Universidad de las Ciencias Informáticas
Facultad 4

Implementación de una nueva versión del módulo Búsqueda para el Repositorio de Objetos de Aprendizaje RHODA.

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

Autores:

Yidielis Pérez Leiva

Sergio Bello González

Tutores:

Ing. Javier Soler Martín

Ing. Orlando Salvador Broche

Ing. Roxana Cañizares

Ciudad de la Habana, Junio 2011.

“Año 53 de la Revolución”

Declaración de Autoría

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas; para que haga el uso que estime pertinente.

Para que así conste firmamos la presente a los 19 días del mes de Mayo del año 2011.

Autores

Yidielis Pérez Leiva	Sergio Bello González

Datos de Contacto

Datos de Contacto

Ing. Roxana Cañizares González: Profesora de la disciplina Ingeniería y Gestión de Software. Facultad 4, Universidad de las Ciencias Informática (UCI). Carretera a San Antonio de los Baños Km. 2 ½, Torrens, Boyeros, Ciudad de La Habana. Cuba. rcanizares@uci.cu, Ingeniera en Ciencias Informáticas, UCI, 2008. Actualmente es directiva del Centro para la Creación de Recursos Educativos (FORTES).

Ing. Javier Soler Martín: Profesor de la disciplina de Programación. Facultad 4, Universidad de las Ciencias Informática (UCI). Carretera a San Antonio de los Baños Km. 2 ½, Torrens, Boyeros, Ciudad de La Habana. Cuba. jsoler@uci.cu, Ingeniero en Ciencias Informáticas, UCI, 2008. Actualmente es jefe de un subproyecto en el desarrollo de un Repositorio de Objetos de Aprendizaje (RHODA).

Ing. Orlando Salvador Broche: Profesor de la disciplina de Programación. Facultad 4, Universidad de las Ciencias Informática (UCI). Carretera a San Antonio de los Baños Km. 2 ½, Torrens, Boyeros, Ciudad de La Habana. Cuba. jsoler@uci.cu, Ingeniero en Ciencias Informáticas, UCI, 2008. Actualmente es jefe del proyecto en el desarrollo de un Repositorio de Objetos de Aprendizaje (RHODA).

Agradecimientos

A nuestros padres, abuelos y otros familiares, por guiarnos por el camino correcto y ayudarnos a realizar este sueño.

A Dios por haber estado de nuestro lado durante estos 5 años.

A nuestros tutores Roxana, Orlando y Javier por el apoyo, la confianza depositada y por contribuir a nuestra formación como profesionales.

A los compañeros de proyecto por sus contribuciones y su tiempo.

A nuestros amigos y compañeros de estudio de la facultades 10 y 4 por ser una gran familia, que siempre nos han brindado su amistad incondicional.

A todos, Muchas Gracias.

Dedicatoria

Dedicatoria

Dedico este trabajo a toda mi familia, en especial a mi padre Sergio, mi madre Teresa, mi hermano Gilberto y a mi abuelita Josefa.

A mi abuelo *Berto*, que aunque no se encuentra físicamente entre nosotros sirva este trabajo como homenaje a una mente tan brillante.

A mi compañera de tesis *Yidi*, por haberme brindado todo su amor y apoyo incondicional durante estos 5 años.

A mis tutores Roxana, Javier y Orlando, por haber sido los verdaderos impulsores del proyecto.

Sergio Bello González

Dedico este trabajo a toda mi familia, en especial a mi abuela Juana, mi madre Tania, mi tía *Negra* y a mi hermanito Jorgito.

A mi abuelo Luis, que aunque no se encuentra físicamente entre nosotros su integridad me ha servido como guía en estos largos 5 años.

A mi compañero Sergio, por apoyarme, quererme y respetarme siempre.

A todos aquellos que alguna vez creyeron en mí.

Yidielis Pérez Leiva

Resumen

Los Objetos de Aprendizaje son una tendencia importante en lo que respecta a la producción de contenidos educativos, por ello su almacenamiento y gestión es un punto fundamental en la industria del software educativo. La Universidad de Ciencias Informáticas como una de las instituciones educativas vinculadas a esta industria, desarrolla un Repositorio de Objetos de Aprendizaje que actualmente está en su versión 2.0. Aunque constituye una versión funcional, el proceso de búsqueda y recuperación de información presenta características precarias que deben ser actualizadas, tales como la carencia de criterios de búsqueda, falta de exhaustividad, entre otros.

El presente trabajo tiene como objetivo la implementación de un módulo de búsqueda y recuperación de información. Para ello se analizan los referentes teóricos actuales relacionados con el proceso de recuperación de información, cómo es desarrollado por repositorios y las tecnologías que en la actualidad pueden ser usadas para la implementación del sistema.

Como resultado del presente trabajo, se generan algunos de los artefactos (diagrama de despliegue, diagrama de componentes, diagrama de procesos) correspondientes al flujo de implementación que establece la metodología Rational Unified Process. Se utilizan técnicas para el proceso de búsqueda como la búsqueda de texto completo y difusa, además de la utilización del índice invertido para el proceso de indexación, ordenando los resultados mediante el modelo vectorial. Se valida el sistema a través de los métodos de prueba de Caja Blanca y Caja Negra.

Palabras clave: Repositorio, Objetos de Aprendizaje, metadatos, Recuperación de Información, índice, indexación, modelo, expansión de consultas, IMS-DRI.

Índice de contenido

Índice de Contenido

Índice de contenido

Índice de figuras

Índice de figuras

Introducción

Introducción

Con el auge de las Tecnologías de la Información y las Comunicaciones, ha aumentado el volumen de información y por tanto la necesidad de gestionarla. Con el paso de los años se han propuesto numerosas estructuras y componentes, con el objetivo de permitir un acceso eficiente a la información almacenada en enormes bases de datos documentales. La *Recuperación de Información* (RI) es el área de investigación que centra sus esfuerzos en este objetivo. Constituye un campo de investigación iniciado en los trabajos de Salton en los años 60 (G., 1963), que en la actualidad ha presentado un desarrollo ascendente motivado por el avance de Internet y la creciente necesidad de realizar búsquedas en la Web.

Esta área de investigación es ampliamente utilizada en la implementación de bases de datos documentales, motores de búsqueda en la Web y desde hace unos años atrás en los repositorios de información. Estos últimos surgen por la necesidad de almacenar, organizar y reutilizar eficientemente la información digital, en correlación al gran desarrollo del *e-learning* como modalidad de educación a distancia, contribuyendo a realizar dichas operaciones sobre los numerosos recursos digitales provenientes de todos los centros que adoptan dicha modalidad.

Son muchas las definiciones que se le dan a repositorio de información teniendo en cuenta varias bibliografías (Ruiz, 2010), (Guzmán, 2005), (Garrido, et al., 2007), para fines de este trabajo se deduce como: "*un lugar donde la información digital es almacenada, para ser localizada y distribuida a través de la red*" (Guzmán, 2005). Definición sencilla pero objetiva, ya que plantea de forma explícita las funciones principales de los mismos: almacenar, localizar y reutilizar. Como ejemplo se pueden mencionar CogPrints (Psicología), ArXiv (Matemática, Física Computación y ciencias afines) y REPEC (Economía).

Con el desarrollo de estos repositorios de información surgen dos clasificaciones en cuanto a la manera de gestionar los recursos educativos: Repositorios Abiertos y Repositorios de Objetos de Aprendizaje (ROA) (Ruiz, 2010). Los repositorios abiertos, se caracterizan fundamentalmente por permitir la publicación inmediata de artículos sin previa autorización, trayendo la ventaja de lograr una rápida divulgación de la información (Castillo, 2008), mientras que los ROA surgen por la necesidad de almacenar, localizar, recuperar y reutilizar contenidos etiquetados bajo estándares, lo que se conoce como Objetos de Aprendizaje (OA): recurso digital con una marcada intención formativa y estructura didáctica variable, compuesto por uno o varios objetos de información (elementos digitales o recursos digitales), descrito con metadatos, con un comportamiento secuenciado que asegure el correcto enlace

entre los elementos de su estructura didáctica, que puede ser utilizado y reutilizado en entornos *e-learning*. Constituyen ejemplos de ROA Connexion, SLOR, MERLOT, entre otros.

Con el objetivo de establecer una compatibilidad entre las herramientas que gestionan dichos OA se realiza una estandarización de los contenidos educativos, surgiendo así estándares con diversas funciones: empaquetamiento, catalogación, interoperabilidad, entre otros. Entre los más usados se encuentran SCORM[1] (Sharable Content Object Reference Model), LOM[2] (Learning Object Metadata), SQI [3] (Simple Query Interface), IMS [4] (Instructional Management Systems) en sus diferentes especificaciones, entre otros. Entre las especificaciones de IMS se encuentra IMS-DRI (Digital Repositories Specification), donde se define una serie de funcionalidades que deben poseer éstos repositorios, destacando la búsqueda y recuperación de OA (2011).

Los OA están catalogados por metadatos, el uso de los mismos permite facilitar el proceso de búsqueda, definiendo campos que especifican autores, títulos, fechas, entre otros. Aunque esta es una forma bastante aplicable, existe otra que se centra en el *objeto de información* del OA: *"elementos digitales que conforman su contenido"* (Belén, 2010), permitiendo la indexación de todos para la recuperación de información. Este proceso se refiere a la acción de registrar ordenadamente información para elaborar su índice, "*estructura de datos que almacena trazas de contenido, como palabras o números, a sus localizaciones en archivo, permitiendo el desarrollo de búsquedas más rápidas*." (Zobel, y otros, 2006), con la finalidad de obtener resultados de una forma rápida y relevante al momento de realizar la búsqueda.

Para la realización de este proceso existen librerías en la actualidad que brindan potencialidades en el mismo, tales como facilidades para el desarrollador en los procesos de indexación y búsqueda. Poseen algoritmos y modelos de recuperación ya probados y de gran eficiencia, entre otras. Entre las más usadas se encuentran Lemur, Xapian, Terrier, Lucene, entre otros. Todas permiten la indexación y búsqueda de archivos de diversos formatos, tales como Plain text (TXT), Portable document format (PDF), Microsoft Word document (DOC), HyperText Markup Language (HTML), Microsoft PowerPoint document (PPT), entre otros.

---

[1] Para más información http://www.asmoz.org

[2] Para más información http://www-gist.det.uvigo.es

[3] Para más información http://www.slideshare.ne

[4] Para más información http://www.elearningworkshops.com

Introducción

Actualmente, muchas instituciones y centros de formación, principalmente las universidades, están utilizando los ROA para gestionar sus contenidos. Esta demanda ha propiciado la optimización de todas sus funcionalidades, dando lugar a una evolución de estas herramientas en correlación con el desarrollo del e-learning. En Cuba diversas instituciones han incorporado estos tipos de sistemas, ejemplo de esto es Infomed[5] que cuenta con su propio repositorio, la Universidad de la Habana (UH) utiliza un ROA basado en DSpace[6], la Universidad de Oriente posee un desarrollo propio de Objetos de Aprendizaje (RODA), con funcionalidades básicas y el Instituto Superior Politécnico José Antonio Echeverría (CUJAE) cuenta con un Centro Virtual de Recursos.

La Universidad de las Ciencias Informáticas (UCI) en el 2008 comienza a desarrollar un sistema con este objetivo (Cañizares, 2008), surge así el RHODA en su versión 1.0 para la UCI, que almacenaba y gestionaba los OA con un servicio de búsqueda básico. Aunque esta versión posee una gama de características que lo hacían utilizable no se encuentra a la par de tecnologías que surgen con respecto al e-learning: nuevas versiones de estándares, funcionalidades, entre otros. Posteriormente se mejora incluyendo una búsqueda avanzada basada en metadatos, lo que permite mejorar la precisión del proceso de búsqueda y la configuración, al ser posible la selección de los metadatos involucrados en dicho proceso, alcanzando ya una madurez con la versión 2.0.

Aunque la mejora del proceso de búsqueda y recuperación en el repositorio representó una solución para la nueva versión de la herramienta, la búsqueda por metadatos no es exhaustiva, ya que el campo de acción se basa en un resumen del contenido del OA, desperdiciando la mayor parte de la información contenida en sus objetos de información. Además carece de parámetros de búsqueda que mejoren la precisión del proceso y posee una estructura pobre, carente de métodos y algoritmos de búsqueda ya existentes que garanticen confiabilidad en el buscador.

Por lo expuesto anteriormente se plantea el siguiente **problema de investigación:** ¿Cómo mejorar el proceso de búsqueda y recuperación de Objetos de Aprendizaje en el Repositorio de Objetos de Aprendizaje RHODA?

Se plantea como **objeto de estudio:** Procesos de búsquedas en repositorios de información.

[5] Red de salud de Cuba.

[6] Más información en: http://www.dspace.org

Introducción

Objetivo General: Introducir modelos y técnicas de recuperación de información en la implementación de la nueva versión del módulo búsqueda, para perfeccionar el proceso de recuperación de Objetos de Aprendizaje en RHODA.

El **campo de acción**: Modelos y técnicas para la Recuperación de Información.

Idea a defender: Con la introducción de modelos y técnicas de recuperación de información en la implementación de la nueva versión del módulo búsqueda, se permitirá perfeccionar el proceso de recuperación de Objetos de Aprendizaje en RHODA.

El objetivo general se desglosa en las siguientes **tareas de investigación:**

- Análisis de los referentes teóricos relacionados con la investigación.
- Análisis del proceso de búsqueda en RHODA.
- Implementación del módulo de búsqueda en RHODA.
- Realización de pruebas al módulo desarrollado.

Los métodos utilizados para la investigación son: el **Histórico-Lógico** para el estudio del origen y desarrollo de los sistemas de búsqueda y recuperación de información, así como los conceptos, estándares y técnicas involucrados. El **Analítico-Sintético** para el análisis de la documentación actual vinculada al tema, así como los avances en el mundo del e-learning y particularmente en los ROA; todo lo que pueda ser útil para dar solución al problema planteado. Para la representación gráfica de los artefactos de implementación, la **Modelación**.

El presente trabajo se estructura en tres capítulos:

Capítulo 1: Se realiza un análisis de los elementos esenciales de los procesos de búsqueda en los ROA, que constituyen el fundamento teórico del problema a resolver. Se hace referencia a los sistemas de búsqueda y recuperación de información, las técnicas que emplean, cuáles son sus principales funciones y toda la terminología correspondiente. Se hace referencia a las principales tecnologías involucradas en el tema.

Capítulo 2: Se propone la solución al problema planteado, se realiza una descripción detallada de todos los aspectos a tener en cuenta para el perfeccionamiento del módulo de búsqueda y recuperación de información en el RHODA. Se exponen los principales artefactos del flujo de implementación de la metodología utilizada.

Capítulo 3: Se obtiene un módulo de búsqueda con la utilización de Lucene, configurable, accesible y presenta varias secciones de búsqueda con exhaustividad y rapidez. Se describe como se realiza la validación de la solución propuesta, a través de pruebas de Caja Blanca y Caja Negra.

Capítulo 1

Capítulo 1: Fundamentación teórica

Introducción

En este capítulo se recogen los principales conceptos, modelos, técnicas y procesos de búsqueda, con el objetivo de establecer la base teórica de los temas abordados en el resto del documento. Se realiza un estudio sobre los sistemas de recuperación de información y los ROA, resaltando sus características principales y estructura, así como las principales tendencias, tecnologías, metodologías, plataformas de desarrollo y herramientas que hicieron posible la realización de la presente investigación.

1.1 Repositorios de Objetos de Aprendizaje

Formalmente no existe una única definición para el concepto de OA, muchos autores han hecho su aporte a la misma entre los que se encuentran (Guzmán, 2005), (Chiarani, et al., 2006), (Belén, 2010), entre otros. Debido a la existencia de diversidad de criterios en cuanto a cuáles son los elementos principales que caracterizan los OA y a la evolución de los conceptos con el surgimiento de nuevas características que lo definen por los autores antes mencionados; los autores de la presente investigación realizan una recopilación, utilizando como nueva definición de OA: recurso digital con una marcada intención formativa y estructura didáctica variable, compuesto por uno o varios objetos de información (elementos digitales o recursos digitales), descrito con metadatos, con un comportamiento secuenciado que asegure el correcto enlace entre los elementos de su estructura didáctica, que puede ser utilizado y reutilizado en entornos e-learning.

Los repositorios de forma general han estado presentes en los entornos e-learning, pero naturalmente todos no se ajustan a las necesidades educativas, haciéndose necesaria una herramienta capaz de explotar todas las potencialidades de un OA. Surgiendo así lo que se conoce como Repositorio de Objetos de Aprendizaje, que según (Belén, 2010) es: *"una herramienta diseñada para almacenar organizadamente OA, permitiendo así su localización, recuperación, catalogación y reutilización".*

Los ROA pueden clasificarse en dos tipos: están los que almacenan metadatos, que son aquellos que contienen solo la descripción de los OA y una vez localizado el recurso lo referencia a una fuente externa, generalmente otro repositorio, y los que almacenan recursos con sus metadatos, donde el objeto y su descripción se encuentran en un mismo servidor (Chiarani, et al., 2006).

Los ROA más conocidos comúnmente funcionan de forma independiente (stand-alone). Son aplicaciones con una interfaz web, un mecanismo de búsqueda y listados con algún tipo de

clasificación. Otra clase de ROA operan solo como módulos adicionales a otros productos (LMS[7] o LCMS[8]) que utilizan los contenidos de forma exclusiva y sin que el usuario tenga acceso directo al repositorio. Lo deseable es que los ROA tengan ambas capacidades, tanto ofrecer una interfaz web, para que los usuarios humanos puedan acceder a la colección, así como la capacidad de comunicarse directamente con las plataformas de aprendizaje y hacer posible la interoperabilidad entre sistemas de diferente naturaleza (Guzmán, 2005).

Otra forma de clasificarlos es mediante la forma en que almacenan los metadatos, por un lado se pueden apreciar los distribuidos, donde existen varios servidores y cada uno posee un grupo de metadatos, que de alguna forma colaboran entre sí, y por otro lado los centralizados, que son aquellos que tienen todos los metadatos en un mismo servidor (Chiarani, et al., 2006).

IMS cuenta con la especificación IMS-DRI (Instructional Management Systems - Digital Repositories Specification) que define un conjunto de funcionalidades básicas que deben poseer los ROA (IMS, 2011):

Buscar / Exponer: el modelo de referencia de la búsqueda define la búsqueda de los metadatos asociados con el contenido expuesto por los repositorios.

Reunir / Exponer: define la solicitación de metadatos expuestos por repositorios, la adición de los metadatos para su uso en las búsquedas posteriores, y las inclusiones de los metadatos para crear un nuevo repositorio de metadatos.

Enviar / Almacenar: se refiere a la forma en que un OA se mueve a un repositorio de un determinado lugar accesible desde la red, y cómo el OA será representado en ese repositorio de acceso.

Solicitar / Entregar: el componente de solicitud funcional permite que un utilizador de recursos que ha localizado un registro de metadatos a través de la búsqueda (y posiblemente a través de la alerta) para solicitar el acceso a los objetos de aprendizaje u otro recurso descrito por los metadatos. Entregar se refiere a la respuesta del repositorio que permite el acceso al recurso.

[7] Learning Management System

[8] Learning Content Management System

Alertar / Exponer: constituye un posible componente de un repositorio digital o un servicio agregado intermediario y prevé que e-mail / SMTP (Simple Mail Transfer Protocol) puede proporcionar esta funcionalidad.

Como se observa en las funciones anteriores, el estándar IMS-DRI atribuye gran importancia a los procesos de búsqueda y recuperación en los ROA. Este proceso constituye una funcionalidad básica y está involucrado en todos los demás. Está apoyado en los metadatos que caracterizan el contenido, aunque no está estrictamente sujeto a los mismos.

1.1.1 Búsqueda por metadatos

Metadatos (del griego μετα, meta, «después de» y latín datum, «lo que se da», «dato»), literalmente «sobre datos», son datos que describen otros datos. El concepto de metadatos es análogo al uso de índices para localizar objetos en vez de datos. Por ejemplo, en una biblioteca se usan fichas que especifican autores, títulos, casas editoriales y lugares para buscar libros. (Lapuente, 2009)

Así, los metadatos ayudan a ubicar datos en el proceso de búsqueda. Los ROA en su gran mayoría soportan el uso de búsqueda por metadatos, ya que estándares como SCORM, LOM, entre otros, establecen la asociación de ciertos metadatos a los mismos. Algunos ejemplos serían: fecha de creación o edición del OA, título, palabras clave, autor, formato y tamaño del paquete, entre otros. Al añadir esta capacidad, el usuario podría lanzar una búsqueda de la palabra clave “comunicación” junto a la fecha de creación “21-05-2009” para encontrar exactamente el OA requerido.

Los metadatos poseen varias ventajas: algunos son aplicados de forma automática, poseen un resumen que caracteriza el contenido, entre otros. En RHODA se autocompletan ciertos metadatos como “fecha de creación”, “fecha de edición”, entre otros, sin necesidad de intervención por parte del usuario, y otros tales como el título y descripción con la intervención del usuario.

La desventaja de los metadatos es que están limitados a un resumen de la información del OA construido por el usuario, y que en ocasiones no es fiable, ya que no depende del sistema sino de la capacidad de resumen del creador. Por consiguiente, no es completamente precisa la búsqueda por metadatos, a menos que se combine con otros métodos de búsqueda, como la de recuperación de información, para profundizar en sus objetos de información.

1.2 Recuperación de información

En la actualidad el desconocimiento de las personas no relacionadas al campo que verdaderamente abarca este concepto, representa uno de los problemas principales en la recuperación de información.

Capítulo 1

Se han publicado muchos libros de texto [(Baeza-Yates, et al., 1999), (Manning, et al., 2008)], cursos (Allan, 2009) y otros recursos que abordan este tema y proporcionan excelentes definiciones aclarando qué es exactamente la recuperación de información (Manning, et al., 2008). Realizan una definición muy intuitiva de este concepto: *"la recuperación de información (RI) se ocupa de encontrar material (normalmente documentos) de una naturaleza no estructurada (normalmente texto) que se encuentra en grandes colecciones (normalmente almacenada de forma digital) y satisface una necesidad de información."* Esta definición aborda tres elementos clave que la distinguen y son de importancia para la presente investigación. El primer elemento es el *material de naturaleza no estructurada* que especifica la búsqueda literalmente de documentos y no en tablas de bases de datos, el segundo punto especifica su naturaleza digital y el tercer punto deja explícito que siempre es con el objetivo de satisfacer la necesidad de información del usuario.

Anteriormente se menciona explícitamente una de las principales confusiones al hablar del tema: la naturaleza no estructurada del material a buscar. Esto deja fuera del ámbito de la RI temas como la búsqueda en bases de datos, donde la información se encuentra estructurada en un conjunto de campos, las búsquedas son exactas y los sistemas deben satisfacer ciertos requisitos de control de concurrencia, recuperación, entre otros. Además, la definición es lo suficientemente general como para no dejar fuera otros temas que sí se encuentran en el campo de la RI, como que la recuperación no es solamente de documentos.

Como se puede observar RI presenta un conjunto de tareas mediante las cuales el usuario localiza y accede a los recursos de información que son pertinentes para la resolución del problema planteado. En estas tareas desempeñan un papel fundamental los lenguajes documentales, "*sistemas lingüísticos artificiales utilizados para la representación del contenido de los documentos y de las necesidades de información de los usuarios de los sistemas de recuperación de la información"* (Molina, 2009), las técnicas de resumen, la descripción del objeto documental, entre otros.

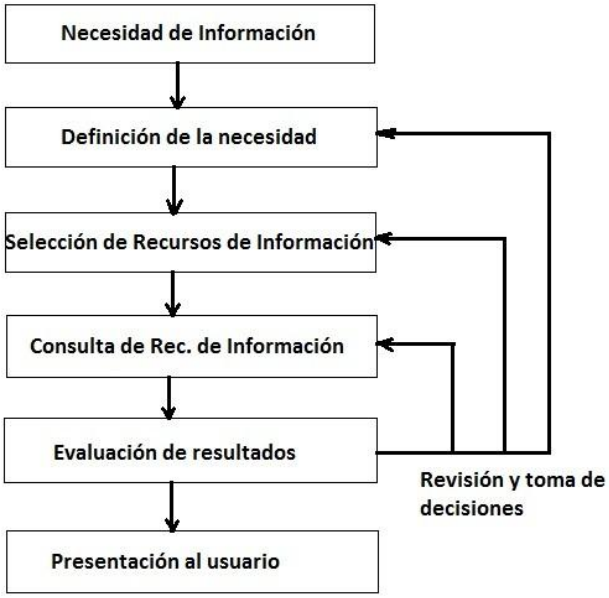

Figura 1. Proceso genérico de recuperación de información (Tramullas, et al., 2007).

Este proceso se lleva a cabo mediante consultas a la base de datos donde se encuentra almacenada dicha información, basadas en un lenguaje de interrogación adecuado. Es necesario tener en cuenta los elementos clave que permiten realizar la búsqueda, brindando un mayor grado de pertinencia y precisión, como son: los **índices**, **palabras clave**, **tesauros** y los diferentes fenómenos que se pueden dar en el proceso como son el ruido y silencio documental.

Una dificultad que surge en el proceso de búsqueda de información es si el resultado es "mucho o poco", es decir, dependiendo de la estrategia de búsqueda se pueden obtener como resultados multitud de elementos o un número reducido. Este fenómeno se denomina silencio o ruido documental. (Molina, 2009)

Los componentes esenciales de un sistema para la recuperación de información son los documentos y las bases de datos (Manning, y otros, 2008), en el primero es necesario establecer un proceso donde se definan herramientas de indización y el control de los términos utilizados, mientras que en el segundo estarán almacenados dichos documentos estableciendo el lenguaje de consulta que será utilizado y los operadores que soportará, además de establecer las combinaciones que serán permitidas.

Este esquema, aunque simplificado, representa las tareas más importantes que intervienen en la recuperación de información. Estas tareas representadas en la figura anterior, son las que diferencian

los distintos métodos o modelos de recuperación de información. Es decir, la manera de representar los documentos y las consultas, y "*la forma de medir la similitud entre ambos son algunas de las características distintivas de los modelos de recuperación de información"*. (Naveiras, 2009)

1.3 Modelos de recuperación

El vocablo modelo es un *"esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y el estudio de su comportamiento"* (Naveiras, 2009). De forma general y para fines de la presente investigación, se puede considerar un modelo de recuperación como un método para representar tanto documentos como consultas en sistemas de RI y comparar la similitud de esas representaciones. Para lograr esto, los modelos de recuperación proveen de manera implícita o explícita una definición de relevancia en cuanto a los resultados obtenidos.

Modelo Booleano

El modelo booleano es uno de los modelos de recuperación de información más básico. Está basado en la hipótesis de conjuntos y el álgebra de Boole. Los documentos y las consultas se simbolizan empleando términos clave (*keywords*) pero, las consultas también se crean como expresiones lógicas combinando los términos clave con operadores del álgebra de Boole (AND, OR, NOT). Este modelo se basa en que un término clave puede estar presente o no en un documento, por ello sólo será efectivo en aquellos documentos que contengan los términos clave que se establecen en la consulta (Naveiras, 2009).

Este modelo es sencillo y eficaz, pero presenta algunos inconvenientes, dificultad a la hora de formular justamente lo que el usuario necesita respecto a información utilizando el álgebra de Boole y términos clave, además de eso solo adquiere los documentos que concuerdan perfectamente con la consulta y no suministra *rankings* con los resultados. Estos inconvenientes provocan que la eficacia del modelo sea inferior a la de otras opciones.

Modelo probabilístico

El modelo probabilístico (S.K., et al., 1976) precisa la recuperación como un proceso de clasificación. Las consultas son catalogadas en dos clases: la de los documentos relevantes y la de no relevantes. Por ello, si se tiene un documento *D* y se establece una consulta, se consigue calcular la probabilidad de pertenencia del documento a cada una de esas clases. Las disímiles formas para estimar las probabilidades son la base de los modelos probabilísticos.

Este modelo considera que la importancia de un documento para una consulta es independiente del resto de documentos de la recopilación. Asimismo, considera que hay una serie de documentos que deben estar incluidos en el resultado de la consulta por preferencia del usuario y que con la elaboración de un *ranking* en orden descendente, basados en la probabilidad de relevancia de los documentos, se logra la eficacia suprema (pues, la probabilidad de error disminuye) (Molina, 2009).

La principal desventaja del este modelo es su forma estricta de catalogación donde plantea que hay solamente dos clases de documentos, sin tener en cuenta términos medios. Todo basado en diferentes formas de calcular la probabilidad de pertenencia de los documentos estas clases, con la vía de salida de que el usuario incluya un documento de su preferencia en los resultados para solventar una probabilidad de error existente.

Modelo Vectorial

La idea fundamental en la que se basa el modelo vectorial (G., et al., 1968) es considerar que tanto la importancia de un término clave con respecto a un documento, como los documentos y las consultas se pueden representar como un vector en un espacio de alta dimensión. Para valorar la relación entre un documento y una consulta; o sea, para saber si un documento es relevante para el resultado de una consulta, sencillamente se efectúa una comparación de los vectores que los representan.

Entre los métodos más habituales para comparar el grado de analogía se encuentra, el de calcular el coseno del ángulo que forman los vectores, esta manera es conocida como *fórmula del coseno*. Mientras más similitud halla entre los vectores, más cercano a cero grados será el ángulo que crean y, en efecto, más se acercará a uno el coseno de ese ángulo (Manning, et al., 2008).

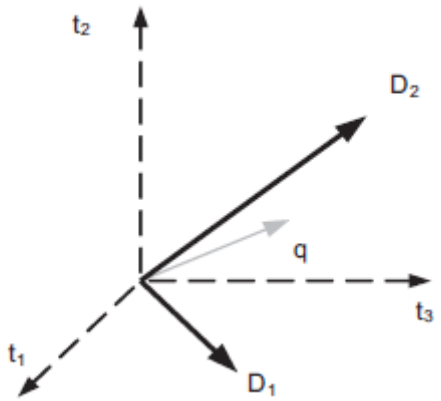

Figura 2. Comparación de similitud en el modelo vectorial. (Manning, et al., 2008).

La construcción de los vectores, para representar tanto documentos como consultas, se efectúa a partir de los vectores de los términos clave que los constituyen (por ejemplo, la suma de dichos vectores) (Naveiras, 2009). Estos vectores de términos clave tienen unos coeficientes (longitudes y pesos) que representan entre otros factores la presencia del término y su importancia.

Básicamente este método se basa en que los pesos están determinados por la frecuencia de aparición del término clave en el documento y la frecuencia inversa de aparición del término en el conjunto total de documentos. En el capítulo 2 se presenta el prototipo, allí se muestra que para la indexación textual se emplea Lucene (Apache, 2009). Lucene utiliza una unión de los modelos vectorial y booleano.

El modelo vectorial alcanza resultados superiores a los demás métodos y, por ello, es el modelo más utilizado actualmente. Entre las aplicaciones que usan este modelo, además del aludido previamente Lucene, está el sistema SMART (G., et al., 1968), creado por Gerard Salton, promotor de la RI, y su equipo en la Universidad de Cornell. La mayoría de los motores web de búsqueda están basados en este modelo.

1.4 Estructuras de indexación textual

Para poder abordar el tema sobre las estructuras de indexación en sistemas de RI, es necesario introducir los diferentes tipos de consultas que se pueden resolver en este tipo de sistemas. Se puede diferenciar tres tipos principales de consultas (Naveiras, 2009):

- Consultas basadas en términos clave.
- Reconocimiento de patrones de texto.
- Consultas sobre la estructura del texto.

El tipo número uno es el eje de la RI pues en dicho grupo están las consultas que se efectúan normalmente en la web. En este grupo de consultas se encuentran las que se efectúan utilizando términos clave de forma individual, frases constituidas por términos clave, y aquellas que se valen de operadores lógicos para tratar conjuntos de términos y documentos. Habitualmente, las consultas de reconocimiento de patrones son más complicadas y se emplean como complemento en las consultas del tipo número uno, para facilitar a los sistemas la implementación de RI más avanzada. Las consultas sobre la estructura de los textos son más concretas y dependen del modelo de texto específico (Manning, et al., 2008). Este tipo de consultas toman en cuenta la estructura de los documentos y no su contenido, esto permite beneficiarse de las ventajas de la estructura del formato HTML.

Capítulo 1

El índice invertido o fichero invertido, es la estructura de indexación textual más popular. Brinda (Zobel, et al., 2006) una exhaustiva descripción de esta estructura, además de los progresos relacionados con ella que se han efectuado en los últimos años. Esta estructura es muy cómoda de mantener y permite resolver eficientemente consultas basadas en términos clave, principalmente cuando se buscan los términos clave de manera individual (Naveiras, 2009).

La figura 3 muestra un ejemplo de construcción de índice invertido sobre dos textos. Es una muestra resumida pues, cuando se construyen este tipo de estructuras, los sistemas reales utilizan una serie de técnicas de procesado de texto como el borrado de palabras sin significado (vocablos muy explotados y con poco beneficio para hacer consultas como *en, la, entre otros*), que provocarían que algunas de las palabras que se muestran no constituyesen parte del índice (Leiva, 2008).

El índice invertido es una estructura dirigida a palabra, combinada esencialmente por dos componentes: el *vocabulario* y la lista de *ocurrencias*. El vocabulario está constituido por el conjunto de todos los vocablos o términos clave que se aluden en los documentos a indexar. Las ocurrencias son las listas que se recopilan para cada término clave mostrando en qué documentos y la posición dentro de ellos en que se encuentra cada término. Las posiciones pueden ser palabras (como en el ejemplo de la figura) o caracteres (Zobel, et al., 2006). En dependencia de la consulta que tenga solucionar usualmente la estructura se utilizan palabras claves o caracteres, pues recurrir a las palabras facilita las consultas de frases y de proximidad, mientras que recurrir a los caracteres proporciona un acceso inmediato al texto.

El algoritmo de búsqueda básico, para solucionar consultas utilizando índice invertido, está compuesto por tres pasos. Primero, se ejecuta una búsqueda en el vocabulario con la idea de localizar los términos clave que constituyen la consulta. Segundo, se adquiere la lista de ocurrencias asociada a cada término clave localizado en el vocabulario en el primer paso. Por último, se tratan las listas de ocurrencias obtenidas para solucionar la consulta. La manipulación de las listas difiere según el tipo de consulta del que se trate, si en la consulta se tratan de localizar términos clave de forma individual, el manejo de la lista de ocurrencias radicará en combinarlas para alcanzar una sola lista ordenada con las soluciones (Manning, et al., 2008).

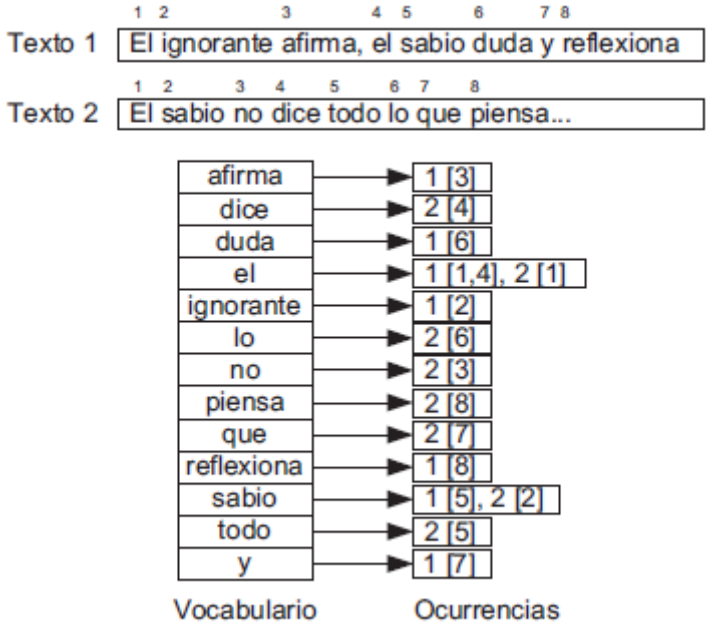

Figura 3. Ejemplo de construcción de un índice invertido (Naveiras, 2009).

A lo largo de los años, se han ido proponiendo variantes sobre esta estructura básica para perfeccionar algunos tipos de consultas y para reducir el espacio de almacenamiento. En (Baeza-Yates, et al., 1999) puede hallar una exhaustiva descripción de algunas de estas variantes. Sin embargo, aunque el índice invertido es la habilidad más eficaz para solventar búsquedas de términos clave de forma individual, cuando se ejecutan otros tipos de consultas menos usuales, por ejemplo las búsquedas de expresiones constituidas por términos clave, existen otras estructuras como los *arreglos de sufijos* (Manber, et al., 1990) que se satisfacen en menos tiempo. No obstante, estas últimas estructuras presentan la dificultad de ser más complejos de construir y de mantener.

Aunque se pueden describir numerosas estructuras de indexación, se considera que la descripción del índice invertido es suficiente para el alcance de esta investigación. También, en la próxima sección se explicará nuevas aproximaciones, fundamentadas en la idea de utilizar la compresión para abreviar el tamaño de los índices y de los documentos a indexar.

1.5 Estructuras de indexación. Compresión.

Con el desarrollo de numerosos buscadores en el campo de la RI y por una necesidad de ahorrar capacidad, se desarrolla un nuevo elemento: las técnicas de compresión de textos. Estas son ventajosas tanto para el ahorro de la capacidad del disco, como para disminuir los tiempos de procesamiento, de transmisión y de transferencia a disco cuando se efectúan las búsquedas. Las

técnicas de compresión delineadas especialmente para textos en lenguaje natural posibilitan buscar en el texto comprimido hasta ocho veces más rápido que en el texto original [(Turpin, et al., 1997), (Moura, et al., 1998)], además de reducir cuantiosamente el espacio necesario para guardar la información. Es posible adquirir **ratios** de compresión de entre el 25% y el 35%; es decir, el texto comprimido ocupará solo un por ciento dentro de este rango del texto original.

"La idea base de la compresión es el reemplazo de los caracteres o de las palabras por códigos, usualmente de longitud variable, que tienden a ser más breves según va ampliando la frecuencia de aparición del carácter o de la palabra y viceversa" (Naveiras, 2009). Las compresiones más sobresalientes se logran con modelos basados en la codificación de palabras (Moffat, 1989) en lugar de caracteres. Basado en la Ley de Zipf (Zipf, 1949), que plantea que las palabras poseen una distribución de frecuencias más sesgada que la de los caracteres. Esto posibilita que los textos, descritos como una secuencia de palabras, sean muy compresibles logrando ratios de compresión del 25% con la codificación óptima de Huffman (Huffman, 1952).

Las técnicas de compresión pueden ser importantes en el proceso de RI, ya que mediante las mismas es posible tener un acceso rápido a posiciones aleatorias en el texto. En la compresión es posible reorganizar el texto comprimido y utilizando un espacio adicional muy pequeño, se incrementan las capacidades de búsqueda en el texto, sin resultar una desventaja los tiempos de compresión y descompresión. Este trabajo de compresión es facilitado por las librerías de búsqueda que facilitan el desarrollo de los buscadores. En RHODA es posible utilizar dichas funcionalidades en la creación y optimización del índice, que constituyen los procesos donde está relacionado este elemento. El almacenamiento de los OA mediante SCORM no se afecta con el mismo, sino que se toman elementos del empaquetamiento, que se indexan mediante la librería utilizada con los medios para la compresión que brinda la misma.

1.6 Expansión de consultas

La expansión de consultas (query expansion) se enfoca en la tarea de mejora de los resultados que se muestra en el esquema simplificado de la RI tratado anteriormente. El objetivo principal de esta tarea es mejorar las consultas efectuadas por los usuarios para resolver los problemas provenientes de la sinonimia (vocablo que alude al uso de disímiles palabras sugiriendo el mismo concepto), que es importante para que funcionen correctamente los sistemas de RI (Baeza-Yates, et al., 1999).

Para mejorar los resultados es necesario depurar la búsqueda efectuada originalmente por el usuario, con más información que se conoce que está vinculada a los términos buscados. En dependencia de

cómo se adquiera esta información para refinar las búsquedas, entonces se utilizan los métodos de mejora de resultados. Estos métodos se dividen en: métodos globales, el que incluye la expansión de consultas, y métodos locales, que incluye la retroalimentación. Los métodos globales tienen la capacidad de reformular los vocablos de una consulta independizándolos de la consulta original y de los resultados adquiridos. No siendo así con los métodos locales, pues estos se identifican por reformular la consulta tomando en cuenta los documentos obtenidos originalmente (Manning, et al., 2008).

La retroalimentación sucede, por ejemplo, cuando se efectúa una consulta y de los resultados, se eligen los que se creen más relevantes para ejecutar nuevamente la consulta. La expansión de consultas sucede, por ejemplo, cuando se emplea un tesauro para ampliar la consulta automáticamente con vocablos que pueden ser afines a los conceptos buscados. Un tesauro define como un diccionario contiene listados de palabras o términos usados para simbolizar conceptos.

Tesauros

Los tesauros constituyen un listado terminológico controlado sobre un área o ámbito de conocimiento que mantiene entre sí relaciones semánticas y genéricas. Su principal característica es que los términos están ordenados jerárquicamente, permitiendo la precisión terminológica en la búsqueda de información. (Leiva, 2008)

Entre sus componentes se encuentran (Leiva, 2008):

Descriptores admitidos o preferentes: son aquellos términos normalizados (donde han sufrido un proceso expurgo denegando plurales, evitando sinónimos, entre otros) que el tesauro lo considera aptos para asignarlos a un documento y que posteriormente facilite la recuperación.

Descriptores no admitidos: son aquellos que aun estando normalizados no se consideran adecuados para emplearlos (suelen ser sinónimos, términos no utilizados en el campo de actuación, entre otros.)

Los tesauros han adquirido una gran importancia debido a que: "*existe la necesidad de describir la semántica de un dominio de forma que el humano lo entienda y que pueda ser procesado por el ordenador*" (Meersman, 2004). Para un proceso de búsqueda los tesauros son un elemento para la precisión en cuanto a terminología se trata, apoyado en que los términos están sujetos a una ordenación jerárquica que facilita su utilización. Aunque los mismos representan una gran ventaja, por la complejidad que representa la creación e implementación de los mismos es utilizado en buscadores avanzados, que por su enorme campo de acción lo necesitan, ejemplo están Google, Yahoo, entre

otros. En el caso de ROA no sería imprescindible el uso de los mismos ya que no se asemejan en complejidad y esto haría que necesiten un mayor consumo de recursos y tiempo de búsqueda que no poseen, siendo la opción más viable la retroalimentación como método para la expansión de consulta.

Diccionarios

Los diccionarios son utilizados para eliminar las palabras que no deben ser consideradas en una búsqueda (stop words o palabras vacías)[9], y para normalizar las palabras con el objetivo de que las diferentes formas derivadas de la misma palabra coincidan. Una palabra exitosamente normalizada se denomina **lexema**[10]. Además de mejorar la calidad de la búsqueda, la normalización y la eliminación de palabras vacías reduce el tamaño de la representación de un documento, lo que mejora el rendimiento. La normalización no siempre tiene significado lingüístico y por lo general depende de la semántica de la aplicación (Leiva, 2008).

Ejemplos de normalización pueden ser: lingüística, donde se trata de normalizar las palabras de entrada, tratar de canonizar direcciones URL[11] para hacerlas equivalentes, tratar de reemplazar los colores por su equivalente hexadecimal, entre otros (Zobel, et al., 2006).

En cualquier sistema que se trate de realizar búsquedas por contenidos, los diccionarios poseen una gran importancia, debido a que en el contenido de los documentos hay una gran probabilidad de que existan términos que necesiten ser normalizados para una mejor precisión en la búsqueda, elemento que convierte en imprescindible su utilización. Todos estos componentes unidos, junto a un lenguaje para que el usuario pueda realizar la búsqueda de una manera más organizada dan estructura al proceso.

Este elemento si es importante para cualquier buscador y generalmente lo desarrollan las librerías que facilitan el proceso de búsqueda, es de vital importancia y no importa la complejidad, ya que está relacionado con mejorar la calidad e incluso el tiempo de respuesta, con la eliminación de términos innecesarios. En un ROA, como en un buscador más complejo como Google, este elemento es necesario para mejorar la precisión del proceso de búsqueda.

---

[9] Las palabras vacías son palabras que son muy comunes, aparecen en casi todos los documentos, y no tienen valor de discriminación.

[10] Monema que posee un significado autónomo e independiente y constituye la parte invariable de una palabra

[11] Uniform Resource Locator

1.7 Lenguaje de interrogación

Todos los sistemas RI poseen su lenguaje de interrogación, siendo el que le permite comunicarse en el mismo lenguaje que la base de datos. El mismo contiene su propia sintaxis, que define las características especiales de la búsqueda. Las reglas gramaticales en dicho lenguaje de interrogación están conformadas por los operadores (Molina, 2009).

- **Lógicos o Booleanos:** Permiten convertir las palabras de la consulta en conjuntos matemáticos, y operar con las palabras como si fuesen conjuntos. Las operaciones básicas son la suma (OR), la resta (NOT) y el producto (AND).
- **Posicionales:** Permiten especificar la posición de las palabras dentro del documento. Cerca (NEAR), Junto (ADJ) y frases.
- **Existencia:** Indica cuando se requiere la presencia o ausencia de una palabra en los documentos resultantes. (Presencia / Ausencia) y (Ausencia).
- **Exactitud:** Este tipo de operador se utiliza cuando la consulta que se pretende realizar es menos específica debido a que da la posibilidad de cortar una palabra de búsqueda a su raíz. Por proximidad o por campos.

El lenguaje de interrogación y las ecuaciones de búsquedas son elementos principales en cualquier proceso de búsqueda. Como se menciona anteriormente no existe un procedimiento estándar que defina como hacer las búsquedas, de ahí la importancia que posee la selección de todas las cuestiones necesarias para la conformación del mismo. Cada uno de estos operadores garantiza precisión en la gramática del lenguaje, razón por la cual todos son indicados para la construcción del lenguaje de interrogación.

Todos estos elementos son utilizados para la realización de distintos tipos de búsquedas, en las cuales les dan un sentido y organización específica dependiendo de su objetivo. Aunque existe gran variedad de tipos de búsquedas, algunas son mucho más empleadas que otras debido a la precisión que logran por las características que poseen, ya sea en su uso simple o mixto.

1.8 Tipos de búsquedas

En la actualidad existe una gran diversidad de búsquedas utilizadas por los motores de búsqueda, cada una con sus características particulares que defienden su precisión y eficiencia ante las demás. La selección del tipo de búsqueda que realizará un buscador es un aspecto esencial para el resultado del mismo, de ahí la importancia que tiene la descripción de cada una.

Full Text Search

"Búsqueda de texto completo (o sólo la búsqueda de texto) proporciona la capacidad para identificar los documentos en lenguaje natural que satisfacen una consulta y, opcionalmente, para ordenarlos por relevancia a la consulta" (Postgres, 2011). En una búsqueda de texto completo, el motor de búsqueda examina todas las palabras en todos los documentos almacenados, para que coincida con las palabras de búsqueda suministradas por el usuario. Las técnicas de búsqueda se hicieron comunes en bases de datos bibliográficas en línea en el 1970 (Coles, 2008). Muchos sitios web y programas de aplicación (tales como procesadores de texto) proporcionaron capacidades de búsqueda de texto completo. Algunos motores de búsqueda web, tales como AltaVista[12] emplean esta técnica de texto completo.

La búsqueda de texto completo provee indexación de texto, permitiéndole a los documentos un pre-procesado y un índice revisado posteriormente a cualquier búsqueda rápida, incluyendo procesos como: análisis de documentos en tokens, conversión de tokens en lexemas y almacenamiento-procesado de documentos optimizados para la búsqueda. Está presente el uso de diccionarios en el proceso de normalización de los tokens con procesos como: definición de *stop words*, mapa de sinónimos de una palabra, variaciones de una palabra a una forma canónica[13] de varias formas, entre otros (Allan, 2009).

Es el tipo de búsqueda más utilizado en la actualidad por las ventajas que provee, aunque presenta una desventaja fundamental: los usuarios no siempre conocen lo que están buscando, es decir, que para arrojar resultados relevantes es imprescindible que el usuario introduzca términos lo más semejante posible al resultado deseado. Existe otro tipo de búsqueda que desarrolla este elemento: Fuzzy Search.

Fuzzy Search

"En informática, una cadena coincidente aproximada (a menudo coloquialmente se refiere a la cadena de búsqueda difusa) es la técnica de búsqueda de coincidencias aproximadas a un patrón en una cadena" (Widmayer, et al., 2002). Las búsquedas fuzzy o difusa tienen sentido cuando la persona no conoce cómo se escribe con exactitud lo que quiere buscar, por lo tanto, el sistema retorna términos que en su deletreo sean parecidos a la entrada de la búsqueda.

---

[12] Disponible en http://es.altavista.com/

[13] Llamada a menudo forma normal, es una forma estándar de representar el objeto.

Inicialmente las búsquedas difusas no se realizaban con índices por lo que requerían más tiempo de procesamiento y gran cantidad de datos, pero con el tiempo este elemento se fue integrando a las mismas, surgiendo algoritmos que la convierten en una de las principales. Un ejemplo de estos algoritmos es el de Levenshtein[14], utilizado para calcular la distancia entre dos palabras en cuanto a términos. Es usado por algunas herramientas como son PostgresSQL[15] y Lucene[16] para la realización de búsquedas (Postgres, 2011).

Como se menciona anteriormente existe una gran variedad de búsquedas y en la selección de las mismas radica el éxito. En cuanto a la búsqueda de texto completo se observa que es la más íntegra, conteniendo una serie de elementos antes mencionados que garantizan una mayor precisión en el proceso. Para la gran mayoría de personas esta representa una elección segura, pero existe un elemento que aunque parece resultar sencillo es importante: los usuarios no siempre tienen el conocimiento exacto de lo que desean buscar, apareciendo la búsqueda difusa como una solución complementaria. Ambas búsquedas complementadas representan una estable solución para cualquier buscador.

En RHODA es necesario el uso de este tipo de búsqueda mixto donde se complementa la capacidad exhaustiva de la búsqueda full text, junto a las características de ayuda en la consulta que propone la fuzzy para lograr resultados relevantes. Esto permitiría la indexación de todos los documentos localizados en los objetos de información de los OA para la posterior realización de la búsqueda, donde el motor de búsqueda examinaría todas las palabras indizadas. Para el usuario que no conozca exactamente la palabra que desea buscar, será posible introducir subcadenas de la misma donde el motor de búsqueda examina los documentos, haciendo coincidir las palabras con el patrón introducido.

1.10 Proceso de búsqueda y recuperación de Información en repositorios

La búsqueda y recuperación de información de información son procesos que deben estar implementados correctamente en un ROA. Como se menciona anteriormente la relevancia de los mismos es definida por el estándar IMS-DRI, donde se precisan como imprescindibles. En la actualidad existe un grupo de ROA que poseen gran prestigio internacional con numerosas funcionalidades al alcance de los usuarios que garantizan su calidad como herramienta. El estudio de

[14] Para más información http://www.merriampark.com/ld.htm

[15] Disponible en http://www.postgresql.org/

[16] Disponible en http://lucene.apache.org/java/docs/index.html

estos sistemas que marcan la punta de las nuevas tecnologías en el área es relevante, debido a que crea un marco en el campo internacional de los cambios y novedades que surgen. Un grupo de los mismos han incluido motores de búsqueda para mejorar los procesos de búsqueda y recuperación de información, de los que se seleccionan dos de los más destacados para un análisis su funcionamiento.

Agrega

"Agrega posibilita la búsqueda de objetos digitales educativos (ODE) partiendo de heterogénea información" (Agrega, 2011). Esta heterogeneidad, la complejidad con la que se corresponden los términos de búsqueda y la funcionalidad de búsqueda federada (difusión de una búsqueda entre varios nodos Agrega) requieren del motor de búsqueda una rapidez excepcional, además de la capacidad de constitución de consultas complejas, y la posibilidad de guardar y gestionar cuantiosos volúmenes de información.

Para satisfacer las necesidades de búsqueda Agrega adopta la librería de búsqueda Lucene. El objetivo principal que implementa la arquitectura de Lucene es el almacenamiento de documentos de los que se guardan campos de texto, así al buscar en esta plataforma con disímiles textos en los campos apropiados, los resultados obtenidos revelan los ODE almacenados que contienen estos datos. En el caso de no haber resultados que se ajusten a los parámetros de búsqueda, Lucene facilita la posibilidad de generar sugerencias a partir de los datos suministrados, lo que incrementa la probabilidad de encontrar el objeto que se está buscando. Con la utilización de Lucene, la experiencia de buscar objetos digitales educativos en Agrega es tan intuitiva como buscar páginas web en un motor de búsqueda de Internet. (Agrega, 2011)

DSpace

Dspace, un sistema de repositorio digital que permite capturar y describir documentos digitales como son Tesis, Artículos de Revistas, Publicaciones, entre otros, (Alvear, 2007) este posibilita las búsquedas de los contenidos que almacena, utilizando el buscador Yakarta Lucene[17].

Los términos de consulta se buscan en diferentes campos (autor, título, materia, serie e identificador de registro) de los archivos almacenados, devolviendo así aquellos documentos que se consideran relevantes (es decir, donde los campos coincidan con los términos de la consulta), en caso de la

[17]Para más información http://incubator.apache.org/lucene.net/

búsqueda por texto completo esté habilitada, permite también buscar estos términos dentro de los textos de los documentos existentes.

Algunas de las directrices utilizadas en ese repositorio para la RI son las *frases,* en las que se buscan la frase exacta, para ello es necesario encerrar lo vocablos entre comillas (por ejemplo, “Ciudad de la Habana”), el *stemming,* que es la expansión automática de las palabras con las terminaciones habituales, como son los plurales y tiempos verbales, la *coincidencia exacta***,** que indica que el término marcado con un signo de adición (+) delante debe aparecer obligatoriamente en el resultado de la búsqueda. Las búsquedas realizadas en este repositorio permiten además desechar las *palabras* vacías (vocablos que no aportan nada al término de consulta, Ej. a, es, en, son, entre otros) y establecer *términos no deseados*, marcando con un signo de sustracción (Ej. casa en permuta – Ciudad de la Habana) (UTPL, 2009). También consiente la introducción de *términos booleanos* en los términos de consulta, como son AND para sumar términos, OR para ampliar la búsqueda a dos términos y NOT para restringirla, por su parte los *paréntesis* permiten agrupar distintos términos de consulta.

La búsqueda avanzada en Dspace brinda más especificidad cuando se realizan las consultas para el proceso de RI, pues permite realizar búsquedas sobre campos específicos del repositorio y asociarles términos booleanos. Además posibilita seleccionar en que parte del repositorio se desea realizar la búsqueda.

Después de un análisis de estas herramientas podemos llegar a las siguientes conclusiones:

- Es necesario la utilización de una librería de búsqueda que facilite y mejore el proceso de recuperación de información.
- Los procesos de búsqueda y recuperación de información deben estar bien delimitados.
- La búsqueda avanzada debe estar dividida en la mayor cantidad de campos para garantizar precisión.
- Es necesario el uso de operadores booleanos.

1.11 Metodología de desarrollo de software

En la actualidad el uso de metodologías es un factor importante en el desarrollo de sistemas informáticos. El principal propósito es contar con un marco de trabajo claramente definido y estandarizado, que permita obtener productos que garanticen los requerimientos de calidad, que

cumplan con las expectativas del cliente y se desarrollen en el tiempo estimado y bajo los costos presupuestados.

Para la realización del sistema se decidió el uso del Proceso Unificado de Desarrollo de Software, RUP (del inglés: Rational Unified Process), debido a que el repositorio constituye una aplicación con una cantidad considerable de módulos, y por políticas del proyecto se decide utilizar dicha metodología.

Proceso Unificado de Desarrollo de Software

Para ejecutar un proyecto, obtener resultados esperados y definir los mismos, es necesario identificar el proceso de desarrollo que se debe seguir. Las metodologías de desarrollo no son más que procesos donde se esquematiza la evolución del proyecto, definiendo **Quién** hará **Qué**, **Cuándo** y **Cómo**. No existe una metodología universal sencillamente porque ningún proyecto es igual ni se maneja de la misma forma, por lo que los procesos de la organización deben ser configurables.

RUP posee una serie de ventajas que la hacen sobresalir entre otras metodologías según (2011):

- Unifica los mejores elementos de metodologías anteriores.
- Preparado para desarrollar grandes y complejos proyectos.
- Orientado a Objetos.
- Utiliza el UML como lenguaje de representación visual.

Debido a las ventajas antes mencionadas y a las necesidades de un proyecto como este se seleccionó Rational Undefined Proccess (RUP).

RUP define nueve flujos de trabajo de los cuales seis son de ingeniería y los restantes de apoyo, cada flujo está destinado a actividades específicas y están divididos en cuatro fases. Estos flujos se desarrollan en paralelo durante todo el ciclo de desarrollo estando presente, unos más que otros en las diferentes iteraciones, en la figura tres se pueden apreciar los mismos, así como su impacto en cada fase e iteración.

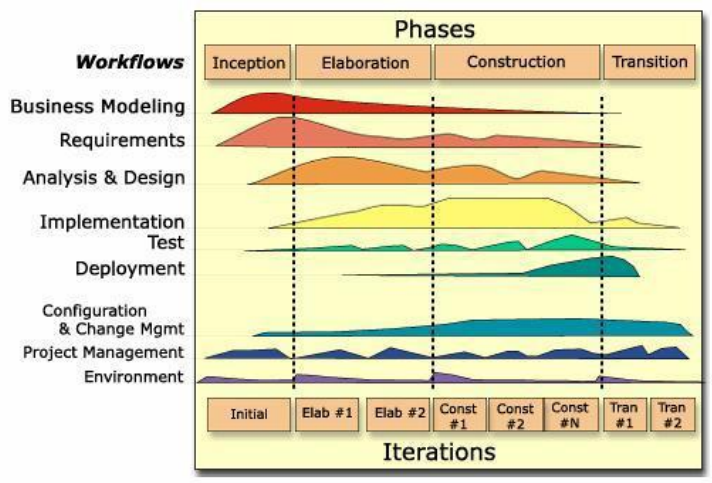

Figura 4. Ciclo de vida de RUP (Rumbaugh, et al., 2000).

1.12 Tecnologías

Para la implementación de una aplicación Web, específicamente en un ROA es necesario una arquitectura **Cliente / Servidor** con una serie de tecnologías que se describen a continuación.

Framework de desarrollo Symfony

Los ROA son aplicaciones implementadas en su gran mayoría sobre tecnologías Web, necesitan un fuerte modelo de datos, gestión de usuarios y otras prestaciones que hacen pensar en el uso del patrón de diseño Modelo-Vista-Controlador, una de las mejores alternativas es el uso de un framework de desarrollo. En su generalidad la mayoría de estos frameworks brindan las siguientes facilidades (Zaninotto, et al., 2005):

- Mapeo Objeto-Relacional u ORM según sus siglas en inglés.
- Seguridad ante diferentes tipos de ataques como la inyección SQL.
- Gran diversidad de librerías y utilidades para realizar tareas muchas veces engorrosas.
- Logran un Modelo-Vista-Controlador con una calidad aceptable.
- Muchas actividades y validación son posibles de implementar mediante archivos de configuración del Framework.
- El acceso a la aplicación es mediante una sola página servidora conocida como Controlador Frontal, por lo cual la aplicación tiene que estar accesible mediante la Web, lo que aporta un grado de seguridad mayor.

Capítulo 1

En la actualidad existe una gran diversidad de frameworks en diferentes plataformas y lenguajes, tal es el caso de Spring[18] para Java, Ruby OnRails[19] para Ruby, siendo este último posiblemente el más famoso de todos. Para el caso de PHP existen varios como: CakePhp[20], Prado[21], ZendFramework[22], Symfony[23], entre otros, en los últimos años Symfony ha tenido un gran auge y desarrollo, y en la UCI ha alcanzado gran aceptación por parte de los desarrolladores, gracias a sus numerosas ventajas.

Symfony pone a la disposición del desarrollador un conjunto de ventajas para la gestión de una aplicación, así como para la automatización de ciertas actividades como: la generación de código SQL, la creación del Modelo de Datos, la creación de módulos, entre otros, provee diferentes entornos de ejecución, siendo los más importantes el Entorno de Desarrollo, que cuenta con un sistema de depuración potente, y el Entorno de Producción para la ejecución de una aplicación ya en producción, aunque existen otros entornos y se pueden definir nuevos.

Symfony cuenta además con un sistema de caché de configuración (no es obligatorio, puede desactivarse), la mayoría en formato Yaml[24], son procesadas y convertidas en código PHP en la primera ejecución, en las restantes se utiliza el código PHP generado. Esta alternativa combinada con un sistema de caché propio de PHP como eAcelerator da una potencia mayor.

Este framework también dispone de un sistema de caché de las vistas generadas, siendo muy configurable y de gran utilidad, además de poseer un sistema de filtros, dándole al desarrollador la posibilidad de definir el orden en que se ejecutan, así como poder crear los suyos propios. Una aplicación en Symfony quedaría estructurada como se muestra en la figura #3 (Zaninotto, et al., 2005).

---

[18] Disponible en http://www.springsource.org/

[19] Disponible en http://rubyonrails.org/

[20] Disponible en http://cakephp.org/

[21] Disponible en http://www.xisc.com/

[22] Disponible en http://framework.zend.com/

[23] Disponible en http://www.symfony-project.org/

[24] Yet Another Markup Language

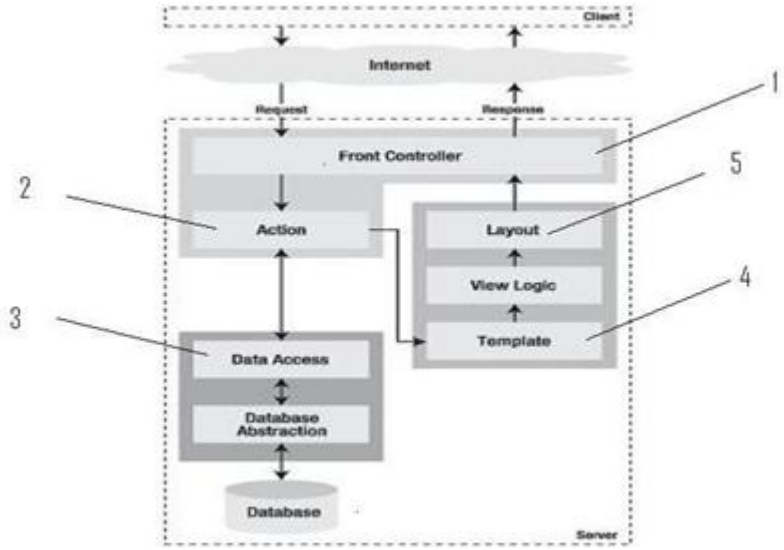

Figura 5. Estructura de una aplicación montada en Symfony (Zaninotto, et al., 2005).

A continuación se argumentan los elementos señalados (Zaninotto, et al., 2005):

Controlador Frontal: es una página servidora generada por el framework que controla todas las peticiones, propiciando así, que se ejecuten los mecanismos definidos por el framework.

Acción: un módulo en Symfony está formado por un conjunto de acciones, por ejemplo: pudiese existir el módulo usuario y dentro las acciones registrarse, editar perfil, entre otros.

Acceso a Datos: en Symfony se utiliza lo que se conoce como Mapeo Objeto Relacional, pudiéndose incluso hacer una ingeniería inversa a partir de una base de datos ya existente, aparte de estas clases generadas existen un conjunto de librerías para todo el manejo de este andamiaje.

Template: después que en la acción ya se ejecutó la lógica pertinente a la petición, estos datos son pasados a la plantilla de dicha acción para generar la vista correspondiente.

Layout: generalmente el layout es el esqueleto de la vista que se mantiene de alguna forma invariable en toda la aplicación; como el banner, los bloques entre otros, pueden existir diferentes layouts.

Symfony además brinda un conjunto de facilidades al desarrollador que no se mencionan como es el caso de los componentes, los slots de componentes, entre otros. Permite hacer pruebas, generar formularios (andamiaje como se le conoce en el mundo de los framework a esta funcionalidad) a partir del modelo y muchas más.

JavaScript. Librerías.

Una aplicación Web utiliza HTML como lenguaje de marcado para el diseño estático de las páginas que se envían al cliente, pero para lograr el dinamismo se debe utilizar JavaScript.

Entre otras razones debido a las incompatibilidades que Java Script presenta de un navegador a otro, hay que codificar una versión distinta de la mayoría de las funcionalidades para cada navegador, para resolver esto, se puede utilizar la librería Prototype[25]. Esta tiene un alto nivel de calidad en cuanto a diseño, facilidad de uso, y documentación, por otro lado, integra a sus helpers de una manera eficiente y le brinda potencialidad al framework.

En la aplicación se deben implementar funcionalidades que requieren un alto nivel de interacción con el usuario, donde se hace necesario el uso de componentes de interfaz de usuario avanzadas, ejemplo de esto es las vistas de árbol que no es soportado por el HTML tradicional ni forman parte de las librerías estándares de JavaScript. Existen numerosas librerías que facilitan dichas tareas, entre las mejores opciones están: ExtJS [26] y JQuery [27], ambas proveen eficientes soluciones a los requerimientos, además de contar con interesantes y ventajosas prestaciones en cuanto a Ajax[28], facilitando muchas tareas.

Hypertext Preprocessor (PHP)

Otras de las tecnologías utilizadas en el desarrollo de aplicaciones Web, es el lenguaje de programación PHP. Es uno de los lenguajes más aceptados para este tipo de desarrollo, y se utiliza de conjunto con el lenguaje HTML del lado del cliente y PHP del lado del servidor, formando así una arquitectura *Cliente/Servidor*. Con este lenguaje se puede realizar procesamiento de información en formularios, manipulación de cookies y páginas dinámicas, entre otros. Se puede integrar fácilmente con varios gestores de base de datos como: MySQL[29], PosgreSQL[30], Oracle[31], entre otros. Además, permite la incorporación de diferentes librerías externas, permitiendo así una mayor cantidad de

[25] Disponible en http://www.prototypejs.org/

[26] Disponible en http://www.extjs.com/

[27] Disponible en http://www.w3.org/XML/Query/

[28] Asynchronous JavaScript And XML

[29] Disponible en http://www.mysql.com/

[30] Disponible en http://www.postgresql.org/

[31] Disponible en http://www.oracle.com/

opciones, para desarrollar una determinada funcionalidad. Como aporte a sus grandes ventajas está el hecho de que es código abierto.

Estas potencialidades son necesarias para el desarrollo en un ROA, por lo que se hace indispensable que se acoja como lenguaje de programación.

Lenguaje Extensible de Marcado

Extensible Marckup Language (XML) es una tecnología que facilita la representación e intercambio de información entre diferentes plataformas, siendo un punto clave para su gran difusión, ya que es una recomendación de la W3C[32]. Básicamente XML es un lenguaje para estructurar datos de forma jerárquica y en texto plano. Como se menciona anteriormente, XML es más que un lenguaje, es una tecnología ya que provee no solo la sintaxis para la representación, sino que posee un conjunto de técnicas y lenguajes para definir el **esquema** de los mismos, para realizar consultas, hacer **referencias**, definir reglas para la **transformación** hacia otros formatos, entre otros (XML, 2011).

Esta tecnología cuenta con grupos de trabajo que se encargan de su definición, mantenimiento y evolución, entre estos se encuentran el grupo para el núcleo del XML, que se encarga de definir aspectos relativos al lenguaje, como los namespaces, atributos, entre otros, otro grupo se dedica a identificar mecanismos para definir el esquema de un XML, creando para este propósito los DocType y los XML Schema , otro se dedica a la transformación de XML en otros formatos como PDF y HTML, dedicándose al desarrollo y coordinación del Extensible Stylesheet Language (XSL), lenguaje mediante el cual se pueden definir reglas para la transformación de documentos XML. Existe un grupo dedicado a recuperar datos de XML, usando un lenguaje de consulta definiéndose principalmente para esta tarea el XPath y el XQuery. (XHTML, 2008) Con todo esto junto, XML se acopla de una manera decisiva en la mayoría de las aplicaciones de hoy en día, por lo que casi todas las plataformas de desarrollo contienen implementaciones para el mismo.

Lenguaje Unificado de Modelado (UML)

UML conocido en español como el Lenguaje Unificado de Modelado desempeña un papel protagónico en cualquier fase de una metodología de desarrollo. Es un lenguaje que permite modelar artefactos, relaciones, entidades, procesos, métodos de una situación determinada. Cabe recalcar que UML no es un lenguaje de programación, su objetivo es solo modelar situaciones, y se encuentra muy relacionado

[32] Disponible en http://www.w3c.es/

con la programación Orientada a Objetos, siendo el complemento que permite culminar de una forma directa lo que se modela con UML. En UML existen varios tipos de diagramas los cuales se agrupan en varias categorías por el tipo de ente que modelan. En la versión 2.0 existen 13 tipos de diagramas de forma categorizada, en esta categorización existen dos clasificaciones básicas basándose en el fin que persiguen los modelos que agrupa (Larman, 2003):

Diagramas de estructura: agrupa seis diagramas dedicados a especificar cómo se modela desde un punto de vista estructural alguna situación del problema a resolver.

Diagramas de comportamiento: agrupa siete diagramas dedicados a modelar el comportamiento, ya sea del sistema o de algún proceso del negocio que se pretende automatizar o servicio a brindar.

Librerías de Búsqueda

En la actualidad existe una gran cantidad de librerías de búsquedas, desarrolladas para enriquecer y facilitar el trabajo a los desarrolladores. Cada una de ellas utiliza criterios y elementos diferentes, como se menciona anteriormente, que definen sus ventajas en su campo de acción. Para la implementación de búsquedas en un ROA, se hace necesaria la utilización de las potencialidades que poseen.

Lucene

Lucene es una librería de búsqueda desarrollada en java, aunque tiene versiones en varios lenguajes. Básicamente permite no solamente reutilizar código propio, sino que permite aprovechar los desarrollos realizados por otros programadores y permite añadir a su núcleo extensiones realizadas por otros desarrolladores. (Zaninotto, et al., 2005)

Lucene ha tenido un gran auge en cuanto a búsquedas e indexación de contenidos se refiere debido a las numerosas ventajas que provee. Lucene posee una estrategia de búsqueda ya definida y eficiente, puede detectar por el mismo el modelo de datos que se está usando en la aplicación, tiene un alto nivel de configuración con la optimización de indizadores, múltiples indizadores e indizadores personalizados, entre otros

En la estrategia que provee Lucene están incluidos elementos como búsquedas tipo **full text** y tipo **fuzzy** con el uso de diccionarios, índices y tesauros. Es posible indexar de forma automática diferentes tipos de documentos como Word, Excel, o páginas web, pero permite de forma general indexar cualquier tipo de información pudiendo tener estos atributos independientes sobre los que luego podrán realizarse búsquedas. A la hora de realizar las búsquedas Lucene dispone de una sintaxis muy

completa que permite realizar búsquedas de todo tipo obteniendo los resultados más adecuados (Apache, 2009).

Características principales (Apache, 2009):

- API para el desarrollo de indización y búsqueda desarrollada en Java.
- Está disponible en C#, C++, Perl, Ruby y PHP.
- Multiplataforma.
- Permite indización incremental.
- Algoritmos de búsquedas fiables y confiables.
- Permite ordenar resultados por relevancia.
- Lenguaje de consulta.
- Stemming.
- Búsqueda por campos, rangos de fecha, entre otras.
- Ordenación por cualquier campo.
- Permite búsqueda mientras se actualiza el índice.

Sphinx

Sphinx (SQL Phrase Index) es un motor que permite buscar texto. Normalmente, es un motor de búsqueda independiente, que provee de forma rápida y eficiente resultados relevantes a otras aplicaciones. Está diseñado para ser integrado con MySQL y lenguajes de programación (actualmente PHP). Los datos se pueden recuperar mediante conexión directa a MySQL o mediante XMLs (Sphinx, 2011).

Dispone de cuatro utilidades: **indexer** para crear índices de texto, **search** para buscar desde la línea de comandos, **searchd** es un demonio que busca en los textos desde aplicaciones externas y **sphinxapi**, un API[33] para lenguajes de programación (PHP).

Actualmente para el framework de desarrollo Symfony existe un plugin que posee las mismas funcionalidades que el motor Sphinx original, llamado sfSphinx que aunque es menor en cuanto velocidad de indexación y búsqueda conserva su esencia. (Sphinx, 2011)

Posee desventajas fundamentales como que no permite actualizar el índice mientras se realiza una búsqueda y no permite la indización incremental.

[33] Application Programming Interface

Lemur

Como sistema de Recuperación de Información Lemur permite todas las etapas desde indización a la búsqueda de documentos. Lemur aporta una poderosa API implementada en C++ y está diseñada para trabajar en todos los sistemas operativos, permite la indización incremental e indiza atributos de los documentos. (2009)

Esta librería no posee implementación en PHP, por consiguiente no se integra con el framework Symfony, trabaja con el modelo probabilístico y no permite búsqueda por cualquier campo.

Xapian

Xapian es una biblioteca de funciones OpenSource de Recuperación de Información para crear motores de búsqueda escrita en C++, pero también se encuentra disponible en otros lenguajes como Perl, Phyton, PHP, Java, C#, y Ruby. Xapian contiene un API potente y adaptable que le facilita al programador los procesos de indización y búsqueda. (2009)

Xapian posee desventajas como que no permite la indización incremental, trabaja con el modelo probabilístico, no posee integración con Symfony y no permite la actualización del índice mientras se realiza la búsqueda.

Terrier

Terrier está implementado en Java, facilita mucho el trabajo a los programadores ya que permite indizar una colección de documentos de forma que sabe cuántos documentos contienen un término determinado, permite la búsqueda. (2010)

Este motor de búsqueda presenta la desventaja que no posee implementación en PHP, no posee integración con el framework Symfony, además de que no permite la indización incremental y trabaja con el modelo probabilístico.

Comparación de Lucene con otras librerías.

Características	Lucene	Sphinx	Lemur	Xapian	Terrier
Multiplataforma	Si	si	si	no	no
PHP	Si	si	no	si	no
Symfony	Si	si	no	no	no

Formatos indexados	Si	si	si	si	si
Stemming	Si	si	si	si	si
Búsqueda y actualización	Si	no	no	no	no
Indexación incremental	Si	no	si	no	no
Modelo	Vectorial	Probabilístico	Probabilístico	Probabilístico	Probabilístico
Licencia	Open Source	Open Source	Open Source	Open Source	Open Source

Tabla 1. Comparación de las librerías de búsqueda.

Debido a las ventajas que posee Lucene sobre las demás librerías analizadas se selecciona para la implementación del módulo de búsqueda en el repositorio.

Bases de Datos Nativas XML

En un ROA una de las funcionalidades centrales es la búsqueda de OA y recursos, según los metadatos que los describen, siendo crucial entonces la indexación de estos metadatos que están en formato XML, para ello surge la alternativa de mapearlos a una base de datos relacional, esto trae varias desventajas (Salvador Broche, et al., 2010):

- El esquema relacional que se obtiene en casos como el esquema LOM se complica en demasía, generando un gran número de relaciones con dependencias unas de otras, lo que puede introducir anomalías debido a la magnitud de este esquema.
- La aplicación debe tener una sincronía constante entre el documento físico y la base de datos.
- Las consultas para hacer ciertas búsquedas suelen ser muy complejas.

Existe otra alternativa que son las Bases de Datos Nativas XML, en las cuales se manejan directamente los documentos XML, su estructura lógica se basa generalmente en colecciones de documentos (análogo a las carpetas) y documentos. La gran ventaja radica en que se puede hacer un uso más completo de XML como tecnología, puesto que es posible hacer uso de lenguajes de consulta como XPath o XQuery, así como usar XSL para la transformación de resultados, algo que se hace muy engorroso mediante el uso de bases de datos relacionales.

En la tabla #1 se listan un conjunto de las principales Base de Datos XML:

Nombre	Licencia	API
XDB	Open Source	C++
CentorInteraction Server	Comercial	C++, Java
eXist	Open Source	Java
Tamino XML Server	Comercial	JSP
DBDOM	Open Source	Java

Tabla 2. Bases de Datos Nativas XML.

Visual Paradigm para UML

Visual Paradigm para UML es una Herramienta Computer Aided Software Engineering (CASE), muy útil para cualquier proyecto que desee generar artefactos con calidad, soporta UML hasta la versión 2.1, así como un grupo de funcionalidades que facilitan el trabajo de casi cualquier rol de un proyecto. Da soporte a casi todas las actividades de modelado que conforman los flujos de trabajo de RUP, además de que se integra con una gran diversidad de lenguajes permitiendo generar código a partir de los diagramas diseñados, también al diseño de base de datos integrándose con una gran diversidad de gestores como PostgreSQL, permitiendo generar SQL a partir de los diagramas e insertarlos en la base de datos seleccionada o generar modelos a partir de la misma, es decir, hacerle una ingeniería inversa a una base de datos creada en un gestor determinado, además de todo lo antes expuesto, esta herramienta es capaz de generar a partir de un proyecto determinado todo un sitio de documentación con las descripciones de cada uno de los diagramas diseñados.

Una de las razones que hace muy interesante al Visual Paradigm es que es multiplataforma, puede ser usada sin problemas en una versión de GNU/Linux, obteniendo los mismos resultados que en cualquier otro sistema operativo, cabe mencionar que esta suite es capaz de integrarse con una gran cantidad de entornos de desarrollo.

NetBeans para PHP

NetBeans es un Integrated Development Environment (IDE) escrito en Java y de código abierto, aunque puede servir para cualquier otro lenguaje de programación. Este ofrece una versión para desarrollar en PHP que comprende una variedad de secuencias de comandos y lenguajes de marcado. El editor de PHP está dinámicamente integrado con HTML, JavaScript y CSS. Posee integración con los framework Symfony y Zend, permitiendo filtrar y ayudar a ver los comandos de los mismos, incluyendo Doctrine o Propel. El editor de PHP entiende espacios de nombres y definiciones de las variables tipo de comentarios que mejora la finalización de código. Además de poseer otras características adicionales que incrementan su usabilidad (2011):

- Fácil navegación de código.
- Cobertura de código.
- Unidad de Prueba de PHP.
- Depuración de PHP.
- Desarrollo de proyectos remotos y locales.
- Integración con MySQL.

Conclusiones parciales

En este capítulo se hizo referencia a aspectos relacionados con los repositorios, específicamente los ROA, framework y tecnologías sobre la Web para el desarrollo de dos procesos importantes como son la búsqueda y recuperación de información, particularmente sobre los OA; arribando a las siguientes conclusiones:

- Es necesario la inclusión en un ROA del método de búsqueda centrado en los objetos de información de los OA, con la indexación de todos los recursos digitales que posea.
- El modelo Vectorial constituye el más indicado para el proceso de relevancia de los resultados por las ventajas que posee ante el modelo Probabilístico y Booleano.
- No es necesario la implementación de Tesauros en el sistema por las desventajas de rendimiento que proporcionaría.
- Se seleccionan un grupo de tecnologías indicadas para la implementación del sistema por las ventajas que ofrecen, entre las que se encuentran Lucene, NetBeans, Visual Paradigm, entre otras.

Capítulo 2

Capítulo 2: Descripción del sistema y propuesta de solución

Introducción

Se expone la descripción de la solución propuesta al presente problema de investigación, donde se presentan los artefactos de implementación arquitectónicamente significativos, se describe el funcionamiento general del sistema en correlación con su arquitectura y sus componentes. Además se muestran las clases fundamentales de la librería Lucene y la descripción de procesos como la indexación y búsqueda, y el modelo de datos del sistema.

2.1 Valoración del diseño propuesto por el analista

La obtención de los artefactos generados durante los flujos de trabajo Requisitos, Análisis y Diseño del sistema es un proceso de gran relevancia para la posterior implementación, ya que permite una visión más profunda de los aspectos afines a los requisitos funcionales y componentes reutilizables. Este diseño se exploró detalladamente para comprobar que las funcionalidades así como las clases persistentes relacionadas con estas, estén bien concebidas para facilitar la implementación de las mismas.

Para un mejor entendimiento de la solución de implementación, se exponen las principales funcionalidades identificadas para el desarrollo del sistema, que aunque no son parte de la presente investigación resulta necesario para guiar el trabajo de implementación.

2.1.2 Funcionalidades identificadas

Las funcionalidades identificadas para la implementación del sistema se exponen a continuación:

Realizar búsqueda básica

El sistema debe permitir realizar una búsqueda básica. En los términos de consulta se puede incluir los operadores booleanos AND, OR, NOT o el uso de caracteres especiales como son: +, - , “”. Además de explorar los documentos que conforman el objeto de información del OA, se buscan coincidencias parciales en el título y descripción de los OA.

Realizar búsqueda avanzada

El sistema debe permitir realizar una búsqueda avanzada. Esta incorpora la búsqueda por otros elementos como son: categoría, subcategoría, autor, área de conocimiento, formato, intervalo de fechas.

Realizar búsqueda por metadatos

El sistema debe permitir realizar una búsqueda sobre los metadatos de los OA, se muestran tres metadatos para realizar la búsqueda y el usuario debe seleccionar al menos uno.

Ordenar resultados

El sistema debe permitir ordenar los resultados por diferentes criterios, ejemplo: cronológicamente por la fecha de creación y alfabéticamente por el título del OA.

Autocompletar las búsquedas

El sistema debe permitir autocompletar la búsqueda. Este se apoyará en otras búsquedas similares realizadas previamente, según el término de consulta que haya arrojado más resultados.

Configurar las búsquedas avanzada y básica

El sistema debe permitir la configuración de la búsqueda por parte del usuario, este no necesita estar autenticado para realizarlo. El usuario debe seleccionar aspectos como: si desea que el sistema le autocomplete la búsqueda, número de resultados por página, formato de resultados, y si desea omitir o agregar algún campo para la realización de la búsqueda.

Guardar historial de búsqueda

El sistema debe permitir guardar un historial de búsqueda del usuario, permitiendo al mismo volver a realizar una búsqueda determinada si así lo desea, para ello el sistema debe almacenar los términos de consulta.

Localizar documento

El sistema debe permitir localizar los documentos del OA seleccionado en los resultados. Consiste en representar una vista semejante a la visualización del OA, donde se resalte la localización del mismo. En caso que los documentos no se encuentren referenciados en el OA, solo podrán descargarse.

Visualizar HTML y PDF

El sistema debe permitir visualizar, en la vista de localización del documento contenido en el OA resultante, los archivos HTML y PDF. Los demás archivos de otras extensiones solo se podrán descargar.

Testear búsquedas del historial

El sistema debe permitir testear de nuevo la búsqueda en el historial. El usuario podrá afirmar una búsqueda anteriormente realizada para obtener un resultado ya adquirido previamente, o nuevos resultados.

Autocompletar autor

El sistema debe permitir en la búsqueda avanzada autocompletar en el campo de autor con los usuarios del sistema, tanto por el nombre como por el usuario.

Sugerir resultados similares

El sistema debe brindar para cualquier resultado de la búsqueda, una sugerencia de otros OA similares al seleccionado de los resultados. Los OA similares serán aquellos de la misma categoría que coincidan con la búsqueda anterior.

Borrar historial de búsqueda

El sistema debe permitir borrar el historial de búsqueda. Una vez que el usuario considere puede limpiar el historial de búsqueda, esto posibilita claridad a la hora de reutilizar los términos de consulta almacenados.

Visualizar estadísticas del resultado

El sistema debe permitir mostrar en la visualización de un resultado, la cantidad de veces visualizados y descargados.

Resaltar vocablos coincidentes

El sistema debe permitir resaltar las palabras que coincidan con la búsqueda en los documentos; es decir, los vocablos en el texto del resultado arrojado por la búsqueda que coincidan con los vocablos utilizados en la consulta.

Retroalimentación de la búsqueda.

El sistema debe permitir una retroalimentación de la búsqueda. Si el usuario desea refinar la búsqueda, el sistema debe posibilitar al usuario realizar otra búsqueda en los resultados obtenidos de una búsqueda anterior para acotar el número de resultados.

Mostrar ficha técnica de resultado.

El sistema debe permitir visualizar una ficha técnica de cada resultado. La ficha técnica muestra la descripción del mismo, titulo, fecha de creación y palabras clave.

Mostrar tiempo de consulta y cantidad de resultados.

El sistema debe mostrar la cantidad de resultados arrojados y el tiempo en que se consultó. El tiempo de consulta es el tiempo en que el buscador obtiene los resultados.

Seleccionar formato de resultado.

El sistema debe permitir seleccionar el formato de visualización del resultado. Existen dos formatos, el formato corto y la vista general. El primero muestra el título, descripción y palabras claves del resultado arrojado, el segundo formato además de los aspectos del formato corto, el autor, la fecha de creación, categoría, palabras clave y cantidad de descargas.

Indexar documentos.

El sistema debe permitir indexar todos los documentos de un OA en el momento que se publique.

Actualizar resultados.

El sistema debe permitir actualizar la cantidad de resultados de las búsquedas que se repitan, debido a la posible indexación de nuevos documentos con la publicación de nuevos OA.

Optimizar índice.

El sistema debe permitir optimizar el índice de todos los documentos cada vez que se publique un nuevo OA en el repositorio.

Actualizar índice.

El sistema debe permitir actualizar el índice de todos los documentos cada vez que se publique un nuevo OA en el repositorio.

Mostrar búsquedas relacionadas.

El sistema debe mostrar al realizar una búsqueda, búsquedas relacionadas a la misma que hayan arrojado al menos algún resultado.

2.2 Principales componentes

Se abordan los principales componentes o partes del sistema, se muestran cada una de las acciones más importantes que contienen. Se enuncia gráficamente la relación entre los mismos (figura 6).

Los estereotipos utilizados en los diagramas de componentes son:

- <<file>>: son ficheros de configuración o código fuente.

- <<framework>>: representa un subsistema framework.
- <<library>>: representa una librería estática o dinámica.
- <<database>>: representa una base de datos.
- <<module>>: representa una parte modular del sistema.
- <<executable>>: representa un fichero binario ejecutable.
- <<script>>: representa un fichero con un código fuente interpretado.
- <<table>>: representa una tabla de la base de datos.

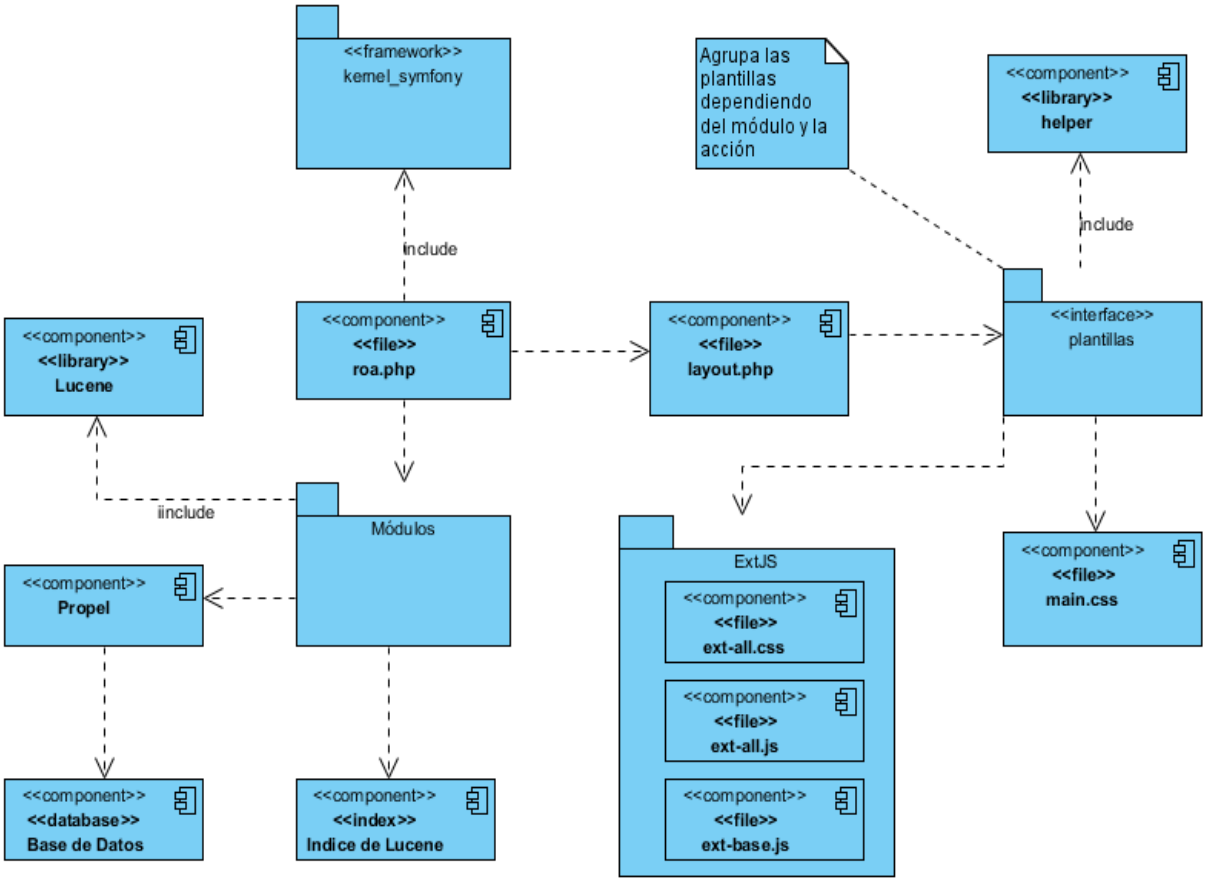

Figura 6. Diagrama de componentes del sistema.

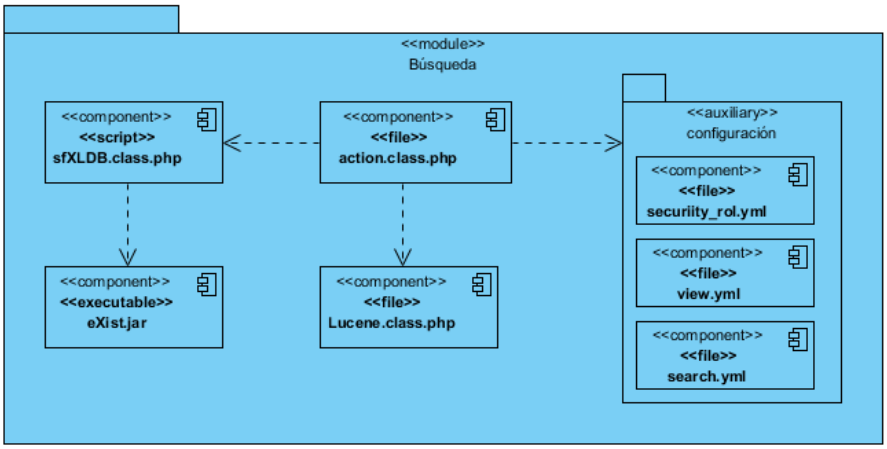

Figura 7. Diagrama de componentes del módulo de búsqueda.

2.3 Modelo de datos

El RHODA usa una base de datos relacional y otra nativa XML, para almacenar documentos XML usados en los OA para facilitar su uso, así como la indexación para búsquedas por metadatos. En los acápites siguientes se exponen criterios que se tienen en cuenta en el diseño de estos dos modelos.

2.3.1 Modelo de base datos relacional

El esquema relacional de la aplicación cuenta con 50 tablas, muchas de las cuales se encuentran interrelacionadas y otras sirven solamente para llevar tabulación de las diferentes relaciones de tipo muchos-a-muchos, las principales relaciones del mismo son:

- **roa_user:** almacenar datos de los usuarios.
- **rol:** para llevar los datos de los diferentes roles del sistema.
- **message:** para lo relativo a la mensajería interna del sistema.
- **category:** para tabular las categorías y sub-categorías del sistema.
- **visits:** lleva las trazas de las visitas de los usuarios del sistema.
- **log:** el sistema de trazas se guarda en esta relación.
- **pif:** guarda información relacionada con los diferentes objetos existentes en el repositorio.
- **revisor_records:** relación que se utiliza para guardar el historial de revisiones.
- **notify_general:** guarda información relacionada con las notificaciones de los usuarios.

2.3.2 Modelo de datos nativo XML

Para la gestión de los documentos XML que el sistema maneja, exactamente el imsmanifest.xml concerniente a cada OA, se utiliza una base de datos nativa XML, siendo eXist el gestor utilizado. Para la recuperación de datos acerca de los mismos se usa XQuery y XUpdate para la actualización de dichos documentos.

El modelo consiste en una colección, la cual a su vez se encuentran compuesta por 3 subcolecciones: **edition** donde se almacenan los OA en edición, **edited** donde se encuentran los OA en revisión y **revised** donde se almacenan los OA publicados.

2.4 Modelo de despliegue

Como un esbozo macro de la arquitectura del RHODA se presenta el modelo de despliegue del mismo, pudiéndose observar el tipo de arquitectura Cliente-Servidor[34] con el patrón Modelo Vista Controlador[35] para el diseño de sus módulos, así como los principales componentes externos.

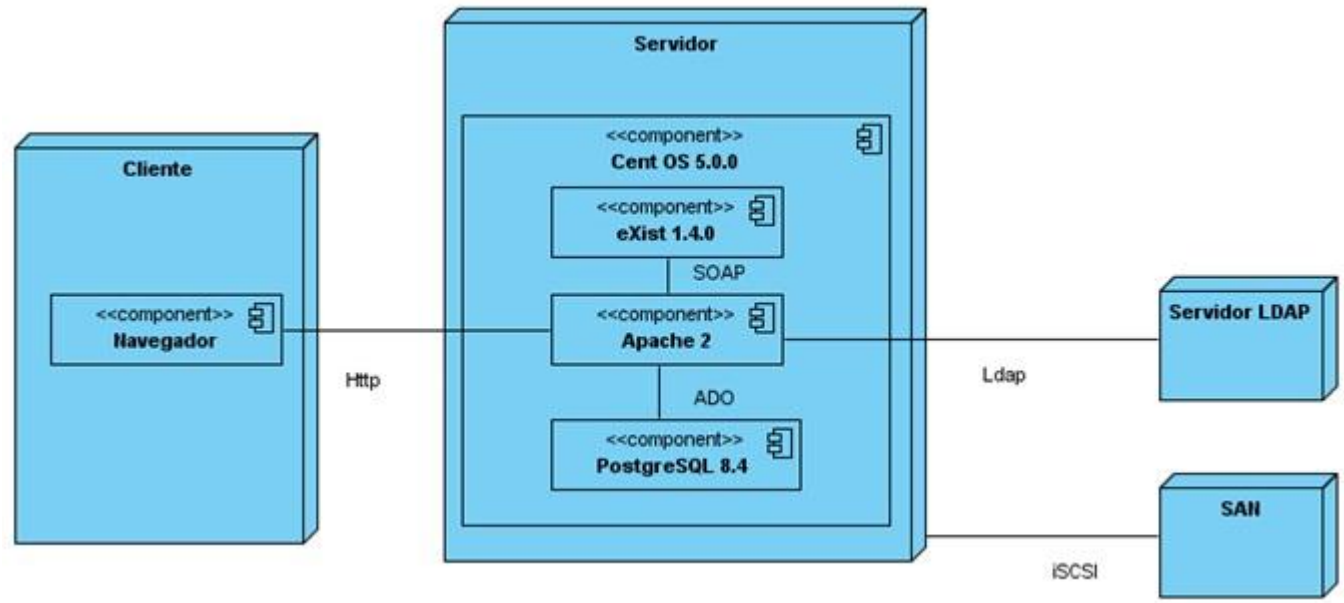

Figura 8. Diagrama de despliegue.

Se recomienda que el sistema sea instalado en un servidor de aplicaciones apache usando php 5, sobre el sistema operativo CentOS 5.0.0, donde se debe usar a su vez un gestor de base de datos

[34] Explicada en el glosario de términos

[35] Descrita en el glosario de términos

PostgreSQL 8.4 y un servidor de base de datos eXist versión 1.4.0, el sistema es capaz de autenticar usuarios basándose en un servidor LDAP. Para el almacenamiento de los ficheros es recomendable utilizar los servicios de un Storage Area Network (SAN). En el anexo 1 se encuentran las características del hardware para que funcione correctamente.

2.5 Principales funcionalidades

En este epígrafe se abordan las principales acciones en que se divide el módulo, se muestran cada una de sus interfaces, se argumenta su funcionamiento, objetivos y los segmentos del modelo de datos en que se apoya.

2.5.1 Búsqueda general

La búsqueda general representa la primera interfaz con que el usuario interactúa en el sistema. Contiene un campo de entrada y desde la misma se pueden acceder a las demás. Puede realizarla cualquier usuario sin necesidad de estar autenticado (invitado). Permite localizar los OA que se encuentren publicados por una coincidencia parcial en el título, la descripción, palabras clave y en el contenido de sus ficheros. En la misma se puede hacer uso de los operadores booleanos: AND, NOT, OR y cuando se desea obtener una frase exacta se debe encerrar entre comillas (“”). En la siguiente figura se muestra la interfaz:

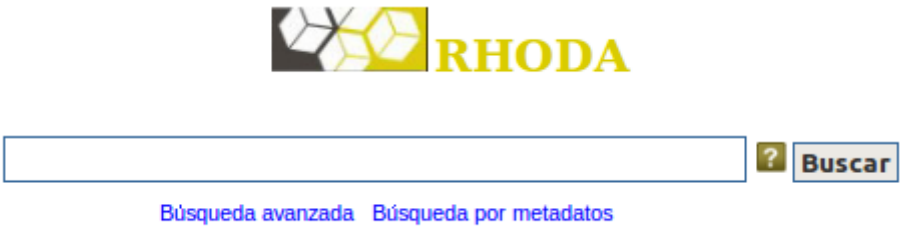

Figura 9. Búsqueda general.

Este tipo de búsqueda se apoya en la base de datos nativa XML eXist, que almacena el fichero imsmanifest.xml que contiene los datos a buscar. Además de hacer uso de Lucene para buscar en los contenidos indexados.

Implementación y algoritmia

Para la implementación de la búsqueda general es necesaria una secuencia de pasos, que garanticen la eficacia y eficiencia del proceso. Inicialmente se captura la cadena de texto que representa la

consulta realizada por el usuario y se parsea, utilizando Lucene para crear un objeto consulta como se muestra en el siguiente ejemplo:

```
$query = $this->getRequestParameter('query');

$obj_query = Zend_Search_Lucene_Search_QueryParser::parse($query);
```

Cuando se tiene el objeto consulta que define Lucene se procede a realizar la búsqueda. En este paso lo primero es abrir el índice donde se encuentran indexados los documentos, para posteriormente realizar la búsqueda sobre el mismo utilizando el objeto consulta previamente parseado.

```
$index = Zend_Search_Lucene::open(sfCOnfig::get('sf_data_dir').'/index');

$results = $index->find($obj_query);
```

Como la búsqueda general muestra coincidencias en características propias de los OA, que no se encuentran indexadas en el índice de documentos, es necesario realizar una búsqueda en la base de datos, luego se agrupan dichos resultados por OA, que constituye la unidad básica para la representación de los resultados.

Debido a que el sistema posee un historial de búsqueda, el siguiente paso es comprobar que la misma arrojó al menos un resultado. Si fue el caso, se procede a almacenarla con la cantidad de resultados respectivamente, de ya estar registrada se realiza una actualización de la cantidad de resultados solamente. Finalmente se muestran los resultados al usuario en el formato que haya definido en la configuración de la búsqueda.

2.5.2 Búsqueda avanzada

La búsqueda avanzada, semejante a la anterior, la puede realizar cualquier usuario sin necesidad de estar autenticado. A diferencia de la búsqueda anterior se suman otros criterios de búsqueda, se puede buscar por el autor, la categoría, la subcategoría, el contenido de los archivos de texto, las extensiones de dichos archivos, por fecha de creación y los resultados pueden ordenarse cronológicamente o alfabéticamente. Por último, se puede seleccionar el formato en que se mostrarán los resultados. En la siguiente figura se muestra la interfaz de la búsqueda avanzada:

Figura 10. Búsqueda avanzada.

En este tipo de búsqueda se hace un mayor uso de las facilidades que brinda la librería Lucene con los índices y la base de datos relacional PostgreSQL.

Implementación y algoritmia

La búsqueda avanzada posee una mayor cantidad de criterios de búsqueda, con el objetivo de garantizar mayor facilidad al usuario para introducir lo que desea buscar. Por ello en la implementación posee muchos más pasos a seguir y funcionalidades secundarias a tener en cuenta.

Paso 1: Verificar por qué campos desea buscar el usuario, definidos anteriormente en la configuración de la misma. Los campos o funcionalidades que sean omitidos en la configuración, son omitidos en la búsqueda avanzada también.

Paso 2: Mostrar los campos de la preferencia del usuario.

Paso 3: Validación de los campos que lo requieran. En este caso solo es necesario validar que seleccione al menos un criterio y que el intervalo de fechas de la petición sea correcto, utilizando las facilidades que brinda symfony con su archivo YML validate y la utilización del método validate para la realización de la misma.

Paso 4: Se procede a parsear la consulta, muy semejante a la funcionalidad anterior, con la única diferencia de que la consulta se divide en 4 campos. Debido a esto primeramente se construye la cadena de texto que representaría la consulta textual, y posteriormente se construye el objeto consulta con dicha cadena de texto.

```
$query = $this->ParseQuery($exact_frase, $all_words, $some_words, $not_words);

$obj_query = Zend_Search_Lucene_Search_QueryParser::parse($query);
```

Paso 5: Se accede al índice y se realiza la búsqueda utilizando Lucene (ver funcionalidad anterior).

Paso 6: Después de obtener dichos resultados se filtran por formato y se agrupan por OA.

Paso 7: Se filtran los resultados por los demás criterios de búsqueda (autor, categoría/subcategoría, fecha).

En esta funcionalidad se utiliza Ajax para el autocompletamiento con los usuarios del sistema en el campo autor y la actualización del campo subcategoría cuando es seleccionada una categoría específica. Para la paginación de los resultados fue necesario crear una clase que hereda de la clase sfPropelPager definida por Symfony, debido a que para la obtención de los resultados no se utiliza solamente el objeto Criteria (consulta), sino que se emplea el índice de Lucene, por lo que se hace necesario definir una clase que adquiera por parámetro el arreglo a paginar sin necesidad de que sea por una consulta a la base de datos.

```
Class sfArrayPager extends sfPager
{
        protected $resultsArray = null;
        public function __construct($class = null, $maxPerPage = 10)
        {
                parent::__construct($class, $maxPerPage);
        }
        //metodos para el nuevo arreglo
}
$pager = new sfArrayPager(null, 10);
$pager->setResultArray($oas);
```

En la búsqueda avanzada se realizan procesos semejantes a la búsqueda general, como el almacenamiento en el historial, el almacenamiento en la base datos si arroja algún resultado o la

actualización de la cantidad de los mismo de ya existir. Finalmente se muestran los resultados de la misma en el formato definido por el usuario y ordenados según la configuración.

2.5.3 Configuración de la búsqueda

Mediante esta funcionalidad el usuario desarrolla un papel fundamental en el proceso de búsqueda. Es posible configurar varios elementos como son la paginación por defecto, el autocompletamiento de las búsquedas, el formato de los resultados por defecto y los criterios de búsqueda. Dicha configuración no necesita de estar autenticado en el sistema. En la siguiente figura se muestra la interfaz de configuración:

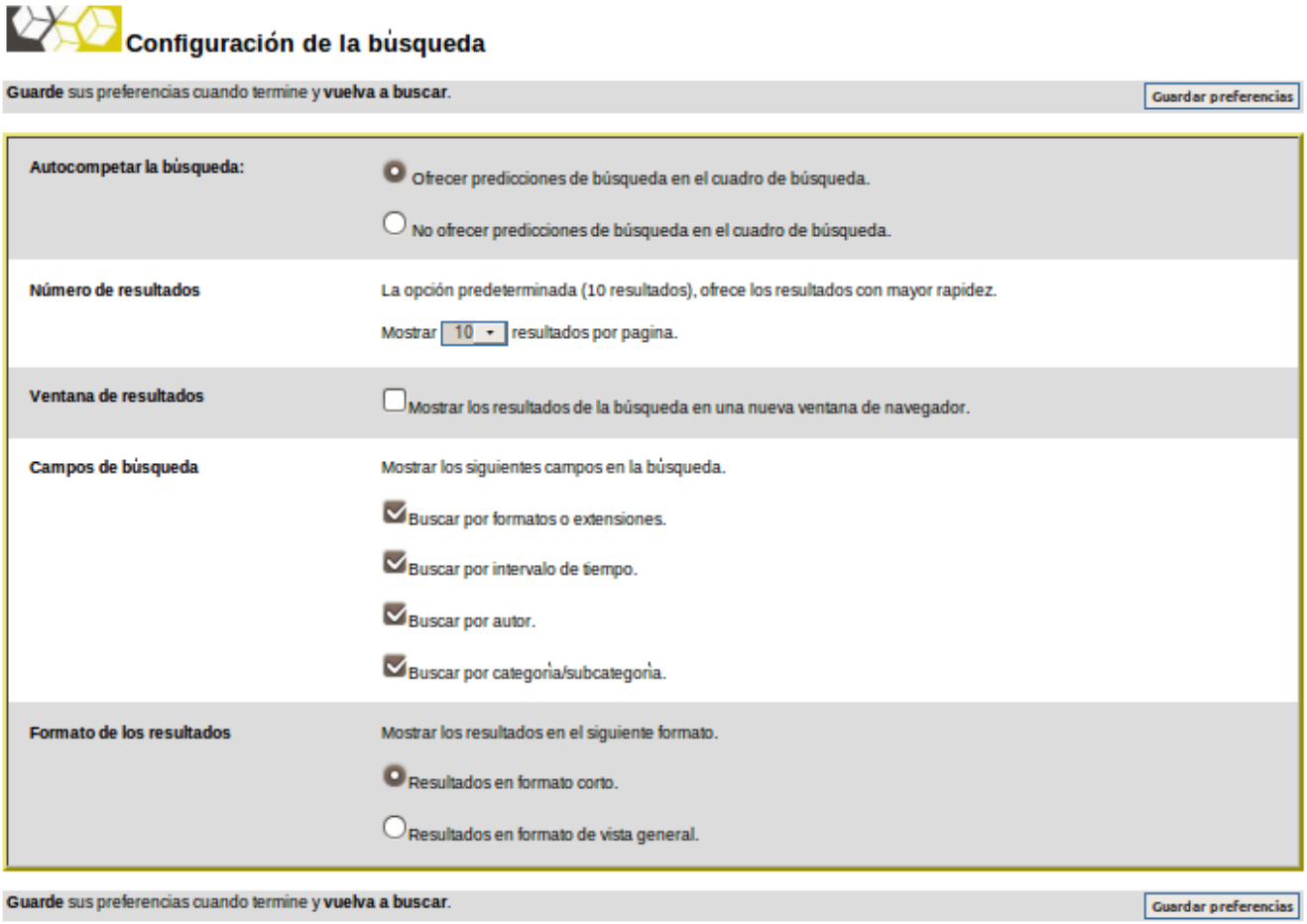

Figura 11. Configuración de la búsqueda.

En este proceso la configuración se almacena de forma temporal; es decir, que cuando el usuario cierre el navegador se pierden todos los cambios realizados.

Implementación y algoritmia

La configuración de la búsqueda en cuanto a algoritmia e implementación no representa gran complejidad aunque es de gran ayuda para el usuario. Se basa fundamentalmente en mostrar los diversos criterios de configuración para el usuario, donde se encuentre seleccionada la configuración por defecto del sistema. Para la implementación de la configuración es muy útil la sesión de usuario definida por symfony (aunque no necesariamente autenticado en el sistema) donde se almacenan dichos datos de manera temporal, e individualmente para cada computadora que se conecte al repositorio.

```
$this->getUser()->setAttribute('paginate', $paginate);
$this->getUser()->setAttribute('window', $window);
$this->getUser()->setAttribute('ext', $ext);
```

En este ejemplo anterior se muestra como se almacenan tres criterios de configuración en la sesión de usuario.

2.5.4 Historial de búsqueda

Esta funcionalidad puede realizarla cualquier usuario, no requiere de estar autenticado. Tiene como objetivo principal almacenar todas las búsquedas realizadas por el invitado (aunque no arrojen resultados), y testearlas de nuevo en el sistema. Esta funcionalidad posee gran importancia, ya que permite al usuario regresar a criterios anteriores de búsqueda que pudieron arrojar resultados de su interés y fueron olvidados. En la siguiente figura se muestra la interfaz del historial:

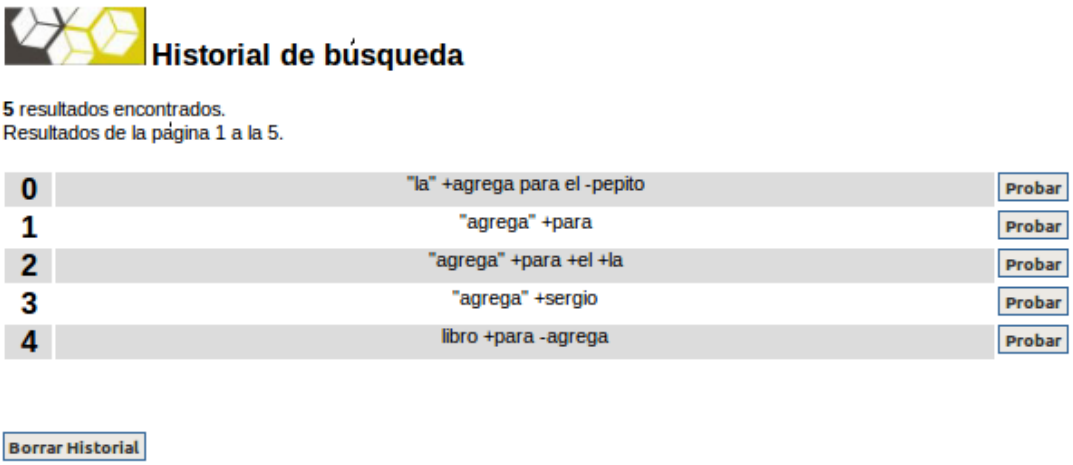

Figura 12. Historial de búsqueda.

En la misma se hace uso de la sesión de usuario de symfony, para el almacenamiento temporal del historial, muy semejante a la configuración de la búsqueda.

Implementación y algoritmia

El historial de búsqueda es una funcionalidad muy útil y que está presente en las funcionalidades de búsqueda general y avanzada, ya que en las mismas es donde se realiza el proceso de almacenamiento en la sesión de usuario.

```
$array = unserialize($this->getUser()->getAttribute('history'));
if($query != "" && !in_array($query, $array))
$array[] = $query;
$this->getUser()->setAttribute('history', serialize($array));
```

En el ejemplo anterior se utiliza una función propia de PHP que ayuda a convertir el array en una cadena de texto para poder almacenarlo en la sesión de usuario y la de unserialize para el proceso inverso. Cada búsqueda realizada en el historial es posible testearla de nuevo en el sistema y es posible borrar dicho historial totalmente. Además está presente también el uso de la clase sfArrayPager vista anteriormente, ya que no es necesario el uso de Criteria para la selección de los elementos a paginar.

2.5.5 Retroalimentación

Semejante a las anteriores, esta funcionalidad se puede realizar sin autenticarse. Su objetivo principal es refinar los resultados obtenidos en cualquier búsqueda, permitiendo realizar una búsqueda muy semejante a la general sobre los resultados. Contribuye a la precisión del resultado y tributa también facilitar al usuario encontrar los elementos de su interés con una mayor rapidez. En la misma se puede hacer uso de los operadores booleanos antes mencionados. En la siguiente figura se muestra la interfaz de la retroalimentación:

Figura 13. Retroalimentación.

Implementación y algoritmia

La retroalimentación es una funcionalidad muy importante para la precisión en el proceso de búsqueda debido a que, constituye un método de refinar la misma realizando una consulta sobre los resultados arrojados.

Para la implementación de la misma existen varias variantes entre las que se encuentran:

- La creación de un índice con los resultados, para la posterior consulta sobre los mismos. Esta variante consume tiempo y almacenamiento considerables cuando los resultados son muchos.
- La búsqueda en el mismo índice filtrando los resultados con la nueva consulta. Esta variante consume menos recursos pero cuando los resultados son muchos aumenta considerablemente el tiempo de consulta.
- La reformulación de la consulta para que incluya los resultados anteriores que coincidan con la nueva consulta realizada en la retroalimentación. Esta variante es similar a realizar una búsqueda básica sin incremento en el tiempo de consulta ni en el almacenamiento, ya que utiliza el mismo índice.

De las tres variantes se selecciona la tercera por las ventajas que ofrece, reformulando así la consulta en el método con el uso de los operadores booleanos. En esta se hace uso tanto de la base de datos relacional PostgreSQL como de la librería Lucene para la búsqueda en el contenido de los archivos de texto.

2.5.6 Búsqueda por metadatos

Muy semejante a la versión anterior, la búsqueda por metadatos se realiza sin autenticarse. Se mantiene casi idéntica, con la diferencia de que se separa de la búsqueda avanzada, ubicándose en una interfaz separada de la misma. Este tipo de búsqueda representa una búsqueda avanzada rápida que no incluye el contenido de los archivos de texto incluidos en los OA. Es más específica que la general pero menos profunda que la avanzada. En la siguiente figura se muestra la interfaz:

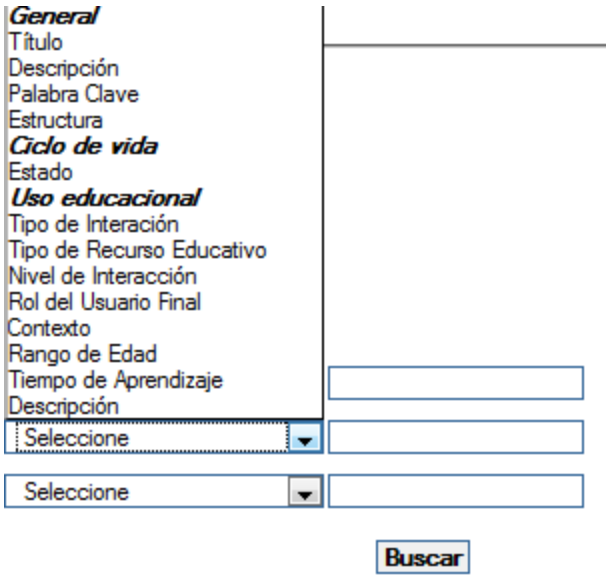

Figura 14. Búsqueda por metadatos.

En esta acción se hace uso de la base de datos relacional PostreSQL y la nativa XML eXist para efectuar la búsqueda.

Implementación y algoritmia

En el proceso de búsqueda por metadatos lo primero sería mostrar los metadatos seleccionados por el administrador en la configuración que especifica los metadatos por los que es posible realizar la búsqueda. Esto se realiza con el uso del archivo search.yml que almacena dicha configuración.

```
$this->arr_select = $this->_getSelectSearch();
```

Esta línea de código permite almacenar en un array la configuración guardada en el archivo YML mediante el método _getSelectSearch().

Después de mostrar los metadatos definidos por el administrador se procede a validar los datos entrados por el usuario. En este caso solo se valida que al menos se introduzca un metadato, con apoyo del método validate proporcionado por symfony.

Cuando se asegura la validación de los datos introducidos se realiza la consulta a la base de datos XML para luego mostrar los resultados de la misma.

```
$dbxml = new sfXmldb(sfXmldb::Query);
$data_xquery = $dbxml->xquery($xquery);
```

2.6 Indexación. Clases básicas.

Para el proceso de indización de documentos en Lucene existen una serie de clases básicas, descritas a continuación:

- IndexWriter
- Analyser
- Directory
- Document
- Field

Analyser

Esta clase permite la implementación de las fases de supresión de palabras vacías, el desarrollo de stemming (obtención de raíces), sinonimia (relación de semejanza de significados entre determinadas palabras), polisemia (cuando un símbolo o palabra tiene varias acepciones), entre otros. Permite entre sus funcionalidades el tratamiento de mayúsculas/minúsculas, acentos, caracteres especiales, entre otros.

IndexWriter

Representa el componente principal en el proceso de indización, permitiendo la creación y utilización de índices. Mediante esta clase es posible agregar documentos a dichos índices ya existentes. IndexWriter permite el uso de índices solamente en el proceso de indización, no siendo así en el de búsqueda.

Directory

La clase Directory representa la ubicación de un índice en Lucene. Esta a su vez utiliza clases hijas tales como FSDirectory para guardar dichos índices en el sistema de archivos. Dicha clase es la más usada en el almacenamiento de índices.

La clase IndexWriter hace uso de FSDirectory cuando necesita recibir como parámetro el directorio donde se almacenarán los índices. Otra clase hija de Directory es RAMDirectory, que a diferencia de

FSDirectory, se usa para el almacenamiento en memoria de los índices, ya que es recomendable cuando se crean índices pequeños o para la realización de pruebas de indización o búsqueda.

Document

La clase Document representa una colección de campos. El documento objeto de indización es dividido en campos o en metadatos, tales como: título del documento, fecha de modificación, autor, entre otros. Estos se almacenan en archivos distintos cuando se indizan.

Cuando se hace referencia a un documento se refiere a todo archivo que contenga texto como PDF, DOC, HTML, entre otros.

Field

Cada documento puede contener uno o más campos, en Lucene hay cuatro métodos *Fields* diferentes, que tiene como objetivo definir donde y el tratamiento de dichos campos:

Keyword: Se almacena y se indiza sin modificaciones, no es objeto de análisis, es utilizado para los campos que necesitan guardarse en el índice sin modificaciones como el directorio donde se encuentra el documento, el id del objeto de aprendizaje al que pertenece, entre otros.

UnIndexed: Se almacena pero no es usado en las búsquedas, como la llave primaria en una base de datos.

UnStored: Es totalmente opuesta a UnIndexed. Es objeto de análisis y se indexa, utilizando para todos los documentos de texto donde solo se requiera guardar título o contenido.

Text: Se analiza e indexa, almacenándose también el valor original.

Tipo de campo	Stored	Indexed	Tokenized	Binary
Keyword	si	si	No	no
UnIndexed	si	no	No	no
Binary	si	no	No	si
Text	si	si	Si	no
UnStored	no	si	Si	No

Tabla 3. Métodos Fields para la indexación.

2.6.1 Índices en Lucene

La creación de un índice representa el punto de partida para el trabajo con Lucene, puesto que una vez creado, se procede a adicionar todos los documentos que serán indizados.

A continuación se muestra como crear un índice con Lucene. Vale aclarar que aunque la librería Lucene es originalmente implementada en Java, los ejemplos de código que se muestran son en su versión para PHP, pero la idea de las clases y los métodos es lo importante.

```
$index = Zend_Search_Lucene::create("directorio_del_index");

$index = Zend_Search_Lucene::open("directorio_del_index");

$hits = $index->find($obj_query);
```

directorio_del_index: Especifica la cadena string que representa la ruta al directorio del índice a crear o abrir. Puede utilizarse una cadena de texto o en el caso de Symfony los métodos que devuelven la ruta de sus directorios habituales.

La función *create* y *open* son la que se utilizan inicialmente en los dos procesos fundamentales, la primera en el de indización y la segunda en el de búsqueda. Posteriormente continuaría la indexación de los documentos.

2.6.2 Indexar documentos

Después de abierto el índice donde se adicionan los documentos, como se muestra anteriormente, el sistema debe ser capaz de indexar los documentos. Consiste en pasar por parámetro la dirección donde se encuentran los documentos y el sistema debe recoger documento por documento, hacer el proceso de indexación, así como crear los índices de los documentos.

```
$doc = Zend_Search_Lucene_Document_TYPE::loadTYPEFile("ruta_al_documento");

$doc->addField(Zend_Search_Lucene_Field::METODO(parametro, valor));

$index->addDocument($doc);

$index->commit();
```

```
$index->optimize();
```

Inicialmente se carga el Documento de acuerdo con el tipo o la extensión del mismo (TYPE), donde se especifica la ruta hacia el mismo como parámetro. Posteriormente se procede a adicionarle los campos que se quieren indexar junto a él de acuerdo a los métodos de Lucene vistos anteriormente (METODO), donde se especifica el nombre del parámetro y el valor del mismo. Finalmente se adiciona el documento al índice abierto y se procede enviarlo (commit), para en último lugar optimizar dicho índice. Esta operación no es imprescindible pero resulta muy útil debido a que con la misma es posible optimizar el índice disminuyendo considerablemente el espacio de almacenamiento.

2.7 Búsqueda. Clases básicas

Para desarrollar una búsqueda en Lucene es importante familiarizarse con las siguientes clases:

- IndexSearcher
- Term
- Query
- Hits

IndexSearcher

En el proceso de búsqueda la clase IndexSearcher realiza una función semejante a IndexWriter en el proceso de indización. Es la clase principal encargada de abrir el índice para buscar en él, ofrece diferentes métodos de búsqueda que en general consisten en pasar la query como parámetro y retornar los objetos hits.

Term

El término es la unidad básica para la búsqueda. Muy similar al objeto Field, consiste en un par de elementos: el nombre del campo y su valor. A continuación se muestra un ejemplo donde se busca la palabra *lucene* en el contenido del documento:

```
Query q = new TermQuery(new Term("contents", "lucene"));
Hits hits = is.search(q);
```

Query

La clase Query es el objeto consulta que resulta de los datos introducidos por el usuario. Existen muchas clases que heredan de la misma como TermQuery, ParseQuery, entre otras. Todas con funciones específicas dependen del tipo de consulta que se realice.

Hits

La clase Hits es la encargada de almacenar los puntos de referencia a los resultados de la búsqueda, es decir, todos los documentos encontrados que están relacionados con la Query realizada.

Vistas las clase principales en los dos proceso fundamentales de Lucene, es importante ver realmente la funcionalidad de esta tecnología, por lo cual a continuación se muestran ejemplos de su utilización.

2.8 Estándares de codificación

Es necesario establecer un estándar de codificación con el objetivo de lograr que el equipo de desarrollo comprenda el código de los demás programadores, además el uso de un estándar para programar propicia que el producto termine con calidad. También existen ciertas reglas definidas por el framework, relacionada con la nomenclatura, que hay que respetar para lograr un correcto funcionamiento, estas tributan al estándar de codificación a usar.

Para el desarrollo del módulo se definieron un conjunto de pautas a seguir, en cuanto a la forma de escribir código fuente, teniendo en cuenta los siguientes aspectos:

Idioma.

Para nombrar variables, funciones, clases y demás se usarán nombres descriptivos y en idioma inglés.

Identificadores.

Los identificadores se comienzan a escribir con minúscula y en el caso de componerse de más de una palabra se escriben una a continuación de la otra, comenzando con mayúscula a partir de la segunda. Como se puede apreciar en el fragmento de código que describe a continuación:

```
classItem{
private $str ;
private $icon ;
private $actionPerform ;
```

Capítulo 2

Documentación del código fuente.

Una de las mejores prácticas para generar la documentación de los módulos es que esté autocontenida en los mismos, para ello en esta investigación, debido a que se utiliza PHP como lenguaje de programación, se utilizó el estándar phpDoc, el cual, la mayoría de los Entornos de Desarrollo Integrado que dan soporte a PHP lo interpretan y usan para ayuda al programador, además de que ofrece la gran ventaja de poder generar páginas HTML con documentación del código fuente.

```
/**
* This method returns an object with the SCORM schema for
* theXPath passed by parameter
*
* @param string $path
* @return stdClass
*/
private static functiongetItems( $path ){
```

Identación

Para identar cualquier fragmento de código se utilizan espacios en blanco y nunca tabulaciones, las estructuras anidadas usan 4 espacios para la identación, además los lexemas que se usan para comenzar y cerrar bloques “{” y “}” no poseen identación alguna.

```
public functionpreExecute()
{
    $id=$this->getRequestParameter('id');
    if ($id == 1)
    {
        $this->redirect404();
    }
}
```

Capítulo 2

Conclusiones parciales

En este capítulo se expuso la solución del problema a resolver, presentándose primeramente una valoración del análisis y diseño, donde se resalta principalmente los requisitos funcionales que son la guía principal de la implementación. Se hace referencia a otros aspectos como son los diagramas de componentes del sistema y del nuevo módulo de búsqueda con la incorporación de Lucene.

Se describen las principales funcionalidades del nuevo módulo, con la incorporación de las interfaces, así como se realizó la implementación y la algoritmia para darle solución. Además se describe el uso de las principales clases de la librería Lucene para potenciar el proceso de búsqueda y como se realiza en la implementación el proceso de búsqueda e indexación mediante la misma. Por último se define el estándar de codificación a seguir para una mejor comprensión del código.

Capítulo 3

Capítulo 3: Validación de la solución propuesta

Introducción

La prueba del software es un instrumento indispensable para garantizar la eficiencia del software. Esta etapa tiene como objetivo procurar la calidad del producto, sacando a luz diferentes clases de errores en la menor cantidad de tiempo y esfuerzo.

El proceso de prueba se basa en la lógica interna del software y las funciones externas, este puede ser aplicado a los artefactos generados en los flujos de trabajo. Constituye un proceso crítico para medir el grado en que el software cumple con los requerimientos. En este capítulo se abordan los tipos de pruebas Caja Blanca y Caja Negra, utilizados en el flujo de trabajo Pruebas de la metodología RUP, así como algunos de los artefactos que se generaron.

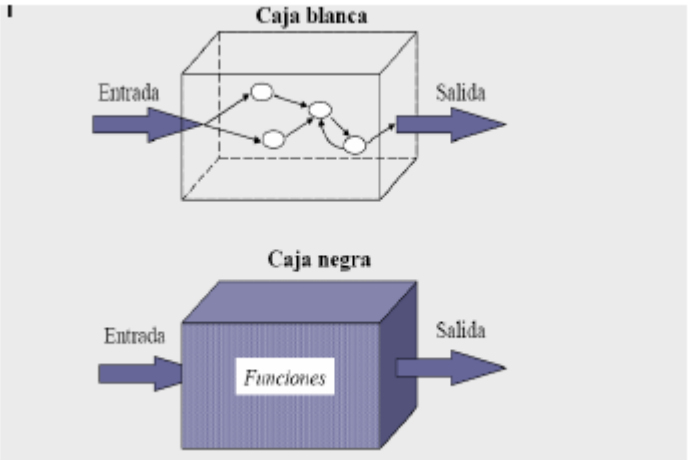

Figura 15. Métodos de prueba (Saucedo, 2007).

3.1 Pruebas de Caja blanca

Es necesario, para garantizar la calidad del software, iniciar el proceso de pruebas sobre las partes críticas del código fuente del mismo, evitando así que la aplicación falle solucionando los errores detectados. Para ello es necesario realizar pruebas de Caja blanca al módulo desarrollado, seleccionando una estrategia que permita ejercitar las decisiones lógicas, ejecutar los bucles, ejecutar las estructuras de datos internos, entre otros.

Capítulo 3

La selección de una herramienta para la automatización de las pruebas es un punto necesario para concebir una estrategia, esta debe minimizar el tiempo de prueba sin que afecte la calidad de la misma, además de registrar de forma legible los resultados obtenidos.

3.1.1 Pruebas unitarias. Utilizando Symfony

El framework utilizado en el desarrollo del producto es Symfony, el mismo trae incorporado otro framework para la realización automática de pruebas llamado Lime, este se apoya en la librería Test::More de Perl y es compatible con TAP, esto permite que los resultados obtenidos se registren con el formato definido en el "Test Anything Protocol", facilitando la lecturas de los mismos. Este framework posee las siguientes ventajas (Daniel, 2007):

- Ejecuta los archivos de prueba en un entorno independiente para evitar interferencias entre las diferentes pruebas.
- Las pruebas de Lime son fáciles de leer y sus resultados también lo son. En los sistemas operativos que lo soportan, los resultados de Lime utilizan diferentes colores para mostrar de forma clara la información más importante
- Symfony utiliza Lime para sus propias pruebas y su “regression testing”, por lo que el código fuente de Symfony incluye muchos ejemplos reales de pruebas unitarias y funcionales.
- Está escrito con PHP, es muy rápido y está bien diseñado internamente. Consta únicamente de un archivo, llamado lime.php, y no tiene ninguna dependencia.

3.1.2 Métodos que se probaron

Para realizar las pruebas se seleccionaron los métodos más importantes del sistema, y así garantizar que su funcionamiento sea correcto. Para ello se seleccionaron algunos parámetros previamente y se probó su respuesta al ser llamado con los mismos.

A continuación se listan las clases con los respectivos métodos que se comprobaron mediante este tipo de pruebas:

- SearchResultPeer
    - countResults(): Integer
    - getFirst(): Integer
    - getAll(): Array
- sfArrayPager

  - Page(): Void
  - countPages(): Integer
  - getFirst(): Page
- Action (módulo de búsqueda “Search”)
  - addFieldValues(): Void
  - indexAll(): Void
  - indexOA(): Void

En cada uno de los casos seleccionados se validó que los resultados obtenidos cumpliesen con los tipos de datos especificados, así como las respuestas ante situaciones excepcionales, validando que las acciones ante éstas sean las adecuadas en cada caso.

A modo de ejemplo, los dos casos de prueba o juegos de datos previstos para el método SearchResultPeer::countResults se incluyen en el archivo test/unit/userTest.php, probándose de la siguiente manera.

```
sergio@uci-23e62871f8a:php symfony test-unit user
1..2
#  sfArrayPager::count::Pages
ok 1 - Este metodo debuelve un Entero
#     Failed test (./lib/symfony/vendor/lime/lime.php at line 103)
#            got: 2
#       expected: '3'
 Looks like you failed 1 tests of 2.
Violación de segmento
sergio@uci-23e62871f8a:
```

Figura 16. Prueba unitaria con Lime.

3.1.3 Iteraciones

Se realizaron tres pruebas después de configuradas las pruebas en el framework, según los casos listados, las dos primeras proyectaron condiciones adversas en diversos métodos, en los próximos subacápites se muestran los resultados de las pruebas efectuadas.

Primera iteración

En la primera iteración realizada se obtuvieron los resultados que se muestran en la tabla 4.

Métodos	Cantidad de juegos de datos	Salidas correctas	Salidas erróneas
SearchResultPeer::countResults	3	2	1
SearchResultPeer:: getFirst	2	2	0
SearchResultPeer::getAll	7	5	2
sfArrayPager::Page	3	3	0
sfArrayPager::countPages	10	7	3
sfArrayPager::getFirst	2	0	2
Action::addFieldValues	1	1	0
Action::indexAll	12	11	1
Action::indexOA	10	8	2
Total	50	39	11

Tabla 4. Primera iteración.

En esta primera iteración se detectaron 6 métodos para los que existían entradas que no producían los resultados deseados, por lo que se realizaron las correcciones pertinentes en un intervalo de 12 horas y luego se procedió a pasar a una próxima iteración en busca de nuevas dificultades. Los casos de prueba se conservan con la misma estructura para garantizar la corrección de los errores antes detectados.

Segunda iteración

Métodos	Cantidad de juegos de datos	Salidas correctas	Salidas erróneas
SearchResultPeer::countResults	3	3	0
SearchResultPeer:: getFirst	2	2	0
SearchResultPeer::getAll	7	7	0
sfPropelArrayPager::Page	3	3	0

sfPropelArrayPager::countPages	10	9	1
sfPropelArrayPager::getFirst	2	1	1
Action::addFieldValues	1	1	0
Action::indexAll	12	12	0
Action::indexOA	10	9	1
Total	50	47	3

Tabla 5. Segunda iteración.

La segunda iteración de pruebas unitarias arroja tres métodos con resultados no deseados, se procede a una tercera iteración de prueba y los casos de prueba continúan con la misma estructura por lo antes descrito.

Tercera iteración

Métodos	Cantidad de juegos de datos	Salidas correctas	Salidas erróneas
SearchResultPeer::countResults	3	3	0
SearchResultPeer:: getFirst	2	2	0
SearchResultPeer::getAll	7	7	0
sfPropelArrayPager::Page	3	3	0
sfPropelArrayPager::countPages	10	10	0
sfPropelArrayPager::getFirst	2	2	0
Action::addFieldValues	1	1	0
Action::indexAll	12	12	0
Action::indexOA	10	10	0
Total	50	50	0

Tabla 6. Tercera iteración.

Capítulo 3

Al concluir los tres ciclos de pruebas unitarias realizados con el framework Lime, se concluye que estas funcionalidades están listas para cumplir las tareas para las que fueron realizados. Este mismo procedimiento fue ejecutado para el resto de las funcionalidades más complejas del sistema.

3.2 Pruebas de Caja negra

Las pruebas de Caja negra se realizan sobre las interfaces de usuario; con el fin de certificar que las funcionalidades sean operativas. Estas se orientan en el exterior del módulo, sin tener en cuenta el código, están dirigidas a encontrar fallas tales como (Mañas, 1994):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Entre las técnicas de realización de pruebas de Caja negra se encuentran (Pressman, 2005): Técnica del Análisis de Valores Límites, Técnica de Grafos de Causa-Efecto y la Técnica de la Partición de Equivalencia.

Para la realización de las pruebas se seleccionó la Técnica de la Partición de Equivalencia, esta divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software, reduciendo el posible conjunto de casos de prueba en uno más pequeño (un conjunto tratable que evalúe bien la aplicación), además posee una documentación con vasto desarrollo. Esta técnica se divide en dos puntos fundamentales: la Identificación de las clases de equivalencia y la Descripción de los casos de pruebas, los cuales son descritos a continuación.

3.2.1 Identificar las clases de equivalencia

La partición equivalente va dirigida a los casos de pruebas que encuentren clases de errores, disminuyendo la cantidad de casos de prueba que hay que ejecutar. Las clases de equivalencia son un grupo de situaciones aceptadas o no aceptadas para condiciones de entrada. Normalmente, una condición de entrada es un número específico, un rango de valores, un grupo de valores vinculados o una circunstancia lógica (Pressman, 2005).

Para identificar las clases de equivalencia, se toma en cuenta un dominio de valores, que respondan a un funcionamiento correcto del caso de prueba y otro conjunto que se encuentre fuera de ese rango, con la intención de que devuelva el error especificado, como se muestra en el breve ejemplo de la tabla 8, para el caso de uso: “Realizar búsqueda”.

Condición externa	Clases de equivalencia válida	Clases de equivalencia no válidas
El campo es obligatorio.	Cualquier valor.	Dejar el campo vacío (valor nulo).

Tabla 7. Clases de equivalencia.

Para la selección de las clases de equivalencia se tienen en cuenta los siguientes criterios (Mañas, 1994):

- Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen tres clases de equivalencia: por debajo, en y por encima del rango.
- Si una entrada requiere un valor concreto, aparecen tres clases de equivalencia: por debajo, en y por encima del rango.
- Si una entrada requiere un valor de entre los de un conjunto, aparecen dos clases de equivalencia: en el conjunto o fuera de él.
- Si una entrada es booleana, hay dos clases: sí o no.
- Los mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases.

Luego de la selección de las clases de equivalencia se pasa al siguiente paso de la técnica seleccionada, la descripción de los casos de pruebas.

3.2.2 Descripción de los casos de pruebas

Para desarrollar software de calidad y libre de errores, los casos de prueba son muy importantes. Un caso de prueba bien diseñado tiene gran posibilidad de llegar a resultados más fiables y eficientes, mejorar el rendimiento del sistema, y reducir los costos en tres categorías: (Perry, 1995).

Productividad, menos tiempo para escribir y mantener los casos

Capacidad de prueba, menos tiempo para ejecutarlo

Programar la fiabilidad, estimaciones más fiables y efectivas

Por cada caso de uso se realizó un caso de prueba, para identificar los escenarios de pruebas se hizo un minucioso estudio de las descripciones de dichos casos de uso. Una vez identificados los elementos que complementan los escenarios de pruebas se procede a la confección de la Matriz de Datos para los diferentes casos de uso.

En la descripción de cada uno de los casos de prueba identificados se muestran los resultados esperados del mismo, con el objetivo de ser comparado con el resultado real, cuando ambos coincidan se considera que la prueba fue realizada con éxito.

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**
SC 1 Realizar búsqueda	EC 1.1 Realizar búsqueda general de OA	El sistema brinda la opción de realizar una búsqueda general de OA.
	EC 1.2 Campo vacío	En caso de que el invitado no introduzca ningún valor, el sistema emite un mensaje “Debe introducir la frase a buscar”.

Tabla 8. Escenarios para el caso de uso “Realizar búsqueda”.

Escenario	**Variable 1**	**Respuesta del sistema**	**Resultado de la prueba**	**Flujo central**
EC 1.1 Realizar búsqueda general de OA	V	Realiza una búsqueda general en los OA del	Satisfactorio	Click en Texto a buscar. Introducir la frase. Click en

		sistema.		el botón Buscar.
EC 1.2 Campo vacío	NA	El sistema emite un mensaje: "Debe introducir la frase a buscar"	Satisfactorio	

Tabla 9. Matriz de datos.

El equipo de desarrollo realiza una próxima iteración, para dar respuesta a las pruebas que arrojaron errores, así será sucesivamente hasta que no se detecten fallas y el sistema sea liberado. Los demás casos de prueba y matrices de datos se encuentran en los anexos.

3.2.3 Resultados de las pruebas de Caja Negra

Se realizaron pruebas al sistema por parte del equipo de calidad del proyecto, además de las descritas anteriormente. Estas pruebas constaron de cuatro iteraciones hasta que el producto fue liberado. Los resultados obtenidos en las iteraciones fueron descritos en la tabla 10, en cada iteración se muestra la cantidad de No conformidades, las que se le dieron respuesta (Cerradas) y las que por un motivo justificado no pueden ser resueltas (No proceden).

Iteración	No Conformidades	Cerradas(s)	No procede(n)
1ra	16	14	2
2da	8	6	2
3ra	4	4	0
4ta	0	-	-

Tabla 10. Resultado de las pruebas funcionales.

Las no conformidades detectadas en la primera y segunda iteración que no proceden no afectan el resultado de la propuesta de solución, ya que están fuera del alcance del campo de acción y se tienen en cuenta en las recomendaciones para la proposición de una posterior investigación.

Capítulo 3

Conclusiones parciales

Las pruebas unitarias arrojaron resultados relevantes, ya que permitieron validar el código fuente desarrollado durante la etapa de implementación del sistema, facilitando además la corrección de los errores identificados.

Se seleccionaron un grupo de clases para ser probadas por incluir implementaciones concretas y que además involucran las funcionalidades más importantes de la aplicación; permitiéndonos corroborar el correcto funcionamiento de las mismas y lograr un producto con calidad.

Se utilizó el framework Line para la automatización de las pruebas unitarias y para la realización de las pruebas funcionales se utilizaron los métodos implementados por el grupo de calidad del proyecto. Se mostraron ejemplos de los artefactos generados; mostrando como se realizó el proceso general, y los resultados obtenidos.

Conclusiones y recomendaciones

Conclusiones

Se cumplieron los objetivos trazados en la investigación, y los resultados fueron aplicados en el ámbito esperado inicialmente, arribando a las siguientes conclusiones:

- Se generaron los artefactos fundamentales de los flujos de implementación y pruebas que establece RUP.
- Se implementó el módulo de búsqueda en el Repositorio de Objetos de Aprendizaje RHODA con la utilización de Lucene como motor de búsqueda.
- Para el proceso de recuperación de Objetos de Aprendizaje se utilizó el modelo vectorial y el índice invertido para la indexación, lo que contribuye a una mayor precisión en el resultado de las búsquedas.
- El módulo de búsqueda implementado es configurable por los usuarios desde una interfaz de configuración.
- Se realizaron las pruebas de Caja Blanca y Caja Negra corroborando que el mismo cumple con todos los requisitos trazados.

Recomendaciones

- Realizar un estudio de posibles nuevas configuraciones para la indexación de nuevos formatos en los procesos de búsqueda.
- Valorar el uso de tesauros como método de expansión de consultas en el proceso de búsqueda y recuperación de Objetos de Aprendizaje.

Bibliografía

Bibliografía

2011. AltaVista. [En línea] 2011. [Citado el: 2011 de 1 de 23.] http://es.altavista.com/help/search/help_adv.

Cañizares, Roxana. 2008.*Framework para la gestión de contenidos educativos. Trabajo de Diploma en opción al título de Ingeniero en Ciencias Informáticas.* La Habana : s.n., 2008.

Coles, Michael. 2008.*Pro Full-Text Search in SQL Server 2008 (Version 1 ed.).* s.l. : Apress Publishing Company, 2008. ISBN 1-430-21594-1..

2011. Conferencia #1. Introducción a la Ingeniería de Software. ISW I. [En línea] 2011. [Citado el: 18 de 1 de 2011.] http://eva.uci.cu.

2010. DocShare. [En línea] 2010. [Citado el: 17 de 1 de 2011.] http://www.docshare.es/gestion.php.

Durant, Aliocha. 2008.*Arquitectura de un Repositorio de Objetos de Aprendizaje. Trabajo de Diploma en opción al título de Ingeniero en Ciencias Informáticas.* La Habana : s.n., 2008.

2011. Google. Ayuda. [En línea] 2011. [Citado el: 2011 de 1 de 23.] http://www.google.com/support/websearch/bin/answer.py?hl=es&answer=35890.

Guzmán, Clara López. 2005.*Los Repositorios de Objetos de Aprendizaje como soporte para entornos e-learning.* Universidad de Salamanca. España : s.n., 2005. Tesis de grado.

2011. IMS Digital Repositories Interoperability. [En línea] 2011. [Citado el: 22 de 1 de 2011.] http://www.imsglobal.org.

Larman, Craig. 2003.*UML y Patrones.* 2003.

Leiva, Isidoro Gil. 2008.*Manual de indización. Teoría y práctica.* Gijón : Trea, 2008. ISBN: 978-84-9704-367-0.

Meersman, R. 2004.*On the move to meaningful internet systems 2004: OTM 2004 Workshops.* 2004.

Molina, María Pinto. 2009. Electronic Content Management Skills. [En línea] 13 de 2 de 2009. [Citado el: 17 de 1 de 2011.] http://www.mariapinto.es/e-coms/index.htm.

2009. Posicionamiento Web. [En línea] 2009. [Citado el: 2011 de 1 de 23.] http://www.posicionamiento-web.org/news/historia_de_altavista/2009-09-05-21.

2011. PostgresSQL. [En línea] 2011. [Citado el: 18 de 1 de 2011.] http://www.postgresql.org/docs/8.4/static/textsearch-dictionaries.html.

Bibliografía

Pressman, Roger S. 2005.*Ingeniería de Software. Un enfoque práctico 5ta edición.* 2005.

Rumbaugh, James, Booch, Grady y Jacobson, Ivar. 2000.*El Proceso Unificado de Desarrollo de Software.* 2000.

Salvador Broche, Orlando y Soler Martín, Javier. 2010.*Implementación de un Repositorio de Objetos de Aprendizaje para la Universidad de las Ciencias Informáticas.* Ciudad de la Habana : s.n., 2010.

2011. Sphinx. [En línea] 2011. [Citado el: 18 de 1 de 2011.] http://www.sphinx-metal.net.

Tramullas, Jesús y Kronos. 2007. Introducción a la Documática. [En línea] 2007. [Citado el: 17 de Enero de 2011.] http://tramullas.com/documatica/3-1.html.

Wheeler, David A. 2004.*Software Configuration Management (SCM) System. [En línea] http://www.dwheeler.com.* s.l. : Free Software (OSS/FS), 2004.

Widmayer, Peter y Ottmann, Thomas. 2002.*Algorithmen und Datenstrukturen (4th ed.).* s.l. : Spektrum Akademischer Verlag, 2002. ISBN 3-8274-1029-0..

2011. WikiSchoenstatt. [En línea] 2011. [Citado el: 23 de 1 de 2011.] http://wikischoenstatt.org/Wiki.

Zaninotto, Francois y Potencier, Fabien. 2005.*Symfony, la guía definitiva.* 2005.