

Universidad de las Ciencias Informáticas

Facultad 6



Título: “Aplicación informática para la gestión de repositorios de ontologías representativas del conocimiento en la Web”.


Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autores: Michael E. Marrero Clark

Leonel Vila Pérez

Tutor: Ing. Edgar Rojas Ricardo

Junio 2011



Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte.

Che.

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis 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 _____.

Michael E. Marrero Clark

Leonel Vila Pérez

Ing. Edgar Rojas Ricardo

Firma del Autor

Firma del Autor

Firma del Tutor

DATOS DE CONTACTO

Autores:

Michael E. Marrero Clark.

Universidad de las Ciencias Informáticas

e-mail: memarrero@estudiantes.uci.cu.

Leonel Vila Pérez

Universidad de las Ciencias Informáticas

e-mail: lvila@estudiantes.uci.cu.

Tutor:

Ing. Edgar Rojas Ricardo.

Universidad de las Ciencias Informáticas.

e-mail: erojas@uci.cu.

AGRADECIMIENTOS

Agradecimientos de Michael:

A mi madre y a mi abuela, por su amor y apoyo incondicional.

A mi familia, que de una u otra forma me ha ayudado a convertirme en quién soy.

A los amigos con los que he compartido estos 5 años en la universidad. En especial a Flavio a quien considero mi hermano, a Elisa, Yanet, Viviana, Héctor, Abel, Elío, Vanía, Leidy, a todos gracias por este tiempo juntos.

A Vía por ser un compañero de tesis inmejorable y excelente amigo.

A Edgar, por habernos guiado en esta última travesía en la universidad.

A todos los que has hecho posible este sueño hecho realidad.

A la Revolución y a Fidel.

Agradecimientos de Leonel:

A mi madre, fuente de inspiración para lograr mis objetivos.

A mi familia, que me han brindado su apoyo incondicional para alcanzar esta meta.

A mi tía Zoraída, por toda la ayuda que me ha dado en los 5 años de carrera.

A mis vecinos y amigos, que me han animado a formarme por el buen camino.

A los amigos que han compartido conmigo a lo largo de mi formación como estudiante.

A mi compañero de tesis Michael, por su esfuerzo y dedicación para que este trabajo saliera bien.

A Edgar, por su excelente labor como tutor.

DEDICATORIA

Dedicatoria de Michael:

A mi madre, por estar siempre presente cuando la necesito y a quien le debo todo lo que he logrado en esta vida.

A mi familia.

Dedicatoria de Leonel:

A mi madre, mi guía en la vida, por todo su apoyo y sacrificio para que pudiera llegar a la meta final.

A mi familia.

RESUMEN

La utilización de ontologías existentes, refinando o extendiendo sus límites, es un elemento fundamental en el desarrollo de aplicaciones para la Web Semántica. Debido al creciente número de ontologías que se han creado ha surgido la necesidad de agruparlas en repositorios con el objetivo de compartir el conocimiento entre la comunidad de desarrolladores. En la actualidad existen varios de estos con estructuras muy heterogéneas y deficiencias que dificultan en mayor o menor medida la recuperación de la información. Con el objetivo de dar solución a la problemática descrita, se desarrolló una aplicación informática para la gestión de repositorios de ontologías representativas del conocimiento en la Web. Para el desarrollo de la misma se seleccionaron las tecnologías y herramientas a utilizar que más se adecuaron a las características del proyecto y a las políticas de uso de software libre de la Universidad. Se identificaron y documentaron las funcionalidades a cumplir por la aplicación y se generaron los artefactos correspondientes a cada fase del proceso de desarrollo de software guiado por la metodología eXtreme Programing.

PALABRAS CLAVES: ontología, OWL, repositorio.

TABLA DE CONTENIDOS

| | |
|---|-----|
| AGRADECIMIENTOS | I |
| DEDICATORIA..... | II |
| RESUMEN | III |
| INTRODUCCIÓN..... | 1 |
| CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA | 5 |
| 1.1 Introducción..... | 5 |
| 1.2 Ontologías | 5 |
| 1.3 Repositorios | 9 |
| 1.4 Metodología de desarrollo | 10 |
| 1.5 Herramientas y tecnologías..... | 14 |
| 1.5.1 Lenguaje de programación..... | 14 |
| 1.5.2 Entorno de desarrollo | 16 |
| 1.5.3 Frameworks | 18 |
| 1.5.4 Servidor de aplicaciones..... | 23 |
| 1.5.5 Sistema gestor de base de datos..... | 24 |
| Conclusiones del capítulo..... | 26 |
| CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL SISTEMA | 28 |
| 2.1 Introducción..... | 28 |
| 2.2 Descripción de la solución propuesta | 28 |
| 2.3 Requerimientos | 29 |
| 2.4 Fase de planificación | 31 |
| 2.4.1 Historias de usuario..... | 31 |
| 2.4.2 Estimación de esfuerzo por Historia de Usuario..... | 35 |
| 2.4.3 Plan de iteraciones | 36 |
| 2.4.5 Plan de duración de las iteraciones | 38 |
| 2.4.6 Plan de entregas | 39 |
| 2.5 Fase de diseño | 39 |
| 2.5.1 Tarjetas CRC | 40 |

| | |
|--|-----------|
| 2.5.2 Patrones de diseño..... | 42 |
| 2.5.3 Diseño de la base de datos..... | 45 |
| Conclusiones del capítulo..... | 46 |
| CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA | 47 |
| 3.1 Introducción..... | 47 |
| 3.2 Implementación | 47 |
| 3.2.1 Tareas de ingeniería..... | 47 |
| 3.2.2 Estándar de código..... | 51 |
| 3.2.3 Código de algunos de los principales métodos implementados | 53 |
| 3.3 Pruebas..... | 55 |
| 3.3.1 Pruebas de aceptación | 56 |
| 3.3.2 Casos de Pruebas | 56 |
| Conclusiones del capítulo..... | 57 |
| CONCLUSIONES GENERALES..... | 59 |
| RECOMENDACIONES..... | 60 |
| REFERENCIAS BIBLIOGRÁFICAS..... | 61 |
| BIBLIOGRAFÍA | 64 |
| ANEXOS | 67 |
| GLOSARIO DE TÉRMINOS | 69 |

ÍNDICE DE FIGURAS

Figura 1: Proceso de desarrollo de software.....11

Figura 2: Patrón de diseño Modelo – Vista – Controlador.43

Figura 3: Clases del Modelo43

Figura 4: Páginas de la Vistas44

Figura 5: Clases y páginas controladoras.....45

Figura 6: Flujo de datos Adicionar dominios.45

Figura 7: Diagrama del diseño de la base de datos.....46

Figura 8: Ejemplo de código54

Figura 9: Ejemplo de código54

Figura 10: Ejemplo de código55

ÍNDICE DE TABLAS

| | |
|--|----|
| Tabla 1: Roles de Usuarios..... | 29 |
| Tabla 2: Historia de usuario #1..... | 32 |
| Tabla 3: Historia de usuario #3..... | 32 |
| Tabla 4: Historia de usuario #10..... | 33 |
| Tabla 5: Historia de usuario #11..... | 33 |
| Tabla 6: Historia de usuario #12..... | 34 |
| Tabla 7: Historia de usuario #14..... | 34 |
| Tabla 8: Historia de usuario #18..... | 35 |
| Tabla 9: Estimación de esfuerzo por puntos..... | 36 |
| Tabla 10: Plan de duración de las iteraciones..... | 38 |
| Tabla 11: Plan de entregas..... | 39 |
| Tabla 12: Tarjeta CRC para la clase Conexion..... | 40 |
| Tabla 13: Tarjeta CRC para la clase ModeloDatos..... | 41 |
| Tabla 14: Tarjeta CRC para la clase Control..... | 41 |
| Tabla 15: Tarjeta CRC para la clase Repositorio..... | 41 |
| Tabla 16: Tarjeta CRC para la clase Dominio..... | 42 |
| Tabla 17: Tarjeta CRC para la clase Ontologia..... | 42 |
| Tabla 18: Tareas asociadas a cada historia de usuario..... | 47 |
| Tabla 19: Tarea #1 HU: Autenticar usuario..... | 48 |
| Tabla 20: Tarea #2 HU: Autenticar usuario..... | 49 |
| Tabla 21: Tarea #3 HU: Autenticar usuario..... | 49 |
| Tabla 22: Tarea #22 HU: Caracterizar repositorio..... | 49 |
| Tabla 23: Tarea #23 HU: Caracterizar repositorio..... | 50 |
| Tabla 24: Tarea #31 HU: Comparar repositorios..... | 50 |
| Tabla 25: Tarea #32 HU: Comparar repositorios..... | 50 |
| Tabla 26: Caso de prueba de aceptación HU3-1..... | 57 |

INTRODUCCIÓN

La búsqueda constante del hombre por satisfacer sus necesidades de comunicación y de compartir el conocimiento ha impulsado el desarrollo de herramientas cada vez más poderosas y veloces en el intercambio de datos. La creación, búsqueda y obtención de información son acciones esenciales de la naturaleza humana, que han condicionado grandes saltos evolutivos de la humanidad con el hito de instaurar algún nuevo instrumento de comunicación.

Sin duda alguna un fenómeno que ha revolucionado al mundo en muchos aspectos, pero con especial impacto en el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), ha sido Internet. Su surgimiento ha determinado una nueva etapa en la historia del hombre, a la cual diferentes autores han denominado indistintamente como era de la información o del conocimiento.

La rápida extensión y propagación de Internet ha sido posible en gran medida por la invención de la *World Wide Web* (WWW según sus siglas en inglés) en 1989 por el británico Tim Berners-Lee. La WWW o simplemente la Web, es un sistema hipermedia basado en el Protocolo de Transferencia de Hipertexto (*Hypertext Transfer Protocol*, HTTP) que permite el acceso a fuentes de información en Internet, presentándolas como documentos denominados páginas Web. La red de redes, como también es conocida la WWW, brinda a sus usuarios acceso a cantidades prácticamente ilimitadas de información a un costo relativamente bajo. Constituye además uno de los medios más difundido y utilizado para las comunicaciones, el comercio, el entretenimiento y los negocios.

El crecimiento exponencial que ha experimentado el número de usuarios de la Web en los últimos años y el aumento de sus funcionalidades, ha hecho necesario la incorporación de nuevas tecnologías que incrementen el potencial de la misma. Sin embargo, en la actualidad, estas no son suficientes para gestionar el cúmulo cada vez mayor de información contenida en la Web. A pesar de que se han desarrollado numerosas aplicaciones que implementan la búsqueda y recuperación de información de Internet, estas se enfrentan a la limitación de que la misma está estructurada para el entendimiento de las personas, pero el ordenador no es capaz de interpretarla.

La Web actual es un espacio preparado para el intercambio de información diseñado para el consumo humano. Con los estándares del momento, no se puede diferenciar entre información personal, académica, comercial, etc. Es decir, cuando un buscador Web realiza una consulta con algunas palabras claves, normalmente la información que se genera como respuesta no es útil en su totalidad porque no corresponde a las necesidades de los usuarios. Por otro lado, los agentes de búsqueda

actuales no se diseñan para “comprender” la información que reside en la Web, porque es prácticamente imposible conocer la representación de los datos ubicados en las páginas (1).

A finales de la década de los 90, Tim Berners-Lee, propuso una solución para superar estos límites: la Web Semántica.

La Web Semántica tiene un propósito tan amplio como la Web actual: crear un medio universal para el intercambio de datos; donde la gestión de la información sea eficiente, se integren aplicaciones empresariales y compartan globalmente datos comerciales, científicos y culturales. Donde se puedan compartir datos de una forma que los ordenadores puedan “entender” y procesar, posibilitando así tanto la interacción entre ordenadores como la de estos con los usuarios (2).

Para que una Web más inteligente sea posible es necesario que la información de las páginas esté estructurada, descrita y clasificada de manera que esté al alcance de las máquinas comprender su significado exacto; de esta manera podrán procesar y manipular la información adecuadamente. El medio principal para lograr este objetivo son las ontologías, estas facilitan la definición formal de las entidades y conceptos presentes en los diferentes dominios del conocimiento; garantizando una representación formal de la información legible por las máquinas.

En el contexto actual de la informática una ontología es un modelo abstracto de algún fenómeno del mundo, construido mediante la identificación de los conceptos relevantes a ese fenómeno. Los conceptos utilizados en la ontología, y las restricciones para su uso, están claramente definidos. Además, para que las máquinas puedan interpretarlas debe estar expresada mediante una sintaxis que permita a un ordenador operar sobre ella (la W3C propone OWL) y el conocimiento contenido en la misma deberá estar consensuado en algún grado por los expertos del dominio en que se enmarque.

Para lograr la comunicación entre sistemas que propone la Web Semántica resultaría provechoso en el desarrollo de las aplicaciones utilizar ontologías existentes, refinando o extendiendo sus límites. Debido al creciente número de ontologías que se han creado y la necesidad de su reutilización, muchas instituciones se han dado a la tarea de agruparlas en repositorios. La forma en que estos han sido estructurados es muy heterogénea y en algunos casos el acceso a ellos se dificulta por el uso excesivo de requisitos de autenticación. Además carecen de información sobre su contenido, que pudiera hacer más eficiente la búsqueda; en la mayoría de los casos no se potencia el intercambio de información en el marco de la comunidad, puesto que no permiten compartir los aportes individuales de sus miembros. Los elementos mencionados anteriormente provocan que el proceso de gestión de la

información contenida en los repositorios destinados a almacenar ontologías se dificulte en gran medida.

Una vez analizada la problemática anteriormente descrita y con el objetivo de darle solución se define como:

Problema de la investigación: ¿Cómo gestionar repositorios de ontologías representativas del conocimiento en la Web?

Objetivo general: Desarrollar una aplicación informática que gestione repositorios de ontologías representativas del conocimiento en la Web para facilitar los procesos de almacenamiento, búsqueda y recuperación de ontologías.

Objeto de estudio: La gestión de la información de los repositorios de ontologías representativas del conocimiento en la Web.

Campo de acción: Las aplicaciones informáticas para la gestión de repositorios de ontologías representativas del conocimiento en la Web.

Objetivos específicos:

- ✓ Realizar el análisis de la aplicación informática.
- ✓ Realizar el diseño la aplicación informática.
- ✓ Realizar la implementación y prueba de la aplicación informática.

Tareas de la investigación:

- ✓ Análisis de los conceptos fundamentales de las ontologías, los repositorios y sus aplicaciones.
- ✓ Selección de la metodología, herramientas y tecnologías para el desarrollo de la aplicación informática.
- ✓ Definición de los requerimientos no funcionales de la aplicación informática.
- ✓ Especificación de las historias de usuario.
- ✓ Elaboración de las tarjetas CRC.
- ✓ Implementación de la aplicación informática.
- ✓ Evaluación de la aplicación informática mediante pruebas de aceptación.

Posibles resultados:

Una aplicación informática que gestione repositorios de ontologías representativas del conocimiento en la Web para facilitar los procesos de almacenamiento, búsqueda y recuperación de ontologías.

El presente trabajo de diploma tiene como estructura:

Capítulo 1: Fundamentación Teórica.

Este capítulo contiene los fundamentos teóricos para entender el problema a solucionar, se abordan los conceptos fundamentales sobre las ontologías, repositorios y la Web Semántica; además se realiza un estudio y selección de las herramientas y tecnologías a utilizar en el desarrollo de la aplicación.

Capítulo 2: Planificación y diseño del sistema.

Este capítulo describe la propuesta de solución para el problema de la investigación planteado. Se describen las características que poseerá la aplicación informática a desarrollar, definidas mediante la especificación de los requisitos no funcionales y funcionales expresados mediante historias de usuario. Se generan todos los artefactos durante el ciclo de vida del proceso de desarrollo de software seleccionado.

Capítulo 3: Implementación y pruebas del sistema.

En este capítulo se presentan las tareas de ingeniería en que se organizó la implementación de la aplicación, se representan el código fuente de las principales funcionalidades y los estándares utilizados durante este proceso. Se efectúa una revisión final de las especificaciones del diseño y de la codificación mediante la realización de pruebas de aceptación, garantizando la calidad del software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realizará un estudio de los principales conceptos, necesarios para comprender el problema de la investigación. Se abordarán las definiciones de varios autores sobre la Web Semántica, las ontologías y los repositorios, imprescindibles para comprender el objetivo y alcance de la aplicación a desarrollar. Se analizarán algunas de las metodologías de desarrollo de software más utilizadas y se seleccionará la que más se ajuste a las características del proyecto y equipo de desarrollo. Se seleccionarán las herramientas y tecnologías que más se adecúen para el desarrollo del tipo de aplicación que se desea implementar.

1.2 Ontologías

Antecedentes del uso de ontologías en la Web: la Web Semántica.

El término “Web Semántica” fue presentado al dominio público tras la publicación del artículo “*The Semantic Web*” aparecido en *Scientific American* en mayo de 2001 y del que fueron coautores Tim Berners-Lee, James Hendler y Ora Lassila. En este artículo se establecen diversos escenarios imaginarios en los que agentes software son capaces de realizar numerosas tareas accediendo al contenido de diferentes páginas de la WWW. Los autores señalan que, para que este escenario sea factible, debería cambiar la manera de representar contenido en la Web (hasta ahora diseñado para que los seres humanos puedan leerlo) para incluir una “semántica bien definida” que permitiese a componentes software acceder al mismo (3).

La Web Semántica añadirá estructura al contenido lleno de significado de las páginas Web, creando un entorno donde agentes software, transitando de página en página, puedan llevar a cabo fácilmente tareas sofisticadas para los usuarios” (2).

La Web Semántica propone modificar la forma en que se presentan los contenidos en la Web de modo que no sólo se indique información sobre su formato, sino que también se incluya información que lo describa. Se plantea la necesidad de una evolución en cuanto a la estructuración de los contenidos en la Web tradicional para permitir las funcionalidades que se esperan de la Web Semántica. Si la conexión entre recursos en la Web tradicional se produce a través de enlaces únicamente entendibles por usuarios humanos, la estructura de la Web Semántica se basa en relaciones semánticas que son capaces de interpretar tanto humanos como entidades software (3).

La definición oficial dada por la W3C de Web Semántica resume que: El propósito de la iniciativa de la Web Semántica es tan amplio como el de la Web actual: crear un medio universal para el intercambio de datos. Se considera para interconectar eficazmente la gestión de la información personal, la integración de las aplicaciones empresariales y compartir globalmente datos comerciales, científicos y culturales. Los servicios para poner datos comprensibles por las máquinas se están convirtiendo rápidamente en una prioridad para muchas organizaciones, individuos y comunidades. La Web sólo puede alcanzar todo su potencial si llega a ser un sitio donde se puedan compartir datos y estos sean procesados por herramientas automáticas, así como por personas. Para adaptar la Web, los programas del mañana deben ser capaces de compartir y procesar datos incluso cuando estos programas se hayan diseñado de forma completamente independiente (4).

Esta nueva aproximación de la Web Semántica a la estructuración de la información se basa en la utilización de ontologías como mecanismo para la representación del conocimiento. En este entorno, las ontologías se utilizan para aportar un vocabulario que describa las relaciones entre diferentes términos, permitiendo a los sistemas computarizados (y a los humanos) interpretar su contenido flexiblemente y sin ambigüedades (5).

¿Qué es una ontología?

En filosofía, una ontología es una teoría que trata de la naturaleza y organización de la realidad, es decir de lo que "existe". Conocimiento del ser (del griego **onto**: ser y **logos**: conocimiento). Por lo que el Diccionario de la Real Academia Española lo define como parte de la metafísica que trata del ser en general y de sus propiedades trascendentales.

En el campo de la inteligencia artificial, las ontologías fueron definidas para compartir y reutilizar conocimiento. Aportan un lenguaje de comunicación necesario en entornos distribuidos que involucran agentes software, como la Web Semántica, y una descripción semánticamente formal para el procesamiento del conocimiento. En sentido general, una ontología es la base del procesamiento semántico; es una red de conceptos, relaciones y axiomas para representar, organizar y entender un dominio de conocimiento; proporciona el marco de referencia común para todas las aplicaciones en cierto entorno (6).

Una de las primeras definiciones en el área de la ciencia de la información la hizo Neches y su equipo de trabajo: "una ontología define los términos básicos y relaciones incluyendo el vocabulario de un área así como las reglas para la combinación de términos y relaciones para definir ampliaciones de un vocabulario" (7). Se puede decir que esta definición aporta unas líneas a seguir para crear una

ontología: identificar términos básicos y relaciones entre los términos, identificar reglas para combinar las relaciones, aportar definiciones de los términos y las relaciones. Así que según esta definición, una ontología no incluye solo los términos que son explícitamente definidos en ella, sino que también los términos que pueden ser deducidos usando reglas (8).

En 1993, Gruber dio una de las definiciones más empleadas: "las ontologías se definen como una especificación explícita de una conceptualización" (9). En 1997, Borst modificó ligeramente la definición de Gruber diciendo que: "las ontologías se definen como una especificación formal de una conceptualización compartida" (10).

Para Guarino "una ontología es una fuerte estructura semántica que codifica reglas implícitas restringiendo la estructura de una porción de la realidad" (11).

Las ontologías constituyen lenguajes y a su vez sistemas de representación de la información y del conocimiento, que describen diferentes recursos de la información, para su posterior recuperación, lo que también se ha venido usando para mejorar la recuperación de información en la Web, en términos de efectividad, rapidez y facilidad de acceso a la información, a lo que comúnmente se denomina "la Web Semántica" (12).

Las ontologías han sido conceptualizadas por diversos autores, coincidiendo en mayor o menor medida en que son una forma de representación del conocimiento. En el ámbito de la informática las ontologías, expresadas en los diferentes lenguajes creados para su representación, serán la herramienta que permitirá hacer realidad la visión que constituye la Web Semántica o Web 3.0. Una Web donde la recuperación de la información será eficiente y donde las máquinas podrán realizar múltiples procesos que hoy resultan complejos para las personas.

Componentes de una ontología

Las diferentes disciplinas que se encargan del estudio de las ontologías no identifican con exactitud los mismos elementos que la componen o presentan en ocasiones ambigüedad en los términos con que los designan. De manera general se ha llegado al consenso de que las ontologías están compuestas por cinco elementos que permitirán representar el conocimiento de algún dominio: conceptos, relaciones, funciones, instancias y axiomas.

- ✓ **Conceptos:** son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc. Las clases en una ontología se suelen organizar en taxonomías a las que se les pueden aplicar los mecanismos de herencia.

- ✓ Relaciones: representa la interacción y enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio.
- ✓ 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.
- ✓ Instancias: se utilizan para representar objetos determinados de un concepto.
- ✓ Axiomas: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Permiten junto al mecanismo de la herencia de conceptos, inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos (9).

Cada componente de la ontología tiene una implicación en la gestión del conocimiento. Así, los conceptos, las instancias y las relaciones pueden representar el conocimiento tácito de los integrantes de la organización. Las funciones son muy usadas para describir los pasos para desarrollar un proceso. Y los axiomas, permiten hacer inferencias, lo que es de gran utilidad para la toma de decisiones (13).

Leguajes de representación del conocimiento

Para poder explotar la Web Semántica, se necesitan lenguajes de marcado apropiados que representen el conocimiento de las ontologías. W3C publicó una especificación denominada *Resource Description Framework* (RDF), que sería la base de la mayoría de lenguajes ontológicos de la actualidad. Estos lenguajes permiten mediante relaciones taxonómicas, crear una jerarquía de conceptos (16).

RDF es un modelo de datos para representar los recursos y las relaciones que se pueden establecer entre ellos. Además, el modelo de datos RDF se ha definido con una semántica básica que puede representarse mediante XML. Está basado en tripletas formadas por sujeto, predicado y objeto que aportan una forma de expresar enunciados simples acerca de recursos usando sus propiedades y valores.

Web Ontology Language (OWL) es una extensión del modelo RDF, y en ella se redefinen recursos y propiedades, al igual que se le añaden nuevas. Con OWL se pueden definir clases mediante restricciones u operaciones booleanas sobre otras clases, aparecen nuevas relaciones entre clases como la inclusión, disyunción y la equivalencia, se pueden definir restricciones de cardinalidad en propiedades o dar propiedades sobre las relaciones (transitividad, simetría) así como permitir clases enumeradas. OWL está separado en tres niveles:

- ✓ *OWL Lite*: la versión más simple para los programadores principiantes. Permite la jerarquía de *clasificación y las restricciones simples*.
- ✓ *OWL DL*: esta versión ya tiene todo el vocabulario OWL completo. Las limitaciones son que las clases no son instancias ni tipos y los tipos no son ni instancias ni clases. No permite restricciones de cardinalidad en propiedades transitivas. Posee gran expresividad sin perder las propiedades de completitud y decidibilidad.
- ✓ *OWL Full*: esta versión también incluye todo el vocabulario de OWL pero en este caso no hay limitaciones para explotar todo su potencial. Sin garantías computacionales.

OWL Full se considera la más completa de todas y se supone una extensión de DL que a su vez es una extensión de Lite, por lo que toda ontología correcta en *OWL Lite* es una ontología correcta en *OWL DL*, y toda conclusión correcta en *OWL Lite* es una conclusión correcta en *OWL DL* (pero no a la inversa). De la misma manera esto también ocurre con *OWL DL* y *OWL Full* respectivamente (8).

OWL es el lenguaje de especificación de ontologías más completo, principalmente porque está basado fundamentalmente en lógica descriptiva. OWL es desde febrero de 2004 una recomendación de W3C, por lo que la mayoría de desarrolladores e investigadores de la Web Semántica van a centrar sus esfuerzos en desarrollar herramientas y sistemas orientados a este lenguaje (8).

1.3 Repositorios

Un repositorio es una ubicación centralizada donde se almacena información digital. El término deriva del latín *repositorium*, que significa armario, alacena; actualmente el Diccionario de la Real Academia Española lo define como: lugar donde se guarda algo.

En el contexto informático surgen a partir de la década de los 90 como centros auténticos de intercambio de material didáctico, entendidos como almacenes digitales que reúnen las aportaciones individuales de los miembros de una comunidad para ser compartidos y evaluados. Hoy en día muchas universidades e instituciones de todo el mundo han dado marcha a proyectos de repositorios para la organización y difusión de sus contenidos.

Características generales (14):

- ✓ Pueden ser accesibles a través de una red informática o pueden estar contenidos en un medio físico como un disco compacto, una memoria flash o un disco externo.
- ✓ Pueden ser de acceso público o estar protegidos y necesitar de una autenticación previa.
- ✓ Proporcionan un acceso fácil, controlado y estandarizado a los objetos.

- ✓ Asegura la identificación y persistencia de los objetos.
- ✓ Contienen metadatos.
- ✓ Ofrece funciones de administración.
- ✓ Son sostenibles en el tiempo.

Clasificaciones de los repositorios (15):

Por su contenido:

- ✓ General.
- ✓ Especializado (temáticos).

Por su cobertura:

- ✓ Institucional.
- ✓ Nacional.
- ✓ Mundial.

Por su función:

- ✓ Académico.
- ✓ Corporativo.
- ✓ General

Por el tipo de usuarios (audiencia):

- ✓ Interno.
- ✓ Externo.

Por su distribución e interconexión:

- ✓ Centralizado.
- ✓ Distribuido.
- ✓ Híbrido.

Un repositorio de ontologías es un sitio donde se almacenan ontologías, de forma centralizada, que pueden o no estar relacionadas entre ellas.

1.4 Metodología de desarrollo

Según Pressman: “Una metodología de desarrollo de software es un conjunto de reglas e instrucciones relativas al proceso de desarrollo de software” (17).

Rumbaugh, da una definición más abarcadora donde expresa: “Una metodología de ingeniería de software es un proceso para la producción organizada del software, empleando para ello una colección de técnicas predefinidas y convencionales en las notaciones. Una metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada uno de ellos. Los pasos de la producción del software se organizan normalmente en un ciclo de vida consistente en varias fases de desarrollo” (18).

Las metodologías pretenden guiar a los desarrolladores en el proceso de desarrollo de software definiendo artefactos, roles e indicando paso a paso las actividades que se van a realizar mediante prácticas y técnicas recomendadas. Su principal meta es lograr un producto final de calidad y que cumpla con los requisitos del usuario, pero los requisitos son tan diversos y cambiantes, que han dado lugar al surgimiento de una variada cantidad de metodologías de desarrollo, que atendiendo a sus particularidades, es posible clasificarlas en dos grandes grupos: robustas y ágiles.



Figura 1: Proceso de desarrollo de software.

Para decidir que metodología seguir en un proyecto de desarrollo de software se debe tener en cuenta las características del mismo: complejidad, tiempo estimado de desarrollo, cantidad de personal, etc. Una vez que estén bien definidos estos factores, se realiza un análisis preliminar para identificar cual se ajusta más al proyecto. A continuación se valoran las características principales de las metodologías más conocidas y utilizadas en la actualidad.

Metodologías robustas

Las metodologías robustas o pesadas están orientadas al control de los procesos, detallan rigurosamente tanto las tareas y actividades del equipo de desarrollo como los artefactos de software. Establecen rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Son las más tradicionales y altamente recomendadas para proyectos grandes y complejos, donde se requiere de una gran organización.

Dentro de las metodologías pesadas se encuentra el Proceso Unificado de Desarrollo (Rational Unified Process, RUP), que por ser la más completa constituye un ejemplo académico cuando se quieren

estudiar estas metodologías. RUP cuenta con cuatro fases de trabajo (Inicio, Elaboración, Construcción y Despliegue) divididas en flujos de trabajo, esta estructura hace que el desarrollo con RUP sea dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.

Estas metodologías demandan de un numeroso equipo de proyecto y generan una gran cantidad de documentación, que en ocasiones resulta inconveniente para proyectos sencillos y con poco tiempo para el desarrollo, como es el caso que se plantea. Por estas razones se decide desechar este tipo de metodología y analizar las ágiles.

Metodologías ágiles

El término “ágil” se adoptó tras una reunión desarrollada en Utah-EEUU en febrero del 2001, con la participación de 17 expertos del mundo de la industria de software. El resultado del intercambio quedó reflejado en el “Manifiesto ágil”, un documento que describe los fundamentos de las metodologías ágiles. Algunos de ellos son:

- ✓ Se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados.
- ✓ Se hace mucho más importante crear un producto de software que funcione que escribir mucha documentación.
- ✓ El cliente está en todo momento colaborando en el proyecto.
- ✓ Es más importante la capacidad para responder a un cambio realizado que el seguimiento estricto de un plan.

Ventajas de las metodologías ágiles

- ✓ Rápida respuesta a cambios en los requisitos a lo largo del proyecto.
- ✓ Entrega continua y en plazos cortos de software funcional.
- ✓ Trabajo conjunto entre el cliente y el equipo de desarrollo.
- ✓ Minimiza los costos frente a cambios.
- ✓ Importancia de la simplicidad, al eliminar el trabajo innecesario.
- ✓ Atención continua a la excelencia técnica y al buen diseño.
- ✓ Mejora continua de los procesos y el equipo de desarrollo.
- ✓ Evita malentendidos de requerimientos entre el cliente y el equipo.

Cada componente del producto final ha sido probado y satisface los requerimientos.

Proceso Unificado Abierto (Open Unified Process, OpenUP)

OpenUP, define claramente las actividades, roles y sus responsabilidades, logrando mayor productividad en menor tiempo. Hereda características de RUP, el desarrollo es iterativo e incremental, guiado por casos de uso, centrado en la arquitectura y además permite la gestión de riesgos. Propone no utilizar muchos artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del cliente pudiendo ser este modificado, mejorado y extendido. Algo a tener en cuenta con esta metodología es que utiliza solo los procesos que sean necesarios y para ello necesita de trabajadores que sean capaces de distinguir entre los necesarios y los no necesarios para garantizar que el producto no tenga errores.

Scrum

Scrum más que una metodología de desarrollo de software, es una forma de auto-gestión de los equipos de desarrollo de software. Cada equipo de desarrollo decide cómo hacer sus tareas y cuánto van a tardar en ello, además ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro. Básicamente es un framework iterativo e incremental para desarrollar cualquier producto o manejar cualquier trabajo. Permite que los equipos entreguen un conjunto de funcionalidades del sistema en cada iteración, proporcionando la agilidad necesaria para responder rápidamente a los cambios de requisitos. Desafía constantemente a sus usuarios a centrarse en la mejora, y sus sprints¹ proporcionan la estabilidad para tratar las necesidades evolutivas que ocurren en cualquier proyecto (19).

Programación Extrema (Extreme Programming, XP)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (20). Está concebida para proyectos con poco tiempo de desarrollo y equipos pequeños, con pocos roles, por lo que la programación se realiza en parejas. Propone que el diseño debe ser sencillo, que funcione con todas las pruebas, sin lógica duplicada y con el menor número de clases y métodos posible.

¹ Sprint: unidad básica de desarrollo de la metodología de desarrollo Scrum y otras metodologías de desarrollo ágil.

Fundamentación de la metodología seleccionada

OpenUP necesita de trabajadores especializados para identificar que procesos son necesarios y cuáles no. El equipo con que se cuenta para desarrollar la aplicación no tiene experiencia en el campo de acción en que está enmarcada, lo que podría causar indecisiones en determinado momento o trabajo innecesario, ambos casos, se traducen a retrasos en el cronograma. Una vez analizado este riesgo teniendo en cuenta el poco tiempo que se tiene para el desarrollo, se decide no optar por esta metodología.

De las dos propuestas restantes se seleccionó XP, porque está más enfocada a la técnica de desarrollo mientras que Scrum se preocupa más por la gestión del proyecto. Además, con Scrum es difícil hacer cambios una vez comenzada una iteración, generalmente hay que esperar a que concluya para poder realizarlos; por el contrario XP es más flexible al cambio.

1.5 Herramientas y tecnologías

En el proceso de desarrollo del software, la selección de las herramientas y tecnologías a utilizar juega un papel fundamental. Estas facilitan el diseño e implementación del sistema puesto que se especializan en áreas determinadas del proceso, ahorrando tiempo y esfuerzo al equipo de desarrolladores. Una adecuada elección puede incidir directamente en el éxito o fracaso del proyecto.

1.5.1 Lenguaje de programación

Un lenguaje de programación es una construcción mental del ser humano para expresar programas. Está constituido por un grupo de reglas gramaticales, un grupo de símbolos utilizables, un grupo de términos monosémicos (es decir, con sentido único) y una regla principal que resume las demás. Para que ésta construcción mental sea operable en un computador debe existir otro programa que controle la validez o no de lo escrito. A éste se le llama traductor (21).

Java

La compañía Sun Microsystems, creadora del lenguaje Java, lo describe como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”.

Java es un lenguaje multiplataforma que incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Muchos expertos opinan que Java es el lenguaje ideal para aprender la informática moderna, porque incorpora

todos estos conceptos de un modo estándar, mucho más sencillos y claro que con las citadas extensiones de otros lenguajes (22).

Los programas en Java generalmente son compilados y luego interpretados por una máquina virtual. Esta última sirve como una plataforma de abstracción entre la máquina y el lenguaje permitiendo que se pueda "escribir el programa una vez, y correrlo en cualquier lado" (22). Por tales razones se decidió implementar la aplicación que describe este trabajo con Java Server Page (JSP), que es una de las herramientas más poderosas y fácil de usar.

Java Server Pages (JSP)

La tecnología JSP combina HTML y XML con Java Servlet (extensión para el servidor de aplicaciones) y JavaBeans para crear un ambiente, altamente productivo, fiable, interactivo y con un alto rendimiento para el desarrollo de sitios Web en plataformas independientes, facilita la creación dinámica de contenido en el servidor e integra una parte de la plataforma Java como una alternativa a las tecnologías de programación del lado del servidor (23).

JavaScript

JavaScript es un lenguaje de programación creado por la empresa Netscape (creadora de uno de los navegadores más conocido). Es el lenguaje de programación más utilizado en Internet para añadir interactividad a las páginas Web. Un programa en JavaScript se integra en una página Web (entre el código HTML) y es el navegador el que lo interpreta (ejecuta). Es decir el JavaScript es un lenguaje interpretado, no compilado (no se genera ningún tipo de fichero objeto o .exe) (24).

JavaScript es un lenguaje que se integra directamente en páginas HTML. Tiene como características principales las siguientes (25):

- ✓ Es interpretado (que no compilado) por el cliente.
- ✓ Está basado en objetos. No es, como Java, un lenguaje de programación orientada a objetos (OOP). JavaScript no emplea clases ni herencia, típicas de la OOP.
- ✓ Su código se integra en las páginas HTML, incluido en las propias páginas.
- ✓ No es necesario declarar los tipos de variables que van a utilizarse.
- ✓ Las referencias a objetos se comprueban en tiempo de ejecución, por lo tanto no se compila.
- ✓ No puede escribir automáticamente al disco duro.

La ventaja que presenta JavaScript sobre el HTML es que permite crear páginas más dinámicas, lo que las hace más atractivas para el usuario. Para utilizar y dominar el JavaScript es prerequisite indispensable saber HTML.

1.5.2 Entorno de desarrollo

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés), es un programa donde es posible escribir código fuente, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para hacer Debug gráficamente. Permiten desarrollar las aplicaciones de forma mucha más rápida, incorporando en muchos casos librerías con componentes previamente implementados (22).

Para el lenguaje Java existen varios entornos de desarrollo, entre los más usados en la actualidad se encuentran Eclipse, Netbeans y JCreator.

Eclipse Galileo

Eclipse es un IDE para todo tipo de aplicaciones libres, inicialmente desarrollado por IBM, y actualmente gestionado por la Fundación Eclipse. Es una herramienta desarrollada principalmente para el desarrollo de aplicaciones Java que facilita al máximo la gestión de proyectos colaborativos mediante el control de versiones, también es posible exportar e importar proyectos.

La arquitectura de módulos (*plugins*) de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otros componentes o librerías como accesorios que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

Las principales características de este IDE son:

- ✓ Multiplataforma (GNU/Linux, Solaris, Mac OSX, Windows).
- ✓ Soportado para distintas arquitecturas (x86, 64).
- ✓ Estructura de plugin que hace sencillo añadir nuevas características y funcionalidades.
- ✓ Control de versiones con CVS o con Subversión.
- ✓ Resaltado de sintaxis, autocompletado, tabulador de un bloque de código seleccionado, etc.
- ✓ Asistentes para la creación, exportación e importación de proyectos; para generar esqueletos de códigos, etc.

NetBeans 6.9

NetBeans es un exitoso proyecto de código abierto fundado en junio del año 2000 por Sun Microsystems, quien continúa siendo su patrocinador principal. Cuenta con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo (26).

NetBeans supera las deficiencias asociadas a otras plataformas de desarrollo y se abren nuevas posibilidades para alcanzar el desarrollo rápido y eficiente de aplicaciones multiplataforma. En su núcleo, NetBeans IDE es una herramienta de desarrollo escrita puramente sobre la base de la tecnología Java, de modo que puede ejecutarse en cualquier ambiente donde se ejecute este lenguaje, lo cual quiere decir, casi en todas partes. El enfoque de código abierto ha permitido una mayor capacidad de uso con cada nueva versión. Las amplias posibilidades de desarrollo multiplataforma, su facilidad de uso, su cumplimiento de regulaciones, sus perfiles de rendimiento, además de su flexibilidad entre plataformas, hacen que se convierta en el IDE de preferencia para muchos programadores (27).

Con NetBeans se pueden desarrollar aplicaciones de escritorio, Web, para celulares y empresariales. Soporta los lenguajes Java, C/C++, Ruby on Rails, PHP, Groovy, Python, JavaScript, y otros.

Algunas de las características más notables de NetBeans respecto a otros entornos son: (26)

- ✓ Mejoras en el editor de código (completamiento de código más inteligente y resaltado).
- ✓ Soporta Ruby, JRuby y Ruby on Rails.
- ✓ La instalación y actualización es más simple.
- ✓ Enlace de datos con el Swing GUI.
- ✓ Características visuales para el desarrollo Web.
- ✓ Creador gráfico de juegos para celulares.
- ✓ Mejoras para SOA y UML.
- ✓ Soporte para PHP.
- ✓ Soporte para Python.
- ✓ Desarrollo Colaborativo

JCreator Pro 5.00.007

JCreator es un IDE de Xinox Software que le permite manejar archivos de proyectos de Java. Ofrece al usuario una amplia gama de funcionalidades como: Gestión de proyectos, plantillas de proyecto, código de terminación, interfaz de depuración, editor con resaltado de sintaxis, asistentes y una interfaz de usuario totalmente personalizable

Con JCreator se puede compilar directamente o ejecutar el programa Java sin necesidad de activar el documento principal en primer lugar. JCreator encuentra automáticamente el archivo con el método principal o el documento html de soporte para la applet de Java y a continuación se inicia la herramienta adecuada. Está escrito enteramente en C++, lo que hace que sea rápido y eficiente en comparación con otros entornos basados en Java.

El editor de JCreator tiene las características básicas usuales: insertar, borrar y actualizar texto, soporte para copiar, cortar y pegar, deshacer ilimitado entre otras. Además, tiene las siguientes características:

- ✓ Asistente de clases: permite agilizar el proceso de crear clases, ingresar métodos y atributos con este asistente.
- ✓ Números de línea: por defecto enseña líneas con número mientras se ingresa el código. Esto le permite referenciar más fácil el código fuente referenciándolo con estos números de línea.
- ✓ Resaltado de sintaxis: resalta varios elementos de código fuente, como nombres de métodos y palabras claves como new y class.

Fundamentación del entorno de desarrollo seleccionado

Después de analizar los diferentes entornos de desarrollo integrado se decide utilizar Netbeans en su versión 6.9 para la implementación de la aplicación informática por ser el que más se adecúa a las necesidades del proyecto. Es un software libre y multiplataforma, características que cumplen con las políticas de desarrollo de la Universidad con respecto a la independencia tecnológica. Además tiene un alto grado de aceptación para el trabajo con java y JSP entre la comunidad de desarrolladores y es un entorno con el que el grupo de desarrollo de la aplicación está familiarizado. Por otra parte el IDE Eclipse a pesar de ser uno de los más populares presenta como deficiencias la complejidad en el desarrollo de interfaces visuales y la compatibilidad de versiones; muchos plugins requieren versiones específicas para funcionar correctamente. En el caso de JCreator no se decide su utilización en el proyecto por ser un software propietario, que aunque posee una versión libre, esta no posee tantas funcionalidades como la versión propietaria.

1.5.3 Frameworks

Un framework, es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado para ayudar a desarrollar y unir los diferentes componentes de un proyecto, acelerando su proceso de desarrollo.

Framework para el trabajo con ontologías

Para representar el conocimiento mediante ontologías de forma que fuera comprensible por sistemas computarizados se crearon lenguajes como RDF, RDF Schema y OWL. La aparición de estos provocó el desarrollo de un conjunto de frameworks, la mayoría escritos en Java, que permiten manipular las ontologías casi en su totalidad y también razonar en base a ellas.

Sesame

Sesame es un API para Java, es decir, un entorno para el desarrollo de aplicaciones en el lenguaje de programación Java para la Web Semántica. Es un marco de desarrollo para almacenamiento, consulta y razonamiento con RDF y RDF Schema. Puede ser usado como base de datos para RDF y RDF Schema, o como una librería de Java para aplicaciones que necesitan trabajar internamente con RDF.

De manera más general, Sesame proporciona a los desarrolladores de aplicaciones un conjunto de herramientas muy útil para hacer cualquier cosa por uno mismo con RDF.

Un concepto fundamental en el framework de Sesame es el repositorio. Un repositorio es un contenedor de almacenamiento para RDF. Éste puede ser un simple objeto Java en memoria o incluso una base de datos relacional. Para cualquiera de las formas de almacenamiento que se elija es importante darse cuenta de que casi todas las operaciones en Sesame están relacionadas con un repositorio: cuando se añaden datos RDF se incluyen en un repositorio, cuando se realiza una consulta, se pregunta a un repositorio en particular.

SeRQL (*Sesame RDF Query Language*) es un lenguaje de consulta en RDF o RDF Schema que en la actualidad está siendo desarrollado como parte de Sesame. Combina los mejores aspectos de otros lenguajes de consulta (p.ej. RQL, RDQL, N3, etc.) con algunos añadidos nuevos propios.

Algunas de las funcionalidades más importantes que ofrece SeRQL son:

- ✓ Transformación gráfica.
- ✓ Soporta RDF Schema.
- ✓ Soporta tipos de datos de XML Schema.
- ✓ Sintaxis para Path Expressions.
- ✓ Opcionalmente, Path Matching.

Jena

Jena es un framework de código abierto para desarrollar aplicaciones en Java con tecnologías de la Web Semántica. Incluye un motor de inferencia basado en reglas, entornos para generar modelos RDF, RDF Schema, OWL y un intérprete y servidor de SPARQL, además de diversos servicios de almacenamiento como adaptadores a bases de datos relacionales.

Fue desarrollado por HB Labs para manipular metadatos desde una aplicación Java. En la actualidad existen dos versiones:

Jena 1:

- ✓ Principalmente soporte para RDF.
- ✓ Capacidades de razonamiento limitadas.

Jena 2:

- ✓ Incluye además una API para el manejo de ontologías.
- ✓ Soporta el lenguaje OWL.

Jena permite gestionar todo tipo de ontologías (añadir hechos, borrarlos y editarlos), almacenarlas y realizar consultas contra ellas. Soporta RDF, DAML y OWL y es independiente del lenguaje. Los recursos no están ligados estáticamente a una clase java particular.

Incluye varios componentes:

- ✓ ARP: un analizador de RDF.
- ✓ API RDF.
- ✓ API de Ontologías para OWL, DAML y RDF Schema.
- ✓ Subsistema de razonamiento.
- ✓ Soporte para persistencia.
- ✓ RDQL: Lenguaje de consultas de RDF.
- ✓ API RDF de Jena. Permite crear y manipular modelos RDF desde una aplicación Java.

Jena brinda la posibilidad de realizar una validación básica de OWL. Esta validación sólo comprueba la sintaxis, no infiere ni razona. Para validaciones más complejas existe Jena 2, que ofrece soporte para inferencias y detecta la violación de las restricciones definidas en el schema por las instancias.

Jena ofrece mecanismos para añadir nuevos razonadores e incluye un conjunto básico de éstos:

- ✓ OWL Reasoner

- ✓ DAML Reasoner
- ✓ RDF Rule Reasoner
- ✓ Generic Rule Reasoner

Para hacer inferencias debemos crear un modelo inferido a partir de un razonador y a partir de ahí, todas las consultas que se le hagan al modelo inferido devolverán información inferida.

RDQL (RDF Data Query Language) es un lenguaje de consultas para RDF desde un enfoque totalmente declarativo. Considera un modelo RDF como un conjunto de tripletas: (Objeto, Propiedad, Valor). Permite especificar patrones que son mapeados contra las tripletas del modelo para retornar un resultado. Se pueden realizar consultas en RDQL desde una aplicación Java. Para ello se usan las siguientes clases: Query, QueryExecution, QueryEngine, QueryResults, ResultBinding.

Jena permite crear modelos persistentes: son mantenidos de forma transparente al usuario en una base de datos relacional. Jena 2 soporta MySQL, Oracle, PostgreSQL.

Fundamentación del framework para el trabajo con ontologías seleccionado

Sesame y Jena son los framework más populares para desarrollar aplicaciones en java con tecnologías de la Web Semántica. Entre ambos marcos de trabajo se decide utilizar Jena en su versión 2, puesto que tiene soporte para el lenguaje OWL; elemento fundamental para el desarrollo de la aplicación informática que se implementará.

Framework de presentación

Uno de los factores del crecimiento y expansión de la Web ha sido la evolución de las tecnologías que sustentan la infraestructura de desarrollo de software. Dentro de las diferentes opciones de desarrollo para los sitios Web han surgido plugins y herramientas que facilitan el desarrollo de las páginas. Estas herramientas por lo general están enmarcadas dentro de los que se denomina frameworks. Con la utilización de un framework de presentación, ya no es necesario que los desarrolladores tengan un nivel técnico tan alto, puesto que este se encarga de facilitar el desarrollo y estandarizar las páginas.

Ext JS 3.0

Ext JS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores y que permite crear páginas e interfaces Web dinámicas (28). Fue desarrollada por Sencha, originalmente como una extensión de la librería Yahoo User Interface (YUI), pero desde su versión 1.1 puede ejecutarse como una aplicación independiente. Ofrece una gran cantidad de widgets para crear interfaces de usuario complejas. Esta librería incluye (29):

- ✓ Componentes UI del alto performance y personalizables.
- ✓ Modelo de componentes extensibles.
- ✓ Un API fácil de usar.
- ✓ Licencias de código abierto y comercial.

Ventajas de Ext

- ✓ Código reutilizable
- ✓ Independiente o adaptable a frameworks diferentes (prototype, jquery, YUI)
- ✓ Orientada a la programación de interfaces tipo desktop en el Web.
- ✓ El API es homogeneizado independientemente del adaptador usado. Los controles siempre se verán igual.
- ✓ Soporte comercial
- ✓ Una extensa comunidad de usuarios

Dojo Toolkit

Dojo Toolkit es un popular framework javascript de código abierto para la creación de aplicaciones Web dinámicas. Nace en el 2004 producto del aumento en la demanda de aplicaciones Web con mejores características tanto de diseño como de uso, además de esto hubieron otros factores que influyeron en el nacimiento de este Framework entre los cuales se pueden encontrar: (30)

- ✓ Incompatibilidades en el cumplimiento de los estándares por parte de los Navegadores.
- ✓ El surgimiento de AJAX generó nuevos desafíos para los desarrolladores y diseñadores.
- ✓ La Web 2.0 generó nuevas oportunidades tanto para el desarrollo de servicios como para la evolución de las tecnologías.

Dojo está ganando popularidad rápidamente entre la comunidad de desarrolladores, convirtiéndose en una de los framework más usados para el desarrollo de aplicaciones con Ajax. Proporciona una API bien concebido y un conjunto de herramientas para solucionar los problemas que se experimentan habitualmente en el desarrollo Web.

Características (31):

- ✓ Comunicación asíncrona.
- ✓ Sistema de paquetes.
- ✓ Almacenamiento en el cliente de datos.
- ✓ Almacenamiento en el servidor.
- ✓ Manejo de Eventos.

- ✓ Gráficos.
- ✓ Manipulación de DOM (*Document Object Model*).
- ✓ Animaciones.
- ✓ Apis para el manejo de Ajax y Cometd.
- ✓ Conjunto de Componentes Reutilizables (widgets).
- ✓ Soporte nativo para vector gráfico 2D y 3D.

Estructura de Dojo Toolkit

El proyecto está dividido en tres grandes paquetes

- ✓ Dijit : Todos los widget o componentes reutilizables que permiten la interacción con el usuario
- ✓ Dojo : Core de framework
- ✓ Dojox: Dojo Experimental y de otros proveedores
- ✓ Util : Librerías utilitarias del Framework

Fundamentación del framework de presentación seleccionado

Se decidió el uso de Dojo Toolkit en su versión 1.3 como framework de presentación para el desarrollo de las interfaces visuales de la aplicación por su perfecta integración con JSP, lenguaje que se utilizará para la implementación de las funcionalidades fundamentales. Además Dojo se presenta bajo las licencias *Academic Free License* (AFL) y BSD; ambas otorgan amplios derechos a utilizar y construir en y con Dojo, tanto en código abierto o comercial. Por su parte ExtJS ha decidido cambiar su licencia *open-source* LPGL por el modelo GPLv3 a partir de su versión 2.1 que si bien es una licencia creada por la fundación de software libre, en sus clausulas compromete a los desarrolladores al uso de esta en todo software que utilice en su desarrollo código licenciado con GPLv3.

1.5.4 Servidor de aplicaciones

Un servidor de aplicaciones o servidor Web es un programa que se ejecuta continuamente en un computador, manteniéndose a la espera de peticiones de ejecución que le hará un cliente o un usuario de Internet. El servidor Web se encarga de contestar a estas peticiones de forma adecuada, entregando como resultado una página Web o información de todo tipo de acuerdo a los comandos solicitados. En este punto es necesario aclarar lo siguiente: mientras que comúnmente se utiliza la palabra servidor para referirnos a una computadora con un software servidor instalado, en estricto rigor un servidor es el software que permite la realización de las funciones descritas.

Apache Tomcat 6.0.26

Es un servidor de aplicaciones que funciona como un contenedor de servlets. Es la implementación de referencia de las especificaciones de servlets 2.5 y de *Java Server Pages* (JSP) 2.1, especificaciones para *Java Community Process*, usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Apache Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Glassfish

GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. La versión comercial es denominada Oracle GlassFish Enterprise Server. Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.

Fundamentación del servidor de aplicaciones seleccionado

Ambos servidores presentan similares características y muy buenas potencialidades. El equipo de desarrollo de este trabajo decidió utilizar Apache Tomcat, debido a que es el más utilizado en la actualidad, se ejecuta sobre cualquier plataforma que tenga instalada la máquina virtual de Java (jdk) y soporta un alto nivel de tráfico.

1.5.5 Sistema gestor de base de datos

Los sistemas gestores de Base de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de consulta. Estos permiten manejar con facilidad grandes volúmenes de información en muy poco tiempo y ofrecen independencia y seguridad en el tratamiento de información.

El propósito general de los sistemas de gestión de Base de Datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, para un buen manejo de datos. Dentro de los SGBD más utilizados se encuentran MySQL y PostgreSQL

PostgreSQL 8.4

PostgreSQL es un potente sistema gestor de base de datos relacional libre. Fue desarrollado en la Universidad de California, en el departamento de Ciencias de la Computación. Posee más de 15 años

de activo desarrollo y arquitectura probada que le han ganado una muy buena reputación por su confiabilidad e integridad de datos. Funciona en los principales sistemas operativos, incluyendo las distribuciones de GNU/Linux, las variantes de UNIX y Windows.

Principales características (32):

- ✓ Soporta casi toda la sintaxis SQL, tiene soporte total para llaves foráneas, uniones, vistas, disparadores y procedimientos almacenados.
- ✓ Usa una arquitectura cliente/servidor.
- ✓ Tiene soporte interno para lenguajes procedurales como Python, Perl, TCL, los cuales permiten agregarle mayor dinamismo al sistema.
- ✓ Soporta herencia de tablas.
- ✓ Incluye la mayoría de los tipos de datos de SQL92 y SQL99.
- ✓ Posee puntos de recuperación a un momento dado.
- ✓ Replicación asincrónica.
- ✓ Copia de seguridad en línea.

Ventajas:

- ✓ Máximo tamaño de base de datos ilimitado.
- ✓ Máximo tamaño de tabla 32 TB.
- ✓ Máximo tamaño de tupla 1.6 TB.
- ✓ Máximo tamaño de campo 1 GB.
- ✓ Máximo de tuplas por tabla ilimitado.
- ✓ Máximo columnas por tabla 250 - 1600 dependiendo del tipo de columnas.
- ✓ Máximo de índices por tabla ilimitado.

MySQL 5.0

Este gestor de bases de datos presenta un diseño multihilo que le permite soportar una gran carga de manera muy eficiente. MySQL fue creada por la empresa sueca MySQL AB. Es probablemente el gestor de base de datos más usado en el mundo, por su gran rapidez y facilidad de uso. Esta gran aceptación se debe, en parte, a que existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración (33).

A pesar de que MySQL 5 es un proyecto básicamente Open Source, la propiedad real es de Sun Microsystems, a su vez propiedad de Oracle. Estas empresas ofrecen soporte y fomentan el desarrollo de este estupendo sistema de gestión de bases de datos (SGDB).

Características de MySQL (33):

- ✓ Arquitectura Multihilo, múltiples clientes tienen acceso concurrente. Cachea el resultado de las consultas comunes.
- ✓ Interfaz de línea de comandos y herramientas gráficas.
- ✓ Múltiples usuarios tienen acceso concurrente a una o más bases de datos simultáneamente. Sistema de privilegios de usuarios potente y flexible.
- ✓ Esquemas de autenticación basados en usuario – máquina.
- ✓ Funciona en diferentes plataformas: Linux, Solaris, Windows, etc.
- ✓ Base de datos de aplicaciones para Escritorio y la Web. APIs (*Application Programming Interface*, o Interfaz de Programación de Aplicaciones) para: C/C++, Java, PHP, Perl, Python, Ruby.
- ✓ Procedimientos y funciones almacenadas y disparadores.
- ✓ Escribe los datos en almacenamiento persistente. Motores de almacenamiento conectable, permite que el motor de almacenamiento sea cargado y/o cambiado dinámicamente en tiempo de ejecución. Algunos de los motores que utiliza son: HEAP, MyISAM, CSV, Falcon, Cluster, y su propio motor.
- ✓ Sistema de asignación de memoria basado en hilos. Tablas temporales o tablas virtuales, formadas por consultas SQL son implementadas en tablas hash en memoria.

Fundamentación del SGBD seleccionado

Se decidió utilizar PostgreSQL como sistema gestor de base de datos puesto que es una estrategia de la universidad la migración hacia este software. PostgreSQL es un proyecto de código abierto soportado por una gran comunidad de desarrollo, mientras que MySQL es propiedad de la empresa privada Oracle, la cual posee el copyright de la mayor parte del código.

Conclusiones del capítulo

En este capítulo se analizaron los principales conceptos sobre las ontologías, la Web Semántica y los repositorios de información para una mayor comprensión del problema de investigación. Se valoraron diferentes metodologías de desarrollo para optar por la que más se ajusta a las necesidades del proyecto, quedando seleccionada XP. También se identificaron las principales herramientas y tecnologías a usar para el desarrollo de este tipo de aplicación, donde se eligió JSP como lenguaje de

programación para la implementación utilizando NetBeans 6.9 como entorno de desarrollo, además el Framework Jena en su versión 2.6.4 para el trabajo con las ontologías, Apache Tomcat 6.0.26 como servidor de aplicaciones y PostgreSQL 8.4 como sistema gestor de base de datos.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL SISTEMA

2.1 Introducción

En este capítulo se describe la propuesta de solución para el problema de la investigación planteado. Se establecen las principales características y funcionalidades con que contará la aplicación informática reflejadas 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.

2.2 Descripción de la solución propuesta

Para darle solución al problema de la investigación identificado se propone el desarrollo de una aplicación informática que permita gestionar repositorios de ontologías representativas del conocimiento en la Web. Debido a que desde febrero de 2004 *Web Ontology Language* (OWL) se convirtió en recomendación de la W3C como lenguaje de especificación de ontologías, la aplicación estará diseñada para el trabajo con ontologías escritas con este formato.

La aplicación contará con diferentes niveles de privilegios para acceder a sus funcionalidades atendiendo al tipo de usuario que interactúe con ella. Los tipos de usuario identificados son: usuario común, usuario autenticado, gestor y administrador en ese orden de jerarquía; cada usuario podrá hacer uso de las funcionalidades propias de su nivel y las de niveles inferiores.

La aplicación permitirá gestionar un repositorio local e incorporar contenidos de repositorios remotos con el objetivo de brindar a los usuarios la mayor cantidad de información posible.

Las funcionalidades que la aplicación deberá brindar según el tipo de usuario son:

Para el usuario común:

- ✓ acceder a la página principal (inicio).
- ✓ visualizar la lista de repositorios.
- ✓ caracterizar un repositorio, clasificando su contenido (las ontologías) por dominios del conocimiento.
- ✓ visualizar la caracterización del repositorio mediante gráficas de barra y pastel.
- ✓ buscar ontología
- ✓ visualizar ontología
- ✓ descargar ontología
- ✓ comparar el contenido de los repositorios mediante gráficas

- ✓ ver listado de dominios del conocimiento asociados a la aplicación.
- ✓ registrarse

Para el usuario autenticado:

- ✓ visualizar su perfil de usuario
- ✓ editar (actualizar) su perfil de usuario.
- ✓ subir ontologías al repositorio local.

Para el usuario gestor:

- ✓ gestionar (insertar, eliminar, editar) dominios del conocimiento

Para el usuario administrador:

- ✓ gestionar usuario (eliminar, actualizar, asignar roles)
- ✓ gestionar repositorios
- ✓ actualizar repositorio

Roles de usuarios

Tabla 1: Roles de Usuarios

| Roles | Descripción |
|--------------------|---|
| Usuario común | Toda persona que acceda a la aplicación sin autenticarse. |
| Usuario registrado | Usuario con un perfil creado en la aplicación. |
| Gestor | Es el encargado de gestionar todo lo referente a los dominios del conocimiento con que trabajará la aplicación. |
| Administrador | Usuario con todos los privilegios sobre la aplicación, encargado de gestionar los repositorios y usuarios. |

2.3 Requerimientos

La ingeniería de requisitos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos del sistema y el papel asignado al software.

El análisis de requisitos es una tarea de ingeniería del software que cubre el hueco entre la definición del software a nivel sistema y el diseño de software. El análisis de requerimientos permite al ingeniero de sistemas especificar las características operacionales del software (función, datos y rendimientos), indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software (17).

Los requisitos del software son la descripción de los servicios y restricciones de un sistema, es decir, lo que el software debe hacer y bajo qué circunstancias debe hacerlo (34).

Requisitos no funcionales.

Los requisitos no funcionales describen atributos sólo del sistema o del ambiente del sistema que no están relacionados directamente con los requisitos funcionales. Los requisitos no funcionales incluyen restricciones cuantitativas, como el tiempo de respuesta o precisión, diseño de la interfaz, lenguajes de programación y/o sistemas operativos, etc. (34)

Software

Cliente: Navegador Mozilla Firefox 3.4 o superior o Internet Explorer 5.0 o superior.

Debido a que la estrategia de la Universidad de las Ciencias Informáticas es la completa migración al software libre, con el objetivo de lograr la independencia tecnológica, el servidor donde se instalará la aplicación deberá contar con los siguientes requisitos:

Servidor:

Multiplataforma.

El sistema gestor de base de datos PostgreSQL 8.2 o una versión superior.

Servidor de aplicaciones Apache Tomcat 6.0.

Hardware

Los requerimientos mínimos de hardware para el correcto funcionamiento de la aplicación son:

Cliente: Procesador con frecuencia de 2.0 GHz o superior, 128 MB de RAM, 100 MB libres en el disco duro.

Servidor: Procesador con frecuencia de 3.0 GHz, 512 MB de RAM, 10 GB libres en el disco duro.

Seguridad

Los requisitos vinculados a la seguridad de la aplicación son complicados de gestionar y pueden provocar los mayores riesgos sino se manejan correctamente. La seguridad informática en toda aplicación, pero especialmente en las desarrolladas para desplegarse en la Web, puede ser tratada en tres aspectos diferentes: confidencialidad, integridad y disponibilidad.

Confidencialidad: La aplicación informática no maneja información personal o sensible para los usuarios por lo los requisitos de seguridad en este aspecto son mínimos. La información del perfil de cada usuario solamente podrá ser modificada por el propio o por el administrador de la aplicación.

Integridad: La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes, de la misma forma será considerada igual a la fuente o autoridad de los datos.

Disponibilidad: Los usuarios tendrán acceso a la información de forma ininterrumpida. Los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán a los usuarios para obtener los datos deseados en un momento dado. Los usuarios que juegan determinados roles en la aplicación (entiéndase administrador y gestor) deberán de autenticarse para acceder a las funcionalidades específicas de cada uno de ellos.

Restricciones en el diseño y la implementación

El software deberá ser una aplicación Web desarrollada utilizando herramientas de software libre en su totalidad. El diseño estará basado en el patrón modelo-vista-controlador apoyándose en los patrones GRAPS para la implementación de las clases.

2.4 Fase de planificación

La metodología XP plantea la planificación como un permanente diálogo entre las partes involucradas en el proceso de desarrollo del software; la parte empresarial (deseable) y la técnica (posible). En esta fase las personas del negocio necesitan determinar las posibles funcionalidades de la aplicación, generalmente representadas por historias de usuarios, y establecer prioridades y estimaciones de su realización. Se determinan además los elementos que van a componer cada una de las versiones así como la fecha de entrega de las mismas.

2.4.1 Historias de usuario

El primer paso de cualquier proyecto que siga la metodología XP es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso pero con algunas diferencias: Constan de 3 ó 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia

de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas (35).

Las historias de usuario son una forma rápida de administrar los requerimientos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requerimientos cambiantes.

En los intercambios con el cliente se identificaron y redactaron 19 historias de usuarios donde se describen las funcionalidades con que deberá contar la aplicación en su versión final. A continuación se muestran algunas de las historias más significativas, el resto está plasmado en el expediente de proyecto de la aplicación.

Historias de usuarios identificadas

Tabla 2: Historia de usuario #1

| Historia de Usuario | |
|---|-----------------------------|
| Número: 1 | Nombre: Autenticar usuario |
| Usuario: administrador, gestor, usuario autenticado | |
| Prioridad en Negocio: Alta | Riesgo de desarrollo: Medio |
| Puntos Estimados: 0.5 | Iteración asignada: 1 |
| Programador responsable: Michael E. Marrero Clark | |
| Descripción: Los usuarios administrador, gestor o usuario autenticado deberán autenticarse para acceder a la aplicación, donde podrán realizar las tareas asignadas según su rol. Datos para autenticarse: *Nombre del Usuario *Contraseña | |
| Observaciones: Los campos con (*) son obligatorios. | |

Tabla 3: Historia de usuario #3

| Historia de Usuario | |
|------------------------|----------------------------|
| Número: 3 | Nombre: Registrar usuario. |
| Usuario: usuario común | |

| | |
|---|-----------------------------|
| Prioridad en Negocio: Media | Riesgo de desarrollo: media |
| Puntos Estimados: 0.5 | Iteración asignada: 1 |
| Programador responsable: Michael E. Marrero Clark | |
| <p>Descripción: El usuario común podrá registrarse en el sistema si desea acceder a la funcionalidad de subir archivos al repositorio. Para efectuar el registro deberá llenar los campos obligatorios:</p> <p>*Nick:</p> <p>*Contraseña:</p> <p>*Email:</p> <p>*Nombre completo:</p> <p>Teléfono:</p> <p>Ocupación:</p> <p>Institución a la que pertenece:</p> | |
| Observaciones: Los campos con (*) son obligatorios. | |

Tabla 4: Historia de usuario #10

| Historia de Usuario | |
|--|----------------------------------|
| Número: 10 | Nombre: Caracterizar repositorio |
| Usuario: usuario común | |
| Prioridad en Negocio: Alta | Riesgo de desarrollo: Alto |
| Puntos Estimados: 1.5 | Iteración asignada: 2 |
| Programador responsable: Michael E. Marrero Clark | |
| <p>Descripción: Cuando el usuario seleccione un repositorio la aplicación mostrará un sumario con información sobre el contenido del mismo:</p> <ul style="list-style-type: none"> -Total de ontologías contenidas en el repositorio. -Total de ontologías pertenecientes a un dominio u otro. | |
| Observaciones: El usuario tendrá acceso a esta funcionalidad sin necesidad de algún tipo de autenticación; bastará con sólo acceder a la aplicación. | |

Tabla 5: Historia de usuario #11

| Historia de Usuario | |
|---------------------|--------------------------------------|
| Número: 11 | Nombre: Graficar caracterización del |

| | |
|--|-----------------------------|
| | repositorio |
| Usuario: usuario común | |
| Prioridad en Negocio: Media | Riesgo de desarrollo: Medio |
| Puntos Estimados: 1.0 | Iteración asignada: 3 |
| Programador responsable: Leonel Vila Pérez | |
| Descripción: Cuando el usuario seleccione un repositorio, deberán aparecer las opciones para visualizar su caracterización en gráficas de pastel y de barra. | |
| Observaciones: El usuario tendrá acceso a esta funcionalidad sin necesidad de algún tipo de autenticación; bastará con sólo acceder a la aplicación. | |

Tabla 6: Historia de usuario #12

| | |
|--|--------------------------------|
| Historia de Usuario | |
| Número: 12 | Nombre: Gestionar repositorios |
| Usuario: administrador | |
| Prioridad en Negocio: Alta | Riesgo de desarrollo: Alta |
| Puntos Estimados: 1.5 | Iteración asignada: 3 |
| Programador responsable: Leonel Vila Pérez | |
| Descripción: Esta funcionalidad solo estará disponible para los usuarios administradores, quienes podrán insertar nuevos repositorios además de modificar o eliminar los existentes. Cada repositorio tendrá asociado la dirección de internet y un código java que deberá especificar el administrador, el cual será ejecutado cuando se desee actualizar el repositorio. | |
| Observaciones: Los usuarios que inserten o modifiquen los repositorios en la aplicación deberán tener conocimiento del lenguaje de programación Java. | |

Tabla 7: Historia de usuario #14

| | |
|----------------------------|--------------------------------|
| Historia de Usuario | |
| Número:14 | Nombre: Actualizar repositorio |
| Usuario: administrador | |
| Prioridad en Negocio: Alta | Riesgo de desarrollo: Medio |

| | |
|---|-----------------------|
| Puntos Estimados: 1.0 | Iteración asignada: 3 |
| Programador responsable: Leonel Vila Pérez | |
| Descripción: El administrador tendrá la posibilidad de actualizar el contenido de los repositorios. Cuando este ejecute esta opción, la aplicación deberá conectarse a la ubicación original en Internet del repositorio y descargar las nuevas ontologías que pudieran haberse incorporado al mismo. | |
| Observaciones: | |

Tabla 8: Historia de usuario #18

| Historia de Usuario | |
|---|-----------------------------|
| Número: 18 | Nombre: Descargar ontología |
| Usuario: usuario común | |
| Prioridad en Negocio: Media | Riesgo de desarrollo: Media |
| Puntos Estimados: 1.0 | Iteración asignada: 4 |
| Programador responsable: Michael E. Marrero Clark | |
| Descripción: Los usuarios podrán descargar ontologías de la aplicación. Esta opción deberá estar disponible siempre que el usuario acceda a una ontología, ya sea como resultado de una búsqueda realizada o por la simple navegación por los repositorios. La descarga será de un fichero de extensión owl (fichero.owl) y el usuario podrá especificar donde se almacenará. | |
| Observaciones: El destino de la descarga será la PC del usuario que la solicite. | |

2.4.2 Estimación de esfuerzo por Historia de Usuario

Dentro de la fase de planificación la estimación de esfuerzo por historia de usuario es un elemento esencial. En este paso se determina el costo asociado a la implementación de cada historia, utilizando como medida el punto. Un punto equivale a una semana ideal de programación, donde el equipo de desarrollo trabaja bajo condiciones adecuadas y sin interrupciones.

La estimación del costo de la implementación de las historias de usuarios es establecida por los programadores, tomando valores que por lo general no exceden los 3 puntos. Esto permite al equipo llevar el registro de la velocidad de desarrollo del proyecto establecida en puntos por iteración.

La velocidad del proyecto es una medida que representa la rapidez con la que se desarrolla el proyecto; estimarla es muy sencillo, basta con sumar los puntos para el total de historias de usuario que se implementaron en la iteración precedente; de esta forma, se sabrá el cupo de historias que se podrán desarrollar en la siguiente iteración. Usando la velocidad del proyecto controlaremos que todas las tareas se puedan desarrollar en el tiempo del que dispone la iteración.

Tabla 9: Estimación de esfuerzo por puntos.

| No. | Historias de Usuario | Puntos Estimados |
|-------|--|------------------|
| 1 | Autenticar usuario | 0.5 |
| 2 | Gestionar usuario | 1.0 |
| 3 | Registrar usuario | 0.5 |
| 4 | Editar perfil de usuario | 0.5 |
| 5 | Visualizar perfil de usuario | 0.5 |
| 6 | Filtrar usuarios | 0.5 |
| 7 | Buscar usuario | 0.5 |
| 8 | Gestionar dominio del conocimiento | 1.0 |
| 9 | Listar dominios del conocimiento | 0.5 |
| 10 | Caracterizar repositorios | 1.5 |
| 11 | Graficar caracterización del repositorio | 1.0 |
| 12 | Gestionar repositorios | 1.5 |
| 13 | Comparar repositorios (mediante gráficas). | 0.5 |
| 14 | Actualizar repositorio | 1.0 |
| 15 | Visualizar ontologías | 1.0 |
| 16 | Buscar ontología | 1.0 |
| 17 | Subir ontología | 1.0 |
| 18 | Aceptar ontología | 1.0 |
| 19 | Descargar ontología | 1.0 |
| Total | | 16 |

2.4.3 Plan de iteraciones

El plan está compuesto por iteraciones que no exceden las cuatro semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fueren la creación de esta arquitectura, sin

embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del plan de iteraciones son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores (36).

Una vez identificadas las historias de usuario descritas por el cliente y realizada la estimación del esfuerzo por puntos de cada una de ellas, se establecieron cuatro iteraciones para el desarrollo de la aplicación. En la selección de las historias de usuario a implementar se tuvieron en cuenta la prioridad que estas presentan para el negocio y la relación en cuanto a funcionalidad entre las mismas.

Iteración 1

En esta iteración se implementarán desde la historia de usuario número 1 hasta la 7, todas referente a la autenticación, gestión y registro de usuarios en la aplicación, así como la gestión de sus perfiles. Estas historias son imprescindibles puesto que determinan a que funcionalidades podrá acceder cada usuario de acorde a los roles y privilegios asignados.

Iteración 2

Al inicio de la iteración, como propone la metodología XP, se revisan y corrigen las posibles no conformidades o historias de usuario que no superen las pruebas de aceptación de la iteración anterior. Luego se desarrollan las historias de usuario 8, 9, 10 y 11 referentes a la gestión de los dominios y caracterización del repositorio. Al concluir la iteración deberá ser posible caracterizar el repositorio por los dominios definidos y representar dicha caracterización mediante gráficas.

Iteración 3

Al igual que en la iteración anterior se corrigen las disconformidades o errores y luego se implementan las historias de usuario número 12, 13, 14 y 15. Al concluir la implementación de las mismas, la aplicación brindará información visual mediante graficas que permitirán establecer criterios de comparación entre los diferentes repositorios indexados a la aplicación y el propio repositorio local que maneja. Se podrán incorporar repositorios remotos y actualizar su contenido (ontologías), además se podrá visualizar el contenido de cualquier ontología seleccionada.

Iteración 4

En esta iteración se implementan las historias de usuario 16, 17, 18 y 19 referentes a la gestión de las ontologías contenidas en la aplicación a través de las funcionalidades básicas buscar, subir y descargar. Una vez desarrolladas estas funcionalidades los usuarios podrán interactuar e incidir directamente sobre el contenido de los repositorios, aunque siempre estará sujeto a la supervisión del administrador de la aplicación.

Como resultado de cada iteración se obtiene una versión de la aplicación cuyas funcionalidades se van incrementando en la medida que estas se vayan realizando. La fecha entrega de cada una de estas versiones se refleja en el plan de entrega y está condicionada con el plan de duración de las iteraciones.

2.4.5 Plan de duración de las iteraciones

Para realizar el plan de duración de las iteraciones se tuvieron en cuenta factores como la estimación del esfuerzo por puntos de cada historia, la velocidad del proyecto y el tiempo con que dispone el equipo para el desarrollo de la aplicación. La duración de cada iteración se pactó entre las partes involucradas.

Tabla 10: Plan de duración de las iteraciones.

| Iteraciones | Historias de usuario | Duración de las iteraciones |
|-------------|--|-----------------------------|
| Iteración 1 | Autenticar usuario | 4 semanas |
| | Gestionar usuario | |
| | Registrar usuario | |
| | Editar perfil de usuario | |
| | Visualizar perfil de usuario | |
| | Filtrar usuarios | |
| Iteración 2 | Buscar usuario | 4 semanas |
| | Gestionar dominio del conocimiento | |
| | Listar dominios del conocimiento | |
| | Caracterizar repositorio | |
| Iteración 3 | Graficar caracterización del repositorio | 4 semanas |
| | Comparar repositorios | |
| | Gestionar repositorios | |
| | Actualizar repositorio | |

| | | |
|-------------|-----------------------|------------|
| | Visualizar ontologías | |
| Iteración 4 | Buscar ontología | 4 semanas |
| | Subir ontología | |
| | Aceptar ontología | |
| | Descargar ontología | |
| Total | | 16 semanas |

Durante el desarrollo de la iteración y en particular al concluir esta, se realiza un seguimiento del plan de duración para determinar si se está cumpliendo con lo previsto o si es necesario hacer reajustes en el mismo y llegar a nuevos acuerdo con el cliente.

2.4.6 Plan de entregas

El plan de liberaciones tiene como objetivo definir el número de entregas que se cumplirán a lo largo del período de vida del proyecto y las iteraciones necesarias para desarrollar cada una. Las entregas se corresponden con una iteración de desarrollo, obteniéndose como resultado un prototipo del sistema con las funcionalidades descritas en las historias de usuarios definidas a implementar en la iteración.

Tabla 11: Plan de entregas.

| Aplicación informática para la gestión de repositorios de ontologías representativas del conocimiento en la Web | |
|---|-------------------------|
| Número de versión | Fecha de entrega |
| Versión 0.2 | Segunda semana de marzo |
| Versión 0.5 | Segunda semana de abril |
| Versión 0.8 | Segunda semana de mayo |
| Versión 1.0 | Primera semana junio |

2.5 Fase de diseño

La metodología XP propone que el diseño de la aplicación ha de ser lo más simple posible, siempre y cuando cumpla con las funcionalidades especificadas por el cliente. La complejidad innecesaria y el código extra debe ser removido inmediatamente. Kent Beck, creador de XP, plantea que en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos (36).

Para el diseño de aplicaciones informáticas XP no se requiere la presentación del sistema mediante diagramas utilizando UML como lenguaje de modelado. En su lugar se utilizan otras técnicas como las tarjetas CRC (Contenido, Responsabilidad, Colaboración).

2.5.1 Tarjetas CRC

Las tarjetas CRC fueron presentadas por Beck y Cunningham² en 1989 para la enseñanza de la Programación Orientada a Objetos (OOP). Desde entonces, esta técnica ha sido propuesta también para el modelado conceptual y/o diseño detallado de sistemas orientados a objetos. La característica más sobresaliente de las tarjetas CRC es su simpleza y ductilidad, ya que una tarjeta CRC no es más que una ficha de papel o cartón que representa a una entidad del sistema, a las cuales asigna responsabilidades y colaboraciones. El formato físico de las tarjetas CRC facilita la interacción entre clientes y equipo de desarrollo, en sesiones en las que se aplican técnicas de grupos como tormenta de ideas o juego de roles, y se ejecutan escenarios a partir de especificación de requisitos o historias de usuarios. De esta forma, van surgiendo las entidades del sistema junto con sus responsabilidades y colaboraciones. Luego en un estado de diseño avanzado o ya en la implementación del sistema, las tarjetas CRC se convierten en clases con métodos, atributos, relaciones de herencia, composición o dependencia (37).

Durante el proceso de diseño de la aplicación se elaboraron un total de ocho tarjetas CRC. A continuación se muestran algunas de las más significativas, el resto está plasmado en el expediente de proyecto de la aplicación.

Tabla 12: Tarjeta CRC para la clase Conexion.

| Tarjeta CRC | |
|--|----------------|
| Clase: Conexion | |
| Responsabilidades | Colaboraciones |
| Establecer la conexión a la base de datos. Ejecutar consultas de selección en la base de datos. Ejecutar consultas de modificación a la base de datos. Cerrar la conexión a la base de datos. | |

² Howard G. Cunningham, pionero en el desarrollo de la programación extrema y en el desarrollo de patrones de programación.

Tabla 13: Tarjeta CRC para la clase ModeloDatos.

| Tarjeta CRC | |
|--|-----------------|
| Clase: ModeloDatos | |
| Responsabilidades | Colaboraciones |
| Abre la conexión a la base de datos, ejecuta consultas de selección o modificación y cierra la conexión. | <i>Conexion</i> |

Tabla 14: Tarjeta CRC para la clase Control.

| Tarjeta CRC | |
|---|----------------|
| Clase: Control | |
| Responsabilidades | Colaboraciones |
| Encargada de controlar el sistema. Carga en memoria el directorio raíz del repositorio. Controla la gestión de los repositorios. Actualiza los dominios en el sistema. Crea la estructura del árbol de navegación en el menú. | |

Tabla 15: Tarjeta CRC para la clase Repositorio.

| Tarjeta CRC | |
|---|-----------------------|
| Clase: Repositorio | |
| Responsabilidades | Colaboraciones |
| Devuelve la información asociada al repositorio (directorio, nombre, ruta, etc). Carga los dominios de la base de datos. Caracteriza las ontologías por dominios. Actualiza el contenido del repositorio de su ubicación real en Internet. | <i>ModeloDominios</i> |

Tabla 16: Tarjeta CRC para la clase Dominio.

| Tarjeta CRC | |
|--|------------------|
| Clase: Dominio | |
| Responsabilidades | Colaboraciones |
| Devuelve la información asociada al dominio. Comprueba si una ontología pertenece al dominio. Caracteriza el dominio. Crea la estructura para el árbol de navegación. | <i>Ontologia</i> |

Tabla 17: Tarjeta CRC para la clase Ontologia.

| Tarjeta CRC | |
|---|----------------|
| Clase: Ontologia | |
| Responsabilidades | Colaboraciones |
| Devuelve la información asociada a la ontología (fichero .owl). Devuelve si la ontología ha sido seleccionada por algún dominio. | |

2.5.2 Patrones de diseño

Un patrón describe un problema que ocurre una y otra vez en nuestro entorno y el núcleo de la solución al problema, de forma que puede utilizarse en ilimitadas ocasiones sin tener que hacer dos veces lo mismo.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular (38).

Patrón Modelo – Vista – Controlador (MVC)

Cuando las aplicaciones en general comienzan a ser más complejas y grandes, con un gran número de componentes y procedimientos, se comienza a mezclar el código de acceso a las bases de datos, el código referente a la lógica de negocio e incluso el código referente a la presentación.

El resultado de esto son aplicaciones difíciles de mantener ya que la relación entre los componentes causa efectos colaterales cada vez que se realiza algún cambio. En definitiva, se dificulta o imposibilita

la reutilización del código por causa de las dependencias de muchas clases lo que obliga a volver a codificar y a reescribir funciones y aplicaciones enteras nuevamente.

El patrón de diseño Modelo-Vista-Controlador intenta resolver este problema a través de la separación del acceso a los datos, la lógica de la aplicación y la interacción del usuario (39).

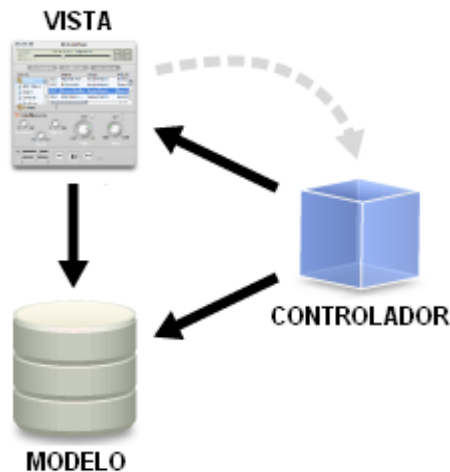


Figura 2: Patrón de diseño Modelo – Vista – Controlador.

El modelo

El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, en el caso de la aplicación que se desea desarrollar estaría compuesto por repositorios, dominios, ontologías y usuarios. No tiene en cuenta ni la forma en la que esta información va a ser mostrada ni los mecanismos que hacen que estos datos estén dentro del modelo, es decir, no tiene relación con ninguna otra entidad dentro de la aplicación.

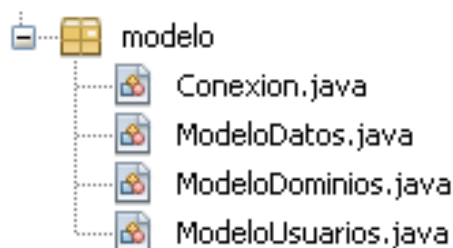


Figura 3: Clases del Modelo

Las vistas

Las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.

Una vista obtiene del modelo solamente la información que necesita para desplegar y se actualiza cada vez que el modelo del dominio cambia por medio de notificaciones generadas por el modelo de la aplicación (39).

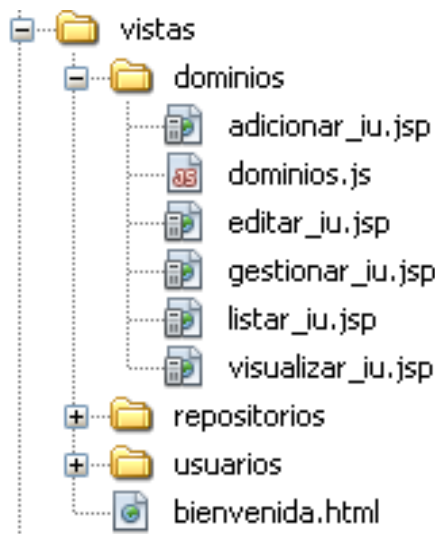


Figura 4: Páginas de la Vistas

El controlador

El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador (40).



Figura 5: Clases y páginas controladoras

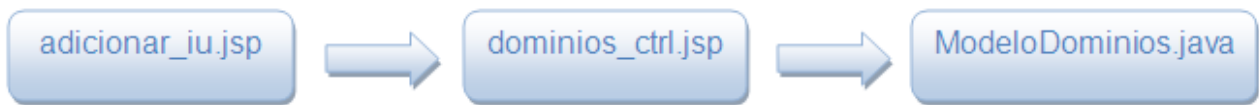


Figura 6: Flujo de datos Adicionar dominios.

Patrones GRASP³

Una de las tareas más complicadas en la programación orientada a objetos consiste en elegir las clases adecuadas y decidir cómo deben interactuar. Incluso cuando se utilizan metodologías ágiles como programación extrema (XP) y el proceso se centra en el desarrollo continuo, es inevitable elegir cuidadosamente las responsabilidades de cada clase en la codificación y fundamentalmente en la refactorización del programa. Los patrones GRASP representan una guía para realizar un diseño eficiente de la aplicación (41).

Entre los patrones que se tienen en cuenta para el desarrollo de la aplicación se encuentran:

Bajo Acoplamiento: La dependencia entre clases es la mínima posible. El nivel de acoplamiento de las clases se puede determinar en las tarjetas CRC.

Alta Cohesión: Cada elemento del diseño realiza una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

Experto: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.

Controlador: Se asignó la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilitó la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.

2.5.3 Diseño de la base de datos

Una base de datos correctamente diseñada permite obtener acceso a información exacta y actualizada. Puesto que un diseño correcto es esencial para lograr los objetivos fijados para la base de datos, parece lógico emplear el tiempo que sea necesario en aprender los principios de un buen diseño

³ GRASP : acrónimo de General Responsibility Assignment Software Patterns

ya que, en ese caso, es mucho más probable que la base de datos termine adaptándose a sus necesidades y pueda modificarse fácilmente.

La aplicación presenta un diseño sencillo de la base de datos, donde existen dos tablas; una con la información referente a los usuarios registrados en el sistema y la otra donde se almacenan los diferentes dominios del conocimiento con que cuenta la aplicación y las palabras claves asociados a estos. Entre ambas tablas existe una relación no identificativa de una a muchos con el objetivo de poder identificar el usuario creador de cada dominio, y que la información de estos persista en caso de que su autor sea eliminado del sistema.

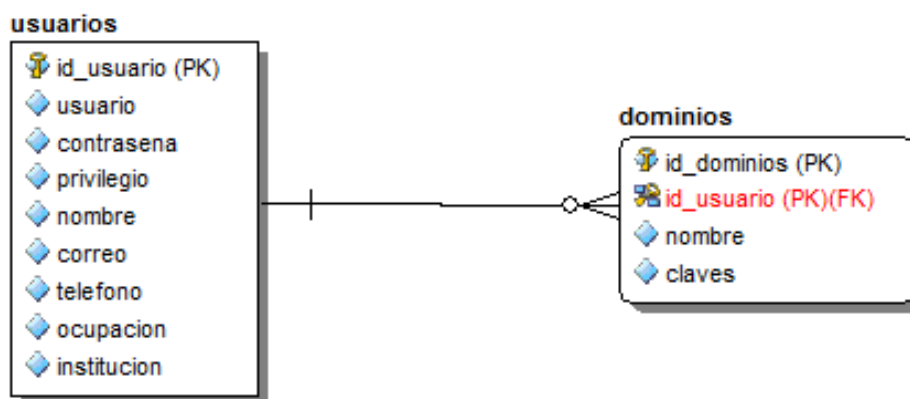


Figura 7: Diagrama del diseño de la base de datos.

Conclusiones del capítulo

En el presente capítulo se describió la solución propuesta para el problema de la investigación, se identificaron los requisitos no funcionales y 19 historias de usuario. Se establecieron los planes de iteraciones, de duración de las iteraciones y de entrega que determinarán el desarrollo de la aplicación. Se mencionaron los diferentes patrones de diseño que se aplicarán en la implementación del software. Se creó el diseño de la base de datos y se generaron ocho tarjetas CRC donde se reflejan las diferentes clases que intervendrán en la aplicación con sus responsabilidades e interacciones, por lo que se sentaron las bases para comenzar con la fase de desarrollo.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA

3.1 Introducción

En este capítulo se presentan las tareas de ingeniería, generadas por historia de usuario, en que se fraccionó el proceso de implementación de la aplicación. Se muestra el estándar de codificación utilizado y parte del código de los algoritmos de las principales funcionalidades de la aplicación. Se realizan, acorde a lo que propone la metodología XP, las pruebas de aceptación a la aplicación, reflejadas en el informe mediante los casos de prueba.

3.2 Implementación

En la disciplina de implementación se lleva a cabo la producción del software. El equipo de desarrollo se encarga de implementar las funcionalidades especificadas por el cliente en las historias de usuario utilizando el lenguaje de programación elegido. XP propone entre sus técnicas la programación en pareja, las principales ventajas de introducir este estilo de programación son: muchos errores son detectados conforme son introducidos en el código, por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor, los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y finalmente, los programadores disfrutan más su trabajo(36).

3.2.1 Tareas de ingeniería

Entre los artefactos que se generan durante la fase de implementación en el desarrollo de una aplicación con la metodología XP, se encuentran las tareas de ingeniería. Estas surgen de la división de las historias de usuario en tareas más sencillas para planificar el trabajo desde un punto de vista técnico. Cada una de las historias de usuario se transformará en tareas que serán desarrolladas por programadores, dentro del equipo de desarrollo, aplicando la práctica de la programación en parejas. Cada tarea de desarrollo corresponderá a un período de uno a tres días de desarrollo.

Distribución de las tareas por historia de usuario

A continuación se relaciona una muestra de las tareas de ingeniería asociadas a algunas de las historias de usuario identificadas.

Tabla 18: Tareas asociadas a cada historia de usuario.

| Historia de usuario | Tareas de ingeniería |
|---------------------|----------------------|
|---------------------|----------------------|

| | |
|---------------------------|---|
| Autenticar usuario | <ul style="list-style-type: none"> ✓ Diseñar interfaz de usuario. ✓ Implementar la funcionalidad iniciar sesión. ✓ Implementar la funcionalidad cerrar sesión. |
| Caracterizar repositorios | <ul style="list-style-type: none"> ✓ Diseñar interfaz de usuario. ✓ Implementar la funcionalidad caracterizar repositorio. |
| Comparar repositorios | <ul style="list-style-type: none"> ✓ Diseñar interfaz de usuario para mostrar gráficas que comparen el contenido de los repositorios. ✓ Implementar la funcionalidad comparar repositorios. |
| Visualizar ontologías | <ul style="list-style-type: none"> ✓ Diseñar interfaz de usuario. ✓ Implementar la funcionalidad para visualizar una ontología. |
| Subir ontologías | <ul style="list-style-type: none"> ✓ Diseñar interfaz de usuario. ✓ Implementar la funcionalidad subir ontología. |

Tareas detalladas

Tabla 19: Tarea #1 HU: Autenticar usuario

| Tarea de ingeniería | |
|---|---|
| Número de tarea: 1 | Número de historia de usuario: 1 |
| Nombre de la tarea: Diseñar interfaz de usuario. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.1 |
| Fecha de inicio: 07/02/2011 | Fecha de fin: 07/02/2011 |
| Programador responsable: Leonel Vila | |
| Descripción: Los campos para la autenticación del usuario estarán ubicados en la parte izquierda de la barra de menús situada en la sección superior de la página. Una vez que el usuario haya iniciado sesión se mostrará el nombre del usuario autenticado y un botón para finalizar la sesión. | |

Tabla 20: Tarea #2 HU: Autenticar usuario

| Tarea de ingeniería | |
|--|---|
| Número de tarea: 2 | Número de historia de usuario: 1 |
| Nombre de la tarea: Implementar funcionalidad iniciar sesión. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.2 |
| Fecha de inicio: 07/02/2011 | Fecha de fin: 08/02/2011 |
| Programador responsable: Leonel Vila | |
| Descripción: Crear una página que se encargue de iniciar la sesión. Primeramente debe comprobar que no haya sesión iniciada para luego capturar los datos enviados para la autenticación. Si los datos del usuario son correctos inicia la sesión, de lo contrario muestra un cartel indicando que no se ha iniciado la sesión. En todos los casos redirecciona a la página principal. | |

Tabla 21: Tarea #3 HU: Autenticar usuario

| Tarea de ingeniería | |
|--|---|
| Número de tarea: 3 | Número de historia de usuario: 1 |
| Nombre de la tarea: Implementar funcionalidad cerrar sesión. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.2 |
| Fecha de inicio: 08/02/2011 | Fecha de fin: 09/02/2011 |
| Programador responsable: Leonel Vila | |
| Descripción: Crear una página que se encargue de finalizar la sesión. Si existe una sesión iniciada la invalida. Siempre redirecciona a la página principal. | |

Tabla 22: Tarea #22 HU: Caracterizar repositorio

| Tarea de ingeniería | |
|---|--|
| Número de tarea: 22 | Número de historia de usuario: 10 |
| Nombre de la tarea: Diseñar interfaz de usuario. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.1 |
| Fecha de inicio: 23/03/2011 | Fecha de fin: 23/03/2011 |

| |
|--|
| Programador responsable: Leonel Vila |
| Descripción: Crear una página que muestre el nombre y la caracterización por dominios de los repositorios. Debe incluir una barra de herramientas que brinde las posibles opciones a realizar en dependencia con el nivel de privilegio. |

Tabla 23: Tarea #23 HU: Caracterizar repositorio

| | |
|--|--|
| Tarea de ingeniería | |
| Número de tarea: 23 | Número de historia de usuario: 10 |
| Nombre de la tarea: Implementar funcionalidad caracterizar repositorio. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 1.4 |
| Fecha de inicio: 24/03/2011 | Fecha de fin: 01/04/2011 |
| Programador responsable: Leonel Vila | |
| Descripción: Implementar todas las clases involucradas en el proceso (Control, Repositorio, Dominio, Ontología). | |

Tabla 24: Tarea #31 HU: Comparar repositorios

| | |
|---|--|
| Tarea de ingeniería | |
| Número de tarea: 31 | Número de historia de usuario: 13 |
| Nombre de la tarea: Diseñar interfaz de usuario para mostrar gráficas que comparen el contenido de los repositorios. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.2 |
| Fecha de inicio: 20/04/2011 | Fecha de fin: 21/04/2011 |
| Programador responsable: Michael Marrero | |
| Descripción: Crear una página que muestre en tabs distintos tipos de gráficas (barras, áreas y combinada). | |

Tabla 25: Tarea #32 HU: Comparar repositorios

| |
|----------------------------|
| Tarea de ingeniería |
|----------------------------|

| | |
|---|--|
| Número de tarea: 32 | Número de historia de usuario: 13 |
| Nombre de la tarea: Implementar la funcionalidad comparar repositorios. | |
| Tipo de tarea: Desarrollo | Tiempo estimado: 0.3 |
| Fecha de inicio: 21/04/2011 | Fecha de fin: 22/04/2011 |
| Programador responsable: Michael Marrero | |
| Descripción: Crear las funciones Javascript para mostrar las gráficas. En la vista comparar repositorios pedir los datos a los repositorios para crear la estructura de datos que se mostrarán en las gráficas. | |

3.2.2 Estándar de código

XP enfatiza la necesidad de comunicación entre los programadores a través del código, por lo cual es indispensable que se sigan ciertos estándares de programación (del equipo o de la organización). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios. El estándar comprende: la estructuración del programa, funciones, definición de clases, métodos, definición de variables globales, definición de variables locales, etc.

Las convenciones de código son importantes para los programadores por un gran número de razones, entre ellas:

- ✓ El 80% del coste del código de un programa va a su mantenimiento.
- ✓ Casi ningún software mantiene toda su vida el autor original.
- ✓ Mejoran la lectura del software, permitiendo entender código nuevo mucho más rápido y más a fondo.
- ✓ Si distribuyes el código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.

Para la implementación de la aplicación el estándar aplicado consiste en:

Ficheros de código fuente

Cada fichero fuente de Java contiene una única clase o interface pública. Cuando algunas clases o interfaces privadas están asociadas a una clase pública, pueden ponerse en el mismo fichero que la clase pública. La clase o interfaz pública debe ser la primera clase o interface del fichero.

Los ficheros están ordenados de la siguiente manera:

- ✓ Comentarios de comienzo.
- ✓ Sentencias package e import.

- ✓ Declaraciones de clases e interfaces.

Indentación

- ✓ Se emplearon cuatro espacios como unidad de indentación.
- ✓ Se evitaron las líneas de más de 80 caracteres, ya que no son bien manejadas por muchas terminales y herramientas.
- ✓ Cuando una expresión sobrepasa la longitud de la línea, esta se rompe de acuerdo a estos principios:
 - ✓ Después de una coma.
 - ✓ Antes de un operador.
- ✓ Se prefirieron roturas de alto nivel (más a la derecha que el "padre") antes que las de bajo nivel (más a la izquierda que el "padre").

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Declaraciones

- ✓ Se realizó una declaración por línea, ya que facilita los comentarios.
- ✓ Las variables locales se inicializaron donde se declaran. Con excepciones donde el valor inicial depende de algunos cálculos que deben ocurrir.
- ✓ Se evitaron las declaraciones locales que ocultaran declaraciones de niveles superiores.
- ✓ Al codificar clases e interfaces de Java, se siguieron las siguientes reglas de formato:
- ✓ No se utilizó ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- ✓ La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- ✓ La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{".
- ✓ Los métodos se separaron con una línea en blanco.
- ✓ Las sentencias if-else siempre usan llaves "{ }". El else se coloca inmediatamente después del cierre del if.

Espacios y líneas en blanco

Se utilizaron líneas en blanco:

- ✓ Entre las secciones de un fichero fuente

- ✓ Entre las definiciones de clases e interfaces.
- ✓ Entre métodos
- ✓ Entre las variables locales de un método y su primera sentencia
- ✓ Antes de un comentario de bloque o de un comentario de una línea
- ✓ Entre las distintas secciones lógicas de un método para facilitar la lectura.

Se utilizaron espacios en blanco en las siguientes situaciones:

- ✓ Una palabra clave del lenguaje seguida por un paréntesis.
- ✓ Después de cada coma en las listas de argumentos.
- ✓ Entre los operandos de todos los operadores binarios excepto “.”.
- ✓ Después de un casteo.

Nombres

- ✓ El prefijo del nombre de un paquete se escribió siempre con letras ASCII en minúsculas.
- ✓ Los nombres de las clases son sustantivos, cuando son compuestos tienen la primera letra de cada palabra que lo forma en mayúsculas. Se utilizaron nombres completos y descriptivos, evitando los acrónimos y abreviaturas.
- ✓ Los nombres de las interfaces siguen las mismas reglas que la clase.
- ✓ Los nombres de los métodos comienzan con verbos, cuando son compuestos se escribió la primera palabra con minúsculas y la primera letra de las palabras siguientes en mayúscula.
- ✓ Las variables, excepto las constantes, se escribieron con minúsculas. En los casos en que son palabras compuestas la primera letra de cada palabra después de la primera se escribió en mayúscula. Los nombres son cortos, pero con significado (mnemotécnico).
- ✓ Las constantes se escribieron con mayúsculas. En el caso de palabras compuestas se separaron con un guion (“_”).

3.2.3 Código de algunos de los principales métodos implementados

A continuación se muestran algunos ejemplos del código de los principales métodos implementados en el desarrollo de la aplicación con una breve descripción de la funcionalidad del mismo.

Método cargarOntologías

El método cargarOntologías se encuentra en la clase Repositorio de la aplicación.

```
57 private List<Ontologia> cargaOntologias() {
58     List<Ontologia> lista = new LinkedList<Ontologia>();
59     File[] onts = new File(directorio, "owl").listFiles();
60     for (int i = 0; i < onts.length; i++) {
61         lista.add(new Ontologia(onts[i]));
62     }
63     return lista;
64 }
```

Figura 8: Ejemplo de código

Este algoritmo se encarga de almacenar en un arreglo las ontologías (en este caso ficheros con extensión “.owl”) que existen en el directorio correspondiente (directorio es un atributo de la clase Repositorio donde se especifica su dirección), y luego las adiciona a una lista que es retornada por el método.

Fragmento de código del constructor de la clase Control

```
17 File[] directorios = DIRECTORIO_RAIZ.listFiles();
18 for (int i = 0; i < directorios.length; i++) {
19     File dir = directorios[i];
20     Repositorio r = new Repositorio(dir);
21     r.caracterizaPorDominios();
22     if (dir.getName().equals("Local")) {
23         repositorios.add(0, r);
24     } else {
25         repositorios.add(r);
26     }
27 }
```

Figura 9: Ejemplo de código

Este fragmento de código, implementado en el constructor de la clase Control, almacena en un arreglo los directorios correspondientes a cada repositorio. Estos luego son pasados por parámetro a la instancia de cada repositorio que una vez creada caracteriza su contenido a partir de los dominios almacenado en la base de datos. Por último la instancia es adicionada a la lista de los repositorios que controla el sistema.

Método comprobarOntologia

```
public boolean comprobarOntologia(Ontologia ont) {
39     boolean pertenece = false;
40     try {
41         BufferedReader entrada = new BufferedReader(new FileReader(ont.archivo()));
42         String linea;
43         while ((linea = entrada.readLine()) != null && !pertenece) {
44             linea = linea.toLowerCase();
45             int posNumeral = linea.lastIndexOf("#");
46             if (linea.contains("rdf:about") && posNumeral != -1) {
47                 linea = linea.substring(posNumeral + 1);
48                 for (int j = 0; j < claves.length && !pertenece; j++) {
49                     if (linea.contains(claves[j].toLowerCase())) {
50                         pertenece = true;
51                     }
52                 }
53             }
54         }
55     } catch (FileNotFoundException ex) {
56     } catch (IOException ex) {
57     } finally {
58         return pertenece;
59     }
60 }
```

Figura 10: Ejemplo de código

El método, atendiendo al patrón de diseño Experto, ha sido implementado en la clase Dominio. Tiene como objetivo comprobar si una ontología pertenece o no al dominio a partir de la comparación de cada elemento de la ontología con sus palabras claves. Para ganar en tiempo la variable “*pertenece*” controla la búsqueda de semejanzas, permitiendo que se realicen las comparaciones mientras no se ha encontrado ninguna similitud. Ambas partes a comparar, tanto el elemento de la ontología como la palabra clave, son convertidas a letras minúsculas para garantizar una mayor eficiencia en la comparación.

3.3 Pruebas

Uno de los pilares de la programación extrema es el proceso de pruebas. XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

3.3.1 Pruebas de aceptación

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

En la mayoría de las metodologías de desarrollo, principalmente las robustas, se conoce de cada requisito funcional el camino principal, los caminos alternativos, errores posibles y su tratamiento, precondiciones, postcondiciones e invariantes; lo que permite explorar exhaustivamente todas las combinaciones posibles asegurando que, en todo momento, el sistema responde de acuerdo a sus requisitos.

Sin embargo, toda esa información sobre los requisitos del sistema no está disponible en XP. En un desarrollo XP, los requisitos, es decir lo que el sistema debe hacer, se recogen en historias de usuario durante la etapa de planificación. Una historia de usuario sólo recoge una descripción en lenguaje natural, en general ambigua, incompleta e informal, de un fragmento de la funcionalidad del sistema; aunque las historias de usuario pueden ser completadas con notas, el tamaño recomendado para la tarjeta no permite añadir información clara ni exhaustiva. Debido a la escasez de información sobre el comportamiento esperado del sistema se hace imprescindible contar con la colaboración del cliente para el desarrollo de las pruebas de aceptación

Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento y no se considera completa hasta que no supera sus pruebas de aceptación. A continuación se muestra una representación de los casos de pruebas que fueron realizados al sistema en cada una de las iteraciones.

3.3.2 Casos de Pruebas

Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. Este tipo de prueba permite obtener un conjunto de condiciones de entrada que ejerciten de forma completa todos los requisitos funcionales de un programa, es un enfoque complementario que intenta descubrir diferentes tipos de errores como por ejemplo: errores de interfaz, errores en estructuras de datos o acceso a las bases de datos, entre otros (17).

Tabla 26: Caso de prueba de aceptación HU3-1

| Caso de Prueba de Aceptación | |
|--|--|
| Código Caso de Prueba: HU3-1 | Nombre Historia de Usuario: Registrar usuario. |
| Nombre de la persona que realiza la prueba: Michael E. Marrero Clark | |
| Descripción de la Prueba: Se llenaron los campos referentes a la información para conformar el perfil del usuario registrado. | |
| Condiciones de Ejecución: La aplicación deberá estar en ejecución en el servidor de aplicaciones con su respectiva conexión a la base de datos. | |
| Entrada / Pasos de ejecución: Solicitar en la aplicación la opción registrar usuario. Introducir los datos en los campos de texto (no debe dejarse sin llenar algún campo obligatorio) Una vez introducidos los datos pulsar la opción registrar. | |
| Resultado Esperado: La aplicación debe mostrar la página principal de la aplicación donde el usuario deberá poder autenticarse en el sistema. | |
| Evaluación de la Prueba: Satisfactoria | |

Con el objetivo de probar las funcionalidades de la aplicación se diseñaron 30 casos de pruebas de aceptación (ver expediente de proyecto), los cuales arrojaron como resultado seis no conformidades; tres en la primera iteración, dos en la segunda y una en la tercera. Las mismas fueron solucionadas inmediatamente, garantizándose de esta manera que la aplicación cumple con todas las especificaciones del cliente.

Conclusiones del capítulo

En el presente capítulo se describieron las tareas de la ingeniería a desarrollar para la implementación de las historias de usuario. Se definieron y reflejaron en el informe los estándares de codificación empleados durante la implementación de la aplicación; imprescindibles para el entendimiento entre los miembros del equipo de desarrollo. Se mostraron los algoritmos fundamentales a través de su código fuente y una breve descripción de su funcionamiento. Además se hizo un análisis de los resultados de las pruebas de aceptación aplicadas a la aplicación a través de los diferentes casos de pruebas que

cubrieron la totalidad de sus funcionalidades, permitiendo encontrar errores que afectaban el funcionamiento de esta. Los errores encontrados con las pruebas fueron corregidos lo que permitió que actualmente la aplicación esté al 100% de su funcionamiento.

CONCLUSIONES GENERALES

Al finalizar la investigación puede concluirse lo siguiente:

- ✓ El proceso de desarrollo de la aplicación informática estuvo guiado por la metodología XP, utilizando Java como lenguaje de implementación y Jena como framework para acceder al contenido de las ontologías.
- ✓ Se definieron las funcionalidades de la aplicación reflejadas en 19 historias de usuario y se elaboraron ocho tarjetas CRC que guiaron el proceso de implementación.
- ✓ Se implementó una aplicación informática que gestiona repositorios de ontologías representativas del conocimiento en la Web que hará más viable el proceso de recuperación de ontologías para su futura reutilización en el desarrollo de aplicaciones para la Web Semántica.
- ✓ Las pruebas de aceptación aplicadas para validar las funcionalidades, demuestran que el sistema cumple satisfactoriamente con los requisitos que garantizan su correcto funcionamiento.

RECOMENDACIONES

Durante el desarrollo del trabajo han surgido algunas ideas que podrían ser incorporadas en un futuro con el objetivo de fortalecer el sistema propuesto, por lo que se recomienda:

- ✓ Incorporar nuevos criterios de clasificación de las ontologías en la caracterización de los repositorios.
- ✓ Desarrollar un componente para el trabajo con ontologías, que cubra las necesidades que hoy hacen necesario el uso de Jena en la aplicación, puesto que este último consume muchos recursos del servidor.
- ✓ Integrar en una única plataforma la Aplicación informática para caracterizar repositorios de ontologías representativas del conocimiento en la Web con las siguientes aplicaciones: Aplicación informática para caracterizar ontologías representativas del conocimiento en la Web y Aplicación informática para comparar ontologías representativas del conocimiento en la Web.

REFERENCIAS BIBLIOGRÁFICAS

1. **Lozano, A., y otros.** Uso de Ontologías en Páginas Web para Mejorar su Accesibilidad a Invidentes. s.l. : QUERCUS Software Engineering Group. Departamento de Informática. Universidad de Extremadura.
2. *The Semantic Web.* **Berners-Lee, Tim, Hendler, James y Lassila, Ora.** s.l. : Scientific American, 2001.
3. **García Sánchez, Francisco.** *Sistema Basado en Tecnologías del Conocimiento para Entornos de Servicios Web Semánticos.* s.l. : Universidad de Murcia, Departamento de Ingeniería de la Información y las Comunicaciones, 2007.
4. **Abián, Miguel Ángel.** El Futuro de la Web XML, RDF/RDFS, Ontologías y la Web Semántica. *javahispano*. [En línea] 2005. <http://www.javahispano.org>.
5. **Horrocks, I, Patel-Schneider, P y Van Harmelen, F.** From SHIQand RDF to OWL: The making of a Web Ontology Language. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web.* 2003.
6. **Simón, Alfredo, y otros.** Modelo unificado de representación de conocimientos en mapas conceptuales y ontologías. [ed.] A. J. Cañas y J. D. Novak. San José, Costa Rica : s.n., 2006.
7. *Enabling Technology for Knowledge Sharing.* **Neches, Robert, y otros.** s.l. : AI Magazine, 1991, Vol. 12. 0738-4602.
8. **Samper Zapater, José Javier.** Ontologías para servicios web semánticos de información de tráfico: descripción y herramientas de explotación. s.l., España : Servei de Publicacions, 2005. 84-370-6270-5. Departamento de Informática, Universidad de Valencia.
9. *Toward Principles for the Design of Ontologies Use for Knowledge Sharing.* **Gruber, Thomas.** s.l. : Stanford Knowledge Systems Laboratory, 1993.
10. **Borst, Willem Nico.** Construction of Engineering Ontologies. s.l. : University of Tweenty, 1997. Centre for Telematics and Information Technology.
11. **Guarino, Nicola.** *Formal Ontology in Information Systems.* Trento : IOS Press Amsterdam, 1998. Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy. ISBN:9051993994.
12. **Fernández Hernández, Anisleiby, y otros.** Las Ontologías. Nuevos Retos. Marzo de 2009. IX Congreso ISKO-España, Nuevas perspectivas para la difusión y organización del conocimiento.
13. **Ramírez Céspedes, Zulia.** Las ontologías como herramienta en la Gestión del Conocimiento. s.l. : Departamento de Bibliotecología y Ciencia de la Información, Universidad de La Habana, 2005.

-
14. **Huaroto, Libio.** *Repositorios Digitales de Documentos*. Biblioteca Central - UNMSM. 2007. Taller sobre el Sistema DSpace para crear Repositorios Digitales de Documentos.
 15. **Almagro, Fanny, y otros.** *Repositorios Educativos*. s.l. : Universidad Tecnológica Indoamérica, 2009.
 16. **Martín, Antonio, y otros.** *Ontologías e inteligencia artificial para la recuperación eficiente del conocimiento*. s.l. : Biblioteca Universidad de Sevilla, España, 2009. XV JORNADAS BIBLIOTECARIAS DE ANDALUCÍA.
 17. **Pressman, Roger S.** *Ingeniería de Software. un enfoque práctico*. 5ta. Edición. s.l. : McGraw-Hill, 2002. ISBN 84-481-3214-9.
 18. **Rumbaugh, James, Booch, Grady y Jacobson, Ivan.** *El Proceso Unificado de Desarrollo de software*. 1999.
 19. **Scrum, A.** *Scrum*. 2009.
 20. **Beck, Kent.** *Extreme Programming Explained. Embrace Change*. s.l. : Pearson Education, 1999. Traducido al español como: .Una explicación de la programación extrema. Aceptar el cambio., Addison Wesley, 2000..
 21. Enciclopedia Libre Universal en Español. [En línea] [Citado el: 24 de Febrero de 2011.] http://enciclopedia.us.es/index.php/Lenguaje_de_programaci%C3%B3n..
 22. **García de Jalón, Javier, et al.** *Aprenda Java como si estuviera en primero*. San Sebastián, Navarra, España : s.n., 2000.
 23. **developerWorks, IBM.** *Introduction to JSP technology*.
 24. **Vila, Fermi.** JavaScript. *www.softdownload.com.ar*. [En línea] Junio de 2001. [Citado el: 17 de Marzo de 2011.]
 25. Conocimientos básicos de Javascript. *JAVASCRIPT*. [En línea] [Citado el: 19 de Marzo de 2011.] http://perso.wanadoo.es/javascript_12/.
 26. **Oracle Corporation.** Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto. [En línea] [Citado el: 24 de Febrero de 2011.] <http://www.netbeans.org>.
 27. **Sun Microsystems.** Conozca el nuevo NetBeans. [En línea] [Citado el: 24 de Febrero de 2011.] http://www.sun.com/emrkt/.../0207latam_feature.htm.
 28. <http://extjs.es/>. [En línea] [Citado el: 23 de Abril de 2011.] <http://extjs.es/>.
 29. **Corzo, Giancarlo.** ExtJS lo bueno, lo malo y lo feo. *Desarrollo en Web*. [En línea] 22 de Octubre de 2008. [Citado el: 25 de Abril de 2011.] <http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/>.

30. **Dojo**. DojoToolkit. [En línea] 12 de 4 de 2006. [Citado el: 25 de 2 de 2011.]
<http://www.dojotoolkit.org>.
31. **Fuentes, Alex**. Dojo toolkit + JEE. Generando las aplicaciones para la web2.0. [En línea] [Citado el: 25 de Abril de 2011.] <http://www.jroller.com/afuentes>.
32. **Alarcon, José Manuel**. [En línea] 2006. <http://es.scribd.com/doc/32185176/Manual-de-PostgreSQL-Server>.
33. **Saha, Amit Kumar**. [En línea] 2008.
<http://www.gedlc.ulpgc.es/docencia/abd/Recursos/MySQL-Intro-features-benefits-SPANISH.pdf>.
34. **Sommerville, Ian**. *Ingeniería de Software*. 6ta. Edición. s.l. : Prentice-Hall, 2002. ISBN 970-26-0206-8.
35. Programación Extrema. *Tripod*. [En línea] <http://programacionextrema.tripod.com/fases.htm>.
36. *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. **Letelier, Patricio y Penadés, María del Carmen**. Valencia : Universidad Politécnica de Valencia.
37. *Cuarto Congreso Colombiano de Computación 4CCC, Sociedad Colombiana de Computación, UNAB - UIS*. **Casas, Sandra y Reinaga, Héctor**. 2009.
38. *Design Patterns: Elements of Reusable Object-Oriented Software*. **Gamma, Erich, y otros**. s.l. : Addison-Wesley, 1995. ISBN 0-201-63361-2..
39. **Bergin, Joseph**. Building Graphical User Interfaces with the MVC pattern. [En línea] <http://csis.pace.edu/bergin/mvc/mvcgui.html>..
40. **Burbeck, Steve**. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). [En línea] <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>..
41. **Mora, Roberto Canales**. Tutorial: Patrones GRASP. *Adictos al trabajo*. [En línea] 22 de 12 de 2003. [Citado el: 13 de Abril de 2011.]
<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=grasp>.

BIBLIOGRAFÍA

- ABIÁN, Miguel Ángel.** *El Futuro de la Web XML, RDF/RDFS, Ontologías y la Web Semántica. javahispano.* [En línea] 2005. <http://www.javahispano.org>.
- AEGARAN, Toby, Evans, Colin and Taylor, Jamie.** *Programming the Semantic Web.* [ed.] Mary E. Treseler. First Edition. s.l. : O'Reilly Media, Inc, 2009. ISBN: 978-0-596-15381-6.
- ALARCON, José Manuel.** [En línea] 2006. <http://es.scribd.com/doc/32185176/Manual-de-PostgreSQL-Server>.
- ALLEMANG, Dean and Hendler, James.** *Semantic Web for the Working Ontologist Modeling in RDF, RDFS and OWL.* s.l. : Morgan Kaufmann Publishers, 2008. ISBN-13: 978-0-12-373556-0.
- BECK, Kent.** *Extreme Programming Explained. Embrace Change.* s.l. : Pearson Education, 1999. Traducido al español como: .Una explicación de la programación extrema. Aceptar el cambio., Addison Wesley, 2000.
- BERGIN, Joseph.** *Building Graphical User Interfaces with the MVC pattern.* [En línea] <http://csis.pace.edu/bergin/mvc/mvcgui.html>.
- BERNERS-LEE, Tim, Hendler, James y Lassila, Ora.** *The Semantic Web.* s.l. : Scientific American, 2001.
- BORST, Willem Nico.** *Construction of Engineering Ontologies.* s.l. : University of Tweenty, 1997. Centre for Telematics and Information Technology.
- BURBECK, Steve.** *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC).* [En línea] <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>.
- CASAS, Sandra y Reinaga, Héctor.** *Cuarto Congreso Colombiano de Computación 4CCC, Sociedad Colombiana de Computación, UNAB – UIS.* 2009.
- CORZO, Giancarlo.** ExtJS lo bueno, lo malo y lo feo. *Desarrollo en Web.* [En línea] <http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/>.
- DOJO.** *DojoToolkit.* [En línea] 12 de abril de 2006. <http://www.dojotoolkit.org>.
- FERNÁNDEZ HERNÁNDEZ, Anisleiby, y otros.** *Las Ontologías. Nuevos Retos. Marzo de 2009. IX Congreso ISKO-España, Nuevas perspectivas para la difusión y organización del conocimiento.*
- FUENTES, Alex.** Dojo toolkit + JEE. Generando las aplicaciones para la web2.0. [En línea] <http://www.jroller.com/afuentes>.
- GAMMA, Erich, et al.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1995. ISBN 0-201-63361-2.

-
- GARCÍA DE JALÓN, Javier, et al.** *Aprenda Java como si estuviera en primero*. San Sebastián, Navarra, España : s.n., 2000.
- GARCÍA SÁNCHEZ, Francisco.** *Sistema Basado en Tecnologías del Conocimiento para Entornos de Servicios Web Semánticos*. s.l. : Universidad de Murcia, Departamento de Ingeniería de la Información y las Comunicaciones, 2007.
- GILL, Rawld, Riecke, Craig and Russell, Alex.** *Mastering Dojo: JavaScript and Ajax Tools for Great Web Experiences*. Dallas, Texas : The Pragmatic Bookshelf. ISBN: 978-1-934356-11-1.
- GUARINO, Nicola.** *Formal Ontology in Information Systems*. Trento : IOS Press Amsterdam, 1998. Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy. ISBN:9051993994.
- GRUBER, Thomas.** *Toward Principles for the Design of Ontologies Use for Knowledge Sharing*. s.l. : Stanford Knowledge Systems Laboratory, 1993.
- HORROCKS, I, Patel-Schneider, P y Van Harmelen, F.** From SHIQand RDF to OWL: The making of a Web Ontology Language. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*. 2003.
- LETELIER, Patricio y Penadés, Maria del Carmen.** *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. Valencia : Universidad Politécnica de Valencia.
- LOZANO, A., y otros.** *Uso de Ontologías en Páginas Web para Mejorar su Accesibilidad a Invidentes*. s.l. : QUERCUS Software Engineering Group. Departamento de Informática. Universidad de Extremadura.
- MARTÍN, Antonio, y otros.** *Ontologías e inteligencia artificial para la recuperación eficiente del conocimiento*. s.l. : Biblioteca Universidad de Sevilla, España, 2009. XV JORNADAS BIBLIOTECARIAS DE ANDALUCÍA.
- MORA, Roberto Canales.** Tutorial: Patrones GRASP. *Adictos al trabajo*. [En línea] 22 de 12 de 2003. <http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=grasp>.
- NECHES, Robert, y otros.** *Enabling Technology for Knowledge Sharing*. s.l. : AI Magazine, 1991, Vol. 12. 0738-4602.
- ORACLE CORPORATION.** Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto. <http://www.netbeans.org>.
- POLLOCK, Jeffrey T.** *Semantic Web For Dummies*. Indianapolis, Indiana : Wiley Publishing, Inc., 2009 . ISBN: 978-0-470-39679-7.
- PRESSMAN, Roger S.** *Ingeniería de Software. un enfoque práctico*. 5ta. Edición. s.l. : McGraw-Hill, 2002. ISBN 84-481-3214-9.
- Programación Extrema. *Tripod*. [En línea] <http://programacionextrema.tripod.com/fases.htm>.

-
- RAMÍREZ CÉSPEDES, Zulia.** *Las ontologías como herramienta en la Gestión del Conocimiento.* s.l. : Departamento de Bibliotecología y Ciencia de la Información, Universidad de La Habana, 2005.
- RUMBAUGH, James, Booch, Grady y Jacobson, Ivan.** *El Proceso Unificado de Desarrollo de software.* 1999.
- RUSSELL, Matthew A.** *Dojo: The Definitive Guide.* First Edition. s.l. : O'Reilly Media, Inc., 2008. ISBN: 978-0-596-51648-2.
- SAHA, Amit Kumar.** [En línea] 2008. <http://www.gedlc.ulpgc.es/docencia/abd/Recursos/MySQL-Intro-features-benefits-SPANISH.pdf>.
- SAMPER ZAPATER, José Javier.** *Ontologías para servicios web semánticos de información de tráfico: descripción y herramientas de explotación.* s.l., España : Servei de Publicacions, 2005. 84-370-6270-5. Departamento de Informática, Universidad de Valencia.
- SIMÓN, Alfredo, y otros.** *Modelo unificado de representación de conocimientos en mapas conceptuales y ontologías.* [ed.] A. J. Cañas y J. D. Novak. San José, Costa Rica : s.n., 2006.
- SOMMERVILLE, Ian.** *Ingeniería de Software.* 6ta. Edición. s.l. : Prentice-Hall, 2002. ISBN 970-26-0206-8.
- STEVE, Geri, Gangemi, Aldo y Pisanelli, Doménico M.** *Integrating Medical Terminologies with ONIONS Methodology.* [En línea] 1998. [Citado el: 26 de 2 de 2011.] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.4356&rep=rep1&type=pdf>.
- SUN MICROSYSTEMS.** *Conozca el nuevo NetBeans.* http://www.sun.com/emrkt/.../0207latam_feature.htm.
- VAN HEIJST, G., Schreiber, A.T. y Wieling, B.J.** *Using explicit ontologies in KBS development.* Amsterdam : s.n., 1997. International Journal of Human and Computer Studies, 1996. 1071-5819.
- VILA, Fermi.** JavaScript. *www.softdownload.com.ar.* [En línea] Junio de 2001.
- ZAMMETTI, Frank W.** Practical Dojo Projects. 2008. ISBN: 978-1-4302-1065-8.

ANEXOS

Anexo 1: Interfaz de la funcionalidad Registrar usuario

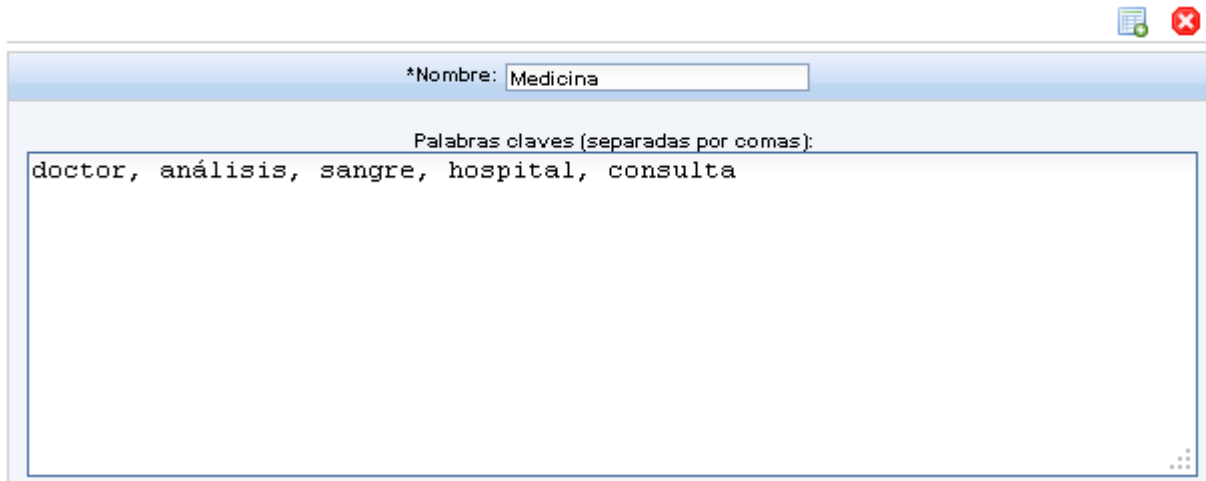


| | | |
|--------------|----------------------|--|
| *Usuario: | <input type="text"/> | *Obligatorio. No mas de 16 caracteres. |
| *Contraseña: | <input type="text"/> | *Obligatorio. Minimo 8 caracteres. |
| *Nombre: | <input type="text"/> | *Obligatorio. |
| Correo: | <input type="text"/> | |
| Telefono: | <input type="text"/> | |
| Ocupación: | <input type="text"/> | |
| Institución: | <input type="text"/> | |

Anexo 2: Interfaz de la funcionalidad Comparar repositorios



Anexo 3: Interfaz de la funcionalidad Insertar dominio

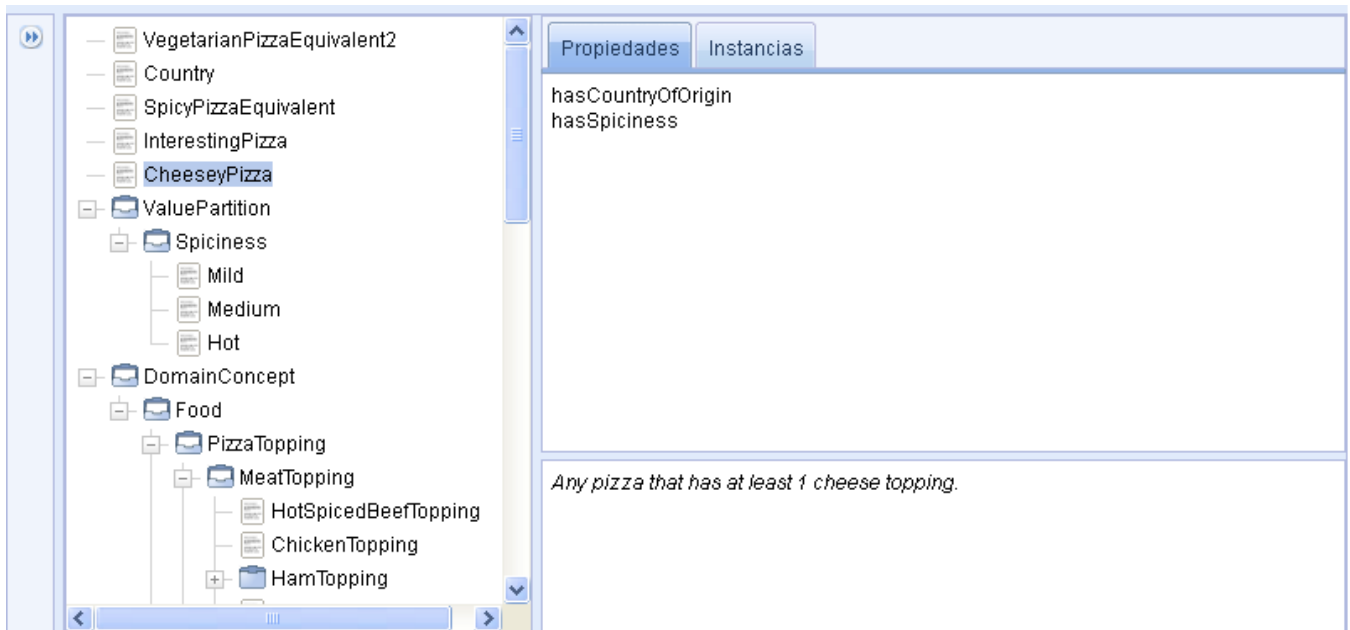


*Nombre:

Palabras claves (separadas por comas):

doctor, análisis, sangre, hospital, consulta

Anexo 4: Interfaz de la funcionalidad Visualizar ontología



Propiedades Instancias

hasCountryOfOrigin
hasSpiciness

Any pizza that has at least 1 cheese topping.

GLOSARIO DE TÉRMINOS

Agentes software: programa informático que habita en ordenadores y redes y que ayuda a los usuarios en tareas relacionadas con el uso del ordenador, un agente no es sino una entidad informática autorizada para actuar en nombre de una persona.

Indentación: anglicismo (de la palabra inglesa indentation) de uso común en informática que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente, lo que en el ámbito de la imprenta se ha denominado siempre como sangrado o sangría. En los lenguajes de programación de computadoras, la indentación se utiliza para mejorar la legibilidad del código fuente por parte de los programadores, teniendo en cuenta que los compiladores o intérpretes raramente consideran los espacios en blanco entre las sentencias de un programa.

Internet: gran red descentralizada de ordenadores, de ámbito global y públicamente accesible, que proporciona una ingente cantidad de servicios de comunicación de varios tipos, incluyendo la World Wide Web, el correo electrónico y muchos otros.

Plugins: pequeño programa que proporciona alguna funcionalidad específica a otra aplicación mayor o más compleja.

Widgets: son pequeñas aplicaciones o programas, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están los de dar fácil acceso a funciones frecuentemente usadas y proveer de información visual. Sin embargo los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet.

W3C: World Wide Web Consortium, organización de carácter internacional que se encarga de velar por la regulación y desarrollo de la multitud de estándares empleados en la Web. Fue fundada en 1994 por Tim Berners-Lee y realiza una labor imprescindible para el correcto desarrollo de este gigantesco entramado.

World Wide Web (cuya traducción en informática es Red Global Mundial o "Red de Amplitud Mundial"): es un sistema de documentos de hipertexto enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza sitios Web compuestos de páginas Web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.