Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Biblioteca en C++ para obtener las características de las ontologías representativas del conocimiento en la herramienta HABD.

Autores:

Amarilys Rodríguez Lorenzo.

Jeovany Jesús Cabrera Pedroso.

Tutores:

Ing. Glennis Tamayo Morales.

Ing. Flavio Roche Rodríguez.

La Habana, 2013.

Declaración de Autoría

Declaramos ser autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Amarilys Rodríguez Lorenzo

_____

[Firma Autor]

Jeovany Jesús Cabrera Pedroso

_____

[Firma Autor]

Ing. Glennis Tamayo Morales

_____

[Firma Tutor]

Ing. Flavio Roche Rodríguez

_____

[Firma Tutor]

Datos de Contacto

Tutores:

Ing. Glennis Tamayo Morales.

Universidad de las Ciencias Informáticas

Email: gtamayo@uci.cu

Ing. Flavio Roche Rodríguez.

Universidad de las Ciencias Informáticas

Email: feroche@uci.cu

De Amarilys:

Para poder lograr este sueño fue necesario el apoyo de muchas personas a las cuales les quiero agradecer:

No hay palabras en este mundo que puedan expresar mis agradecimientos a las personas más importantes en mi vida, mis padres: porque sin ellos no estaría hoy aquí, porque cada vez que les decía no puedo ellos me alentaban, me daban las fuerzas que ya no tenía para poder seguir y conseguir este sueño que ha sido de los tres, porque juntos superamos cada obstáculo que nos ha puesto la vida, gracias por ese apoyo, sacrificio y dedicación en todos estos años. Gracias por creer en mí y apoyarme en cada decisión que he tomado, los amo.

Quiero agradecerles a unas personitas que han dedicado parte de su vida a cuidarme y protegerme: a mis abuelos ellos han sido parte indispensable de mí, siempre han estado presentes en cada momento que los he necesitado. Aunque hoy mi abuela no tenga mucha idea de lo que está sucediendo quiero agradecerle de todo corazón por quererme tanto y cuidarme, a mis abuelos porque sé que hoy están orgullosos de mí.

Existen unas personas que me han dado su apoyo incondicional ellos son Dania, Miguel, Miguelito y Aylen, gracias por estar cada vez que los llamaba para decir que me sentía mal o que suspendía alguna prueba, porque sin pensarlo me ayudaban, gracias por estar conmigo en las buenas y en las malas y formar parte de mi familia.

En general quiero agradecerle a toda mi familia por todo el amor que me han dado, en especial a mi primo y a Saray por estar siempre dispuestos a ayudarme, los quiero.

Le agradezco a mi novio Edgar por estar ahí siempre para mí, por aguantar todas mis malcriadeces y mis malos ratos.

A todos mis amigos, compañeros y profesores con los que he vivido en estos 5 años.

Quiero agradecerle a mis tutores Glennis y Flavio por habernos apoyado e impulsado a lo largo de la tesis, por compartir con nosotros sus conocimientos, gracias.

De Jeovany:

Quisiera expresar mi más profundo agradecimiento a:

Mis abuelos por darme tanto amor el tiempo que los tuve conmigo espero que me estén mirando en este preciso momento y vean que su pequeño pototico ha crecido.

A mi madre, a mi hermana y a mi hermano por todo el amor que me inculcaron y la confianza depositada en mi todo el tiempo a todas horas, por todos estos años difíciles de sacrificio, no hubiese llegado hasta aquí sin ustedes.

A mi familia por todos los momentos que hemos atravesado a mi papa Julián, a mi otro papa Juan, inclusive a mi otro padre Lazarito, todos han tenido una gran influencia en mi educación.

A los que integraron mi grupo de 1er año, los que están y los que no están en la escuela en estos momentos.

A Adrián, a Helen, a Gisela, y a Nancy que aunque no están en estos momentos aquí, cuando estuvieron siempre pude contar con ellos.

A mi piquete fuerte desde 2do año a Yuned, Sureny, Yailema, Darlon, Bernal, Fernan y Yanet.

Doy gracias que nos unieran con la facultad 9 y poder conocer a Leydis, Yunet y Breissy, son un verdadero equipo ellas 3, si en estos momentos me dejaran pedir un deseo, desearía que las amistades que tengo perduren como han perdurado la de ustedes de corazón lo digo.

A Tomás, el Yaboo, Raúl y Marquitos hermanos de tantos momentos en esta escuela y compañeros excepcionales.

A la incansable Lieny y a la bella familia que tiene por todos los momentos lindos que pasamos juntos.

A mis compañeros del apartamento 96, Marinas, Osniel, el gato, Reisel farandula, el Buti record que tantas veces me ayudo con el código y no olvidar sus cabalgatas a plena luz de la tarde como dios lo trajo al mundo.

A mis rivales de football del edificio 97, el pequeño Liandry, Javico, Jose Carlo, el Yoco, que tantas veces quisieron ganarnos pero no pudieron. A mis compañeros de los equipos de football que he integrado en la universidad, especialmente al Yonly que es mi otro hermanito.

A mi entrenador Gregorio que me enseñó a caminar por los caminos duros y siempre nos aconsejó tanto en el deporte como en la vida, a todos los que de una forma u otra han influido en mi formación desde que era un niño.

A mi dúo de tesis, que tantos mensajes me mandó en todo el transcurso de la misma y lo mucho que me ayudo, diciéndome día a día que nunca nos rindiéramos.

A mis tutores que tanto me ayudaron en el transcurso de esta larga travesía, les agradezco lo mucho que hicieron por nosotros.

Y por supuesto a todos los que quizás haya olvidado en este agradecimiento.

De Amarilys:

Le dedico este trabajo de diploma a mis padres, a quienes le debo todo lo que soy hoy, porque sin ellos nada de esto fuera posible. Ellos son la fuente de mi inspiración, mi gran ejemplo a seguir como personas y profesionales. A ustedes que se han sacrificado para que yo sea una mejor persona y una profesional les dedico este trabajo, los amo con todo mi corazón.

De Jeovany:

Especialmente dedico este trabajo de diploma a tres personas muy importantes a lo largo de mi carrera no solo en la Universidad, sino desde el mismo momento que me trajo al mundo, a mi madre, mi hermana y mi hermano.

Mami espero que siempre estés a mi lado, que el tiempo se congele y te haga eterna, que la luz de dios siempre te ilumine y poder retribuirte todo lo que has hecho por mí, te quiero muchísimo.
Hermana eres un ángel que me guía en la oscuridad, de verás que sin ti no hubiera llegado hasta donde estoy mucha salud y prosperidad, siempre te tendré a mi lado.

Wacho eres el espejo que siempre miraré, no tienes manchas, eres impecable, intransigente y batallador le doy gracias a mi papá y a Noemí por haber tenido un hermano tan bueno te quiero con la vida.

A todos en especial un abrazo, un te quiero, una sonrisa, porque sé que mientras yo respire en este mundo los tendré siempre a mi lado. Muchas gracias por existir.

Resumen:

Las ontologías constituyen sistemas de representación de la información, que permiten almacenar el conocimiento, de modo que la información se encuentre ordenada y mejor procesada. La presente investigación tiene como objetivo desarrollar una biblioteca que permita obtener las características de las ontologías representativas del conocimiento, para lograr los procesos de búsqueda y visualización de ontologías en el lenguaje de programación C++. Para ello se realizó el análisis y selección de la metodología, tecnologías y herramientas para el desarrollo de la biblioteca. Se identificaron las funcionalidades que debe cumplir la biblioteca, se realizó el diseño y a partir de los patrones seleccionados se implementó la solución, logrando como resultado una biblioteca que permite obtener características en ontologías representativas del conocimiento. Para comprobar el correcto funcionamiento se realizaron pruebas unitarias y de aceptación arrojando resultados satisfactorios.

Palabras clave: Biblioteca, información, ontología.

Índice

Índice de figuras

Índice de tablas

Introducción:

El uso de las Tecnologías de la Información y las Comunicaciones (TIC) constituye un gran avance en el constante desarrollo del hombre, pues ayudan a ampliar sus capacidades físicas y mentales y juegan un importante papel para el desarrollo de la sociedad.

Con el auge y desarrollo de las TIC surgen nuevos conceptos y formas para representar la información. Las ontologías constituyen la base para soportar la representación del conocimiento, permiten almacenarlo de modo que la información se encuentre ordenada y mejor procesada. En la actualidad la utilización de las ontologías se han expandido por todo el mundo en diferentes áreas, como por ejemplo en el comercio electrónico, inteligencia artificial y en la web, permitiendo compartir opiniones sobre la estructura de información entre personas o programas de software.

En Cuba en los últimos años existe un auge en el desarrollo de software y en la utilización de las nuevas tecnologías, donde las ontologías han sido objeto de investigación en diferentes universidades del país, muestra de ello se encuentra en la universidad Marta Abreu en Villa Clara, donde se analizó la aplicación Onto-Satcol, con el objetivo de encontrar inconsistencias en esta herramienta y poder erradicarlas. La Universidad de las Ciencias Informáticas (UCI) también se encuentra inmersa en el estudio y trabajo con las ontologías, en el departamento de PostgreSQL perteneciente al Centro de Tecnologías de Gestión de Datos (DATEC) se han realizado diferentes trabajos relacionados con las ontologías, como por ejemplo el Sistema para el Grupo de Ontologías (SISGO). Este posee los siguientes módulos:

- ✓ Aplicación informática para la gestión de repositorios de ontologías representativas del conocimiento en la Web.
- ✓ Aplicación Informática para establecer semejanzas entre ontologías representativas del conocimiento en la Web.
- ✓ Aplicación informática para caracterizar las ontologías representativas del conocimiento en la Web.

- ✓ Módulo para la gestión de la información de las Bases de Datos Relacionales en ontologías representativas del conocimiento en la Web.

En dicho departamento también se han llevado a cabo proyectos como la Herramienta de Administración de Bases de Datos (HABD), que es una nueva herramienta de administración de base de datos para PostgreSQL. La misma está desarrollada en el lenguaje de programación C++ y tiene como característica fundamental una arquitectura basada en *plugins*, que permite agregar un gran número de funcionalidades, evitando que los usuarios precisen de varias aplicaciones para resolver un único problema.

Los módulos antes mencionados no han podido incorporarse a HABD, pues SISGO está implementado en el lenguaje de programación Java y HABD en C++, siendo incompatibles dichos lenguajes. De lograrse su integración posibilitaría una mayor eficiencia a la hora de buscar y recuperar información relacionada temáticamente, se lograría la validación de los datos procedentes de bases de datos, ya que las ontologías utilizan el razonamiento automático usando reglas que permiten inferir conclusiones. El uso de las ontologías desde HABD en un futuro facilitaría el trabajo, ya que realiza un procesamiento de la información más profundo y de esta forma lograr una comprensión común de contenidos ya sea entre máquinas o entre personas.

Lo anterior expuesto no ha podido materializarse, pues la herramienta HABD está implementada en el lenguaje de programación C++ y actualmente para el mismo no existen mecanismos que permitan acceder a los datos almacenados en una ontología representativa del conocimiento.

Por la situación planteada con anterioridad surge como **problema de la investigación:** ¿Cómo permitir la incorporación los módulos de la herramienta SISGO a la herramienta de Administración de Base de Datos (HABD)?.

La investigación tiene como **objeto de estudio**: La gestión de las características de las ontologías representativas del conocimiento enmarcado en el **campo de acción:** Herramientas informáticas para obtener las características de las ontologías representativas del conocimiento.

Se plantea como **objetivo general de la investigación**: Desarrollar una biblioteca en C++ que posibilite a los módulos de la herramienta SISGO una vez integrados a HABD, obtener las características de las ontologías representativas del conocimiento.

En correspondencia con el objetivo general, se plantean como **tareas a cumplir**:

- ✓ Análisis de los conceptos fundamentales de las ontologías y sus aplicaciones.
- ✓ Caracterización de las metodologías, herramientas y tecnologías a utilizar en el desarrollo de la biblioteca.
- ✓ Identificación de las funcionalidades de la biblioteca a desarrollar.
- ✓ Realización de las tarjetas CRC (Contenido, Responsabilidad y Colaboración) a partir de las funcionalidades identificadas.
- ✓ Implementación de las funcionalidades identificadas.
- ✓ Diseño de los casos de pruebas para la realización de pruebas a la biblioteca desarrollada.
- ✓ Aplicación de los casos de pruebas para validar la biblioteca desarrollada.

Arrojando como **posible resultado** una Biblioteca en C++ para obtener las características de las ontologías representativas del conocimiento.

Capítulo 1: Fundamento teórico.

En este capítulo se describe el marco teórico de la investigación, así como el análisis de conceptos, herramientas y lenguajes a utilizar en el desarrollo del trabajo de diploma.

Capítulo 2: Características del sistema.

En este capítulo se presenta la propuesta de solución al problema de la investigación planteado. Establece las principales características y funcionalidades que tendrá la solución en requisitos no funcionales e historias de usuario respectivamente. Se generan los artefactos correspondientes a las etapas de

planificación y diseño que propone la metodología de desarrollo XP, además se describen los patrones de diseño que se utilizarán.

Capítulo 3: Implementación y prueba

El contenido que se aborda en este capítulo está relacionado con la descripción de la implementación del sistema, para darle solución a las Historias de Usuario definidas en el capítulo anterior, se realizan las tareas de Ingeniería, se define el estándar de codificación y se especifican las pruebas a las que fue sometida la aplicación.

Capítulo 1: Fundamentación teórica.

Introducción.

En este capítulo se describe el marco teórico de la investigación, así como el análisis de conceptos, herramientas y lenguajes a utilizar en el desarrollo del trabajo de diploma.

1.1. Ontologías.

El término ontología viene del campo de la filosofía, donde "*Onto*", en griego significa **ser**; y "*logos*" es **tratado**, con lo cual según su etimología la ontología se ocupa del estudio del ser. De este modo la concibió el filósofo griego Aristóteles, o sea, como la ciencia que se ocupa del ser en cuanto tal.

Gruber[1] en 1993, dio una de las definiciones más empleadas: "*las ontologías se definen como una especificación explícita de una conceptualización*" [1], es decir, permite crear una estructura y contenidos de forma explícita, donde se codifican las reglas contenidas de la realidad.

En 1997, Borst[2] modificó ligeramente la definición de Gruber diciendo que: "*las ontologías se definen como una especificación formal de una conceptualización compartida*" [2], donde hace referencia a un modelo abstracto que ayuda en la identificación de conceptos relevantes de un determinado fenómeno.

El término ontología ha sido una destacada disciplina filosófica desde la Edad Media, pero no ha sido hasta el siglo XXI, que el vocablo ha adquirido un nuevo significado en el área de la informática, donde se representan los conceptos de manera que los usuarios puedan comunicarse con los sistemas y estos comprendan la terminología. Permitiendo facilitar la comunicación y la compartición de la información entre diferentes sistemas y entidades.

[1] GRUBER, Tom R. "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University, CA, 1993.

[2] Borst, W., 1997, Construction of Engineering Ontologies for Knowledge Sharing and Reuse: Ph.D. Dissertation, University of Twente.

De lo anterior se puede concluir que, las ontologías no son más que sistemas que se utilizan para representar la información, de modo tal, que esta se encuentre ordenada y mejor procesada, permitiendo a las máquinas interpretar y referenciar la información de una forma precisa.

1.1.1. Componentes de una ontología.

"*Las ontologías tienen los siguientes componentes que sirven para representar el conocimiento de algún dominio:*

- ✓ ***Conceptos:*** *son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento.*
- ✓ ***Relaciones:*** *representan la interacción y enlace entre los conceptos de un dominio. Suelen formar la taxonomía del dominio. Por ejemplo: subclase-de, parte-de, parte-exhaustiva-de, conectado-a.*
- ✓ ***Funciones:*** *son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología. Por ejemplo, pueden aparecer funciones como: asignar-fecha, categorizar-clase.*
- ✓ ***Instancias:*** *se utilizan para representar objetos determinados de un concepto. Reglas de restricción o axiomas: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Por ejemplo: "Si A y B son de la clase C, entonces A no es subclase de B", "Para todo A que cumpla la condición B1, A es C". Los axiomas, junto con la herencia de conceptos, permiten inferir conocimiento que no esté indicado explícitamente en la taxonomía de concepto."* [3]

Estos componentes serán tenidos en cuenta más adelante para el desarrollo de la biblioteca.

1.1.2. Aplicaciones de las ontologías.

El desarrollo de las ontologías ha favorecido la comunicación entre personas, organizaciones y aplicaciones, ya que proporcionan una comprensión común de un dominio, de modo que se eliminan confusiones conceptuales y terminológicas.

"En los campos de la Inteligencia Artificial, la Teoría de Decisiones[3] y la Teoría de Sistemas Distribuidos[4] (campos muy relacionados con la Web semántica), los investigadores de un campo no pueden leer fácilmente los resultados de los investigadores de los otros, pues se usan diferentes perspectivas y términos para las mismas ideas y conceptos. Construyendo una ontología común para los tres campos, las investigaciones de un campo serían inmediatamente aplicables a los otros." [4]

El mundo empresarial no es ajeno a los problemas derivados de la falta de un entendimiento común, porque algunas empresas utilizan diferentes significados, asociado a un mismo término. Mediante las ontologías se llega a una estandarización de la información, facilitando el entendimiento entre varias empresas.

En el comercio electrónico automático, si cada vendedor llama o clasifica de una manera a sus productos, resulta muy difícil automatizar las operaciones electrónicas. Para evitar la pérdida de oportunidades de negocio se debe tener en cuenta que un mismo término puede tener varios conceptos asociados.

"La Web semántica busca catalogar la información de los recursos web, páginas HTML, documentos PDF, vídeos, archivos de sonido, mediante ontologías, donde los usuarios organizarán la información de manera que los agentes de software podrán interpretar el significado y, por tanto, podrán buscar e integrar datos mucho mejor que ahora. Gracias al conocimiento almacenado en las ontologías, las aplicaciones podrán extraer automáticamente datos de las páginas web, procesarlos y sacar conclusiones de ellos, así como tomar decisiones y negociar con otros agentes o personas." [5]

[3] Concierne a la forma y al estudio del comportamiento y fenómenos psíquicos de aquellos que toman las decisiones (reales o ficticios), así como las condiciones por las que deben ser tomadas las decisiones óptimas.
[4] Sistema donde el cliente es una máquina que solicita un determinado servicio y se denomina servidor a la máquina que lo proporciona.

Las ontologías han tenido un gran impacto en numerosas disciplinas ya que se han desarrollado proyectos que facilitan y agilizan el trabajo, permitiendo una mejor comprensión entre máquinas y personas.

1.1.3. Lenguajes de representación del conocimiento.

Para explorar la Web semántica, es necesario lenguajes de marcado apropiados que representen el conocimiento de las ontologías. *World Wide Web Consortium* (W3C) publicó una especificación denominada *Resource Description Framework* (RDF) el cual es un framework para metadatos en la *World Wide Web* (WWW). RDF es la base de la mayoría de los lenguajes ontológicos de la actualidad que permiten mediante relaciones taxonómicas, crear una jerarquía de conceptos.

Este modelo se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto (conocidas en términos RDF como tripletes). Donde el sujeto es el recurso, es decir aquello que se está describiendo. El predicado es la propiedad o relación que se desea establecer acerca del recurso. Por último, el objeto es el valor de la propiedad o el otro recurso con el que se establece la relación. La combinación de RDF con otras herramientas como Lenguaje de Ontologías Web OWL permite añadir significado a las páginas, y es una de las tecnologías esenciales de la Web semántica.

El Lenguaje de Ontologías Web (OWL) está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente representar información para los humanos.

OWL añade más vocabulario para describir propiedades y clases, entre otros, relaciones entre clases (por ejemplo, desunión), cardinalidad (por ejemplo, "uno exacto"), igualdad, más tipos de propiedades, características de propiedades (por ejemplo, simetría), y clases enumeradas.

“OWL proporciona tres lenguajes, cada uno con nivel de expresividad mayor que el anterior.

- ✓ *OWL Lite está diseñado para aquellos usuarios que necesitan principalmente una clasificación jerárquica y restricciones simples. Por ejemplo, a la vez que admite restricciones de cardinalidad, sólo permite establecer valores*

cardinales de 0 o 1. Debería ser más sencillo proporcionar herramientas de soporte a OWL Lite que a sus parientes con mayor nivel de expresividad, y OWL Lite proporciona una ruta rápida de migración para tesauros y otras taxonomías. OWL Lite tiene también una menor complejidad formal que OWL DL.

- ✓ *OWL DL (Description Logic) está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (todos los cálculos se resolverán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo ciertas restricciones (por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra). OWL DL es denominado de esta forma debido a su correspondencia con la lógica de descripción (Description Logics, en inglés), un campo de investigación que estudia la lógica que compone la base formal de OWL.*

- ✓ *OWL Full está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha. OWL Full permite una ontología para aumentar el significado del vocabulario preestablecido (RDF u OWL). Es poco probable que cualquier software de razonamiento sea capaz de obtener un razonamiento completo para cada característica de OWL Full."* [6]

OWL Full puede ser considerada como una extensión de RDF, mientras que OWL Lite y OWL DL pueden ser consideradas como extensiones de una visión restringida de RDF. Cada documento OWL (Lite, DL, Full) es un documento RDF, y cada documento RDF es un documento de OWL Full, pero sólo algunos documentos RDF serán legalmente documentos OWL Lite u OWL DL.

Se escogió como lenguaje a utilizar OWL Full ya que este contiene toda la expresividad de OWL, permite mezclar definiciones de RDF con definiciones de OWL.

La siguiente figura muestra los 3 niveles de expresión del lenguaje OWL, evidenciándose que OWL Full, contiene dentro los lenguajes OWL Lite y OWL DL, por lo que admite utilizar la máxima expresividad ofrecida por este lenguaje, permitiendo desarrollar herramientas más sólidas.

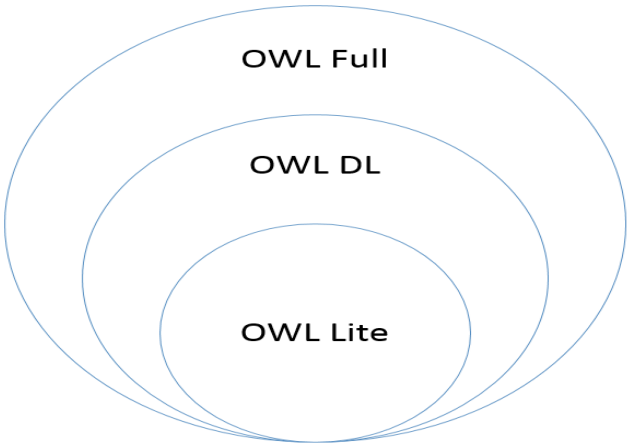

Figura 1: Niveles definidos por OWL.

1.1.4. Biblioteca para la gestión de ontologías.

"*En ciencias de la computación, una biblioteca es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular."* [7]

La mayoría de los sistemas operativos modernos proporcionan bibliotecas que implementan los servicios del sistema. De esta manera, estos servicios se han convertido en una "materia prima" que cualquier aplicación moderna espera que el sistema operativo ofrezca. Como tal, la mayor parte del código utilizado por las aplicaciones modernas se ofrece en estas bibliotecas.

Existen varias bibliotecas que realizan la gestión y manipulación de ontologías, entre ellas se encuentran 3store, Redland, Sesame, Jena y la de biblioteca para obtener las características de las ontologías representativas del conocimiento en la Web. En esta investigación se profundizará en el estudio de Sesame y Jena, ya que en la actualidad son las dos bibliotecas para la gestión de ontologías más utilizadas en el

mundo, la primera está vinculada a diferentes proyectos entre los que se encuentran, ¿Cómo encontrar dinosaurios en toda Europa?, el cual permite conectar las colecciones en línea de organizaciones de patrimonio cultural por toda Europa. Otro proyecto es el "Tesauro Rex", el cual tiene como objetivo la creación de un gran diccionario de sinónimos y "Una aguja en un pajar", este permite la búsqueda de información en grandes base de datos relacionadas con las ontologías. Por su parte Jena es utilizada por Hewlett Packard (HP) en la creación de aplicaciones, ejemplos de estas son: Twinkle, que provee una interfaz gráfica de usuario para trabajar con consultas SPARQL e Infinite Graph, aplicación para el trabajo con grafos RDF. La biblioteca para obtener las características de las ontologías representativas del conocimiento en la Web, también se tendrá en cuenta en esta investigación ya que fue desarrollada en el departamento de PostgreSQL en el curso 2011-2012.

Sesame es una biblioteca RDF, de software libre desarrollado en Java, que soporta inferencia y búsqueda sobre RDF y OWL. Fue diseñado para ser flexible, puede trabajar sobre varios sistemas de almacenamiento (bases de datos relacionales, en memoria y en sistema de archivos) brinda a los desarrolladores de aplicaciones un conjunto de herramientas útiles para trabajar individualmente con RDF, combina los mejores aspectos de otros lenguajes de consulta y proporciona acceso directo a los módulos de funcionalidades, tanto a un cliente como a un servidor, mediante las APIs de acceso. Una gran desventaja que posee esta biblioteca, es que la aplicación del motor de búsqueda es directamente dependiente del repositorio utilizado, por lo tanto la arquitectura perdería la capacidad de cambiar fácilmente entre repositorios.

Jena es un es un biblioteca desarrollada en Java, de software libre, para construir aplicaciones en la Web Semántica basada en las recomendaciones W3C. Provee una API de programación para almacenamiento (persistente y en memoria), permite gestionar todo tipo de ontologías (añadir hechos, borrarlos y editarlos), almacenarlas y realizar consultas sobre ellas. Soporta RDF y OWL y es independiente del lenguaje. Facilita crear modelos persistentes que son mantenidos de forma transparente al usuario en una base de datos relacional. A pesar del gran alcance y el alto nivel de abstracción que proporciona Jena, tiene algunas desventajas, una de ellas es que, al querer recuperar un conjunto de datos, coloca a todos los estados

en la memoria principal, causando a menudo un desbordamiento de la Máquina Virtual Java (JVM). Es decir, necesita de gran cantidad de espacio, dependiendo del número de estados que desean recuperar. La desventaja más importante es el coste del proceso de inferencia. Sin inferencia, una base de conocimiento no sería muy diferente de una base de datos común. Por lo tanto, el número de aristas en el grafo tiende a aumentar rápidamente, lo que requiere una gran cantidad de tiempo y memoria para navegar y localizar un recurso específico dentro de ella.

En Cuba, con igual objetivo que las anteriores, el Departamento de PostgreSQL de la UCI, desarrolló un trabajo de diploma denominado "**Biblioteca para obtener las características de las ontologías representativas del conocimiento en la Web**", desarrollada en el lenguaje de programación Java, permitiendo gestionar las características de las ontologías representativas del conocimiento en la Web, a través de la carga y análisis sintáctico de los ficheros OWL (ontologías) para la obtención de dichas características. Esta biblioteca realiza los procesos de almacenamiento, búsqueda, recuperación y reutilización de ontologías en el Sistema Gestor de Características de Ontologías Representativas del Conocimiento en la Web. Ella tiene como desventajas que trabaja con una API-OWL intermedia que hace que el tiempo de respuesta sea mucho más lento, pero demás, como esta API no le pertenece, la misma puede cambiar de propietario y quedar obsoleta.

En la tabla que se muestra a continuación se analizan las características de las bibliotecas Jena, Sesame y la desarrollada en el Departamento PostgreSQL, en cuanto a: arquitectura.

Tabla 1. Comparación entre las bibliotecas Jena y Sesame y la biblioteca desarrollada en el departamento PostgreSQL.

JENA	SESAME	Biblioteca para obtener las características de las ontologías representativas del conocimiento en la Web
Arquitectura		
Interacción con un	Compuesta por	Interactúa con un modelo

modelo abstracto que traslada operaciones de alto nivel a operaciones de bajo nivel sobre las sentencias que se almacenan en grafos. Permite el almacenamiento persistente. Divide la información en grafos lógicos, para cada uno se optimiza su almacenamiento según una serie de subgrafos especializados (uno para ontologías, otro para sentencias normales o cualquiera otra especialización). Permite la búsqueda concurrente. El sistema busca en cada grafo y une los resultados.	subsistemas: RDF *Model*: conjunto de parsers y escritores para varios formatos RDF. *Storage and inferencelayer* (*Sail*) API: API de bajo nivel (SPI) para almacenamiento RDF e inferenciadores. Repository API: API de alto nivel. Ofrece un gran número de métodos orientados al desarrollador para el manejo de datos RDF. HTTP*Server*: serie de *Servlets* en Java que implementan un protocolo para acceder a los repositorios de Sesame a través de http. HTTP*Client*: usado por HTTP*Repository* que es una librería de este tipo.	abstracto que traslada operaciones de alto nivel a operaciones de bajo nivel. Trabaja con la API-OWL* la cual contiene una serie de métodos que permiten leer ficheros OWL.

"API (Application Program Interface). Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones." [8]

La API proporciona un apoyo limitado para la manipulación OWL genérico. Todos los objetos que se derivan de la clase OWLResource que tiene funciones de acceso para obtener el valor de cualquier

propiedad OWL. También es posible tener acceso al modelo de datos subyacente (que se basa en Jena ahora) para búsquedas más avanzadas. Esta API cuenta con la clase ONTModel la cual soporta las sentencias que conforman OWL, como clases, propiedades, instancias, jerarquía de clases y propiedades.

En resumen, la API OWL agrega un conjunto de clases y métodos que su función es facilitar la escritura de programas.

Del estudio realizado sobre la arquitectura de estas bibliotecas, se concluye que no es aconsejable utilizar ninguna de ellas, ya que todas trabajan con una API intermedia desarrollada en Java, que tiene como desventaja la disminución de la velocidad de respuesta, debido a su elevado consumo de recursos. La nueva biblioteca deberá desarrollarse en C++ que es el lenguaje que utiliza la herramienta HABD. Para eliminar este proceso intermedio se parseará el fichero OWL, permitiendo consultar la información almacenada en ellos.

1.2 Herramienta de Administración de Bases de Datos (HABD).

La Herramienta de Administración de Base de Datos (HABD) surge a partir de los inconvenientes encontrados en herramientas como PgAdmin3, que no permiten el desarrollo de determinadas funcionalidades y obliga al usuario a apoyarse en aplicaciones externas para tener un control más completo de sus bases de datos.

HABD evita tener que contar con aplicaciones distintas que puedan resolver los problemas de administración de las bases de datos; al proponer el desarrollo de una herramienta de administración para PostgreSQL, con una arquitectura basada en *plugins*, posibilita realizar todas las funcionalidades permitidas por el gestor de manera sencilla y eficaz, contribuyendo a la integración de los mismos para lograr la realización satisfactoria de las aplicaciones y que el trabajo sea rápido y menos engorroso, facilitando a los desarrolladores incorporar nuevas funcionalidades que el usuario necesite. La aplicación posee una gran flexibilidad y una interfaz amigable que mejora el intercambio entre el usuario y el gestor PostgreSQL.

1.3 Sistema para el Grupo de Ontologías (SISGO).

SISGO es una herramienta para la gestión de ontologías que posee cuatro módulos integrados, que son:

- ✓ Aplicación informática para la gestión de repositorios de ontologías representativas del conocimiento en la Web: permite gestionar un repositorio local e incorporar contenidos de repositorios remotos con el objetivo de brindar a los usuarios la mayor cantidad de información posible sobre las ontologías.
- ✓ Aplicación Informática para establecer semejanzas entre ontologías representativas del conocimiento en la Web: el desarrollo de esta aplicación informática está orientado a establecer un índice de semejanza entre ontologías que facilite la comprensión de los conceptos en diferentes escenarios.
- ✓ Aplicación informática para caracterizar las ontologías representativas del conocimiento en la Web: permite caracterizar las ontologías representativas del conocimiento en la Web a través de su visualización y obtención de datos estadísticos.
- ✓ Módulo para la gestión de la información de las Bases de Datos Relacionales en ontologías representativas del conocimiento en la Web: permite gestionar la información de una BDR (script SQL) en una ontología (fichero OWL) mediante las reglas de transformación.

La tabla que se reseña relaciona los módulos anteriores con las funcionalidades identificadas de las bibliotecas Sesame, Jena y la desarrollada en el Departamento de PostgreSQL, con el objetivo de identificar cuáles de estas son necesarias para el correcto funcionamiento de los módulos de SISGO.

Tabla 2. Relación entre los módulos de SISGO y las funcionalidades de las bibliotecas Sesame, Jena y la desarrollada en el departamento de PostgreSQL.

	Gestión de repositorios de ontologías.	Establecer semejanzas entre ontologías.	Caracterizar las ontologías.	Gestión de la información de las Bases de Datos Relacionales en ontologías.
Importar fichero OWL.	X	X	X	X

Analizar sintácticamente el fichero OWL.	X	X	X	X
Mostrar listado de clases de la ontología.		X	X	X
Mostrar listado de instancias de una clase.		X	X	X
Mostrar árbol jerárquico de clases.		X	X	
Mostar listado de propiedades de anotación.				X
Verificar existencia de clases en la ontología.		X		X
Verificar existencia de propiedades en la ontología.		X	X	X
Verificar existencia de instancias en la ontología.		X	X	
Obtener las declaraciones de sujeto				
Obtener las declaraciones de predicado				
Obtener de datos con contexto				
Obtener de datos sin contexto				
Listar super-clases				

A partir de las revisiones bibliográficas e investigaciones realizadas sobre los módulos de la herramienta SISGO se concluye que las funcionalidades necesarias para su correcto funcionamiento son:

- ✓ Importar fichero OWL.
- ✓ Analizar sintácticamente el fichero OWL.
- ✓ Mostrar listado de clases de la ontología.
- ✓ Mostrar listado de instancias de una clase.
- ✓ Mostrar árbol jerárquico de clases.
- ✓ Mostrar listado de propiedades.
- ✓ Mostar listado de propiedades de anotación.
- ✓ Verificar existencia de clases en la ontología.
- ✓ Verificar existencia de propiedades en la ontología.
- ✓ Verificar existencia de instancias en la ontología.

Es válido resaltar que la biblioteca a desarrollar contará con una arquitectura que permitirá la incorporación de nuevas funcionalidades sin afectar el funcionamiento de las ya existentes.

1.4 Metodología de desarrollo de software.

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo producto de software, clasificándose en dos grandes grupos las metodologías agiles y las pesadas.

Las metodologías pesadas no se adaptan adecuadamente a los cambios, por lo que no son adecuados cuando se trabaja en un entorno, donde los requisitos pueden variar. Es por ello que se decide utilizar las metodologías ágiles, pues defienden la idea de que el proceso de desarrollo del *software*, se centra en el *softwar*e como tal y no en la documentación alrededor de este, por lo que le da mayor importancia a la programación que a la documentación, aunque no la obvia por completo, sino que toma en cuenta sólo la documentación necesaria y de forma muy sencilla.

1.4.1. Extreme Programming (XP)

"Extreme Programming (XP) surge como una nueva manera de encarar proyectos de software, proponiendo una metodología basada esencialmente en la simplicidad y agilidad. Las metodologías de desarrollo de software tradicionales aparecen, comparados con los nuevos métodos propuestos en XP, como pesados y poco eficientes." [9]

Se escoge como metodología de desarrollo XP puesto que:

- ✓ Se utiliza para la realización de proyectos a corto plazo: para la realización de la biblioteca se definió un tiempo de desarrollo de 10 meses de trabajo.
- ✓ Fomenta el desarrollo de equipos pequeños: XP es una metodología pensada para equipos pequeños. Para solucionar el problema de la investigación planteado el equipo cuenta con solo 2 integrantes para todo el ciclo de desarrollo del software.
- ✓ Consiste en una programación rápida o extrema: esta metodología centra la atención del equipo en desarrollar las funcionalidades y no en una profunda documentación del proceso de desarrollo. Teniendo en cuenta que se debe desarrollar la biblioteca en un tiempo estimado de 10 meses y que el equipo lo conforman 2 integrantes, es fundamental la utilización de esta metodología.
- ✓ Propone la programación en pares: este elemento se ve estrechamente relacionado con las características propias del equipo de desarrollo, teniendo en cuenta que son solo 2 personas que implementarán en parejas ya que con esto se garantiza que muchos errores sean detectados conforme son introducidos en el código, por lo tanto el número de errores del producto final es más baja, el diseño mejor y el tamaño del código menor, los problemas de programación serán resueltos más rápido y se posibilitará la transferencia de conocimientos de programación entre los miembros del equipo.
- ✓ El cliente forma parte del equipo de desarrollo: para el desarrollo de la biblioteca el cliente estará presente y disponible todo el tiempo para el equipo. Gran parte del éxito de la utilización de esta metodología se debe a

que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada.

Además es válido destacar que el equipo que desarrollará la solución cuenta con más de 1 año de experiencia en la utilización de esta metodología pues en la línea a la que pertenece es XP la utilizada en el desarrollo de los proyectos.

1.5. Tecnologías y herramientas a utilizar.

La selección de las herramientas y tecnologías a emplear en el desarrollo de la biblioteca se realizó teniendo en cuenta la compatibilidad de estas con las definidas para el desarrollo de la herramienta HABD. Además de que son las definidas en la arquitectura del proyecto.

1.5.1. Herramientas CASE.

Se puede definir a las Herramientas CASE (*Computer Aided Software Engineering*) como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software, permiten incrementar la productividad y el control de la calidad en cualquier proceso de elaboración. Facilitan la realización de prototipos, el desarrollo conjunto de aplicaciones y la reutilización de componentes.

"***Visual Paradigm*** *es una herramienta de diseño UML y herramienta CASE, profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Diseñada para la ayuda al desarrollo de software, potente, gratuito, fácil de instalar, utilizar y actualizar. Ofrece una serie de facilidades para generar informes que permiten documentar el proyecto como son PDF y Word*." [10]

Características:

- ✓ Producto de calidad.

- ✓ Soporta aplicaciones web.
- ✓ Se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE, Oracle JDeveloper, BEA Weblogic.
- ✓ Fácil de instalar y actualizar.
- ✓ Compatibilidad entre ediciones.

1.5.2. Lenguaje Unificado de Modelado (UML) 2.0.

UML (*Unified Modeling Language*) es un lenguaje de modelado de sistemas de software conocido y utilizado a nivel mundial. Es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software. Permite a los desarrolladores visualizar el producto de su trabajo (artefactos) en esquemas o diagramas estandarizados. Es un lenguaje consolidado no solo en el mundo sino también en la UCI. Se ha convertido en poco tiempo en una notación estándar para la comunidad de investigadores en ingeniería del software.

“Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- ✓ Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- ✓ Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- ✓ Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- ✓ Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.” [11]

1.5.3. Lenguaje de programación C++.

“C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese

sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido." [12]

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT (enlace externo).

Una particularidad del lenguaje C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. C++ está considerado por muchos como un lenguaje potente, debido a que permite trabajar tanto a alto como a bajo nivel.

La biblioteca será desarrollada en este lenguaje debido a que la herramienta para la administración de base de datos HABD y los módulos que se desarrollarán, están implementados en el mismo.

1.5.4. Framework de desarrollo.

Los frameworks son diseñados para facilitar el desarrollo de software, ofreciéndole así a los programadores y diseñadores tener más tiempo para identificar los requerimientos del software, además representan una arquitectura de software que mediante esta se pueden modelar las relaciones generales de las entidades del dominio.

*"**Qt** es un toolkit (conjunto de herramientas) de plataforma cruzada (cross-platform) desarrollado por Trolltech (empresa Noruega) para construir GUIs (Interfaz Gráfica de Usuario) en C++. Es un software multiplataforma ajustable a Microsoft Windows, Linux, Mac OS X."* [10]

Este es distribuido bajo ediciones comerciales o bien Open Source. Incluye una extensa librería de clases C++ y utilidades para construir las aplicaciones de forma rápida y sencilla. Se puede utilizar para el diseño de interfaz gráfica de usuario, o para crear aplicaciones completas con su apoyo a la integración con los entornos de desarrollo integrados (IDEs). Las aplicaciones Qt pueden ampliar su funcionalidad mediante plugins y librerías dinámicas. Los plugins proporcionan *codecs* (funcionalidades) adicionales, conductores de base de datos, formatos de imagen y estilos.

1.5.5. Entorno de desarrollo integrado (IDE).

Un Entorno de Desarrollo Integrado o IDE (por sus siglas en inglés), es una herramienta que permite a los desarrolladores de software escribir sus programas en uno o más lenguajes. Consiste básicamente en una plataforma en la que se integran un editor de código, un compilador, un depurador y una interfaz gráfica de usuario.

*"**Qt Creator** es un excelente IDE multiplataforma para desarrollar aplicaciones en C++ de manera sencilla y rápida. Está basado en la librería QT y posee herramientas para la administración y construcción de proyectos, depurador visual, diseñador de formularios y auto-completado de código. Posee un avanzado editor de código en C++. Además soporta los lenguajes: C#/.NET Languages (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby. Permite realizar programación visual y programación dirigida por eventos."* [13]

1.6. Conclusiones del capítulo.

En este capítulo se abordaron los principales conceptos sobre las ontologías, sus componentes, aplicaciones de las mismas y los lenguajes de representación del conocimiento.

Del análisis realizado sobre las herramientas Sesame, Jena y de la "Biblioteca para obtener las características de las ontologías representativas del conocimiento en la Web" y los módulos de SISGO, se obtuvieron 10 funcionalidades necesarias para el desarrollo de la biblioteca

Con el objetivo de obtener un desarrollo de software de forma rápida y extremo se emplea como metodología de desarrollo XP, Visual Paradigm 8.0 como herramienta CASE y UML 2.0 para el modelado. El lenguaje de programación será C++, como framework de desarrollo Qt 2.5 utilizando como IDE de desarrollo Qt Creator 4.7.

.

Capítulo 2: Características del sistema.

Introducción.

En el capítulo se presenta la propuesta de solución al problema de la investigación planteado. Se establecen las principales características y funcionalidades con que contará la solución en requisitos no funcionales e historias de usuario respectivamente. Se generan los artefactos correspondientes a las etapas de planificación y diseño que propone la metodología de desarrollo XP, además se describen los patrones de diseño que se utilizarán.

2.1. Modelo de dominio.

"*Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés*." [14]

El modelo de dominio permite centrarse en el comportamiento del sistema y en el flujo de datos que lo hace funcionar, además nos ayuda a obtener un modelo de objetos que es una representación del modelo conceptual del sistema. Describe las entidades que participan en el sistema, las relaciones y el flujo de datos que existe entre ellas, dichas entidades se mapean en clases que se componen de propiedades y métodos.

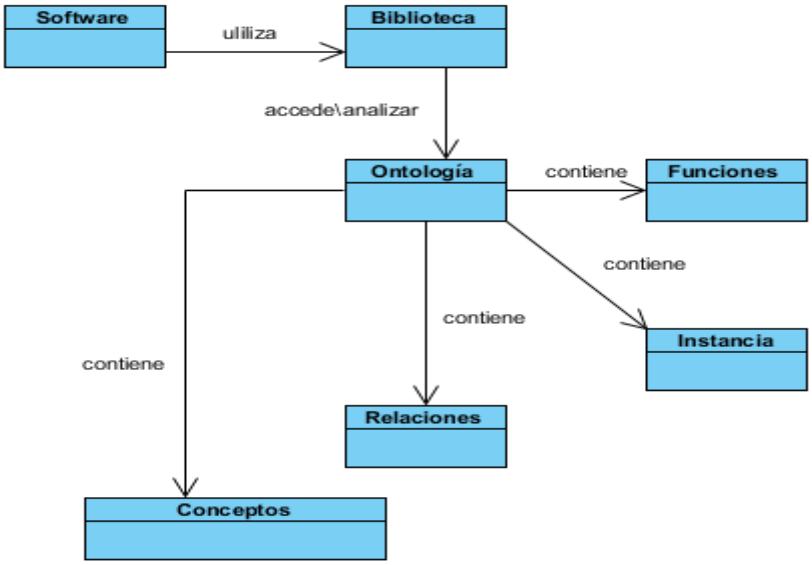

Figura 2: Modelo de Dominio.

Software: el software estará contenido por los cuatro módulos de la herramienta SISGO.

Biblioteca: es la encargada de analizar sintácticamente los ficheros OWL.

Ontología: contiene los conceptos, relaciones, funciones e instancias que permite el análisis de los ficheros OWL.

Conceptos: representan las ideas que se intentan formalizar.

Relaciones: representan la relación entre los conceptos.

Funciones: representan un tipo concreto de relación.

Instancias: representan un objeto determinado de un concepto.

El modelo de dominio presenta un software, el cual está integrado por los 4 módulos de SISGO, este utiliza una biblioteca para acceder a los componentes de las ontologías representativas del conocimiento, la cual va a contener conceptos, relaciones, funciones e instancias.

2.2. Descripción del modelo propuesto.

Se propone desarrollar una biblioteca que permita gestionar las características de las ontologías representativas del conocimiento. Una vez integrada a HABD permitirá a los módulos de SISGO la carga y análisis sintáctico de los ficheros OWL.

Permitiendo a los usuarios consultar listados de clases, instancia y propiedades, así como verificar la existencia de las mismas. Además muestra en forma de árbol jerárquico la lista de las clases existentes en el fichero. Esta biblioteca podrá lograr los procesos de búsqueda y visualización de ontologías en el lenguaje de programación C++.

2.3. Historia de usuario.

“Las Historias de Usuario (HU) es uno de los artefactos más importantes que genera la metodología XP. Éstas tienen similar propósito que los casos de uso y son confeccionadas por el cliente. Expresan su punto de vista sobre las necesidades del sistema. Son descripciones cortas y escritas en el lenguaje del usuario sin

terminología técnica que proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conllevará su implementación." [15]

Las HU describen las funcionalidades que realiza la biblioteca para la gestión de ontologías representativas del conocimiento. Se especifica el grado de importancia (prioridad) que estas alcanzan en el desarrollo, el tiempo estimado de duración para la implementación de las mismas, se detalla la acción a realizar por cada HU, dejando plasmado en las observaciones los inconvenientes que se pueden presentar para la no realización exitosa. En resumen proveen información al desarrollador que implementará dichas funcionalidades.

Para el presente trabajo de diploma se obtuvieron un total de 10 HU para ser implementadas. Las HU identificadas son: 1- Importar fichero OWL, 2- Analizar Sintácticamente el fichero OWL, 3- Mostrar listado de clases de la ontología, 4- Mostrar listado de propiedades de la ontología, 5- Mostrar listado de instancias de una clase, 6- Mostrar árbol jerárquico de clases, 7- Mostrar listado de propiedades de anotación, 8- Verificar existencia de clase en la ontología, 9- Verificar existencia de propiedad en la ontología y 10- Verificar existencia de instancia en la ontología. Estas HU alcanzan un grado de importancia de muy alta, alta y media, y no existe ninguna de ellas que la prioridad en el negocio sea baja, porque es de suma importancia su realización exitosa.

La tabla que se muestra sobre la HU Importar fichero OWL, estará implementada en la primera iteración, con una prioridad muy alta y una duración de una semana.

Tabla 3. Historia de Usuario: Importar fichero OWL.

Historia de Usuario	
Número:1	**Nombre Historia de Usuario:** Importar fichero OWL
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Amarilys Rodríguez Lorenzo	**Iteración Asignada:** 1
Prioridad en Negocio: Muy Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Medio	**Puntos Reales:** 1
Descripción: Esta funcionalidad permite importar el fichero OWL a través de la especificación de su localización.	

Observaciones:
Prototipo de interfaces: No presenta interfaz de usuario.

Para ver el resto remitirse a los documentos complementarios, Planilla de Historias de Usuario del Expediente de proyecto.

2.4. Lista de reserva del producto.

La lista de reserva del producto agrupa los requisitos funcionales organizados por prioridad según la complejidad en el negocio y los requisitos no funcionales con una breve descripción de cada uno de ellos.

"Los requisitos funcionales de un sistema describen lo que el sistema debe hacer." [16]

"Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema." [16]

La etapa de definición de requisitos es una tarea de suma importancia a la hora de desarrollar un sistema. Esta consiste en generar una definición clara y precisa de los aspectos relevantes del producto, cuanto más alto sea el grado de cumplimiento y aceptación de los requerimientos del cliente, más alta será la calidad del sistema desarrollado. Seguidamente se expone el listado de los requerimientos en la lista de reserva del producto.

Tabla 4. Lista de reserva del producto.

Prioridad	Ítem	Descripción	Estimación	Estimado por
Muy Alta				
	1	Importar fichero OWL	1 semana	Programador
	2	Analizar Sintácticamente el fichero OWL	1 semana	Programador
Alta				

	3 4 5 6 7	Mostrar listado de clases de la ontología Mostrar listado de propiedades de la ontología Mostrar listado de instancias de una clase Mostrar árbol jerárquico de clases Mostrar listado de propiedades de anotación	1 semana 1 semana 1 semana 1 semana 1 semana	Programador Programador Programador Programador Programador
Media				
	8 9 10	Verificar existencia de clase en la ontología Verificar existencia de propiedad en la ontología Verificar existencia de instancia en la ontología	0.6 semana 0.6 semana 0.8 semana	Programador Programador Programador
Requisitos no funcionales				
	11 12	**Software** • Máquina instalada con el sistema operativo Ubuntu GNU/Linux en su versión 11.4 o superior, Qt-SDK como maquina virtual de Qt y GCC como compilador de C++. **Hardware** • El ordenador debe tener como mínimo un microprocesador con una velocidad de 2.0 GHz, capacidad libre en disco duro de 5 MB y 512 GB de RAM.		

2.5. Plan de iteraciones.

Después de describir e identificar las HU se procede a la planificación de la etapa de implementación del sistema. Este plan especifica exactamente cuáles HU serán implementadas para cada iteración.

Para el desarrollo del plan de iteraciones se definieron 3 iteraciones, la primera con una duración de 3 semanas, la segunda de 4 semanas y la última de 2 semanas, para un total de 9 semanas.

Tabla 5. Plan de iteraciones.

Iteración	Orden de las Historias de Usuario	Duración de las iteraciones
1	Importar fichero OWL Analizar sintácticamente fichero OWL	2 semanas
2	Mostrar listado de clases de la ontología Mostrar listado de propiedades de la ontología Mostrar listado de instancias de una clase Mostrar árbol jerárquico de clases Mostrar listado de propiedades de anotación	5 semanas
3	Verificar existencia de clase en la ontología Verificar existencia de propiedad en la ontología Verificar existencia de instancia en la ontología	2 semanas
Total		**9 semanas**

2.6. Arquitectura de software.

"Según la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) la arquitectura de software es la organización de un sistema en términos de sus componentes de software, incluyendo los subsistemas y las relaciones e interrelaciones entre ellos, y los principios que guían el diseño de ese sistema de software." [17]

2.6.1. Estilos arquitectónicos.

Un estilo arquitectónico es "*Una familia de términos de un patrón de organización estructural. Más específicamente, un estilo arquitectónico determina el vocabulario de los componentes y conectores que se pueden utilizar en casos de ese estilo,*

junto con un conjunto de restricciones sobre la forma en que se pueden combinar." [18]

Entre los estilos arquitectónicos se encuentran el orientado a objetos, el estratificado y el de llamada y retorno que es el utilizado, pues permite obtener una estructura de programa que resulta relativamente fácil de modificar y cambiar de tamaño. Es el estilo más generalizado en sistemas en gran escala. Dentro del estilo llamada y retorno se encuentra la arquitectura basada en componentes la cual será empleada en la realización de la biblioteca.

Arquitectura Basada en Componentes

"La Arquitectura Basada en Componentes consiste en una rama de la Ingeniería de software en la cual se trata con énfasis la descomposición del software en componentes funcionales. Esta descomposición permite convertir componentes pre-existentes en piezas más grandes de software. Este proceso de construcción de una pieza de software con componentes ya existentes, da origen al principio de reutilización del software, mediante el cual se promueve que los componentes sean implementados de una forma que permita su utilización funcional sobre diferentes sistemas en el futuro." [19]

Existen 4 principios definidos por Clemens Szyperski and David Messerschmitt[5], que definen a un componente de software como elemento de la arquitectura:

- ✓ Múltiple uso: se refiere al hecho de que un componente es escrito dentro de un contexto que permita que su funcionalidad sea útil en la creación de distintas piezas de software.
- ✓ Contexto no específico: en relación con la orientación conceptual de la especificación de un componente, debe estar planteada de una forma general que permita su adaptación en distintos sistemas, sin que el contexto tenga prioridad.
- ✓ Encapsulación: se refiere a la especificación interna oculta o no investigable a través de la interface. Así se protege que el resto de componentes y piezas

[5] Ingeniero y profesor emérito en la Universidad de California, Berkeley , en el Departamento de Ingeniería Eléctrica y Ciencias de la Computación.

de software dentro de un sistema, no se vean afectados por cambios en el diseño de uno de los componentes.

- ✓ Una unidad independiente de desarrollo con su propio control de versiones: este principio está muy relacionado con la encapsulación, permite que un componente pueda ser desarrollado de manera independiente, cambiando el diseño o agregando nuevas funcionalidades, sin afectar significativamente el resto del sistema.

2.7. Patrones de diseño.

"*Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema general de diseño en un contexto particular.*" [20]

Con el uso de patrones de diseño se evita la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente. Formalizando un vocabulario común entre diseñadores y estandarizar el modo en que se realiza el diseño.

2.7.1. Patrones GRASP.

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones.

GRASP es un acrónimo que significa *General Responsibility Assignment Software Patterns*. El nombre se eligió para indicar la importancia de captar estos principios, con el objetivo de diseñar eficazmente el software orientado a objetos.

Patrón Experto

El patrón experto se usa al asignar responsabilidades. Ofrece como solución asignar las responsabilidades a las clases que tienen la información necesaria para cumplir con estas, para las cuales son creadas, sin depender de ninguna otra. Es un principio básico que suele utilizarse en el diseño orientado a objetos. El uso de este patrón se evidencia en la clase OWL_Reader y la clase Instancia, donde esta clase es la encargada de contener toda la información referente a las instancias.

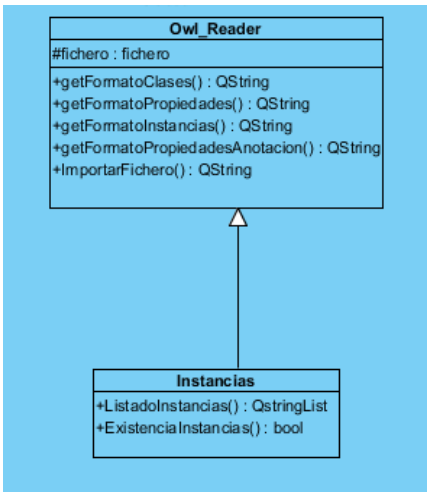

Figura 3: Fragmento del diagrama de clases donde se evidencia la utilización del patrón GRASP: Experto.

Patrón Controlador

El patrón controlador plantea la asignación de las responsabilidades de manejar un sistema a una clase. Con el uso de este patrón se puede obtener como beneficio el incremento del potencial de los elementos que pueden ser reutilizados. El uso de este patrón se evidencia en la clase OWL_Reader la cual es la encargada de controlar todo el flujo de datos de la biblioteca.

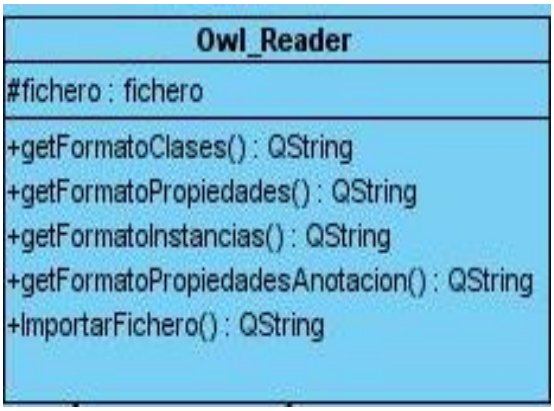

Figura 4: Fragmento del diagrama de clases donde se evidencia la utilización del patrón GRASP: Controlador.

Alta Cohesión

El patrón alta cohesión es la meta principal que ha de buscarse en todo momento, se debe tener presente en todas las decisiones de diseño. El alto nivel de cohesión es presentado por las clases que tienen responsabilidades moderadas en un área funcional y colaboran con otras para llevar a cabo las tareas, no donde dichas clases abarcan el volumen de las responsabilidades a realizar sin importar su complejidad. Este patrón se evidencia en la clase Propiedad ya que esta es la encargada de realizar todo lo referente a las propiedades en la biblioteca.

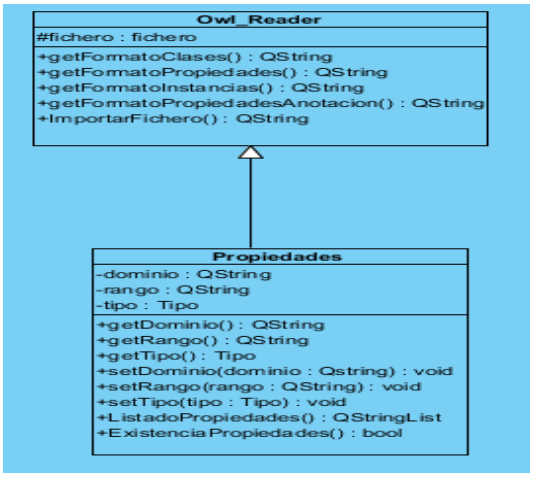

Figura 5: Fragmento del diagrama de clases donde se evidencia la utilización del patrón GRASP: Alta Cohesión.

Bajo Acoplamiento

El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de éstas no incremente el acoplamiento, es decir, significa asignar una responsabilidad para mantener pocas dependencias entre las clases. Se puede evidenciar el uso de este patrón en la clase OWL_Reader y Propiedades puesto que OWL_Reader tiene el número mínimo de dependencias con otras clases.

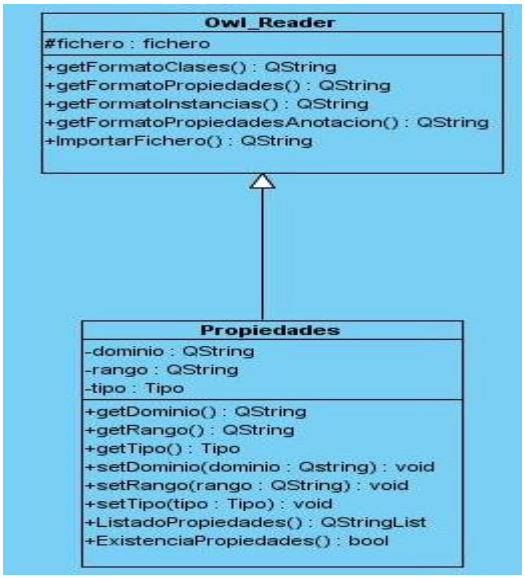

Figura 6: Fragmento del diagrama de clases donde se evidencia la utilización del patrón GRASP: Bajo Acoplamiento.

2.7.2 Patrones GoF.

Fue por los años 1994, que apareció el libro *Design Patterns: Elements of Reusable Object Oriented Software* escrito por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Estos expertos recopilaron y documentaron 23 patrones de diseño aplicados usualmente por diseñadores de software orientado a objetos. [21]

Los patrones de diseño del grupo de GoF se clasifican en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento. En la biblioteca se aplica el patrón observador, que se encuentra dentro del grupo de los de comportamiento.

Comportamiento: Los patrones de comportamiento facilitan y definen la comunicación e iteración entre los objetos de un sistema y que estos estén lo menos mezclados posible.

Observador: Este patrón tiene la responsabilidad de que al cambiar el estado de un objeto todas las dependencias del mismo se actualicen. Este patrón QT lo trae por defecto, en este trabajo se emplea en todas las interfaces, ya que todas emiten señales. Utiliza el mecanismo de la interacción con las clases visuales mediante las señales y los Slot donde de forma asincrónica los objetos que están suscritos a esa

señal se enteran del cambio de estado. En la programación de Interfaces Gráficas de Usuario (GUI) se necesita que los cambios o eventos producidos sobre un objeto sean comunicados al resto de ellos. Qt posee el mecanismo de *Signals* y *Slots*, el cual posibilita entre estos una comunicación segura, flexible y orientada a objetos. Básicamente, al producirse un evento sobre un objeto se emite una señal (*Signal*), que permite que se ejecute una determinada función (*Slot*) en respuesta a la señal emitida, debido a ello el patrón observador se encuentra inherente en Qt..

```
connect(ui->actionVerificar_Existencia_de_Propiedades,SIGNAL(triggered())
  ,this,SLOT(VerificarExistenciaPropiedades1()));

 connect(ui->actionMostrar_Listado_de_Propiedades_Anotacion,SIGNAL(triggered())
  ,this,SLOT(MostrarListadoDePropiedadesDeAnotacion1()));

 connect(ui->actionVerificar_Existencia_de_Propiedades_de_Anotacion,SIGNAL
   (triggered()),this,SLOT(VerificarExistenciaPropiedadesDeAnotacion1()));
```

Figura 7: Ejemplo donde se evidencia el patrón Observador.

2.8. Modelo de diseño.

"Para el diseño de aplicaciones informáticas la metodología XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML. En su lugar se usan otras técnicas como las tarjetas CRC. No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante." [22]

2.8.1. Tarjetas CRC.

Las tarjetas CRC trabajan con una metodología basada en objetos, estas garantizan que el equipo completo contribuya en la tarea del diseño.

Las tarjetas CRC están divididas en 3 partes:

Clase: Representa una colección de objetos similares.

Responsabilidades: Describen las funciones que debe realizar una clase, es aquello que la clase sabe o hace.

Colaboraciones: Describen las demás clases con las que trabaja una clase en conjunto para llevar a cabo sus responsabilidades.

Para el presente trabajo se identificaron un total de 6 tarjetas CRC: 1-OWLParser, 2-OWLRedear, 3- Clases, 4- Instancias, 5- Propiedades_Anotación, 6- Propiedades.

Seguidamente se muestra el ejemplo de la tarjeta CRC OWLParser.

Tabla 6. Tarjeta CRC "OWLParser".

Tarjeta CRC	
Clase: OWLParser	
Responsabilidades	**Colaboraciones**
Permite la carga dinámica de cualquier fichero OWL	OWLRedear

Para ver el resto remitirse a los documentos complementarios, Planilla Modelo de diseño del Expediente de proyecto.

2.8.2. Diagrama de clases.

"El diagrama de clases es una herramienta esencial durante el proceso de análisis y diseño del sistema (…). Representa de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, y sus relaciones con los demás en el modelo." [23]

El Diagrama de Clases es el diagrama principal para el análisis y diseño. Representa las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. Aunque la metodología XP no define un diagrama de clases del sistema, se presenta a continuación el diagrama realizado para un mejor entendimiento de la aplicación, puesto que este se puede utilizar siempre y cuando contribuya para el mejoramiento de la comunicación y comprensión del mismo ya que es una guía esencial para los desarrolladores a la hora de implementar. El diagrama cuenta con 6 clases, está compuesto por las relaciones de herencia y composición. A continuación se detalla una breve descripción de las principales clases del diagrama.

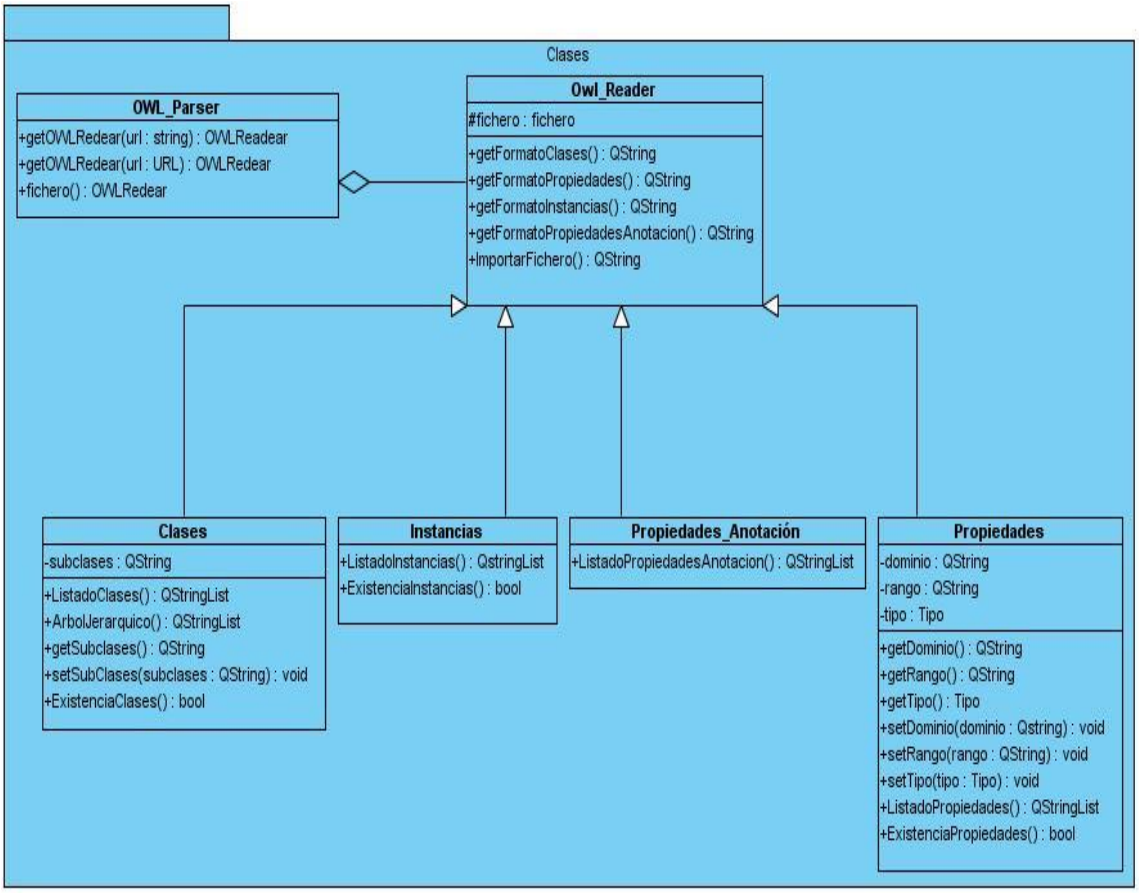

Figura 8: Diagrama de clases.

OWL_Parser permite la carga dinámica de cualquier fichero OWL, especificando una dirección URL.

OWL_Reader está compuesta por un fichero OWL y por métodos que son necesarios en la implementación de las funcionalidades necesarias para analizar sintácticamente el fichero OWL.

Clases es la encargada de verificar la existencia de las clases, mostrar el listado de las mismas y un árbol jerárquico.

Instancias se encarga de listar las instancias y verificar la existencia de las mismas.

Propiedad_Anotación permite mostrar un listado de propiedades de anotación.

Propiedades es la encargada de mostrar una lista de propiedades y verificar la existencia de las mismas.

2.9. Conclusiones del capítulo.

En este capítulo se desarrollaron los artefactos que genera la metodología XP, identificándose 10 Historias de Usuario, se planificó su implementación para 3 iteraciones comenzando por las HU de mayor prioridad del negocio. Se confeccionaron 6 tarjetas CRC. Se definieron como patrón arquitectónico el basado en componentes, describiéndose a través de los patrones de diseño GRASP, identificándose los patrones experto, controlador, alta cohesión y bajo acoplamiento y los GoF, mediante el patrón observador.

Capítulo 3: Implementación y prueba.

Introducción.

El contenido que se aborda en este capítulo está relacionado con la descripción de la implementación del sistema, para darle solución a las Historias de Usuario definidas en el capítulo anterior, se realizan las tareas de Ingeniería, se define el estándar de codificación y se especifican las pruebas a las que fue sometida la aplicación.

3.1. Tareas de ingeniería.

Para lograr desarrollar las HU definidas en el capítulo dos, con la calidad requerida, se desglosan en tareas de ingeniería, y se les asignan a los programadores para que sean implementadas durante cada iteración, de esta forma el programador obtiene una mejor visión a la hora de implementar las HU.

La tabla que se muestra refleja la Tarea de Ingeniería número 1 vinculadas a la HU Importar Fichero OWL, con un tiempo estimado de una semana.

Tabla 7. Tarea de ingeniería "Importar fichero OWL".

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Importar fichero OWL	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 4/02/2013	**Fecha fin:** 8/02/2013
Programador Responsable: Amarilys Rodríguez Lorenzo	
Descripción: Se introduce el *String* con la ruta de localización del fichero OWL, de esta manera se importa la información del mismo guardándose en un objeto *TextStream* para su posterior análisis sintáctico.	

Las tablas que se muestran reflejan las Tareas de Ingeniería 1 y 2 de la HU, Mostrar listado de clases de la ontología, con un tiempo estimado de una semana.

Tabla 8 Tarea de ingeniería "Obtener listado de clases".

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 3
Nombre Tarea: Obtener listado de clases	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6
Fecha inicio: 18/02/2013	**Fecha fin:** 20/02/2013
Programador Responsable: Jeovany Jesús Cabrera Pedroso	
Descripción: Después de haber cargado el fichero OWL con el método *CargarFicheroOwl*, se obtiene un *fichero* con toda la información de la ontología. A partir de dicha captura se parsea el fichero OWL y se construye una lista con el nombre de las clases existentes en dicho fichero.	

Tabla 9 Tarea de ingeniería "Mostrar listado de clases".

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 3
Nombre Tarea: Mostrar listado de clases	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.4
Fecha inicio: 21/02/2013	**Fecha fin:** 22/02/2013
Programador Responsable: Jeovany Jesús Cabrera Pedroso	
Descripción: Se retorna el *QlistString* de las clases obtenidas.	

Para ver el resto remitirse a los documentos complementarios, Planilla de Tareas de Ingeniería del Expediente de proyecto.

3.2. Estándares de codificación.

"Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código." [24]

Los estándares de codificación permiten entender de manera rápida, fácil y sencilla, el código empleado en el desarrollo de un *software*. Es de gran importancia el usar técnicas de codificación y realizar buenas prácticas de programación con vistas a generar un código de alta calidad. Si se aplica de forma continua un estándar de codificación bien definido, y posteriormente se efectúan revisiones del código, caben muchas posibilidades de que un proyecto de *software* se convierta en un sistema fácil de comprender y de mantener, garantizando un mantenimiento óptimo de dicho código por parte del programador. A continuación se presenta un fragmento del estándar definido para la implementación del plugin.

Identación

La unidad de identado es de 4 espacios. El uso de la tabulación debe ser evitado pues no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

```
if(lista.length()>1)
{
....QString lista2=lista[0];
    QString lista3=lista[1];
    if( lista2.startsWith(formatoPropiedadesdeAnotacion))
    {
    ....lista1=lista3.left(lista3.size()-3);

        istaFinal.append(lista1);
    }
```

Figura 9: Ejemplo de código donde se evidencia la identación.

Longitud de la Línea

Evitar líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta deber ser separada en otras. Realizar la separación después de un operador, preferentemente luego de una coma. Realizar la ruptura después de un operador disminuye la probabilidad de que al realizar un copiado del código se cometa un error por olvido de la inserción del punto y coma. La siguiente línea debe ser identada con 5 espacios.

```
while(!fichero.atEnd())
{
    QString formatodePropiedadesAnotacion="<DataProperty IRI=";
    QString linea=fichero.readLine().trimmed();
    if(linea.startsWith(formatodePropiedadesAnotacion))
    {
        resultado=true;
    }
```

Figura 10: Ejemplo de código donde se evidencia la longitud de la línea.

Comentarios

Es conveniente dejar información que pueda ser leída tiempo después por personas (posiblemente la misma persona que programó) que necesitan entender que fue lo que se hizo en el fragmento de código. Los comentarios deben ser escritos correctamente y claros. Generalmente deben usarse comentarios de una sola línea. Se debe reservar los comentarios de bloques para la documentación formal o para comentar porciones de código.

```
if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    qFatal("No pudo abrir el fichero");
    QString fichero= archivo.readAll(); // Lee todo y lo guarda en un QString
```

Figura 11: Ejemplo de código donde se evidencia los comentarios.

Declaración de variables

Cada variable debe de ser declarada en una línea y comentada. También deben aparecer ordenadas alfabéticamente: El nombre de las variables debe de comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula.

```
QStringList MainWindow::MostrarListadoDeInstancias() {
    QString dre=DireccionUrl();
```

Figura 12: Ejemplo de código donde se evidencia la declaración de variables.

Declaración de funciones

No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros. Debe haber un espacio entre el paréntesis derecho y la llave de comienzo del cuerpo de la función. Las sentencias del cuerpo deben estar en la línea siguiente. La llave que cierra debe estar alineada con la línea de declaración de la función. Los nombres de las funciones se rigen por las mismas características que las variables.

```
QStringList MainWindow::MostrarListadoDeInstancias() {
```

Figura 13: Ejemplo de código donde se evidencia la declaración de funciones.

Identificadores

Los identificadores pueden estar formados por cualquiera de las 26 letras minúsculas o mayúsculas (A…. Z, a…. z), los 10 dígitos (0... 9) y el carácter subrayado “_”. Debe evitarse el uso de caracteres internacionales (ej. ñ, ü) porque no siempre pueden ser leídos o entendidos correctamente en todos los lugares. No se debe usar el símbolo dólar “$” o la barra invertida “\” en los identificadores.

```
while(!fichero.atEnd())
{
    QString formatoInstancias="<NamedIndividual IRI=";
    QString lista1;
    QString linea=fichero.readLine().trimmed();
    QStringList lista=linea.split("#");
```

Figura 14: Ejemplo de código donde se evidencia los identificadores.

Etiquetas

Las sentencias etiquetadas son opcionales. Solo estas sentencias deben ser etiquetadas: while, do, for, foreach, switch.

```
for(int i=0;i<listaFinal.size();i++)
{
    if(!aux.contains(listafinal.at(i)))
    {
        aux.append(listaFinal.at(i));
    }
}
```

Figura 15: Ejemplo de código donde se evidencian las etiquetas.

Sentencia return

Una sentencia return no debe utilizar paréntesis "()" alrededor del valor que se retorna. La expresión cuyo valor se retorna debe comenzar en la misma línea que la palabra reservada return, terminada con un punto y coma.

```
    for(int i=0;i<listaFinal.size();i++)
    {
        if(!aux.contains(listafinal.at(i)))
        {
            aux.append(listaFinal.at(i));
        }
    }
    return aux;
}
```

Figura 16: Ejemplo de código donde se evidencia la sentencia return.

Espacios en blanco

Las líneas en blanco facilitan la lectura determinando secciones de código lógicamente relacionados. Los espacios en blanco pueden ser (o no deben ser) utilizados en las siguientes circunstancias:

- ✓ No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre la lista de parámetros. Esto ayuda a distinguir entre palabras reservadas y llamadas a funciones.
- ✓ Para cualquier operador binario excepto el punto ".", el paréntesis que abre "(" y el corchete que abre "[" todos deben ser separados por un espacio entre operandos y operador.
- ✓ No se debe utilizar el espacio para separar un operador unario de su operando excepto cuando ese operador es una palabra como typeof.

- ✓ Cada punto y coma ";" en una sentencia de control debe ser seguido por un espacio.
- ✓ Debe dejarse un espacio luego de cada coma ",".

3.3. Validación del sistema.

La validación y comprobación del funcionamiento del sistema antes que pueda ser liberado, consiente en aumentar su calidad, reduciendo el número de errores no detectados. Permite aumentar la seguridad para evitar efectos colaterales no deseados a la hora de realizar modificaciones.

Uno de los pilares fundamentales de la metodología XP, es proporcionar al cliente la posibilidad de concretar las funcionalidades de las HU y verificarlas en el proceso de pruebas.

3.3.1. Pruebas de software.

"Las pruebas de software constituyen un pilar indispensable para evaluar y determinar la calidad de un software. Concretamente se puede definir pruebas de software como:

- ✓ *El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final.*
- ✓ *Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente."* [25]

La metodología XP propone para validar las necesidades de los usuarios y dirigir la implementación las pruebas de aceptación y las unitarias. Las pruebas de aceptación están destinadas a evaluar la funcionalidad requerida del software. Son definidas y diseñadas por el cliente, permitiendo validar todas las funcionalidades. Posibilitan que los programadores estén al tanto de lo que resta por realizar. Su principal objetivo es asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas.

Las unitarias están encomendadas a verificar el código, son diseñadas por los programadores, garantizan que un determinado módulo cumpla con un

comportamiento esperado antes de ser integrado al sistema. Se emplean cuando la implementación es complicada y la interfaz de un método no es clara.

3.3.2. Pruebas unitarias

En el desarrollo de aplicaciones las pruebas unitarias tienen un papel crítico para asegurar el nivel apropiado de calidad, se tienen que poder ejecutar sin necesidad de intervención manual. Esta característica posibilita automatizar su ejecución. Dichas pruebas tienen que poder repetirse tantas veces como uno quiera. Por este motivo, la rapidez de las pruebas tiene un factor clave. Deben poder cubrir casi la totalidad del código de la nuestra aplicación, dependiendo de la cantidad que se logre probar garantizará la calidad, si la cobertura es baja, significará que gran parte del código está sin probar.

Las pruebas unitarias utilizan la técnica de caja blanca. *"La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba."*[26]

"Las pruebas de caja blanca intentan garantizar que:

- ✓ *Se ejecutan al menos una vez todos los caminos independientes de cada módulo.*
- ✓ *Se utilizan las decisiones en su parte verdadera y en su parte falsa.*
- ✓ *Se ejecuten todos los bucles en sus límites.*
- ✓ *Se utilizan todas las estructuras de datos internas."* [27]

Dentro de las pruebas de caja blanca existen varias técnicas de prueba entre las que se pueden encontrar camino básico, estructura de control y *domain testing*.

Camino básico

"El método camino básico permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez." [28]

El método de prueba del camino básico se aplica a un diseño procedimental o al código fuente. Para ilustrar como se realiza este método se seguirá una serie de

pasos, tomando como ejemplo la funcionalidad Mostrar listado de clases de la ontología.

Antes de iniciar el método se debe representar una notación simple para representar el flujo de control (o gráfica del programa), permitiendo describir el flujo empleado en la notación.

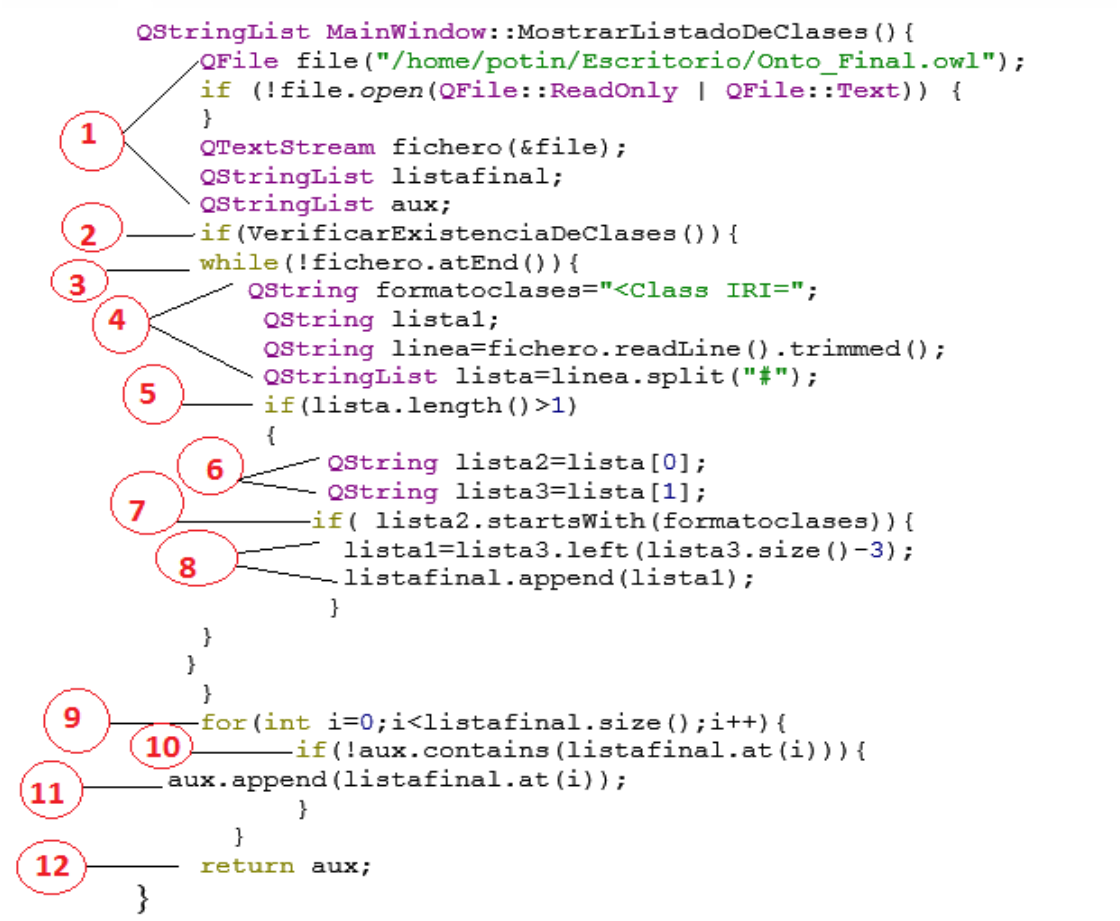

Figura 17: Ejemplo de gráfica de programa "Mostar listado de clases de la ontología".

Seguidamente se describe la estructura de control del programa mediante un diagrama de flujo. Cada círculo representado se denomina nodo del diagrama de flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser que existan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo. Las flechas del grafo se denominan aristas y representan el flujo de

control, una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental. Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más.

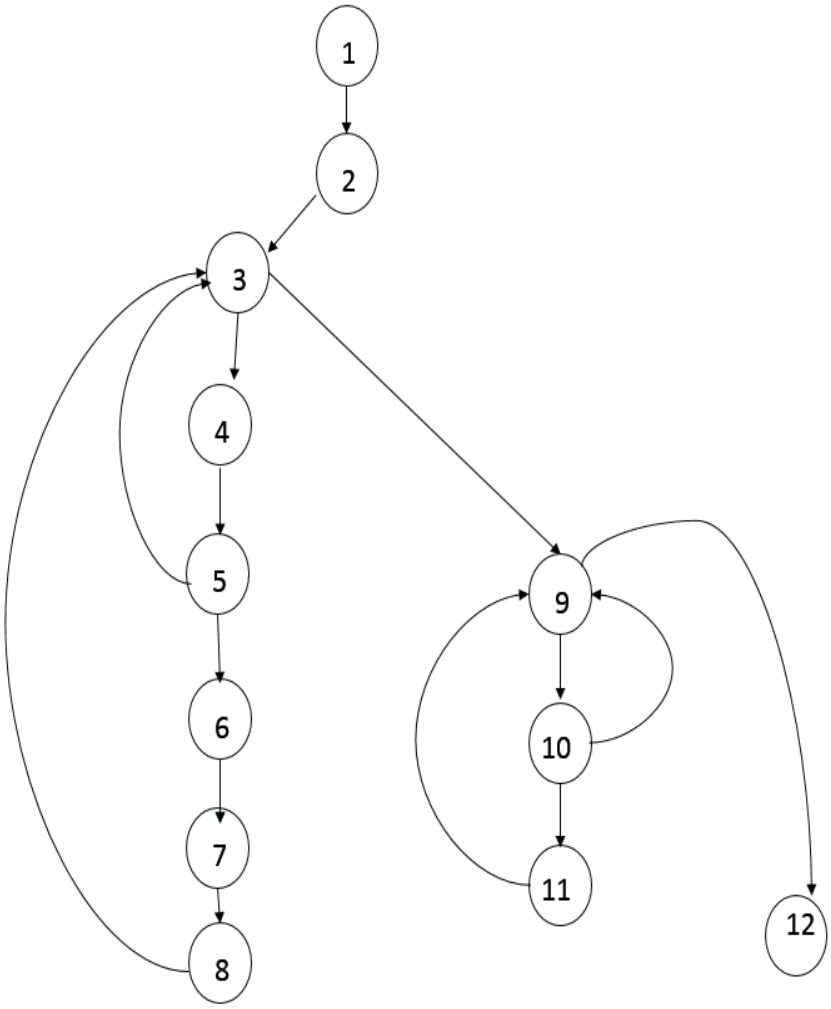

Figura 18: Ejemplo del diagrama de flujo "Mostar listado de clases de la ontología".

Ya realizado este diagrama se procede a identificar todas las rutas independientes existentes, es decir, un nuevo conjunto de instrucciones de procesamiento o una nueva condición. Una ruta independiente debe recorrer por lo menos una arista que no se haya recorrido antes.

Ruta 1:1-2-3-4-5-6-7-8-3-9-10-11-9-12

Ruta 2: 1-2-3-4-5-3-4-5-6-7-8-3-9-10-11-9-12

Ruta 3: 1-2-3-4-5-3-9-10-11-9-12

Ruta 4: 1-2-3-9-10-9-12

Ruta 5: 1-2-3-4-5-6-7-8-3-9-10-9-12

Después de haber identificado todas las rutas existentes, se procede a calcular la complejidad ciclomática, proporcionando una medida cuantitativa de la complejidad lógica del programa, permitiendo definir el número de pruebas que debe aplicarse para asegurar que todas las instrucciones se hayan ejecutado al menos una vez.

"La complejidad se puede calcular de las siguientes formas:

1. *El número de regiones corresponde a la complejidad ciclomática.*
2. *La complejidad ciclomática, V(G), de una gráfica de flujo G se denomina como: V(G) = N – G + 2 donde E es el número de aristas, y N, el número de nodos del gráfica de flujo.*
3. *La complejidad ciclomática, V(G), de un gráfica de flujo, G, también se define como: V(G) = P +1, donde P es el número de nodos predicado incluidos en la gráfica de flujo G."* [25]

Caso de prueba

Los casos de prueba del camino básico se aplican al código fuente mediante un diseño procedimental, para garantizar que por lo menos cada camino se ejecute al menos una vez, para ilustrar como realizar un caso de prueba se tomará como ejemplo la funcionalidad Importar fichero OWL.

1. Representar una notación simple para representar el flujo de control.

```
QString MainWindow::CargarFicheroOwl() {
1   QFile file("/home/potin/Escritorio/Onto_Final.owl");
    QTextStream archivo(&file);
2   if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
        qFatal("No pudo abrir el fichero");
        // O pones tu dialogo de error
3   QString fichero= archivo.readAll(); // Lees todo y lo guardas en un
QString
4   return fichero;
    }
```

Figura 19: Ejemplo gráfica de programa "Importar fichero OWL".

2. Utilizando el código como base se dibuja la gráfica de flujo correspondiente.

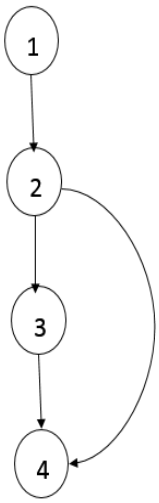

Figura 20: Ejemplo diagrama de flujo "Importar fichero OWL".

3. Determinación de la complejidad ciclomática.

   V(G) = 2 regiones

   V(G) = 4 aristas – 4nodos + 2 = 2

   V(G) = 1nodo predicado + 1 = 2

4. Identificar todas las rutas independientes existentes.

   Ruta 1: 1-2-3-4

Ruta 2: 1-2-4

5. Preparamos los casos de pruebas para la ejecución de cada camino.

   Caso de prueba del camino 1:

   1-2-3-4

   Valor: /home/ potin/ Escritorio/ Onto_Final.owl

   Resultado: muestra un mensaje de error.

   Caso de prueba del camino 2:

   1-2-4

   Valor: /home/ Docencia/ Escritorio/ Onto_Final.owl

   Resultado: carga el fichero OWL satisfactoriamente.

Después de ser obtenido el valor de V(G) y los caminos identificados, se define que cada camino independiente se ejecuta al menos una vez, siendo satisfactorias las pruebas realizadas.

3.3.4. Pruebas de aceptación.

Las pruebas de aceptación se crean sobre la base de las HU, el cliente debe especificar uno o varios escenarios para comprobar que las HU han sido implementadas correctamente. Dichas pruebas son conocidas como “pruebas de caja negra”, estas pretenden demostrar que las funciones del software son operativas, que la entrada se acepta correctamente y se produce una salida correcta. Las pruebas de caja negra se aplican a la interfaz del software, son completamente indiferentes al comportamiento interno y la estructura del programa. Por tanto, están centradas en realizar pruebas del software a través de la funcionalidad. En caso de que fallen varias pruebas, se debe indicar el orden de prioridad de resolución. Una HU no se puede considerar terminada hasta que pase todas las pruebas de aceptación.

“Las pruebas de caja negra intentan encontrar errores en las siguientes categorías.

- ✓ *Funciones incorrectas o ausentes.*

- ✓ *Errores de interfaz.*
- ✓ *Errores en estructuras de datos o en accesos a bases de datos externas.*
- ✓ *Errores de rendimiento.*
- ✓ *Errores de inicialización y de terminación. "* [29]

El método de prueba que se decide aplicar es partición equivalente, utilizando la estrategia de prueba tipo alfa. *"La partición equivalente es un método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de pruebas.*

… se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de pruebas que deben desarrollarse." [29]

"Las pruebas alfa consisten en hacer pruebas funcionales como clientes en un entorno de desarrollo. Se trabaja en un entorno controlado y siempre se tiene un experto a mano para ayudarle a usar el sistema y para analizar los resultados." [30]

A continuación se presenta la Sección (SC) "Verificar existencia de clases en la ontología", del caso de prueba realizado a la HU "Verificar existencia de clases en la ontología".

Tabla 10. Caso de prueba "Verificar existencia de clases en la ontología". SC Verificar existencia de clases en la ontología.

Escenario	Descripción	Usuario	Respuesta del sistema	Flujo central
EC 1.1 Verificar existencia de clases en la ontología.	En este escenario el usuario introduce la clase que desea verificar, el sistema muestra el listado de clases.	V(Actividades)	Se introduce la clase, el sistema muestra un mensaje ¨La clase se encuentra dentro del fichero¨.	1- El usuario introduce clase que desea verificar 2- El sistema muestra un mensaje ¨La clase se encuentra dentro de la ontología¨.

EC 1.2 Verificar existencia de clases en la ontología incorrectamente.	En este escenario el usuario introduce la clase que desea verificar, el sistema verifica que la clase se encuentre dentro del fichero.	I(Actividad) V(Actividades)	Se introduce la clase, el sistema verifica que se encuentre dentro del fichero, sino se encuentra muestra un mensaje de error "ERROR: La clase esta incorrecta o no se encuentra dentro del fichero. "	1- El usuario introduce la clase que desea verificar. 2- El sistema verifica la existencia de la clase, sino se encuentra muestra un mensaje de error "ERROR: La clase esta incorrecta o no se encuentra dentro del fichero. "
EC 1.3 Verificar existencia de clases en la ontología vacía	En este escenario el usuario introduce la clase que desea verificar, el sistema verifica que el campo no se encuentre vacío.	I(-) V(Actividades)	Se introduce la clase, el sistema verifica que el campo no encuentre vacío, si está vacío muestra un mensaje de error "ERROR: El campo se encuentra vacío."	1- El usuario introduce la clase que desea verificar. 2- El sistema verifica la existencia de la clase, si el campo se encuentra vacío muestra un mensaje de error "ERROR: El campo se encuentra vacío. "

Las pruebas realizadas proporcionaron resultados satisfactorios en más del 80 % de los casos de pruebas efectuados. Los resultados que no fueron satisfactorios pasaron a ser no conformidades y se emitieron en el registro de dificultades encontradas. A continuación se muestra un ejemplo de las no conformidades más importantes encontradas, en la HU “mostrar listado de clases en la ontología” y la “Verificar existencia de clases en la ontología”.

Tabla 11 No conformidades encontradas y resueltas.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta del Equipo Desarrollo
Aplicación	1	Muestra clases repetidas	Mostar listado de clases en la ontología	Prueba	S	6/05/13 PD 6/05/13 RA	Se corrigió el error, encontrado, ya no muestra clases repetidas.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta del Equipo Desarrollo
Aplicación	1	No muestra el mensaje de Error cuando el campo queda vacío.	Mostrar mensaje de error	Prueba	S	8/05/13 PD 9/05/13 RA	Se corrigió el error, encontrado, ya muestra el mensaje de error cuando se deja el campo vacío.

En la siguiente gráfica se evidencia los resultados de las no conformidades encontradas en las tres iteraciones realizadas.

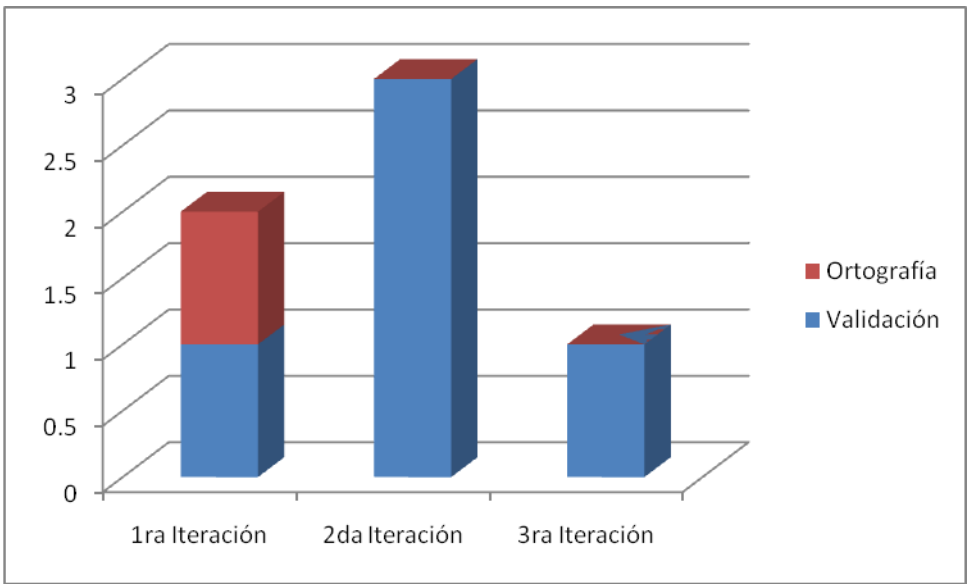

Figura 21: Representación de los tipos de no conformidades encontradas en las iteraciones.

En la siguiente grafica se muestra los resultados de las no conformidades encontradas en la realización de los casos de pruebas por iteraciones.

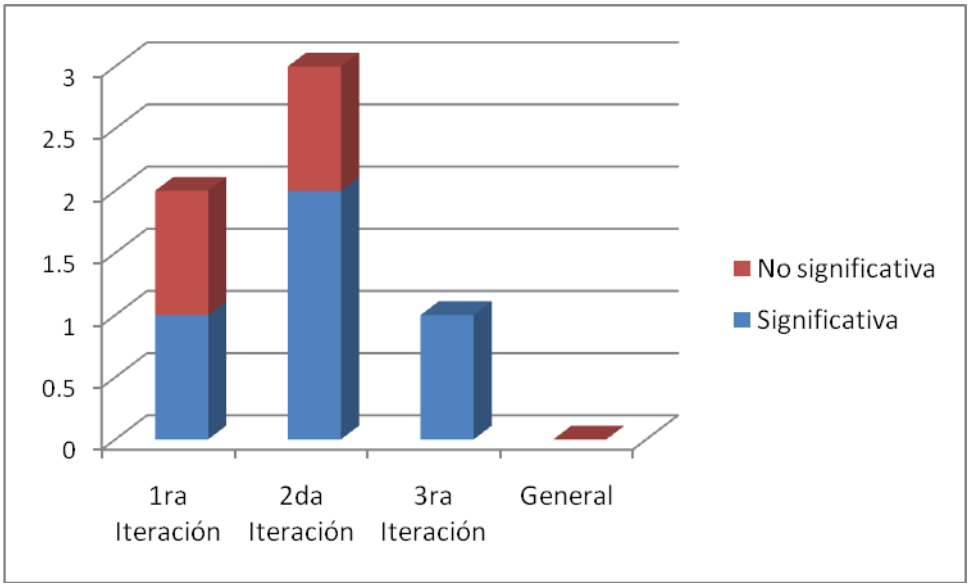

Figura 22: Representación de las clasificaciones de no conformidades encontradas en las iteraciones.

3.4 Conclusiones del capítulo.

De las 10 HU definidas en la fase de análisis y diseño se logró implementar el 100% de las mismas haciendo uso de QT Creator con C++ como lenguaje.

En el desarrollo de este capítulo se realizaron las Tareas de Ingeniería y se definió el estándar de codificación.

Se realizaron las pruebas de aceptación a través del diseño de 9 casos de pruebas, uno por cada Historia de Usuario, utilizando el método caja negra con la técnica, partición de equivalencia, arrojaron como resultado 2 no conformidades en la primera iteración, 3 en una segunda iteración y 1 en la tercera iteración para un total de 6 no conformidades, que fueron solucionadas demostrando que la biblioteca desarrollada cumple con las características especificadas por el cliente.

Además se realizaron pruebas unitarias utilizando el método de caja blanca, mediante la técnica de camino básico, asegurando que cada camino se ejecuta al menos una vez, siendo satisfactorias las pruebas realizadas.

Conclusiones generales

1. Se implementó la primera versión de la biblioteca para gestionar las características de las ontologías representativas del conocimiento.

2. Al realizar el análisis se obtuvieron 10 HU.

3. Se obtuvieron durante el diseño de la biblioteca 6 tarjetas CRC, así como 6 clases que conformaron el diagrama de clases.

4. Se validó la solución mediante la confección de 9 casos de prueba, uno por cada Historia de Usuario demostrando que la aplicación cumple con las funcionalidades identificadas.

Recomendaciones

Incorporar a la biblioteca las funcionalidades que permitan la gestión de las características en ontologías, como son: insertar, modificar y eliminar.

Referenia bibliográfica

[1] **Gruber, Thomas**. 1993. Toward Principles for the Design of Ontologies Use for Knowledge Sharing. s.l. : Stanfort Knowledge Systems Laboratory, 1993.

[2**] Borst, Willem Nico**. 1997. Construction of Engineering Ontologies. University of Tweenty : Centre for Telematics and Information Technology, 1997.

[3]**Gruber,Thomas. 1993**. Translation Approach to Portable Ontology Specifications. s.l. : Knowledge Acquisition, 1993. pp. 199-220. Vol. 5(2).

[4] **PhD. Jaime Alberto Guzmán Luna. M.S. Mauricio López Bonilla, Ing. Ingrid Durley Torres**. 2012. Methodologies and methods for building ontologies. s.l. : Universidad Tecnológica de Pereira, 2012.

[5] **W3C**. [Online] [Cited: 12 5, 2012.] http://www.w3.org/2007/09/OWL-Overview-es.html.

[6] **Rogério Aparecido Sá Ramalho, Mariângela Spotti Lopes Fujita**. APLICABILIDAD DE ONTOLOGÍAS EN BIBLIOTECAS DIGITALES. p. 8.

[7] **Orallo,Enrique Hernández**. El Lenguaje Unificado de Modelado(UML). pp. 3-4. ebernandez@disca.upv.es.

[8] **MODELO DEL DOMINIO Extraído de: *UML y Patrones*. 2ª Edición**. Craig Larman. Prentice Hall. 2003.

[9] **Joskowicz, José.** 2008.

[10] **Análisis comparativo de bibliotecas multiplataforma para el desarrollo de aplicaciones de escritorio, aplicado a la escuela de diseño gráfico**, Mayra Alexandra Macas Carrasco, Ana Elizabeth Jácome Quintanilla, Riobamba-Ecuador

[11]**ingenieriasoftware2011.wordpress/2011/07/08/herramientas-case-ejemplos-para-evaluarlas/**. [Online] [Cited: octubre 25, 2013.]

[12]**ingenieriasoftware2011.wordpress/2011/07/08/herramientas-case-ejemplos-para-evaluarlas/**. [Online] [Cited: octubre 25, 2013.]

[13] **glateriel.wordpress.com/2009/05/15/qt-creator-desarrollo-aplicaciones-rápidamente.** [Online] [Cited: octubre 27, 2013.]

[14] **Reyes, Yunior Mesa y Ortiz, Yudisney Vásquez**. Sitemas de Bases de Datos. Espacio de comunicación e intercambio para la comunidad técnica cubana de postgeSQL. PostgreSQL. Ciudad de la Habana, Cuba : s.n., 2011. pág. 6. 1994-1536.

[15] **Can, Carolina Novelo. 6 de septiembre de 2010.** *Planificación y modelado.* Instituto tecnológico superior Escárcega. 6 de septiembre de 2010.

[16] **Guerrero, Lugo Manuel Barbosa.** *Arquitectura de software como eje temático de investigación.* 2006.

[17] **es.scribd.** [Online] [Cited: marzo 2013, 23.] *es.scribd /doc/23161581/Estilos-Arquitectónico*

[18]**es.scribd.** [Online] [Cited: marzo 2013, 23.] es.scribd/doc/14704374/Arquitectura-Basada-en-Componetes

[19] **Prieto, felix.** Programacion III. Sistemas tema 7. Patrones de Diseño. Universidad de Valladolid. 2005.

[20] **Reyes, Yunior Mesa y Ortiz, Yudisney Vasquez.** Sitemas de Bases de Datos. Espacio de comunicación e intercambio para la comunidad técnica cubana de postgeSQL. PostgreSQL. Ciudad de la Habana, Cuba : s.n., 2011. pág. 7. 1994-1536.

[21] **www.ciberula.com/articulo/diseno patrones j2ee**. [Online] [Cited: abril 20, 2013.]

[22] **Aguilera.M, Carmen. P.** Sistema de información para el registo y control de o procesos de gestión de higiene ocupacional. UNIVERSIDAD DE ORIENTE . MATURÍN / MONAGAS / VENEZUELA : s.n., 2011. pág. 260.

[23] **Hugo, Serrano Cuayahuitl Victor.** Pruebas de unidad, Estándares de codificación y generación automática de código. Facultad de Ciencias básicas, ingeniería y tecnología. Universidad Autónoma de Tlaxcala .

[24] **Sánchez, Linet Lores y Roque, Diana Monné.** Aplicación de las pruebas de libración al sistema informático de genética médica. Universidad de las Ciencias informáticas(UCI). Ciudad de la Habana, Cuba : s.n., 209.

[25] **www.mastermagazine.info/termino/3868.php.** [En línea] [Citado el: 5 de 04 de 2013.]

[26] **Pressman, Roger S**. Ingeniería el software. Un enfoque práctico. 6ta edición cap 13 .

[27] **Pressman, Roger S**. Ingeniería de Software. Un enfoque práctico. 6ta edición, Cap 14. pág. 19.

[28] **www.es.testhouse.net/pruebas-de-aceptacion/**. [En línea] [Citado el: 04 de 04 de 2013.]

[29]**www.adalog.es/mtorres/LP/Prueba.pdf.** [En línea] [Citado el: 23 de 04 de 2013.]

[30]**Pressman, Roger S**. Ingeniería de Software. Un enfoque práctico. 6ta edición, Cap 14. pág. 423.

Bibliografía

Gruber, Thomas. 1993. Toward Principles for the Design of Ontologies Use for Knowledge Sharing. s.l. : Stanfort Knowledge Systems Laboratory, 1993.

Borst, Willem Nico. 1997. Construction of Engineering Ontologies. University of Tweenty : Centre for Telematics and Information Technology, 1997.

usuarios.multimania.es/ontologiainfometrica/with/Definiciones.htm. [En línea] [Citado el: 24 de septiembre de 2012.]

elcorregallego.es/galicia/ecg/nueva-ontologia. [En línea] 09 de 02 de 2009. [Citado el: 25 de septiembre de 2012.]

Gruber,Thomas. 1993. Translation Approach to Portable Ontology Specifications. s.l. : Knowledge Acquisition, 1993. pp. 199-220. Vol. 5(2).

PhD. Jaime Alberto Guzmán Luna. M.S. Mauricio López Bonilla, Ing. Ingrid Durley Torres. 2012. Methodologies and methods for building ontologies. s.l.: Universidad Tecnológica de Pereira, 2012.Bibliografía

The Aplication of Ontologies in the Retrofit of Chemical Processes. I Lopez Arévalo, A Rodríguez Martínez, R Bañares Alcántara y A Aldea. Revista Mexicana de Ingeniería Química.

W3C. [Online] [Cited: 12 5, 2012.] http://www.w3.org/2007/09/OWL-Overview-es.html.

Ora Lassila, Ralph R Swick. ww.sidar.org/recur/desdi/traduc/es/rdf/rdfesp.htm. [En línea] 22 de febrero de 1999.

Miller, Eric.www.dlib.org/dlib/may98/miller/05miller.html. [En línea] Mayo de 1998. [Citado el: 7 de octubre de 2012.]

Deborah L. McGuinness, Frank Van Harmelen. www.w3.org/TR/owl-features/. [En línea] 10 de febreo de 2004. [Citado el: 9 de octubre de 2012.]

Torre, Pablo de la. Almacenes de Datos para la Web.

Delgado, Susana Corbera. Almacenamiento, consulta y razonamiento: Sesame Y Jena

Rogério Aparecido Sá Ramalho, Mariângela Spotti Lopes Fujita. APLICABILIDAD DE ONTOLOGÍAS EN BIBLIOTECAS DIGITALES. p. 8.

Joskowicz, José. *Reglas y Prácticas en eXtreme Programming.* 2008.

www.scrum.org/. [En línea] [Citado el: 20 de octubre de 2012.]

www.slideshare.net/vivi_jocadi/rational-rose. [En línea] [Citado el: 4 de noviembre de 2012.]

Grady Booch, James Rumbaugh, Ivar Jacobson. El lenguaje unificado de modelado.

Permuy, Fernando Bellas. www.tic.udc.es/~fbellas/teaching/cpp/CPP.pd. [En línea] [Citado el: 18 de noviembre de 2012.]

www.slideshare.net/Dunkherz/lenguaje-de-programacion-c-basico-1ra-parte-elementos-basicos-del-lenguaje. [En línea] [Citado el: 20 de noviembre de 2012.]

ww.wxwidgets.org/. [En línea] [Citado el: 25 de noviembre de 2012.]

qt.digia.com/. [En línea] [Citado el: 5 de diciembre de 2012.]

www.eclipse.org/. [En línea] [Citado el: 9 de diciembre de 2012.]

qt-project.org/ Category:Tools::QtCreator_Spanish. [En línea] [Citado el: 15 de diciembre de 2012.]

Reyes, Yunior Mesa y Ortiz, Yudisney Vásquez. Sitemas de Bases de Datos. Espacio de comunicación e intercambio para la comunidad técnica cubana de postgeSQL. PostgreSQL. Ciudad de la Habana, Cuba : s.n., 2011. pág. 6. 1994-1536.

Can, Carolina Novelo. 6 de septiembre de 2010. *Planificación y modelado.* Instituto tecnológico superior Escárcega. 6 de septiembre de 2010.

Guerrero, Lugo Manuel Barbosa. *Arquitectura de software como eje temático de investigación.* 2006.

es.scribd. [Online] [Cited: 23 de marzo de 2013.] *es.scribd /doc/23161581/Estilos-Arquitectónico.*

es.scribd. [Online] [Cited: 23 de marzo de 2013.] es.scribd/doc/14704374/Arquitectura-Basada-en-Componetes

Gastón Coco, Nicolás Passerini, Juan Arias, Gustavo A Brey. apit.wdfiles.com/local--files/start/03_apit_estilos_arquitectonicos.pdf. [En línea] 2005. [Citado el: 12 de enero de 2013.]

Prieto, felix. Programacion III. Sistemas tema 7. Patrones de Diseño. Universidad de Valladolid. 2005.

Lidia Fuentes, José M Troya y Antonio Vallecillo. http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf. [En línea]

Hern, Marcello Visconti y. www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf. [En línea]

geektheplanet.net/5462/patrones-gof.xhtml. [En línea] [Citado el: 9 de febreo de 2013.]

Reyes, Yunior Mesa y Ortiz, Yudisney Vasquez. Sitemas de Bases de Datos. Espacio de comunicación e intercambio para la comunidad técnica cubana de postgeSQL. PostgreSQL. Ciudad de la Habana, Cuba : s.n., 2011. pág. 7. 1994-1536.

Aguilera.M, Carmen. P. Sistema de información para el registo y control de o procesos de gestión de higiene ocupacional. UNIVERSIDAD DE ORIENTE . MATURÍN / MONAGAS / VENEZUELA : s.n., 2011. pág. 260.

Hugo, Serrano Cuayahuitl Victor. Pruebas de unidad, Estándares de codificación y generación automática de código. Facultad de Ciencias básicas, ingeniería y tecnología. Universidad Autónoma de Tlaxcala .

www.slideshare.net/jpbthames/diagramas-de-clases. [En línea] [Citado el: 10 de abril de 2013.]

Sánchez, Linet Lores y Roque, Diana Monné. Aplicación de las pruebas de libración al sistema informático de genética médica. Universidad de las Ciencias informáticas(UCI). Ciudad de la Habana, Cuba : s.n., 209.

www.mastermagazine.info/termino/3868.php. [En línea] [Citado el: 5 de abril de 2013.]

pruebasdesoftware.com/pruebadeaceptacion.htm. [En línea] [Citado el: 15 de abril de 2013.]

Pressman, Roger S. Ingeniería el software. Un enfoque práctico. 6ta edición cap 13.

Pressman, Roger S. Ingeniería de Software. Un enfoque práctico. 6ta edición, Cap 14. pág. 19.

www.es.testhouse.net/pruebas-de-aceptacion/. [En línea] [Citado el: 04 de abril de 2013.]

www.adalog.es/mtorres/LP/Prueba.pdf. [En línea] [Citado el: 23 de abril de 2013.]

Pressman, Roger S. Ingeniería de Software. Un enfoque práctico. 6ta edición, Cap 14. pág. 423.

cppunit.sourceforge.net/doc/lastest/cppunit_cookbook.html. [En línea] [Citado el: 20 de abril de 2013.]

www.codeproject.com/Articles/5660/Unit-testing-with-CPPUnit. [En línea] [Citado el: 20 de marzo de 2013.]

Glosario de términos.

API: Interfaz de Programación de Aplicaciones.

CASE: Ingeniería de software asistida por computadora.

CRC: Artefacto generado por la metodología XP, es una tarjeta dividida en tres partes Clase, Responsabilidad y Colaboración.

C++: Lenguaje de programación.

Framework: estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, que puede servir de base para la organización y desarrollo de *software*.

GoF*:* Patrones de diseños, *Gang of Four* (Pandilla de los Cuatro).

GRASP: Patrones de *Software* para la Asignación General de Responsabilidad.

HU: Historia de Usuario

IDE: Entorno de Desarrollo Integrado.

QT: Nombre de la compañía que lo creó: Quasar Technologies.

OWL Lenguaje de Ontologías Web

RDF Macro de Descripción de Recursos

SC: Abreviatura de la palabra Sección utilizada para enumerar las secciones de los casos de pruebas.

Plugin: Software que se relaciona con otro para portar una función nueva, generalmente muy específico que se ajusta a la aplicación principal.

UML: Lenguaje de Modelado Unificado.