener. La ubicación geográfica desde la cual se envía la consulta sería detectada de manera automática, sin necesidad de indicar el punto de partida, además, elementos de la oración como "mañana" adquirirían significado, siendo convertidos en un día concreto calculado en función de un "hoy". De igual manera ocurriría con el segundo "mañana", el cual sería interpretado como un momento determinado del día. Todo ello a través de una Web en la que los datos, dejan de ser sólo datos y pasan a ser información llena de significado.

La manera como se procesará la información no sólo será en términos de entrada y salida de parámetros de búsqueda, sino en términos de su semántica, apoyándose en una infraestructura basada en meta-datos. Vale acotar, que no se trata de Inteligencia Artificial, sino de dar a las máquinas la capacidad de resolver problemas bien definidos, a través de operaciones bien definidas, ejecutadas sobre datos existentes bien definidos a través de meta-datos.

Dentro de la Web Semántica convergen una serie de estándares y tecnologías, muchas de ellas aún en proceso de definición y desarrollo por parte de investigadores alrededor del mundo y el W3C, las cuales se explican brevemente a continuación:

2.2. Ontologías

Según el W3C (S/F), "una Ontología define los términos utilizados para describir y representar conocimiento en un área". Las Ontologías son utilizadas por personas y aplicaciones que necesitan compartir conocimiento acerca de un área específica, es por ello que las ontologías son relativas a un tópico o área en especial, además, incluyen información acerca de los conceptos básicos y las relaciones entre ellos, esta información puede ser utilizada por las computadoras. De esta manera, una Ontología codifica el conocimiento de un área y lo hace accesible y reutilizable.

Complementando al W3C, Russel y Norvig (2004), definen una Ontología como una representación abstracta de un "algo" en el mundo real, definiéndolo junto con sus relaciones a otros entes, dejando moldes definidos para que nuevo conocimiento pueda ser incorporado posteriormente.

La palabra Ontología, se ha utilizado a lo largo de la historia para definir elementos con distintos grados de estructura: desde taxonomías simples hasta teorías lógicas complejas. La Web Semántica necesita ontologías con cierto grado de estructura y significado para poder especificar descripciones para los siguientes conceptos:

- Clases y dominios de interés.
- Las relaciones que pueden existir entre las clases descritas.
- Las propiedades o atributos que pueden tener las clases descritas.

Las Ontologías son escritas en lenguajes basados en lógica, de esta manera se tiene la garantía de que son precisas, detalladas, consistentes y significativas (W3C, S/F). Muchas herramientas para Ontologías pueden realizar procesos de razonamiento sobre el conocimiento que definen, de esta manera, se puede tener cierta "inteligencia" y se pueden desarrollar aplicaciones con capacidades complejas como consulta de información de manera semántica y conceptual, soporte a la toma de decisiones, gestión de conocimiento, bases de datos inteligentes, comercio electrónico y entendimiento del lenguaje natural.

En la Web Semántica, resulta necesario el uso de Ontologías ya que proporciona una manera sencilla y eficiente de representar la semántica de los recursos descritos y permitir que sean utilizadas y consultadas por aplicaciones y agentes de software. Para el W3C, el uso de ontologías, permitirá a las aplicaciones del futuro actuar de manera "inteligente" para cumplir con su trabajo de una manera más rápida y con mayor exactitud.

2.3. RDF

Sus siglas significan "Resource Description Framework", y es "un lenguaje para la descripción de recursos disponibles en la World Wide Web" (W3C, 2004). Es un modelo estándar para el intercambio de datos en la Web, RDF extiende la estructura de enlaces de la Web utilizando URIs para dar nombre a las cosas, es así como se establecen enlaces entre dos extremos: el sujeto y el objeto, a través de una relación o propiedad definida a través de RDF. Normalmente, a esta manera de expresar relaciones, se le conoce como "triples". Este modelo de relaciones permite que los datos estructurados o semi-estructurados sean mezclados, expuestos y compartidos a través de diversas aplicaciones en distintas plataformas.

RDF se utiliza para declarar las distintas clases y las relaciones existentes entre ellas. Toda la estructura descrita en un RDF forma una estructura de grafo, donde los arcos representan las relaciones entre dos recursos, los cuales son representados por los nodos de dicho grafo (W3C, S/F). Este modelo mental, es la manera más utilizada para lograr

explicaciones visuales y fáciles de comprender.

La sintaxis de RDF, está basada en XML, al igual que muchos otros metalenguajes y lenguajes de marcado, como HTML y XHTML. De esta manera, es posible además definir atributos adicionales a los recursos que son descritos en el documento.

2.4. RDFS

Siendo RDF un lenguaje basado en notación XML, pareciera lógico que siga más o menos el mismo esquema. Si recordamos, y si somos estrictos, cada XML debería tener un XML Schema (o, en su defecto, un DTD, pero estos están siendo poco a poco desplazados por los XMLS), que resulta ser otro archivo XML, que lo define. RFDS significa "RDF Schema" y es "un lenguaje para la descripción de vocabularios en la Web" (GSI, S/F). Es una extensión semántica de RDF para describir vocabulario, RDFS no busca definirlos, sino describirlos, proporcionar la facilidades para describir tipos y clases de un mismo dominio y servir como sistema de tipos para RDF, es decir, RDF define las clases y, a través de RDFS se establece la jerarquía de clases.

Un RDF Schema, se escribe utilizando las mismas reglas de RDF, es decir, el Esquema RDF es un documento RDF que, a su vez, es capaz de definir a otros RDF. El RDFS, extiende al RDF tradicional para poder implementar jerarquías de clases y relaciones un poco más complejas que las que RDF permite definir por sí solo.

RDFS, al igual que RDF, aún se encuentran en proceso de definición y desarrollo, aunque ya el trabajo está bien adelantado y se puede trabajar y desarrollar con ambos. Un ejemplo de esto es el proyecto FOAF o "Friend of a Friend", que es un proyecto de la Web Semántica para describir personas y las relaciones entre ellas utilizando documentos RDF.

2.5. OWL

El W3C (2005), define OWL como "un Lenguaje de Ontologías Web". Existen muchos lenguajes y herramientas para desarrollar y trabajar con Ontologías, pero ninguna de las existentes hasta el momento resultaba compatible con una arquitectura Web y mucho menos con la Web Semántica.

El Lenguaje de Ontologías Web, rectifica esto aprovechando una conexión proporcionada por RDF para dar a las Ontologías las siguientes capacidades (W3C, 2005):

- Capacidad de ser distribuidas y compartidas a través de varios sistemas.
- Escalable a las necesidades de la Web.
- Compatible con los estándares internacionales de accesibilidad Web e Internacionalización (I18N).
- Abierto y extensible.

Adicionalmente, OWL es una extensión de RDF Schema para permitir la expresión de relaciones más complejas entre elementos y clases. Algunos de los recursos que tiene OWL y de los que, según el W3C (2005), carece RDFS son las siguientes:

- Recursos para inferir cuáles elementos que tienen varias propiedades son miembros de una clase en particular.
- Recursos para determinar si la totalidad de elementos de una clase tendrán una propiedad determinada o si puede ser que sólo algunos elementos la tengan.
- Recursos para diferenciar relaciones 1:1, 1:N y N:1, permitiendo que las "claves foráneas" de las bases de datos puedan ser representadas en la Ontología.
- Recursos para expresar relaciones entre clases definidas en documentos diferentes a través de la Web.

- Recursos para definir nuevas clases a partir de uniones, intersecciones y complementos de otras ya existentes.
- Recursos para restringir rangos y dominios para especificar combinaciones de clases y propiedades.

Siendo OWL un lenguaje tan extenso, resulta costoso explotarlo a su máxima expresión en aplicaciones web, ya que resulta muy difícil para el motor de inferencia tomar una decisión rápida respecto a la consulta que se le está realizando, es por ello que OWL se divide en tres (3) subconjuntos, según la extensión y el nivel de detalle y profundidad que se desee dar a la ontología, sin llegar a un modelo de datos en el que la capacidad de decición del razonador no se complique demasiado ni, mucho menos, llegue a ser indecidible. Los subconjuntos de OWL, según Antoniou y Van Harmelen (2003), son los siguientes:

OWL Lite

Es el subconjunto más restrictivo de OWL, pero, aún así, es el más utilizado para la Web Semántica pues incluye los constructores básicos y necesarios para escribir modelos cuya decibilidad no sea muy compleja.

OWL DL

Es un subconjunto de OWL un poco más amplio y muy popular en el ámbito de la Lógica Descriptiva (Descriptive Logic), sigue garantizando eficiencia computacional en la inferencia sobre modelos que requieren una mayor expresividad y descripción de sus conceptos, restringiendo el uso de algunos constructores de OWL.

OWL Full

Es el conjunto completo del lenguaje OWL, sin ningún tipo de restricción de constructores o utilización de los mismos, la ventaja de OWL Full es que es lo más flexible que existe para el modelado de ontologías para Web Semántica, la desventaja es que el nivel de expresividad del lenguaje es tan rico y poderoso que afecta la decibilidad del razonador.

OWL, se encuentra aún en proceso de desarrollo y de definición como estándar, pero el trabajo se encuentra bien adelantado y es posible desarrollar aplicaciones con lo que existe actualmente. Existen numerosas herramientas que permiten modelar y trabajar con RDF, RDFS y OWL, siendo el más utilizado el editor de ontologías Protegé (desarrollado por la universidad de Stanford), permite editar ontologías en OWL de manera gráfica y, además, tiene varios razonadores (motores de inferencia) integrados para validar la integridad de la ontoligía que se está escribiendo.

2.6. SPARQL

En la Web Semántica, toda la información acerca de los recursos se encuentra modelada en Ontologías definidas mediante RDF, RDFS y OWL (en cualquiera de sus tres dialectos). SPARQL, "es un lenguaje de consultas para RDF" (W3C, S/F), es decir, así como en el modelo relacional, pueden consultarse las bases de datos a través de SQL y, de esta manera, obtener subconjuntos de la información almacenada según un conjunto de restricciones, mediante consultas SPARQL puede obtenerse subconjuntos de la información contenida en uno o varios documentos RDF en forma de grafos, dependiendo de las relaciones y de lo que el usuario desee buscar.

SPARQL, viene a ser la evolución de RQL y aún se encuentra en proceso de desarrollo, aunque existen numerosos motores de consulta en SPARQL y se puede trabajar sobre lo que ya existe.

2.7. Motores de Inferencia

Una vez que se tiene la Ontología ya definida, es necesario definir ciertas reglas de inferencia a ser aplicadas sobre ese conocimiento ya descrito. Estas reglas de inferencia son aplicadas por un Motor de Inferencia, que es un programa que explora la base de conocimiento, aplicando ciertas reglas, hasta dar con la solución que mejor se adapte a las necesidades del usuario (Díaz, S/F).

En la Web Semántica, es el motor de inferencia quien traduce la consulta del usuario y evalúa, mediante reglas bien definidas, qué es lo que el usuario realmente está buscando, es por ello que el motor de inferencia juega un papel de suma importancia en este marco, vale acotar, que los motores de inferencia para Web Semántica, deben ser capaces de "razonar" sobre RDF, RDFS y OWL.

CAPÍTULO 3

Marco Metodológico

Las metodologías de desarrollo tradicionales, parecieran haber sido diseñadas para proyectos extensos, con equipos numerosos, en los que se goza de roles bien definidos y que, cada rol, cumple con una labor específica en cada etapa del ciclo de vida.

La realidad de este proyecto es otra, sólo se cuenta con una persona para realizar el trabajo de todos los roles, además, el tiempo de entrega es limitado y resulta, además, conveniente entregarlo lo antes posible. Es por ello que se propone trabajar con una metodología Iterativa Incremental bajo el esquema ágil. Los preceptos del esquema de trabajo enmarcado en el Desarrollo Ágil, fueron definidos por Kent Beck año 2001 en un documento denominado El Manifiesto Ágil (The Agile Manifesto), este manifiesto se cita a continuación:

"Estamos descubriendo nuevas maneras de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de esta experiencia hemos aprendido a valorar:

- **Individuos e interacciones** sobre procesos y herramientas.

- **Software que funciona** sobre documentación exhaustiva.
- **Colaboración con el cliente** sobre negociación de contratos.
- **Responder ante el cambio** sobre el seguimiento de un plan.

Esto es, aunque los elementos a la derecha tienen valor, nosotros valoramos por encima de ellos a los que están a la izquierda".

El Trabajo Especial de Grado, tiene un carácter investigativo y aplicativo, es decir, debe investigarse acerca del tema y realizar un pequeño aporte al área de investigación mediante el desarrollo de algún producto final. Por ello, dado el carácter de un TEG, todos y cada uno de los principios definidos en el Manifiesto Ágil de Kent Beck tienen sentido en un proyecto de este tipo pues, debe valorarse "Individuos e interacciones sobre procesos y herramientas", esto es, debe valorarse más la interacción entre el tesista y el tutor que los procesos y herramientas necesarios para ello, por ejemplo. Debe darse más valor al "Software de funciona sobre la documentación exhaustiva", en este caso, para la presentación final, debe mostrarse algo funcional, si bien la documentación es importante, no se hará de manera exhaustiva, únicamente lo necesario para poner orden en el proyecto y para que, posteriormente, si el presente trabajo resulta de interés para alguien, pueda entender qué fue lo que se hizo. "Colaboración con el cliente sobre negociación de contratos", si bien esto no tiene mucha pertinencia al hablar de un TEG, puede interpretarse como que la Escuela de Ingeniería Informática y el Tesista deberían colaborar pues ambos persiguen un objetivo común: innovar. Finalmente, Responder ante el cambio sobre el seguimiento de un plan"pues al tratarse de un proyecto de investigación, no todos los requerimientos están claramente definidos ni pueden predecirse en su totalidad, es por ello que conforme se va investigando y despejando la incertidumbre en algunos temas, pueden ir surgiendo nuevas cosas que no se esperaban y se debe estar preparado para responder y adaptar el plan a las nuevas condiciones.

Existen numerosas metodologías y ciclos de vida basados en el modelo ágil, para este proyecto en particular se propone utilizar una metodología basada en Scrum para organi-

zar el proyecto y XP (Programación Extrema o eXtreme Programming en inglés) para la organización de cada una de las iteraciones.

3.1. Descripción de la Metodología Planteada

Para describir la metodología que se utilizará para el desarrollo del proyecto, es necesario primero estudiar las dos metodologías antes mencionadas:

SCRUM

SCRUM, segun Roger Pressman (2010), es una metodología creada por Jeff Sutherland y su equipo de desarrollo a principios de la década de 1990. Los principios de SCRUM están enmarcados dentro del Manifiesto Ágil y es un proceso que lleva el Desarrollo de Software a través de las siguientes actividades: requerimientos, análisis, diseño, evolución y entrega. Cada una de esas actividades son realizadas dentro de un patrón de trabajo llamado Sprint, todo el trabajo realizado dentro de un Sprint es adaptado al problema y frecuentemente modificado por el equipo de desarrollo a medida que las condiciones van cambiando.

SCRUM hace énfasis en la utilización de procesos de software que son efectivos en proyectos con tiempos cortos de entrega y requerimientos cambiantes. Esos procesos, en general, definen dos grandes actividades de desarrollo:

- **Backlog:** el backlog, constituye una lista priorizada de requerimientos que agregan valor de negocio al producto, estos requerimientos pueden ser agregados en cualquier momento y, de esta manera, se introducen los cambios en el proyecto (Pressman, 2010). El backlog puede ser, además, modificado en cualquier momento para adaptar las prioridades a los cambios del negocio.
- **Sprints:** los Sprints constituyen paquetes de trabajo que son necesarios para desarrollar un requerimiento o un conjunto de ellos (Pressman, 2010). Los cambios no pueden ser

introducidos dentro de un Sprint, de esta manera se asegura que el trabajo que se realiza es estable, pues los requerimientos que se seleccionan para ser desarrollados en un Sprint ya deben estar definidos y se debe tener la certeza de que, en caso de cambiar, será manejable.

- **Scrum Meetings:** son reuniones cortas que, usualmente, se realizan todos los días durante un sprint (Pressman, 2010). Durante los Scrum Meetings se responden a tres preguntas básicas:
  - ¿Qué has hecho desde la última reunión?.
  - ¿Cuáles obstáculos has encontrado?.
  - ¿Cuál es tu planificación hasta la próxima reunión?.
- **Demos:** se van entregando los resultados de las funcionalidades desarrolladas al cliente para que puedan ser evaluados (Pressman 2010). De esta manera, se va entregando Software Listo y Funcional que agrega valor al negocio del cliente en cada iteración.

De esta manera, SCRUM permite que los equipos trabajen de manera eficiente y estable en proyectos en los que la incertidumbre siempre está presente.

eXtreme Programming

Para Sommenvile (2007), eXtreme Programming (XP) es la más utilizada y la más conocida de las metodologías ágiles. Fue inicialmente concebida por Kent Beck a finales de la década de 1980 y se enfoca en hacer énfasis en la etapa de implementación del ciclo de vida, tal como su nombre lo indica, en la parte de programación o codificación, tomando como buena práctica la programación en parejas, de esta manera, mientras uno de los desarrolladores codifica, el otro está observando y advirtiendo de errores a quien escribe el código.

Kent Beck definió cinco valores para establecer las bases de XP como metodología. Cada uno de esos valores se utilizan para poner en marcha las actividades, acciones y tareas de XP (Pressman, 2010):

- **Comunicación**: entre los clientes y los desarrolladores. XP valora más la comunicación informal por ser más rápida en comparación con volúmenes enormes de documentación como medio de comunicación principal.
- **Simplicidad**: XP restringe a los desarrolladores a realizar el diseño y la implementación de código que satisfaga los requerimientos actuales en vez de pensar en factores futuros, si el diseño puede ser mejorado, puede ser modificado posteriormente.
- **Retroalimentación**: durante un proyecto en necesario muchas veces regresar a una etapa previa y modificar algo para que los requerimientos actuales funcionen sin alterar los que ya fueron implementados. El cambio es algo constante en todo proyecto, pero para poder hacerlo correctamente, se necesita retroalimentación.
- **Coraje**: implementar código sin miedo a las consecuencias, pero siempre de manera coherente.

En eXtreme Programming, no se libera una pieza de software hasta que está totalmente funcional y probada, para asegurar que el software funciona, se desarrollan Pruebas Unitarias.

La metodología propuesta para el desarrollo del presente proyecto, toma la organización de SCRUM y los valores y la filosofía basada en la codificación de XP para llevar a cabo las tareas y actividades concernientes al desarrollo del producto final de este trabajo, obviando la exigencia de XP acerca de la programación en parejas ya que el presente trabajo es realizado por sólo una persona.

Plan General de Trabajo

Se plantea, organizar el desarrollo en las siguientes iteraciones, con una duración de tres a cuatro semanas cada una:

1. Investigación y análisis.
2. Diseño de la arquitectura del sistema.
3. Establecimiento del ambiente de desarrollo.
4. Anlálisis y desarrollo de la ontología.
5. Desarrollo de la herramienta para administrar la Ontología.
6. Desarrollo del traductor de lenguaje natural a SPARQL.
7. Desarrollo del motor de búsqueda.
8. Desarrollo e integración con las vistas.

CAPÍTULO 4

Desarrollo

El desarrollo de este trabajo, se describe siguiendo la metodología planteada anteriormente. Las iteraciones citadas en el *Marco Metodológico* serán descritas a continuación de manera consecutiva:

4.1. [Iteración 1]: Investigación y análisis

Durante esta etapa no se desarrolló ningún tipo de software, es por ello que no se verán historias de usuario, diseño, desarrollo ni pruebas unitarias.

Desarrollo de las tareas

Las tareas de esta iteración se basaron principalmente en la investigación y profundización de los temas relacionados a este trabajo, así como la búsqueda de herramientas para el ensamblaje de la plataforma de desarrollo.

Investigación de los temas relacionados

La mayor parte de la investigación realizada fue descrita en el *Marco Teórico* del presente trabajo. El tópico central del presente trabajo resulta ser *La Web Semántica*, por ello, es necesario tomar en cuenta todos los conceptos que se desprenden de dicho tópico. Los conceptos más importantes que se desprenden de *La Web Semántica* son citados a continuación:

- Ontologías o modelos de conocimiento.
- Motores de inferencia.

Las *Ontologías* o *Modelos de conocimiento*, son el eje central de la *Web Semántica*. Es lo que establece toda la estructura de meta-datos en la que se basa el etiquetado e indexado de los recursos que forman parte de la base de conocimientos. Además, establece las relaciones entre las meta-entidades que conforman la *Ontología*. Define el vocabulario que modela el dominio del problema a ser resuelto.

Los *Motores de inferencia* son herramientas que, dadas ciertas reglas sobre un *modelo de conocimiento*, son capaces de deducir nueva información y nuevas relaciones entre las meta-entidades y, por lo tanto, nuevas relaciones entre los recursos dentro de la base de conocimientos.

El problema y los requerimientos

Del problema general del proyecto, puede plantearse de la siguiente manera: *¿Cómo facilitar el acceso a la información acerca de Ciencias de la Computación a personas interesadas en el área?*. De este problema, puede identificarse los siguientes requerimientos:

- Una interfaz web donde los usuarios puedan interactuar con el sistema.
- Una interfaz de edición que permita a usuarios autorizados agregar nuevos recursos y extender la base de conocimiento.

- Una interfaz de edición que permita a usuarios autorizados agregar meta-información nueva y extender el modelo de conocimiento.
- Un componente de traducción que convierta consultas en lenguaje semi-natural a SPARQL, el lenguaje de consultas sobre RDF/RDFS/OWL.
- Desarrollo de un modelo de conocimiento del área de Ciencias de la Computación e Ingeniería Informática.
- Un componente capaz de realizar consultas e inferencias sobre el modelo de conocimiento realizado.

Selección de la plataforma

Al principio, se planteó llevar a cabo todo el desarrollo utilizando *Python* como lenguaje de programación y las librerías nativas disponibles a través de *easy_install*, pues *Python* es uno de los pocos lenguajes que ofrece librerías nativas para la manipulación de documentos RDF, además de las ventajas propias que ofrece el lenguaje desde el punto de vista sintáctico y de facilidad de aprendizaje y de programación. Pero a la hora de buscar Frameworks que permitieran el desarrollo de aplicaciones basadas en Web Semántica y Motores de Inferencia, la información y la cantidad de herramientas resultó limitada.

Debido a la situación expuesta en el párrafo anterior, se planteó seleccionar la mejor herramienta para cada uno de los componentes del sistema, es decir, la mejor plataforma para la *herramienta para la administración de la ontología* y la mejor plataforma para el buscador como tal.

Herramienta para la administración de la ontología

Las dos opciones más fuertes para el desarrollo de este componente fueron *Python* por un lado, por ser una plataforma 100 % libre y de código abierto y por la gran cantidad de librerías disponibles para extender el lenguaje, y por otro lado *Java* representaba una opción

viable, por ser una plataforma con más de 15 años de desarrollo y estándar *de-facto* de la industria, a continuación se presentan las características más destacables de cada una de las opciones:

- **Python:** según la *Python Software Foundation*, el lenguaje cuenta con las siguientes características
  - Totalmente abierto y libre.
  - Sintaxis clara y legible.
  - Capacidad poderosa de introspección.
  - Multiparadigma: funcional, estructurado y orientado a objetos.
  - Tipos de dato dinámicos.
  - Gestión de errores basada en excepciones.
  - Puede extenderse a través de la escritura de módulos en lenguaje C o C++.
  - Gran cantidad de librerías, entre ellas varias para procesar documentos RDF.
  - Facilidad para interoperar con otras plataformas.
  - Precompilado y semi-interpretado.
- **Java:** según Joyanes y Zahonero (2002), *Java*, es un lenguaje que posee las siguientes características
  - Sencillo, fue diseñado para facilitar las tareas del programador profesional.
  - Orientado a objetos.
  - Distribuido, facilita el desarrollo de aplicaciones que hacen uso de la red mediante la incorporación de clases que manejan protocolos TCP/IP.

- Precompilado y semi-interpretado.
- Arquitectura neutra, debido a que se ejecuta en una máquina virtual.
- Portable, dado que al "compilar" no se produce un archivo ejecutable como ocurre en los lenguajes compilados (como C, C++ y Pascal, por ejemplo), sino un *bytecode* que es ejecutado por la máquina virtual, este *bytecode* puede ser ejecutado en cualquier otro sistema operativo, siempre y cuando exista en él una instancia de la *Máquina Virtual de Java*.
- Multihilo, un programa en *Java* puede ejecutar múltiples tareas de manera simultánea

Ambas plataformas ofrecen prestaciones muy similares, si bien el núcleo de *Python* no es muy amplio, posee gran cantidad de librerías para extender su funcionalidad. De igual manera, *Java* incorpora cientos de clases que lo convierten en un lenguaje amplio y poderoso.

Para este componente, se decidió trabajar con *Java*, ya que, como se verá más adelante, ofrece el mejor Framework para el desarrollo de aplicaciones basadas en Web Semántica y, al seleccionar dicho Framework, la transitividad nos lleva a trabajar con este lenguaje.

Buscador

El buscador, es el encargado de realizar peticiones al motor de consultas sobre RDF e interactuar con el usuario final, es por ello que la generación de vistas es un aspecto importante en esta parte del sistema.

Para el desarrollo de la herramienta de búsqueda, fue seleccionada la plataforma *Python*, debido a todas las razones expuestas anteriormente, además de contar con una gran cantidad de *frameworks* y herramientas que facilitan el desarrollo web.

Selección de los Frameworks de desarrollo

Luego de realizar una investigación en la web, los Frameworks de desarrollo que parecen ser más utilizados para las aplicaciones semánticas son:

- **Sesame:** escrito en *Java*.
- **RedLand:** escrito en C y accesible desde *Python* mediante *Wrapping*.
- **Jena:** escrito en *Java*.
- **CubicWeb:** escrito en *Python*.

En este caso, se tomó la decisión de trabajar con el Framework *Jena* debido a que es el que ofrece la mayor cantidad de documentación disponible en línea, además, es compatible con todos los niveles de representación semántica de meta-información (RDF, RDFS y OWL), posee varios motores de inferencia ya integrados y permite la integración con motores de inferencia externos, tiene un motor de consultas SPARQL y ofrece la posibilidad de integrarse con medios de persistencia externos, además de ser libre y de código abierto (The Jena Community, S/F), sin embargo, los razonadores incluíos en *Jena* no son compatibles con OWL, por ello se seleccionó *Pellet*, un motor de inferencia externo al Framework, escrito en *Java* y compatible con *Jena*.

El Framework *Sesame*, ofrece prestaciones técnicas similares a las de *Jena*, sin embargo, la documentación disponible en línea no es comparable con la que ofrece este último y, a pesar de ser de código abierto, es propiedad de una empresa alemana llamada *ADUNA* (Autor desconocido, S/F) y para poder acceder a información más profunda o solicitar ayuda en algo relacionado al Framework, es necesario pagar por horas de consultoría, lo que hace poco viable la utilización de *Sesame* para este proyecto.

RedLand, escrito en C y accesible desde *Python*, según la comunidad *RedLand* (S/F), luego de leer la documentación, es compatible sólo con RDF y, para este trabajo, el modelo

de conocimientos planteado utilizaría anotaciones definidas en la especificación RDFS y, posiblemente, algunas definidas en el dialecto OWL-Lite, por lo que fue descartado. Además, el proceso de instalación y configuración resulta complicado comparado al de las otras opciones.

Finalmente *CubicWeb*, a pesar de ser un Framework realmente completo pues ofrece toda la plataforma para el desarrollo: desde la persistencia, hasta la generación de vistas, pero fue descartado por no ser compatible con SPARQL, sino con su antecesor: RQL (CubicWeb Community, S/F).

Selección del medio de persistencia

Para la persistencia de datos, si bien los manejadores de base de datos relacionales tradicionales como MySQL y Postgres ofrecen paquetes para la gestión de documentos RDF y OWL, existe una alternativa que ofrece dicha funcionalidad de manera nativa. Se trata de *Virtuoso*, un manejador de base de datos con capacidad de gestionar información en formato RDF y XML, compatible con los estándares ODBC y JDBC, posee un motor de inferencia interno, puede correr en ambientes federados, según información de la comunidad (Virtuoso Community, S/F) y, además, existe una edición *Open Source* bastante completa y muy bien documentada.

4.2. [Iteración 2]: Diseño de la arquitectura del sistema

El sistema estará constituído por dos aplicaciones distintas. La primera, escrita en *Java*, constituye una herramienta de administración para la ontología, la segunda, desarrollada en *Python* constituye una interfaz de consultas sobre la ontología. El servidor, está constituído por el motor de persistencia que, simplemente, almacenará el modelo una vez aplicado el razonamiento a través de un motor de inferencia. Además, se cuenta con una estructura de archivos expuesta vía HTTP a través de Apache, en la que se encuentra el modelo original y

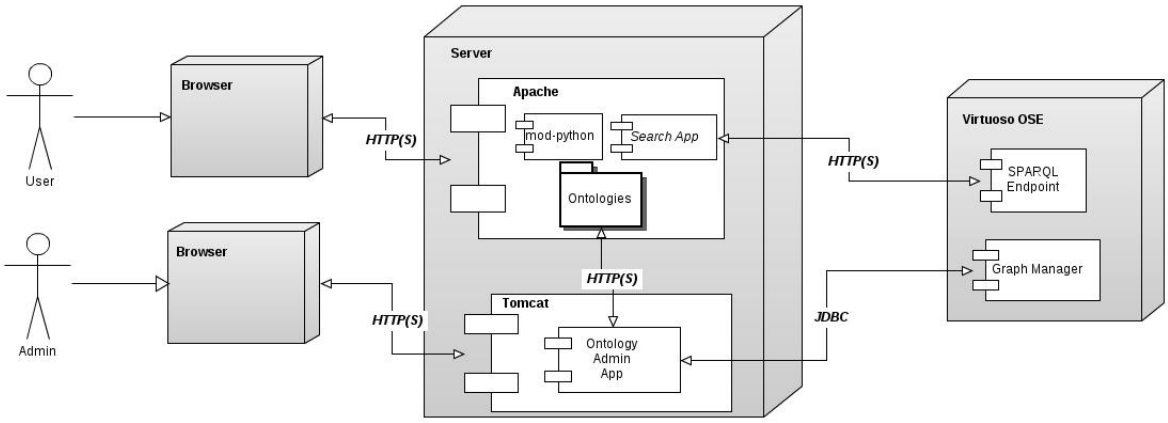

Figura 4.1: Arquitectura del sistema

el modelo razonado para ser consumidos por cualquier otra aplicación que desée hacer uso de la ontología, bien sea para aplicar razonamiento al modelo base o para hacer uso del modelo razonado y sus recursos.

A continuación se presenta el diagrama de arquitectura resultante del análisis previo:

4.3. [Iteración 3]: Establecimiento del ambiente de desarrollo

En esta iteración no se desarrolló ningún tipo de software, por ello, no fue necesaria la realización de pruebas unitarias, sin embargo, se llevó a cabo un proceso de configuración posterior a la selección de las herramientas de desarrollo.

Desarrollo de las tareas

El proceso que dirigirá la etapa de desarrollo, será el proceso de programación o codificación, en dos lenguajes de programación distintos: *Python* del lado del *Cliente* y *Java* del

lado del *Servidor*, es por ello que el *Sistema Operativo* utilizado en el equipo de desarrollo y despliegue del sistema, **debe** facilitar la instalación y configuración de las herramientas necesarias para realizar dicho trabajo, además de albergar el sistema una vez desarrollado.

A nivel de *Sistema Operativo*, se seleccionó una distribución de *GNU/Linux*, muy popular en el mundo de los desarrolladores de software y servidores, ya que ofrece compatibilidad con *Java* a través de una instancia de la máquina virtual y el intérprete de *Python* usualmente viene preinstalado en todos los sistemas *GNU/Linux*. La distribución seleccionada fue *Debian* pues, ofrece alrededor de 30.000 paquetes de software instalables a través del gestor de paquetes *aptitude* o, su front-end gráfico *Synaptic* (Debian Community, S/F), además, las versiones de esta distribución de *GNU/Linux* usualmente tienen años de diferencia pues la comunidad pone especial atención en liberar software que está 100 % probado y estable (Autor Desconocido, 2011).

A nivel de servidores de aplicaciones, se seleccionó *Tomcat* para alojar la aplicación *servidora* por ser estándar *de-facto* en la industria a la hora de alojar aplicaciones web desarrolladas en *Java* y *Apache2* para alojar la aplicación *cliente*, junto con *libapache-mod-python* para activar la compatibilidad del servidor *Apache2* con *Python*.

Para desarrollar las tareas de programación, se seleccionaron las siguientes herramientas:

- Para desarrollar en *Java*, se seleccionó el Entorno Integrado de Desarrollo (IDE) *Eclipse*
- La *Ontología*, ya que debe ser escrita en RDF/OWL, se seleccionó el editor *Protègè*, desarrollado en la Universidad de Stanford, este editor, permite el modelado de manera gráfica y la generación automática de código válido en varias sintaxis de RDF/OWL.
- Para desarrollar en *Python*, los requerimientos no son muchos, por ello, para mantenerlo simple, se seleccionó el editor *VIM*, con el *plugin* NERDTree, que permite la visualización de la estructura de directorios del proyecto y la apertura de varios archivos de código fuente en distintas pestañas a nivel de terminal.

Configuración

Toda la instalación de los paquetes necesarios para el desarrollo, a excepción de los *plugins* para *VIM* y los entornos de desarrollo *Eclipse* y *Protègè*, fueron descargados, instalados y configurados a través del gestor de paquetes de Debian.

En el caso de *Eclipse*, no hizo falta instalarlo, pues la comunidad provée un paquete que ejecuta la aplicación, con el único requisito de tener *Java* instalado. Lo mismo ocurrió en el caso de *Protègè*.

4.4. [Iteración 4] Análisis y desarrollo de la *ontología*

Durante esta iteración, si bien se desarrolló una parte importante del sistema como lo es la *ontología*, no se codificó software como tal. La etapa de pruebas unitarias, fue sustituida por una etapa de validación del modelo de conocimiento.

Desarrollo de las tareas

Para el desarrollo de la *ontología* definitiva, fue necesario realizar un análisis previo para determinar cómo está estructurado el conocimiento en las áreas de *Ciencias de la Computación* e *Ingeniería Informática*, posteriormente, se procedió a realizar el diseño, codificación y validación de la *ontología* hasta llegar al modelo final.

Análisis

Antes de realizar un diseño preliminar de la *ontología*, es necesario realizar un análisis de la estructura del conocimiento en el área, con este análisis se busca:

1. Establecer las *entidades* que conformarán la *ontología*.

2. Determinar las *relaciones* existentes entre las *entidades*.

3. Conocer más a fondo el dominio del problema a ser resuelto.

Los tres (3) ítems enumerados anteriormente, pueden resumirse como *Determinar la estructura del conocimiento en el área seleccionada*. Para ello, es necesario aplicar técnicas de *Ingeniería del Conocimiento*. Según Norvig y Rusell (2004) *Ingeniero de Conocimiento*, es alguien que investiga un dominio concreto, aprende qué conceptos son los importantes de ese dominio y crea una representación formal de los objetos y relaciones del dominio. En este caso, el dominio ya ha sido investigado por el autor durante los años de carrera, simplemente hace falta determinar qué conceptos son importantes en el área y producir la representación formal de dichos conceptos y las relaciones que guardan entre sí.

Diseño

Luego de analizar el área seleccionada y con base en las recomendaciones de la *IEEE Computer Sociery* (IEEE-CS) y la *Assosiation for Computing Machinery* (ACM), descritas en los documentos *Computing Curricula 2001, Computer Science Final Report* para el campo de las *Ciencias de la Computación* y el documento *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Programs in Computer Engineering*, que incorpora elementos de Ingeniería al primero, se llegó a la estructura jerárquica de clases mostrada en la Figura 4.2:

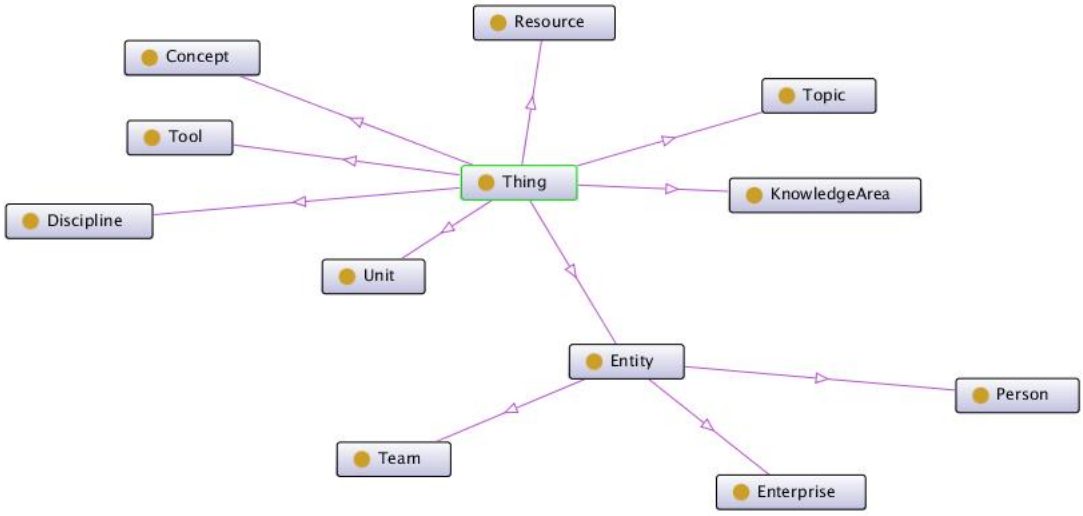

Figura 4.2: Jerarquía de clases

En RDF/OWL, todas las clases o entidades heredan por defecto de la clase *Thing*, un principio similar al de la clase *Object* en *Java*, las líneas moradas expresan precisamente eso, una relación de *herencia* entre dos entidades que puede leerse como *is-a* (*es-un* o *es-una*), esta relación de *herencia* es, además, transitiva, lo que nos permite inferir, por ejemplo, que un elemento de la clase *Person*, es también un *Thing* aunque no exista una relación directa, pues la clase de nivel inmediatamente superior lo es. Las clases *KnowledgeArea*, *Unit* y *Topic*, son las mismas utilizadas en los documentos de la *IEEE-CS* y la *ACM* para dividir los grandes bloques de conocimiento, en unidades más pequeñas, manejables y específicas a cada nivel como se muestra en la Figura 4.3. La clase *Discipline*, fue agregada para hacer referencia al marco global del problema cuyo dominio se está modelando, en este caso *Ciencias de la Computación*, con el objetivo de facilitar la adaptación de este modelo a otros dominios. Por otra parte, la clase *Concept*, fue agregada con el propósito de lograr una granularidad más fina y poder expresar niveles de conocimiento más específico, las clases *Entity*, *Person*, *Team*,

Resource y *Enterprise* serán explicadas posteriormente.

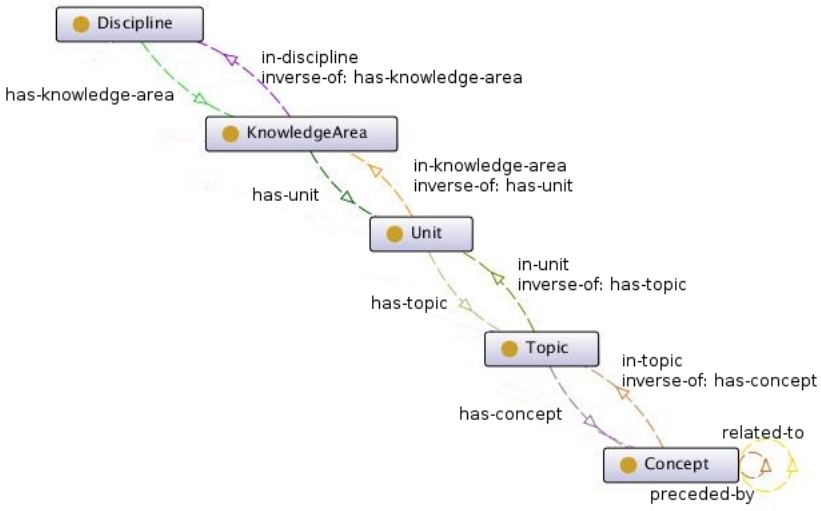

Figura 4.3: Estructura del conocimiento según *IEEE-CS* y *ACM*

En la Figura 4.3 se muestra la estructura del conocimiento según las recomendaciones de la *ACM* y la *IEEE-CS*, puede observarse que es una estructura de *Contención*, donde el nivel superior engloba todo lo que se encuentra en los niveles inferiores. También, según la Figura 4.3, las propiedades a la derecha (las que tienen el prefijo *in*) son propiedades inversas de las de la izquierda (las que tienen el prefijo *has*). Anotar esas propiedades como inversas, permite que a la hora de declarar un *Named Individual* (instancias) en alguna de las clases, sólo sea necesario especificar una de las dos propiedades. Toda la estructura de contención va desde la clase más general (*Discipline*) que contiene a toda la estructura, hasta la clase más específica (*Concept*) que es el último eslabón de la cadena. Sólo con las relaciones descritas anteriormente y representadas de manera gráfica en la Figura 4.3, no basta para expresar lo dicho anteriormente pues ninguno de los atributos dibujados es transitivo, además, son propiedades distintas y no expresan ningún tipo de significado una de la otra sino que tienen significado por sí solas. Para solventar esta situación, se crearon dos súper-propiedades: *contents* y *content-by*, ambas transitivas e inversas una de la otra y que envuelven las propiedades *has* e *in* respectivamente.

De esta manera, además de lo descrito en la Figura 4.3, queda expresada la *Jerarquía de Contención* mostrada en la Figura 4.4 gracias a la propiedad *contents*:

$$Discipline \subseteq KnowledgeArea \subseteq Unit \subseteq Topic \subseteq Concept$$

Figura 4.4: Jerarquía de Contención

De la misma manera, también, a través de su relación inversa *content-by*, queda expresado lo descrito en la Figura 4.4

$$Discipline \supseteq KnowledgeArea \supseteq Unit \supseteq Topic \supseteq Concept$$

Figura 4.5: Jerarquía de Contención Inversa

En la Figura 4.3, se observan dos relaciones recursivas en le entidad *Concept*, estas son: *related-to* y una sub-propiedad *preceded-by*, estas propiedades son *simétrica* y *transitiva* respectivamente, con esta jerarquía de propiedades, se expresa que si un concepto *precede* a otro, entonces, también están *relacionados*.

Cuando se mostró la estructura de clases, quedaron varias entidades pendientes por explicar cuál es su sentido dentro de la *Ontología*. Lo mejor será comenzar por las entidades *Entity*, *Person*, *Team*, *Enterprise* y *Tool*, ya que están todas relacionadas entre sí como se observa en la Figura 4.6:

La clase *Entity*, envuelve a las clases *Person*, *Team* y *Enterprise*. Según lo expresado en la Figura 4.6 una instancia de *Entity* puede enunciar (*states*) un concepto, desarrollar (*develops*) una herramienta o ser referencia (*is-reference-in*) en un área de conocimiento. Por herencia, cualquier sub-clase de *Entity* será capaz de hacer lo mismo. Adicionalmente, se asocian herramientas (*Tool*) a conceptos (*Concept*) con la finalidad de poder expresar que una herramienta, implementa en la práctica un concepto teórico, por ejemplo: *Python* es un *Lenguaje de Programación* que abarca los paradigmas *Estructurado*, *Orientado a objetos* y *Funcional*, todos estos, serían conceptos y *Python* la herramienta que los implementa. Esto se modela para brindar una mayor expresividad

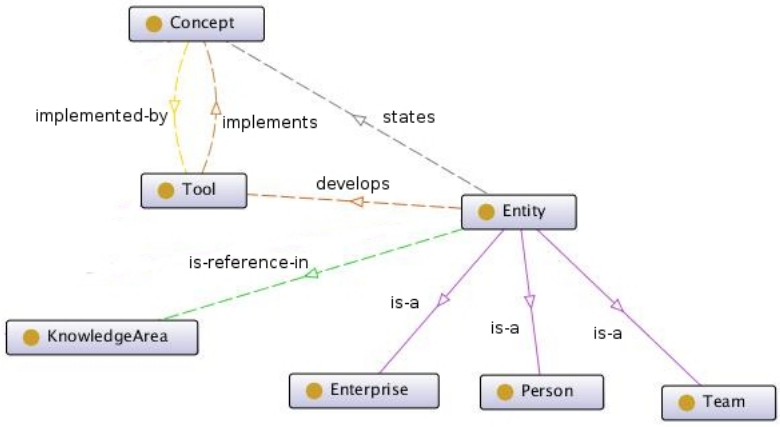

Figura 4.6: La clase *Entity*

al vocabulario creado a manera de poder satisfacer consultas del tipo "Herramientas de *Google* para *Cloud Computing*", por dar un ejemplo.

La clase *Resource*, modela los recursos que hacen referencia a alguna otra clase del modelo de conocimiento generado. Es por ello que cualquier entidad de la *Ontología* puede tener un recurso asociado y, un recurso, puede estar asociado a un concepto si es muy específico, a un área de conocimiento si es más general o, incluso, a la disciplina si es lo suficientemente global como para abarcar toda su extensión, ver Figura 4.7.

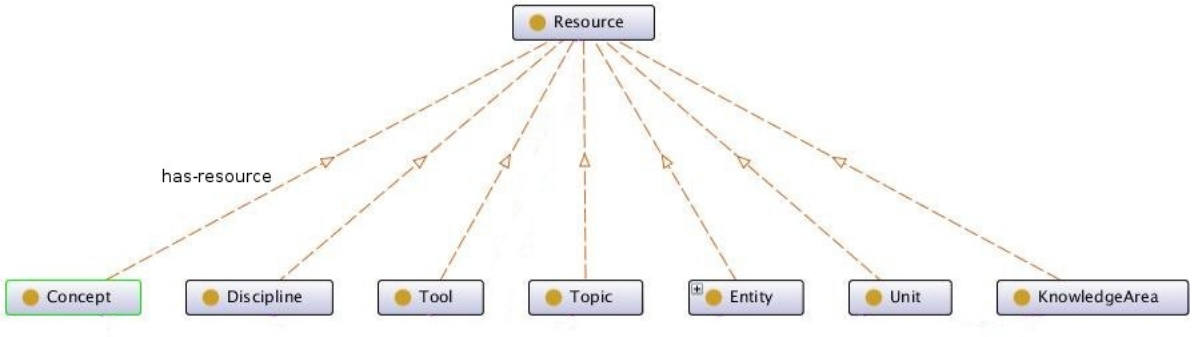

Figura 4.7: La clase *Resource*

Implementación

Para la implementación de la *Ontología*, se utilizó el editor gráfico *Protègè*, este editor permite la construcción de la *Ontología* de manera gráfica, para ello, se reprodujo el modelo mostrado en la sección anterior y se aprovechó la capacidad de la herramienta para la generación de código. Se realizaron las configuraciones necesarias para que el código se genere con la sintaxis RDF/XML.

Se crearon instancias desde la clase *Discipline* hasta la clase *Topic*, según las áreas de conocimiento, unidades y tópicos especificados en los documentos de recomendación de la *IEEE-CS* y la *ACM*, tomando en cuenta sólo los contenidos del núcleo de la carrera.

Validación

Para llevar a cabo la validación de la *Ontología*, se utilizaron dos métodos para determinar dos factores distintos acerca del modelo:

- **OntoClean:** una metodología propuesta por Nicola Guarino y Christopher Welty para la generación de *Ontologías Limpias*, es decir, con una estructura coherente y consistente. En este caso, se utilizó para validar la estructura, es decir, si las

decisiones tomadas respecto a las entidades que fueron modeladas como clases, propiedades e individuales (instancias de clase) fueron acertadas.

- **Domain Questions:** o *Preguntas de Dominio*, es un método propuesto en el texto *Semantic Web for the Working Ontologist*, en el que se realiza una serie de preguntas que el modelo en cuestión debe ser capaz de responder, ya sea por sí solo o a través de inferencia sobre las relaciones y propiedades de cada entidad. Con este método, se busca validar la completitud de la *Ontología*, es decir, si con lo que está expresado en ella, es suficiente para cumplir con el propósito del dominio para el cual fue hecha.

4.5. [Iteración 5]: Desarrollo de la herramienta para administrar la ontología

Durante esta etapa, se desarrolló una herramienta para gestionar la ontología desarrollada en la iteración anterior. Esta herramienta no pretende ser un editor de ontologías, pues no editará ni agregará clases ni relaciones.

A través de esta herramienta podrá agregarse nuevos recursos y nuevas instancias a las clases ya creadas y, además, actualizar la ontología razonada dentro del motor de persistencia.

Finalmente, vale destacar que dado carácter experimental y prototípico del presente trabajo, se prestará especial atención al procesamiento de los datos en términos de sus relaciones, dejando en segundo plano aspectos como usabilidad, seguridad y diseño a nivel gráfico, tópicos importantes para cualquier sistema completo y que, sin lugar a dudas, deben estar presentes en la solución final pero que fueron relegados a un segundo plano para la presentación deste prototipo y se espera que dichas características sean agregadas en trabajos futuros.

Desarrollo de las tareas

Esta herramienta fue desarrollada utilizando *Java* como lenguaje de programación y *Java Server Pages* (JSP) como herramienta para la generación de vistas web. Como motor de persistencia se utilizó *Virtuoso Open Source Edition*, se utilizó *Pellet* para razonar sobre la ontología y las clases provistas por *Jena* para llevar a cabo la integración de todas estas tecnologías y manipular la ontología subyacente escrita en OWL.

Análisis

La herramienta desarrollada es capaz de:

1. Agregar, eliminar, consultar y editar recursos.
2. Agregar, eliminar, consultar y editar anotaciones.

Respecto a las clases y las relaciones definidas en la ontología base, una vez que el conocimiento ha sido modelado, es realmente difícil que aparezca metainformación nueva para ser incorporada a nivel de clases. Es por ello que no se incorporó esta funcionalidad en la herramienta, de ocurrir esto, resulta más conveniente agregar las nuevas clases a través de un editor de ontologías especializado (como Protègè). Lo que sí es posible es la aparición de nuevos conceptos o áreas de conocimiento dentro del campo que se realiza el modelo de conocimiento, estos pueden ser agregados como instancias de las clases ya creadas.

Diseño

La arquitectura de la herramienta es una fusión de una Arquitectura centrada en datos y una Arquitectura por capas.

Una Arquitectura centrada en datos (Data Centered Architecture) es aquella en la que los datos son el corazón del sistema y las aplicaciones acceden a ellos para consultarlos

y manipularlos o modificarlos según sea el caso, por otra parte, una Arquitectura por capas (Layered Architecture), es aquella en la que se divide la aplicación en distintas capas funcionales, donde las capas más internas realizan operaciones con el sistema operativo, mientras las más externas interactúan con el usuario final (Pressman, 2010)

En este caso, la herramienta modifica, consulta y manipula datos almacenados en documentos OWL y en el motor de persistencia Virtuoso, además, está dividida en tres (3) capas cada vez más cercanas a los datos, en las que la más interna realiza la extracción y escritura de los mismos a través de las clases provistas por *Jena*. La capa intermedia, convierte los datos complejos de que recibe de la capa inferior a algo más simple y fácil de manipular, así como también, realiza el proceso inverso. Finalmente, la capa más externa, está compuesta por archivos JSP y clases Servlet que gestionan las vistas, muestran información e interactúan con el usuario. Se destinó, además, un paquete aislado para las pruebas unitarias.

En la Figura 4.8, puede observarse la arquitectura diseñada para la herramienta, el funcionamiento se explicará posteriormente en la etapa de implementación.

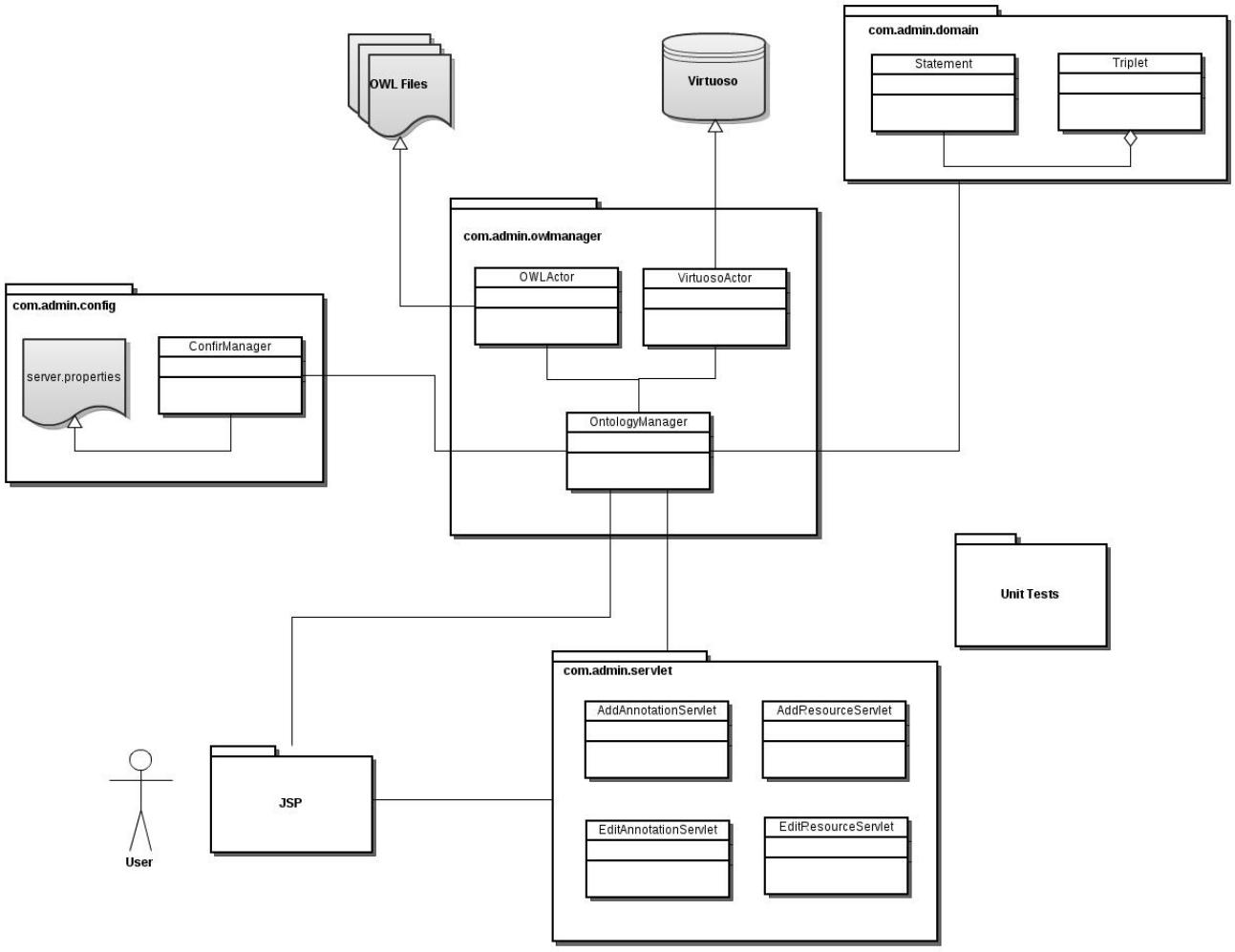

Figura 4.8: Diagrama de clases

Implementación y validación

Toda la codificación de la herramienta se llevó a cabo utilizando la perspectiva *Java EE* (Java Enterprise Edition) de *Eclipse IDE*, esta perspectiva es la que contiene los módulos necesarios para el desarrollo de aplicaciones web.

Al inicio, se configuró el classpath del proyecto con las librerías del JDK y se incluyeron los paquetes precompilados (paquetes *jar*) necesarios para el funcionamiento correcto

de *Jena Framework* y el motor de inferencia *Pellet*, así como también el driver JDBC para *Virtuoso*.

Las clases que conforman la arquitectura diseñada en la etapa anterior, tienen, en líneas generales las siguientes funciones:

- Las clases *OWLActor* y *VirtuosoActor* interactúan con los medios de persistencia en OWL y Virtuoso respectivamente, extrayendo, escribiendo o actualizando datos, según sea el caso, retorna los datos con los tipos tal como son extraídos de su medio de persistencia.

- La clase *OntologyManager* es la encargada de realizar la transformación de los datos complejos que provéen las clases *OWLActor* y *VirtuosoActor* como las distintas abstracciones de recursos como las clases *Individual*, *Resource*, *RDFNode* y *Property* de *Jena* a los *JavaBeans MyStatement* y *Triplet*, definidos en el paquete *com.admin.domain*, así como de realizar transformaciones de colecciones abstractas como el *StmtIterator* a colecciones más manejables y conocidas como el *TreeSet* y el *ArrayList* de *Java*, estos *JavaBeans* y estas colecciones, son más manejables para la capa de presentación. La clase *OntologyManager* también realiza el proceso inverso de conversión y la extracción de las propiedades de configuración mediante llamadas a la clase *ConfigManager*, destinada a interactuar con el archivo de configuración. En pocas palabras, *OntologyManager* orquesta los métodos escritos en las clases subyacentes para satisfacer las peticiones enviadas desde las vistas.

- A nivel de vistas, para mostrar datos, la información va directamente desde la clase *OntologyManager*, pero en el caso contrario, es decir, cuando una vista necesita comunicarse con *OntologyManager*, no lo hace directamente, la solicitud es procesada por un *Servlet* que despacha la operación a los métodos correctos de *OntologyManager* según el evento que se haya invocado en la vista JSP avtiva.

El desarrollo se dividió por características en forma de historias de usuario y fueron desarrolladas en el siguiente orden:

1. Yo como **usuario**, deseo poder crear recursos **para** enriquecer la ontología y extender los resultados de búsqueda.

2. Yo como **usuario**, deseo poder modificar recursos previamente creados **para** corregir errores en el etiquetado de manera fácil y mejorar los resultados de búsqueda.

3. Yo como **usuario**, deseo poder eliminar recursos **para** mantener la ontología limpia de recursos obsoletos o inexistentes.

4. Yo como **usuario**, deseo poder crear anotaciones **para** extender el vocabulario de la ontología e incluir nuevos conceptos.

5. Yo como **usuario**, deseo poder modificar anotaciones **para** corregir errores en el vocabulario y evitar resultados de búsqueda inconsistentes.

6. Yo como **usuario**, deseo poder eliminar anotaciones **para** mantener la ontología limpia de conceptos no vigentes o descritos de manera errada.

Cada característica fue desarrollada "de abajo hacia arriba", es decir, primero se codificaron los métodos de extracción y/o escritura de datos en las clases *OWLActor* o *VirtuosoActor* según fuera el caso y se validaba su correcto funcionamiento mediante *Pruebas Unitarias* a través de *JUnit Framework*, luego se codificaba la lógica de traducción de los datos provistos por la capa de persistencia a los definidos en en paquete *com.admin.domain* y el proceso inverso, así como las transformaciones necesarias para proveer datos que fueran lo más fáciles de manejar posible a la capa de presentación. Luego de eso, se construía la vista básica de los datos, la cual se enriquecía poco a poco utilizando *CSS* para mejorar el estilo de la presentación de los datos y *JavaScript*

para realizar peticiones de datos asíncronas al servidor en los casos que hiciera falta. Finalmente, se codificaba la lógica de envío de datos en los *Servlets* correspondientes.

Únicamente se desarrollaron pruebas unitarias para los métodos de persistencia y en algunos de la capa de transformación, pues no en todos la lógica era lo suficientemente compleja como para requerir validar su funcionamiento a través de este método.

4.6. [Iteración 6]: Desarrollo del buscador

En esta iteración se describe el desarrollo del buscador como tal, su diseño y su implementación utilizando *Python*. Debido al carácted experimental y prototípico de este trabajo, aspectos como usabilidad y diseño gráfico fueron dejados en segundo plano para concentrar la atención en el procesamiento de la información y en el desarrollo de algoritmos que ejecuten las tareas de manera eficiente.

Desarrollo de las tareas

El buscador fue desarrollado utilizando *Python* como lenguaje de programación, el módulo *CherryPy* para la generación de vistas web, un protocolo de comunicación basado en *JSON* para la comunicación con el *endpoint* de *SPARQL* que provée *Virtuoso*, encapsulado dentro del módulo *sparql-wrapper*, y el módulo *Python Lex-Yacc* (PLY) para diseñar un lenguaje de consultas basado en anotaciones y conectores lógicos amigable para el usuario, intuitivo y que, a partir de esa gramática, se pudiera generar la o las sentencias *SPARQL* necesarias para satisfacer la consulta del usuario, finalmente, se utilizó el *framework* para desarrollo web *CherryPy*, junto con el lenguaje de plantillas de *Mako Templates* para generar las vistas del buscador, nuevamente, debido al carácter prototípico de este trabajo, el diseño y aspecto visual, pasan a un segundo plano, prevaleciendo el procesamiento de los datos como prioridad en este desarrollo.

Análisis

El buscador, es un sistema completo que debe recibir consultas, procesarlas, solicitar la información necesaria al medio de persistencia a través de sentencias *SPARQL* y mostrar los resultados al usuario. Este requerimiento general, puede ser separado en las siguientes historias de usuario:

1. Yo como **usuario** requiero un sistema que procese mis consultas *para* conseguir recursos en el área de Ciencias de la Computación de manera sencilla.

2. Yo como **desarrollador** requiero de un componente que separe la consulta del usuario en partes más manejables **para** poder procesarla fácilmente.

3. Yo como **desarrollador** requiero de un componente que traduzca la consulta del usuario a lenguaje *SPARQL* **para** poder realizar solicitudes al medio de persistencia.

El buscador debe ser capaz de satisfacer distintos tipos de consultas, pueden clasificarse en tres (3) grupos:

1. Consultas en las que se requirere que un recurso posea un conjunto de anotaciones específicas.

2. Consultas en las que se require que un recurso posea un conjunto de anotaciones específicas u otro.

3. Consultas de tópicos relacionados a una anotación específica.

Estas consultas se hacen a través del lenguaje *SPARQL*, este es un lenguaje formal en el que no resulta conveniente obligar al usuario a escribir pues, si bien su curva de aprendizaje no es muy empinada, el objetivo es realizar búsquedas utilizando tecnologías de *Web Semántica* de manera sencilla, si quisieramos que el usuario introdujera consultas

utilizando *SPARQL* en nuestro buscador, con el *endpoint* que provée *Virtuoso* sería más que suficiente.

Diseño

Para empezar el desarrollo del buscador, debe resolverse primero el problema de cómo se realizarán las consultas.

Dado que el *Procesamiento de Lenguaje Natural* es un tema que excede el alcance de este trabajo, se decidió diseñar un lenguaje formal y, al mismo tiempo, "amigable al usuario" para la escritura de las consultas. Este lenguaje está basado en anotaciones conectadas a través de conectores lógicos *OR* y/o *AND*, además de un operador para calcular los tópicos relacionados a una anotación dada, el operador *?rel:*.

Para procesar las consultas en el lenguaje lógico, se asume que todas las consultas lingüísticas pueden escribirse, de manera formal, en *Forma Normal Disyuntiva* (DNF) o en *Forma Normal Conjuntiva* (CNF) (Luque et al, S/F), diseñando, para este caso, una gramática que soporte la escritura de consultas en un lenguaje lógico formal en DFN.

```
Query     : OrQuery | AndQuery
OrQuery   : AndQuery || OrQuery | AndQuery
AndQuery  : AndQuery && UnitQuery | UnitQuery
UnitQuery : Annotation | ?rel: Annotation
```

Figura 4.9: Gramática del lenguaje en notación BNF

En la Figura 4.9, se aprecia la gramática formal del lenguaje de consultas diseñado, se define un *Query unitatio* (UnitQuery) como aquel conformado por sólo una anotación o por una consulta de tópicos relacionados. La gramática expuesta en la Figura 4.9 se traduce en lo siguiente:

- Un *Query*, está conformado por una solicitud de tipo *OR* (OrQuery) o una solicitud de tipo *AND* (AndQuery).
- Un *OrQuery* está conformado por un *AndQuery* en disyunción con un *OrQuery* o un *AndQuery*.
- Un *AndQuery* está conformado por un *AndQuery* en conjunción con un *UnitQuery* o un *UnitQuery*.
- Un *UnitQuery* está conformado por una anotación o por un query de tipo *?rel:*.

Esta gramática, primero separa todas las distunciones de la consulta y las agrupa en consultas individuales conjuntivas, separadas por disyunciones, es decir, el lenguaje diseñado procesa consultas en DNF.

El componente diseñado corresponde al lo escrito en la historia de usuario número dos, pues toma la consulta y la separa en partes que pueden ser procesadas y traducidas posteriormente de manera más sencilla. La arquitectura completa del buscador, se presenta en la Figura 4.10

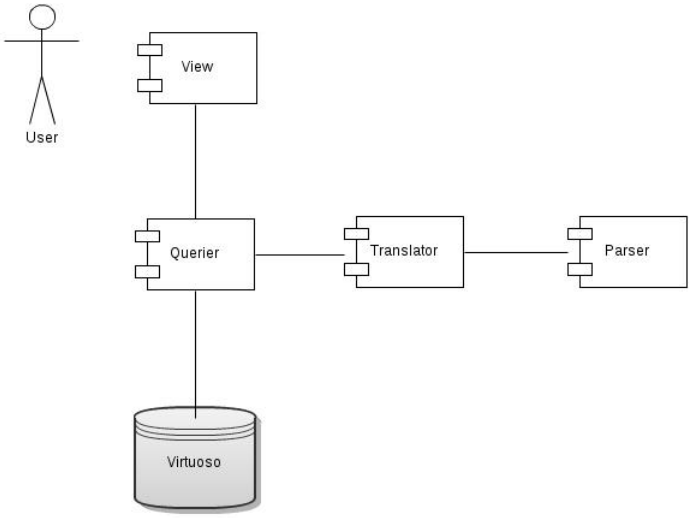

Figura 4.10: Diagrama de componentes

Implementación y validación

Toda la codificación del prototipo del sistema de búsqueda se llevó a cabo utilizando el editor *VIM* (Vi Improved) con la extensión *NERDTree* para navegar a través de los directorios del proyecto y abrir archivos de código fuente en pestañas distintas dentro de la misma instancia del editor en la misma línea de comandos.

La primera etapa del desarrollo, se enfocó en el *Parser* para el lenguaje diseñado, para ello se utilizó el módulo *Python Lex-Yacc* (PLY), que permite escribir compiladores e intérpretes para gramáticas libres de contexto, que es el basamento de los lenguajes formales. PLY, contiene dos clases: *lex* y *yacc* que son, respectivamente, un analizador léxico y un parser.

El analizador léxico, conocido también como *lexer*, separa la entrada en partes conocidas como *tokens*, cada vez que el *lexer* retorna un *token*, lo asocia con un componente léxico del lenguaje o *lexema* (Aho et al, 2007), es decir, este análisis léxico de cada expresión, comprende la separación y clasificación de cada parte de dicha expresión en componentes léxicos conocidos por el intérprete.

En este caso, el *lexer* debe reconocer cuatro (4) *lexemas* básicos: *AND*, *OR* y *REL*, que son los operadores sobre el cuarto *lexema* que compone el lenguaje: *ANNOTATION*. Estos componentes léxicos se describen, a nivel de código fuente, a través de expresiones regulares. Como el público objetivo de este sistema son personas del área de Ciencias de la Computación, los símbolos para los *lexemas AND* y *OR* son las secuencias de caracteres *(*doble ampersand) y *doble barra vertical* respectivamente pues son los símbolos para los operadores lógicos “y” y “o”en muchos lenguajes de programación ampliamente utilizados y difundidos como *C*, *Java* y *PHP*, por lo que debería resultar intuitivo para estas personas la utilización de esa simbología.

El parser, por otra parte, realiza un análisis sintáctico, aplicando las reglas gramaticales definidas anteriormente. Este proceso de interpretación del lenguaje lleva a cabo estas dos etapas, el proceso de traducción, toma el árbol sintáctico que construye el parser y

lo recorre de manera recursiva, traduciendo esas expresiones en consultas *SPARQL* que son enviadas al medio de persistencia a través del módulo de *Python sparql-wraper*.

Finalmente, toda la capa de presentación, fue desarrollada utilizando el módulo *CherryPy*, que constituye un *framework* sencillo para aplicaciones web, no incluye ninguna herramienta *ORM* (Object-Relational Maping), lo que resulta conveniente ya que nuestra fuente de datos no es una Base de Datos Relacional, ni una herramienta de plantillas, por ello, se seleccionó *Mako Templates* para el diseño de las vistas. Tomando el cuenta el carácter prototípico y experimental de este trabajo, el diseño gráfico y los aspectos de usabilidad fueron relegados a un segundo plano. Toda la validación del software, se llevó a cabo mediante la escritura de *Pruebas Unitarias* a través del framework *unittest*, que sigue una filosofía muy similar al framework *JUnit*, utilizado para validar el funcionamiento correcto de los componentes de la herramienta de administración de la ontología desarrollada en la iteración anterior.

CAPÍTULO 5

Resultados

Luego de haber finalizado el desarrollo y la validación del prototipo, se presentan a continuación los resultados obtenidos:

- Se obtuvo una ontología escrita en *OWL* aplicable para modelar cualquier Disciplina de las ciencias o las humanidades a través de la creación de las instancias correspondientes a ella.
- Se obtuvo un modelo de conocimiento del área de *Ciencias de la Computación* escrito en *OWL*, basado en la ontología mencionada en el punto anterior.
- Se obtuvo una herramienta de carácter prototípico para la administración de las instancias (anotaciones) del modelo de conocimiento mencionado en el punto anterior.
- Se obtuvo un lenguaje básico de consultas basado en *Forma Normal Disyuntiva*.
- Se obtuvo un buscador sobre el modelo de conocimiento del área de *Ciencias de la Computación* que implementa el lenguaje mencionado en el punto anterior.

CAPÍTULO 6

Conclusiones y Recomendaciones

6.1. Conclusiones

...

6.2. Recomendaciones

...

Referencias Bibliográficas

[1] Antoniou, Grigori y Van Harmelen, Frank. (2003). A Semantic Web Primer, Massachusetts: MIT Press.

[2] Cobo, Cristóbal y Pardo, Hugo. (2007) Planeta 2.0: Inteligencia Colectiva o Medios Fast Food. México DF: Grup de Recerca d'Interaccions Digitals.

[3] Éxito Explorador (Agosto 31, 2010). Estadísticas Mundiales de Internet [Datos en línea] en http://www.exitoexportador.com/stats.htm [Consulta: 2010, Noviembre 28].

[4] W3C. Guía Breve de la Web Semántica [Documento en línea]. Disponible: http://www.w3c.es/divulgacion/guiasbreves/websemantica [Consulta: 2010, Noviembre 28].

[5] Casanova, Ricardo. (2010). El Modelo Web 2.0. Presentación sobre el modelo de negocios orientado a la Web 2.0.

[6] W3C. RDF Primmer [Documento en línea]. Disponible: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ [Consulta: 2010, Noviembre 28].

[7] W3C. Resource Description Framework (RDF) [Documento en línea]. Disponible: http://www.w3.org/RDF/ [Consulta: 2010, Noviembre 29].

[8] GSI. RDF Schema (RDFS) [Presentación en línea]. Disponible: www.gsi.dit.upm.es/ gfer/ssii/RDFS.pdf [Consulta: 2010, Noviembre 30].

[9] W3C. Preguntas Frecuentes del OWL [Documento en línea]. Disponible: http://www.w3c.es/Traducciones/es/SW/2005/owlfaq [Consulta: 2010, Noviembre 30].

[10] W3C. OWL: Use Cases and Requirements [Documento en línea]. Disponible: http://www.w3.org/TR/2004/REC-webont-req-20040210/#onto-def [Consulta: 2010, Diciembre 01].

[11] W3C. SPARQL Query Language for RDF [Documento en línea]. Disponible: http://www.w3.org/TR/rdf-sparql-query/ [Consulta: 2010, Diciembre 02].

[12] Diaz, Selim. Sistemas Expertos. Un paso en la simulación del razonamiento humano [Documento en línea]. Disponible: http://www.monografias.com/trabajos23/sistemas-expertos/sistemas-expertos.shtml [Consulta: 2010, Noviembre 30]

[13] Beck, Kent et al. Manifesto for Agile Software Development [Documento en línea]. Disponible: http://agilemanifesto.org/ [Consulta: 2010, Diciembre 02].

[14] Pressman, Roger. (2010). Software Engineering: a Practitioner's Approach. New York: McGraw Hill.

[15] Sommervile, Ian. (2007). Software Engineering. New York: Pearson Education.

[16] The Friend of a Friend Project. FOAF Project [Documento en línea]. Disponible: http://www.foaf-project.org [Consulta: 2011, Enero 15].

[17] Python Software Foundation. About Python [Documento en línea]. Disponible en: http://www.python.org/about [Consulta: 2011, Agosto 08].

[18] Joyanes, Luis y Zahonero Ignacio. Programación en Java 2 (2002). Madrid: McGraw Hill.

[19] Jena Community. About Jena [Documento en línea]. Disponible: http://jena.sourceforge.net/index.html [Consulta: 2011, Agosto 08].

[20] Autor desconocido. About Sesame [Documento en línea]. Disponible: http://www.openrdf.org/about.jsp [Consulta: 2011, Agosto 08].

[21] RedLand Community. RedLand FAQ [Documento en línea]. Disponible: http://librdf.org/FAQS.html [Consulta: 2011, Agosto 08].

[22] CubicWeb Community. The Semantic Web is a Construction Game [Documento en línea]. Disponible http://www.cubicweb.org/ [Consulta: 2011, Agosto 08].

[23] Virtuoso Community. Virtuoso FAQ [Documento en línea]. Disponoble: http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSVirtuoso6FAQ [Consulta: 2011, Agosto 08].

[24] Debian Community. About Debian [Documento en línea]. Disponible: http://debian.org [Consulta: 2011, Diciembre 20]

[25] http://www.techradar.com/news/software/operating-systems/10-best-linux-distros-for-2011-704584?artc_pg=2

[26] Norvig, Peter y Russell, Stuart. Inteligencia Artificial Un Enfoque Moderno, Segunda Edición (2004). Madrid: Prentice Hall.

[27] Aho, Alfred et al. Compilers Principles, Techniques Tools, Second Edition (2007). Boston: Pearson Education.