Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología en Internet

Elaboración de un Prototipo de Buscador de Documentos Académicos de la Facultad de Ciencias.

Trabajo Especial de Grado
presentado ante la ilustre
Universidad Central de Venezuela
por el Bachiller

Jorge Ignacio Sánchez Echevarría
C.I: 14.909.483

para optar al título de
Licenciado en Computación

Tutor
Prof. Sergio Rivas

Caracas, Enero / 2008.

Acta

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el **Bachiller Jorge Ignacio Sánchez Echevarría**, **C.I. 14.909.483**, con el título **"Elaboración de Prototipo de Buscador de Documentos Académicos de la Universidad Central de Venezuela"**, a los fines de optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijo el 1 de febrero de 2008 a las 11:00 a.m., para que su autor lo defendiera en forma pública, se hizo en el Centro de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con una nota de puntos, en fe de lo cual se levanta la presente Acta, en Caracas al primer día del mes de febrero del año dos mil ocho, dejándose también constancia de que actuó como Coordinador del Jurado el profesor Sergio Rivas.

Prof. Jossie Zambrano Tutor	Prof. Sergio Rivas Tutor
Prof. Andrés Sanoja Jurado	Prof. Damaris Barrantes Jurado

Agradecimientos y Dedicatoria

Gracias a Dios por darme vida y determinación. A mi familia que en conjunto intervino en mi crecimiento y me formó con valores y principios.

Este Trabajo Especial de Grado lo entrego y se lo dedico a la ilustre Universidad Central de Venezuela que me ha dado la oportunidad de realizarme en esta etapa profesional de mi vida. Muchas gracias a los maestros que le dan prestigio a la Universidad con su vocación y su honorable trabajo.

Muchas gracias a mis amigos y amigas en general, quienes comparten con uno, aportan valiosos puntos de vistas, experiencias y enseñanzas.

Resumen

El presente Trabajo Especial de Grado aborda el tema de la localización de documentos electrónicos con la información solicitada por los usuarios. En el marco conceptual se investigó acerca de la manera en que funcionan los grandes buscadores de Internet, los buscadores mas específicos que operan en una sola computadora y los formatos de documentos electrónicos que éstos manejan. En el marco aplicativo se desarrolló un repositorio y un buscador Web de documentos electrónicos, los cuales están diseñados para albergar a los trabajos producidos por los estudiantes de la Facultad de Ciencias de la Universidad Central de Venezuela.

El sistema desarrollado en el marco aplicativo se llama BUSCONEST, el cual es a su vez un subsistema de CONEST dedicado exclusivamente a los documentos digitales relacionados con trabajos académicos. Durante el desarrollo de la aplicación se emplearon planteamientos y soluciones innovadoras para los requerimientos específicos. Entre los principales aspectos del desarrollo de BUSCONEST tenemos el algoritmo de ordenamiento, la implementación del manejo de la persistencia, la ejecución de las consultas, la integración con CONEST y la incrementabilidad del sistema entre otros.

Contactos

- **Jorge Sánchez:** jorge.ucv@gmail.com
- **Sergio Rivas:** srivas@gmail.com
- **Jossie Zambrano:** jossie.zambrano@gmail.com

Índice general

Índice de figuras

Índice de Tablas

Parte I

Introducción y Propuesta

Introducción

La informática aporta una diversidad de aspectos y ventajas de los cuales nos beneficiamos mucho hoy en día. A medida que avanza el tiempo, son más grandes los volúmenes de datos e información de los que podemos disponer, ya sea en nuestras computadoras personales o en la red Internet.

Tanta diversidad y crecimiento de las tecnologías ha producido una especie de anarquía a la hora de ubicar algún recurso informático, esto a su vez ha obligado a realizar varios trabajos de investigación para lograr otorgar orden y precisión en las búsquedas y localización de la información que se solicita. Este trabajo de investigación se centra en la información que está contenida en los documentos electrónicos.

Por otro lado ante la diversidad y libertad, se tiene el hecho de que se publica o se elabora cualquier tipo de contenido. Por eso se hace mucho énfasis en la calidad de los resultados de las búsquedas, en los filtros, en las correctas disposiciones físicas y lógicas de la información para garantizar que el usuario sea beneficiado con lo que realmente quiere y necesita.

Propuesta de Trabajo Especial de Grado

Se presenta a continuación la propuesta del Trabajo Especial de Grado, la cual habla sobre la problemática de la obtención de información, la necesidad de que las consultas sean confiables y acertadas para referencias y material para los interesados. Se plantea la creación de un repositorio digital de documentos electrónicos y un mecanismo de búsqueda inteligente que permita especificar criterios útiles y variados.

Descripción del Problema

En la actualidad los procesos de búsqueda de información se están haciendo cada vez más por medios digitales y automatizados. Un caso que se puede referenciar es la búsqueda de información por Internet en contraposición a dirigirse a la biblioteca a consultar un libro o publicación (en este contexto se denominará publicación a todo trabajo académico de los estudiantes que haya sido evaluado y esté avalado por ésta casa de estudios específicamente la Universidad Central de Venezuela).

Las tendencia de buscar información por Internet es casi inevitable, esto se debe a las características de los medios digitales con respecto a los libros convencionales tratados en el capítulo 2. También se debe a la serie de dificultades a la que se enfrentan los estudiantes o cualquier interesado actualmente cuando les toca hacer consultas por las vías tradicionales.

Respecto a las publicaciones de los estudiantes como trabajos especiales de grado, seminarios y pasantías; solamente los trabajos especiales de grado se encuentran actualmente guardados en físico en la biblioteca de la Facultad "Alonso Gamero", y además es muy difícil el acceso, muchas veces no se puede disponer de ellos o el préstamo es muy limitado ya sea en tiempo, para sacar copia, etc. Actualmente no existe un repositorio digital de publicaciones.

A la hora de buscar información para trabajos académicos los estudiantes tienen practicamente dos opciones: La primera es buscar la información en las bibliotecas o la segunda, consultar por Internet. De éstas opciones es evidente que se opta más por la consulta de documentos electrónicos por Internet. Esto último puede implicar una serie de inconvenientes entre las cuales podemos mencionar:

- La información no es confiable.

- Se desconoce la fuente.
- La información es limitada o muy breve.
- Cuando hay mucha información suele estar desordenada o sin estructura coherente.

A la hora de buscar documentos tal como están actualmente almacenados en la biblioteca de la Facultad de Ciencias se tienen éstos otros inconvenientes:

- Se tiene un sistema automatizado de búsqueda de los documentos limitado solo al título de la publicación a la cota y al autor.
- Hay problemas de disponibilidad del documento físico.
- Las opciones avanzadas de consultas no existen, solo se puede buscar por nombre o título. No buscar en el contenido es una severa limitación, porque es el contenido el objetivo del interesado.
- En caso de conseguir con éxito los documentos con información valiosa, viene el proceso de extraer la información, por ejemplo haría falta recurrir al fotocopiado y afrontar posibles inconvenientes como la disponibilidad de las impresoras.

El mismo avance de la tecnología ha permitido a que hoy en día las publicaciones sean realizados en formatos digitales practicamente en su totalidad. Pero estos documentos digitales una vez que son evaluados y entregados en sus versiones en papel no se puede disponer de ellos en su versión digital que tantos beneficios aportaría.

Para hacer las consultas más completas y acertadas se dispone de una aplicación web que automatiza los procesos de Control de Estudios en la Facultad de Ciencias que se llama CONEST. Este sistema almacena toda la información correspondiente a los estudiantes, sus historiales, sus calificaciones, los docentes, las aulas, etc. En toda esa información se encuentra algo muy importante para éste problema: Datos acerca de los trabajos académicos de los estudiantes durante el proceso de grado.

A pesar de que en CONEST se tiene información de los trabajos, como el título y la calificación, es curioso el hecho de que no se tenga almacenado al documento digital propiamente dicho. estos datos de los estudiantes relacionados a sus publicaciones son un valioso aporte porque permite tener más información del documento en sí; con esto último estaríamos disponiendo de más criterios ya sea para consultar o para posicionar más acertadamente un documento en el resultado de las consultas.

La propuesta consiste en implementar un repositorio digital de publicaciones académicas en donde se almacenen los archivos de documentos digitales una vez aprobados por las autoridades académicas, para luego ser consultados mediante un buscador Web automatizado basado en el estudio realizado en el capítulo 1.

En este trabajo se propone implementar una solución efectiva y eficiente, implementando búsquedas inteligentes, ofreciendo criterios importantes al interesado que realiza las consultas. También se requiere de más información y criterios para la elaboración del algoritmo de búsqueda y ordenamiento, ya que al haber más información útil, se dispone de más elementos y datos para mejores resultados de las consultas.

Lo anterior se puede lograr mediante la integración con el sistema CONEST. De esta forma se enriquece la información que esta presente en el documento digital y por la tanto se pueden hacer mejores búsquedas.

Para poder otorgarle a la publicación información valiosa y bien estructurada, se recurre al uso del almacenamiento de una metadata relacionada, la cual puede ser entendida y procesada por personas pero principalmente por el algoritmo de búsqueda. Esta idea esta inspirada en el concepto de Web Semántica tratado brevemente en el capitulo 1. El contenido de la metadata es una información variada obtenida ya sea por simple lectura o por procesos de cálculos, con la idea de que a la hora de hacer las consultas, éstas se realicen sin necesidad de repetir estos procesos de obtención de información accediendo directamente hacia la metadata. Otro aspecto importante de la metadata es que es flexible a la hora de expandir su estructura la cual es posible validar.

La solución de estos problemas hace uso de las tecnologías actuales para disipar los inconvenientes que representan los sistemas de búsqueda de información ya sea por Internet o por la actual biblioteca de la Facultad de Ciencias. Se trata de implementar un buscador y repositorio de publicaciones aprovechando que la realidad tecnológica actual hace que se disponga digitalmente de los trabajos formales y evaluados.

Con esta solución superamos los inconvenientes de los dos metodos de búsqueda más utilizados que han sido tratados, obteniendo lo siguientes beneficios:

- La información es confiable ya que el repositorio cuenta con documentos avalados por la Universidad.
- Se conoce la fuente, ya que la información del trabajo, del estudiante y otros detalles están almacenados.
- La información es estructurada, ordenada y coherente porque esas son características que exigen las autoridades académicas.
- Siempre habrá disponibilidad de los documentos digitales debido a las características de los mismos tratados en el capítulo 2.
- Se tiene un sistema automatizado de consultas en el que es posible las opciones avanzadas. A parte de buscar por nombre o título, se pueden hacer consultas por varios criterios, como palabras dentro del contenido, fecha de publicación, criterios de la metadata del documento, etc.

- Debido a la integración con CONEST se tiene más información valiosa y por lo tanto mejorará la calidad de los resultados de las consultas.
- Una vez que el estudiante obtiene el documento mediante la descarga en su computadora, puede extraer su información sin las limitaciones de un trabajo en físico.

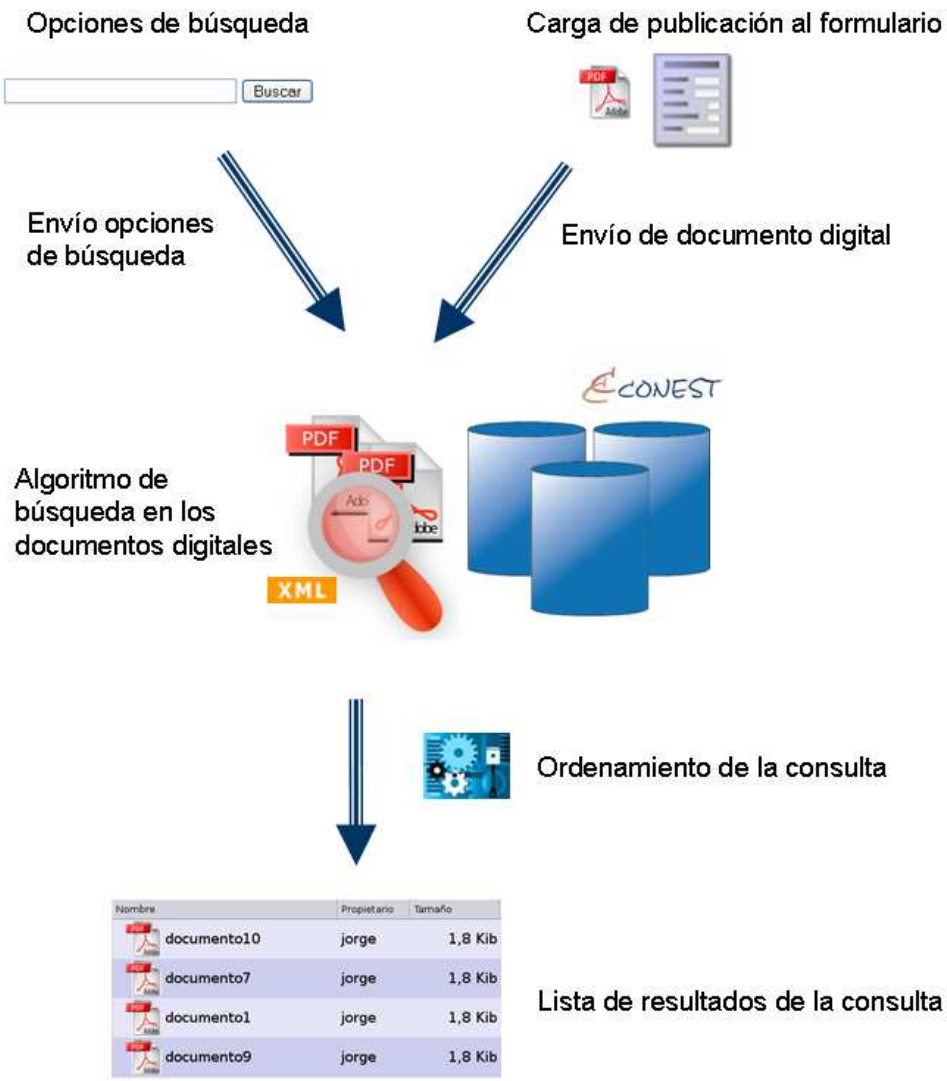

Figura 1: Propuesta: Repositorio digital

Actualmente se está discutiendo la estrategia de publicación de estos documentos para respetar los derechos de autor de los estudiantes. Se pretende que el repositorio virtual se rija por las normas y reglamentos actuales para usuarios de los servicios de la biblioteca "Alonso Gamero" hasta donde puedan ser aplicados a las publicaciones digitales.

Objetivo General

Desarrollar una aplicación Web integrada con CONEST que administre y soporte el repositorio de publicaciones digitales de la comunidad de la Facultad de Ciencias, realizadas y avaladas durante el desarrollo del componente profesional de los estudiantes en las licenciaturas. Ofreciendo criterios de búsqueda que aseguren la obtención de resultados acertados en las consultas.

Objetivos Específicos

- Analizar los procesos y procedimientos de un repositorio de documentos electrónicos, ventajas y desventajas, que permita conocer con alto nivel de detalle la forma de ejecución de cada uno de ellos para determinar de una manera clara y específica los requerimientos.
- Estudio minucioso de la plataforma tecnológica existente empleada por CONEST.
- Adaptar el proceso XP para el desarrollo de una Aplicación Web que administre el repositorio de publicaciones digitales.
- Diseño Físico y Lógico de la Base de Datos que permita almacenar la información relacionada a la problemática. planteada y el guardado de archivos binarios.
- Elaborar el diseño de las interfaces gráficas de usuario (GUI).
- Realizar pruebas.
- Desarrollar un módulo para la consulta de los documentos electrónicos.
- Desarrollar un módulo para la carga de los documentos y validaciones.
- Someter a pruebas rigurosas la aplicación Web.

Proceso de Desarrollo

El proceso de desarrollo a utilizar será Programación Extrema(Extreme Programming) o XP el cual es un proceso de desarrollo ligero que reduce la complejidad del software por medio del trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción.

Tecnologías y Plataformas

Durante el desarrollo se utilizarán las mismas tecnologías que utiliza CONEST ya que el repositorio y sistema buscador estará integrado con éste y las tecnologías de CONEST han demostrado ser una implementación exitosa. las tecnologías usadas por CONEST son de aplicaciones Web, todas ellas *open source* y son tratadas a continuación:

- Lenguaje de programación `Ruby 1.8.5`: Este es un lenguaje de programación sencillo, verdaderamente orientado a objetos, interpretado e independiente de la plataforma. Es muy flexible y dinámico, permite la metaprogramación. A pesar de su sencillez y naturalidad es un lenguaje poderoso que otorga muchas posibilidades a los programadores.

- Framework Web `Rails 1.2`: Es un framework de aplicaciones Web de código abierto, sigue el paradigma de la arquitectura Modelo Vista Controlador (MVC). con este framework se desarrollan aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

- Sistema manejador de base de datos `MySQL Server 5.0`: Es un manejador de base de datos de código abierto. Funciona sobre múltiples plataformas, es bastante rápido que permite un buen rendimiento del hardware gracias a su implementación multihilo, lo cual optimiza los tiempos de respuesta. Es multiplataforma, con una escalabilidad y limites bastante altos para lo que el sistema a desarrollar requiere y ofrece en esta versión el manejo de todas las características requeridas como transacciones, triggers, store procedure, índices, y buen manejo de seguridad entre otras.

Para la ejecución de la aplicación solo es necesario disponer un navegador Web sobre cualquier plataforma.

Parte II

Marco Conceptual

Capítulo 1

Estrategias de Búsqueda en la Web

Este capítulo trata sobre los buscadores de páginas Web. Se hará un estudio de los distintos tipos de buscadores, las ventajas y desventajas de cada uno. Se explicará con mayor detalle el funcionamiento de los motores de búsqueda los cuales son los más útiles y eficaces en la actualidad, analizando cada uno de sus componentes principales. También se hace un breve viaje por el transcurso de la historia de los buscadores.

1.1. Buscadores en Internet

Uno de los usos más extendidos de Internet es la búsqueda de información útil para el usuario. Sin embargo, su localización no resulta siempre una tarea fácil debido a la gran cantidad de datos existentes en la red.

Internet es una extensa red de documentos de múltiples formatos, desde páginas Web en HTML a documentos de distintos tipos como puedan ser textos, imágenes, archivos de sonido o de video, etc. Cuando se necesita buscar cierta información en Internet lo más eficaz es utilizar un buscador.

En nuestro contexto un *buscador* es una herramienta que permite al usuario encontrar un documento electrónico que contenga la información que éste le especifique, ya sea por palabras claves o a través de una serie de pasos.

Como otra definición alternativa tenemos que los buscadores son aquellos sistemas automáticos de recuperación de información que almacenan información sobre páginas Web en sus bases de datos, la cual es consultada por los usuarios a través de interfaces y formularios Web.

Cuando queremos información sobre algún tema las búsquedas se hacen con palabras clave o con árboles jerárquicos por temas; el resultado de la búsqueda es un listado de direcciones Web en los que se mencionan temas relacionados con los criterios solicitados. Algunos

buscadores revisan no sólo en la Web sino además en News, Gopher, FTP, etc.

1.2. Clasificación de los Buscadores

Básicamente existen dos tipos de buscadores: los índices temáticos y los motores de búsqueda. Los otros tipos de buscadores son combinaciones o variaciones de éstos.

1.2.1. Índices Temáticos o Directorios

Son sistemas de búsqueda por temas o categorías jerarquizados. Se trata de agrupamientos y clasificaciones de direcciones Web elaboradas "manualmente", es decir, hay personas que se encargan de asignar cada página Web a una categoría o tema determinado.

Son una barata tecnología y no se requieren muchos recursos de informática en comparación con los motores de búsqueda. En cambio, se necesita más soporte humano y mantenimiento.

Figura 1.1: Directorio venezolano Auyantepui

A continuación sus características:

- Construidos por selección humana. No por computadores o programas automatizados.

- Organizados por categorías de temas, clasificación de páginas por temas. Los temas no están estandarizados y varían de acuerdo con el alcance de cada directorio.
- Nunca contienen los textos completos de las páginas de la Red que enlazan. Usted solamente puede buscar lo que ve (títulos, descripciones, categorías de temas, etc.). Utiliza términos amplios y generales.
- Los hay pequeños y especializados hasta muy amplios, pero son más pequeño que la mayoría de los motores de búsqueda. Los hay en un amplio rango de tamaños.
- Generalmente evaluados y anotados cuidadosamente (¡pero no siempre!).

Ventajas de los Índices Temáticos o Directorios

- La clasificación es resultado de la selección humana, logrando que lo que se obtiene sea pertinente según la clasificación tomada por el directorio.
- Organizan los directorios en forma jerárquica en categorías y subcategorías por materia.
- Son capaces de entregar una mayor calidad de contenido.
- Son la mejor opción para revisar y buscar temas generales (páginas amarillas).
- La navegación por índices temáticos puede sugerir nuevas ideas de búsqueda que no estaban previstas.

Desventajas de los Índices Temáticos o Directorios

- Luego de incluida en el directorio, la página de la Red podría cambiar su contenido sin que los editores lo percibieran.
- Puede suceder que las jerarquías de categorías, a veces muy complejas, desorienten al usuario.
- Los documentos que catalogan no están evaluados.

1.2.2. Motores de Búsqueda (Search engines)

La mayor parte de las veces buscamos temas muy específicos o con ciertas características que no pueden ser localizados o categorizados en directorio Web. Para estos casos se recurre a los motores de búsqueda que nos permiten hacer consultas en la Web según nuestros criterios y especificaciones.

Un motor de búsqueda se podría definir como una herramienta que basa su funcionamiento en *palabras clave*, que tiene el objetivo de recoger información y generar bases de

datos con índices a los recursos y paginas Web disponibles en Internet de manera automatizada.

Figura 1.2: Motor de búsqueda Google

Las *palabras clave* son aquellos términos que utilizamos para describir los conceptos o ideas que buscamos. No sólo pueden ser palabras singulares sino también secuencias de caracteres que sirvan para realizar nuestra búsqueda. Estas palabras están normalmente separadas por espacios en blanco. No se deben incluir signos diferentes a los alfanuméricos, a no ser que representen alguna función especial ya que la mayoría de los buscadores no los tendrán en cuenta.

Es habitual que cuando se realiza una búsqueda por palabras encontremos un elevado volumen de páginas que contienen dicha palabra clave. Cuando esto ocurra, hay que concretar más la búsqueda, añadiendo más palabras clave en la interfaz de búsqueda, de modo que se pueda reducir el número de resultados.

A continuación algunas características:

- Constituidos por programas de computador automatizados que tienen varias funciones como los programas que exploran la red (spiders), los que construyen la base de datos.y los que utiliza el usuario, el programa que explora la base de datos, etc. Sin que intervenga el ser humano en la selección.
- No están organizados por categorías de temas. Todas las páginas están clasificadas por un algoritmo de computador.

- Contienen textos completos (cada palabra) de las páginas de la red que enlazan. Usted encontrará páginas utilizando palabras que coinciden con las que están en las páginas que usted desea.
- Capturan con frecuencia gran cantidad de información. Para búsquedas complejas es necesario especificar más
- No evaluados. Ofrecen tanto lo bueno como lo malo en los reultados. El usuario debe evaluar todo lo que encuentre.

Figura 1.3: Motores de Búsqueda populares

El funcionamiento de los motores de búsquedas será explicado más adelante con mayor detalle.

Ventajas de los Motores de Búsqueda

- Realizan búsquedas recorriendo un gran numero de páginas Web, que aumentan exponencialmente.
- Son útiles para realizar búsquedas sobre temas o sitios específicos.
- Páginas más relevantes aparecen al principio de la lista de resultados.
- Actualización permanente de la información [1].
- Hasta ahora son los mejores medios diseñados para las búsquedas en la red.

[1] Actualización permanente en el sentido de que se recorre la Web y cada cierto tiempo se actualiza la información almacenada en la base de datos indexada

Desventajas de los Motores de Búsqueda

- Retornan muchos resultados irrelevantes.
- No son muy precisos, comparado con una búsqueda realizada por una persona.
- Diferentes motores de búsqueda pueden entregar distintos resultados.

1.2.3. Otros Tipos de Buscadores

Como se dijo anteriormente los otros tipos de buscadores son variaciones o combinaciones de los índices temáticos y los motores de búsqueda. Entre estos podemos enumerar los siguientes:

- **Buscadores de Portal:** Bajo este título, englobamos los buscadores específicos de sitio. Aquellos que buscan información solo en su portal o sitio Web.
- **Los Sistemas Mixtos:** Además de tener características de buscadores, presentan las páginas Web registradas en catálogos sobre contenidos. Informática, cultura, sociedad. Que a su vez se dividen en subsecciones.
- **Metabuscadores:** En realidad, no son buscadores. Lo que hacen, es realizar búsquedas en auténticos buscadores, analizan los resultados de la página, y presentan sus propios resultados.
- **Multibuscadores:** Permite lanzar varias búsquedas en motores seleccionados respetando el formato original de los buscadores.
- **FFA**[2]**:** Página de enlaces gratuitos para todos. Cualquiera puede inscribir su página durante un tiempo limitado en estos pequeños directorios. Los enlaces, no son permanentes.
- **Buscadores Verticales:** Buscadores especializados en un sector concreto, lo que les permite analizar la información con mayor profundidad, disponer de resultados más actualizados y ofrecer al usuario herramientas de búsqueda avanzadas.

1.3. Historia de los Buscadores

Una vez que comenzó la Web a tener contenido, en junio de 1993 y desde el MIT (con Matthew Gray a la cabeza), se desarrolló World Wide Web Wanderer, un robot[3] de búsqueda creado en Perl que pretendía medir el tamaño de la red. Ese robot se amplió pudiendo leer direcciones URL creándose así Wandex, el que se podría considerar el primer buscador

[2]FFA: Free For All

[3]Robot o araña: Software autómata que recorre la Web en busca de datos y que avanza su recorrido basándose en los hipervínculos y partiendo de unas páginas iniciales

de Internet, y que tuvo grandes problemas de infraestructura y velocidad.

El siguiente buscador (quizá mejor llamarlo directorio) fue Aliweb (Archie Like Indexing on the Web), también apareció en octubre de 1993 y todavía está en marcha. Creado por Martijn Koster, lo que hacía era indexar los meta-tags de las páginas que se le daban a su índice, es decir, que no tenía un robot de búsqueda que consumiera gran cantidad de ancho de banda como Wandex.

Tras estos primeros procesos de rastreo en la red, Martijn Koster propuso unas sugerencias para lo que sería el archivo robots.txt que limita la acción de los robots de búsqueda en los sitios Web

Aquí comenzaron a desarrollarse los primeros robots (arañas) como Jumpstation que indexaba el título, URL y cabecera del sitio, al igual que World Wide Web Worm, creado por Oliver Mc.Bryan en 1994 (y comprado en 1998 por Goto.com) que funcionaba de la misma manera. Aunque era interesante que indexaran, el problema de estos motores era la forma de mostrar resultados, ya que lo hacían sin aplicar ningún algoritmo, simplemente mostrando los resultados según la fecha de indexación. Más adelante, en diciembre, también lo hizo así el RBSE (Repository-Based Software Engineering) comenzando a aplicar un primer ranking[4] en base a la relevancia de la palabra dada.

De forma paralela iban apareciendo algunos directorios como EINet Galaxy, que en enero de 1994 podría considerarse el primer directorio tal y como los conocemos en la actualidad. Pero, fue en abril de 1994 cuando David Filo y Jerry Yang crearon Yahoo! (anteriormente conocido como Jerry"s Guide to the World Wide Web), una colección de las páginas Web favoritas. El gran problema de Yahoo! era que comenzó siendo un directorio hecho por personas y eso llevaba mucho tiempo, por lo que tuvo que evolucionar incorporando un buscador para ese directorio

Hasta aquí podríamos hablar de la versión "beta" de los motores de búsqueda, la versión que no era aún ni la primera y en la que todo eran experimentos. Así hasta que el 20 de abril de 1994 Brian Pinkerton, desde la Universidad de Washington, presentase WebCrawler. En realidad este buscador era de escritorio y nació el 27 de enero de 1994 pero en 3 meses se convirtió en un robot de la red. La gran diferencia de lo que podríamos llamar la versión 1 de los buscadores era que indexaba las páginas de forma completa y que buscaba información en ellas, al contrario de sus antecesores, que sólo buscaban en la dirección Web, título o metas. Esto hizo que la relevancia de los resultados fuera mucho mayor. Además, tenía la peculiaridad de poder ver las búsquedas en tiempo real con su Webcrawler Search Voyeur.

InfoSeek también apareció a principios de 1994, y aunque no llegó a tener nada especial, tuvo un gran salto en diciembre de 1995 cuando fue el motor de búsqueda por defecto en Netscape.

[4]Ranking: Proceso de posicionamiento

Tras la aparición de Webcrawler hay que destacar la aparición de Lycos el 20 de julio de 1994, creado por Michale Mauldin en la Universidad de Carnegie Mellon, con un algoritmo interesante que incluía el concepto de proximidad entre palabras. Eso sí, no indexaba de forma completa las páginas, sólo las 20 primeras frases, las 200 primeras de la cabecera y un grupo de las 100 más relevantes de todo el documento.

En esta época comenzaron también a aparecer los primeros meta-buscadores. Este sistema lo que hace realmente es unificar los resultados de varios motores de búsqueda para ofrecer los resultados mezclados. En 1995 apareció el primero de ellos llamado MetaCrawler creado por Erik Selberg y Oren Etzioni en la Universidad de Washington (como Webcrawler). En este caso devolvía resultados de Lycos, Altavista, Yahoo!, Excite, Webcrawler e Infoseek. El problema era su velocidad. Como curiosidad, en noviembre de 1996 ya tenía una nueva versión, en beta, para probar.

En diciembre de 1995 seis estudiantes de Stanford lanzaron Excite gracias al proyecto Architext (iniciado en 1994) que introdujo uno de los conceptos base de las búsquedas. El complicado algoritmo intentaba crear un sistema parecido a los sinónimos mediante estadísticas entre las relaciones de palabras, de forma que se podía realizar una búsqueda obteniendo resultados aunque la misma no existiera en la página (si tenía alguna relación, claro).

El siguiente gran lanzamiento fue AltaVista. Hizo su aparición en escena en Diciembre de 1995 y fue muy importante por las mejoras que proponía: tenía ancho de banda casi ilimitado, permitía consultas en lenguaje natural (las que utilizamos habitualmente para hacer búsquedas), consultas avanzadas mediante operadores lógicos (AND, OR...), añadir o eliminar direcciones Web en 24 horas, comprobar los enlaces entrantes a un sitio Web e incluso permitía hacer búsquedas en los nombres de imágenes y algunos archivos multimedia. No sólo era grande en resultados sino veloz al entregarlos. Además, ofrecía una serie de "ayudas / trucos" para mejorar la calidad de las consultas.

A finales del 95 apareció Ozú como directorio y buscador de la mano de Advernet (todavía funciona su primera dirección). La Base de Datos del buscador y del directorio estaba íntegramente gestionada y actualizada por personas (con ayuda de herramientas de rastreo y gestión de calidad).

También a finales de 1995 apareció un nuevo e importante directorio: LookSmart. Creado por un matrimonio australiano, Evan Thornley y Tracey Ellery, y tras muchos problemas financieros, en 1997 se trasladaron a San Francisco y en 1998 llegaron a tener a Microsoft para como proveedor.

Pocos meses después, el 20 de mayo de 1996 Paul Gauthier y Eric Brewer, desde la Universidad de Berkeley, lanzaban Hotbot, que con su motor Inktomi, llegaron a un acuerdo con el sitio Web de Wired que fue el que le ayudó a darse a conocer. Se consideró el primer motor de búsqueda capaz de indexar los millones de sitios Web que había en ese momento. Tras la burbuja del 2001, perdió muchos de sus usuarios y en 2002 fue comprado por Yahoo!.

El siguiente de la lista es Ask Jeeves, lanzado en 1996 también; su idea era la de poder contestar preguntas de forma natural, tal y como las hacemos habitualmente. Como peculiar, su mayordomo, basado en Jeeves de P.G. Wodehouse. En 1999 compró una empresa llamada Direct Hit (creado por Gary Culli) y aplicó su tecnología en su motor de búsqueda. Apareció el verano de 1998 y mucha gente ya lo utilizaba a finales del mismo debido a la alta relevancia que ofrecía, principalmente porque basaba sus resultados de búsqueda en los clicks que hacían los usuarios (técnica que utilizan algunos en la actualidad).

El proyecto Google comenzó a desarrollarse en enero de 1996 por Sergey Brin y Larry Page en la Universidad de Stanford, llamándose BackRub debido a la tecnología que utilizaba, que calculaba la importancia de un sitio Web en base a los enlaces que recibía. En esa época fue cuando Page fabricó una computadora con piezas de Lego y con tecnología antigua que más adelante se convertiría en lo que hoy es Google. El 15 de septiembre de 1997 el dominio google.com era comprado y el 7 de septiembre de 1998 se creaba Google Inc. Una peculiaridad de Google es que en momentos especiales cambia su logo para adaptarlo a esa ocasión.

Hay que destacar dos razones por las que Google se hizo muy interesante: una interfaz muy clara y sencilla (como la de Altavista en sus inicios) y unos resultados muy relevantes. El secreto de los resultados, la tecnología *PageRank* (patentada el 4 de septiembre de 2001) hizo que el mundo de los motores de búsqueda cambiase completamente dando por iniciado lo que se puede llamar la versión 2 de los buscadores.

Uno de los primeros motores de búsqueda con enfoque profesional fue Norther Light, creado en agosto de 1997 disponía ya de resultados en clustering de forma que daba sugerencias de búsqueda muy interesantes, pero no llegó a ser uno de los más utilizados por el público general. En 2002 dejó de dar servicio y hoy en día vende su tecnología.

En 1998 apareció MSN Search, de la mano de Microsoft, utilizando los datos de Inktomi y también apareció el Open Directory Project (DMOZ), que, aunque no era el primer directorio hecho por personas, sí que era el primero en hacerlo de forma colaborativa. Creado por Rich Skrenta y Bob Truel y llamado inicialmente Gnuhoo, pasó a llamarse Newhoo el 5 de junio de 1998 y finalmente fue adquirido por Netscape en octubre de 1998 cuando pasó a ser el ODP, momento en el cual ya disponía de 100.000 direcciones y cerca de 4.500 editores.

A mediados de 1999 apareció en el mercado AllTheWeb. Utilizaba la tecnología de Fast, una empresa noruega que venía de la Norwegian University of Science and Technology. Este buscador ofrecía algunas mejoras con respecto a Google como por ejemplo una base de datos más actualizada, una búsqueda avanzada mucho más amplia búsquedas clusterizadas[5], pero nunca llegó a tener tanto éxito. En febrero de 2003 fue comprado por Overture y ésta, a su vez, en marzo de 2004 por Yahoo! que redujo algunas de sus funcionalidades. Hay que tener en cuenta que su base de datos pasó de 80 millones a finales de 1999 a 200 millones a

[5]Cluster: Es una agrupacion de elementos relacionados, por ejemplo los clusters para Asturias son ciudades, lugares, mar o playa

principios de 2000 llegando a los 2.000 millones en junio de 2002 quedando en 3.300 millones cuando fue adquirida por su actual propietario.

En 1999 también aparecía un gigante de la red: Baidu. El motor de búsqueda chino sería un punto de referencia hasta la actualidad debido a la presión que mantiene el gobierno chino sobre Internet. Es curioso que la mayor parte de la inversión que tiene viene dada de empresas estadounidenses.

En el año 2000 se lanzó el motor de búsqueda Teoma de mano de Apostolos Gerasoulis en la Universidad de Rutgers. Utilizaba un sistema de clustering para organizar los sitios en base al Subject-Specific Popularity (actualmente Expert Rank) que, al contrario del Pagerank de Google, analizaba los enlaces en un contexto en el que se daba un ranking a una página Web según el tema tratado. El 11 de septiembre de 2001 fue comprado por Ask Jeeves.

El 15 noviembre de 2003, Google implementó uno de los primeros grandes cambios en su motor de búsqueda añadiendo mejoras en la búsqueda semántica. Los motores de búsqueda tenían unas necesidades básicas: clustering y semántica.

En diciembre de 2003 se creó la empresa de Seekport, funcional a partir de enero de 2004. La empresa comenzó asociándose con Arexera, que tenía tratos con la versión alemana de Infoseek. La peculiaridad de Seekport es que dispone de una versión local para cada país con un índice independiente preparado para cada una de las necesidades locales.

En 2004, cuando MSN Search dejó de recibir los datos de Looksmart pasó a utilizar los resultados del motor de Inktomi. De esta misma forma, Yahoo! también dejó los datos de Google para unificar los motores de Alltheweb, Inktomi, Altavista... de forma que en marzo de 2004 esos buscadores comenzaron a utilizar la base de datos de Yahoo!.

En esta época también comenzaron a aparecer nuevos proyectos muy interesantes. Uno de ellos es Nutch, un motor de búsqueda en código abierto y creado en Java. Aunque su desarrollo es costoso ha conseguido el apoyo de Yahoo!. En diciembre de 2006 se ha lanzado Nutch en español, un proyecto que pretende crear una comunidad hispana de desarrollo de Nutch.

El 30 de septiembre de 2004 se lanzó Clusty de la mano de Vivisimo. Entre sus peculiaridades hemos de destacar su filosofía completamente basada en el clustering, lo que da pie a que el idioma inglés tenga resultados razonables, pero falla en muchos otros idiomas. Además fue el primer gran buscador que ofrecía búsquedas en Blogs o la Wikipedia en una de sus opciones.

En noviembre de 2004, MSN Search y de la mano de Christopher Payne y Oshoma Momoh pusieron en marcha una primera fase pública del motor de Microsoft, que se hizo pública el 20 de enero de 2005. El 1 de noviembre de 2005 se presentaba la plataforma Windows Live que será la nueva interfaz del motor de búsqueda de Microsoft. Tiene detalles

interesantes como que deja de existir el paginador como tal.

1993	Wandex, Aliweb, robots.txt
1994	EInet Galaxy, JumpStation, Yahoo! (directorio), WebCrawler, WWWWorm, InfoSeek, Lycos, RBSE
1995	MetaCrawler, Excite, LookSmart, Altavista, Ozú
1996	Hotbot / Inktomi, Dónde?, Ask Jeeves, Backrub / Google
1997	Norther Light
1998	MSN Search, ODP / DMOZ
1999	AllTheWeb, Baidu
2000	Teoma
2003	Seekport
2004	Yahoo! (buscador), Nutch, Clusty, MSN Search (buscador)
2005	Windows Live, Noxtrum
2006	Exalead (motor Quaero)

Tabla 1.1: Resumen historia de los buscadores

1.4. Funcionamiento de los Motores de Búsqueda

De la totalidad de los tipos de buscadores, los motores de búsqueda son los que más se vinculan con la naturaleza dinámica del contexto de la Web, son los más efectivos y eficaces cuando se busca información específica, hay competencia entre ellos y están más desarrollados.

El funcionamiento de los motores de búsqueda está basado en los siguientes componentes principales que se muestran en la figura 1.4:

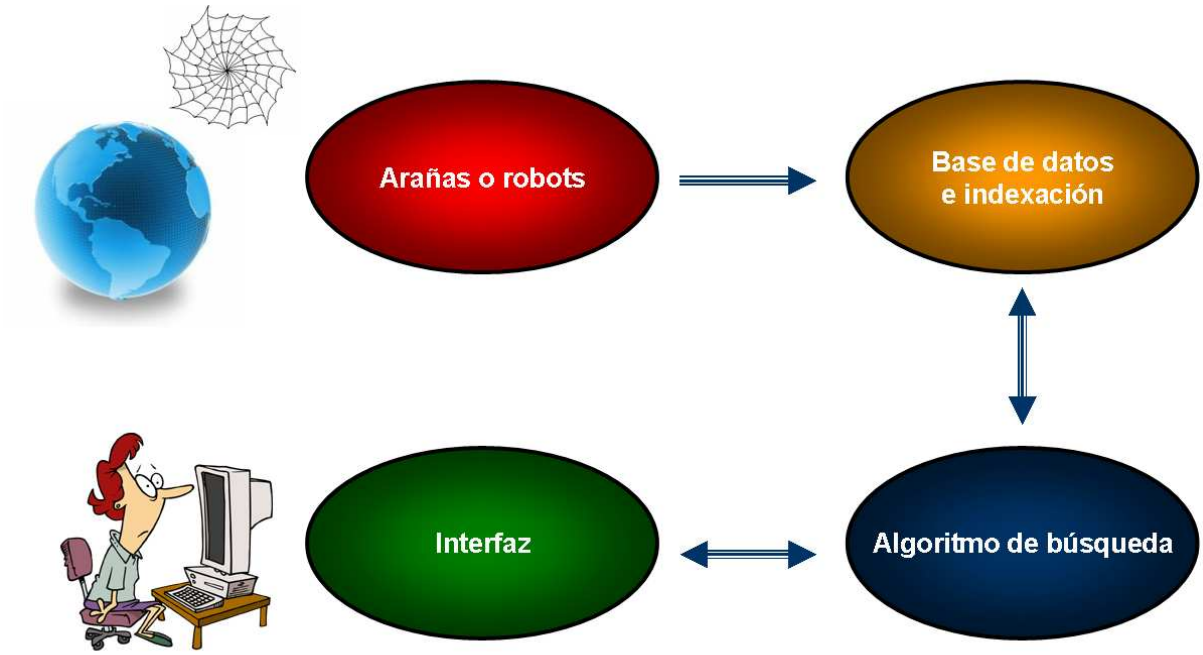

Figura 1.4: Componentes de los motores de búsqueda

1.4.1. Arañas o Robots (Spiders)

Una araña es un programa cuya función es recorrer la Web y recolectar la información que posteriormente será procesada.

Es un programa que rastrea la estructura hipertextual de la Web, recogiendo información sobre las páginas que encuentra. Esa información se indexa y se introduce en una base de datos que será explorada posteriormente.

Estos robots pueden recopilar varios millones de páginas por día, y actualizar la información recogida.

Por regla general, se parte de una lista inicial de direcciones de sitios Web, que son visitados por el robot, y a partir de allí cada robot rastrea a su manera la Web, de allí que la información almacenada en cada base de datos de cada motor sea diferente.

A pesar de lo que puede inducir su nombre y de una amplia serie de definiciones incorrectas, el robot no se mueve por la red, no tiene vida propia, ni se ejecuta sobre las máquinas remotas que visita, ya que realmente el robot funciona sobre el sistema "local" del motor de búsqueda y envía una serie de peticiones a los servidores Web remotos (donde se alojan las páginas a analizar).

Cuando la araña inicia su recorrido a partir de unas páginas iniciales se siguen los enlaces

contenidos en esa relación inicial de páginas evitando repeticiones. El recorrido puede ser de dos modos: por anchura o por profundidad, como en los recorridos de grafos.

Cuando una araña entra en un nuevo servidor, busca un archivo que se llama robots.txt, en el que se le indican los directorios permitidos y los prohibidos. Si este archivo no existiera, los considera a todos permitidos. Las arañas al recorrer nuestra Web van dejando un rastro de logs (bitácoras). De esta forma se podrá saber con un programa de estadísticas qué arañas han visitado el sitio Web entre otras cosas.

1.4.2. Base de Datos e Indexación

A medida que los robots recopilan páginas, la información contenida en las mismas debe ser almacenada e indexada. Existen dos estrategias básicas, no mutuamente excluyentes, para realizar este proceso: usar información que provee el creador o editor del documento, o extraerla directamente de todo el documento.

El volumen de información que gestiona un robot obliga a que el motor de búsqueda implemente algún tipo de indexación automática. En la práctica, los principales motores emplean ambas estrategias para disponer de una completa descripción del contenido de la página analizada. Se toman en cuenta una serie de criterios utilizados para esta descripción: el título del documento, los metadatos, el número de veces que se repite una palabra en un documento, algoritmos para valorar el peso del documento, etc.

La mayoría de los motores calculan el número de veces que se repiten las palabras claves en el cuerpo de una página, después examinan estas palabras en el nombre del dominio o en la URL, posteriormente en el título de la página, en el encabezado y en los metadatos. El orden en que se busca en cada uno de estos elementos varía en función del motor (cada uno usa sus propios algoritmos con criterios diferentes). Si el motor encuentra las palabras claves en todos estos criterios, entonces posee una razón para asignar un peso mayor al documento. Otra metodología se basa en el número de enlaces que la misma reciba o proporcione.

Un ejemplo representativo del comportamiento de un motor clásico a la hora de indexar las páginas Web es el motor Alta Vista [6]:

- Da prioridad alta a las palabras del título y a las palabras que están localizadas en el comienzo de la página.
- Asigna mayor peso a una palabra en un documento según su frecuencia absoluta. Propuesta y desarrollo de un modelo para la evaluación de la recuperación de información en la Web.

[6] No todos aplican estos lineamientos

- El mejor tamaño para una página está entre 4 y 8k. Considera las páginas largas como valiosas en contenido, cuando no están afectadas de spamming[7].
- Indexa las palabras claves y la descripción de los metadatos. Si no se tienen metadatos en la página, indexa las primeras 30 o 40 palabras de la página y las toma como descripción.
- Confiere una mayor prioridad a palabras ubicadas en los metadatos o a las palabras con las cuales se registran las páginas, pero no son tan relevantes como el título y el contenido.
- Es sensible a las palabras claves mayúsculas y minúsculas.
- Puede indexar un sitio que contiene marcos. Pero se debe asegurar que todas las páginas enlacen a la página principal.

Google es el mejor ejemplo de uso extensivo de los enlaces como base para mostrar los documentos a los usuarios de un motor. En este motor, la función de indexación la llevan a cabo dos módulos: el indexador y el clasificador. El primero lee las páginas procedentes del storeserver, descomprime los documentos y selecciona los términos incluidos en los mismos.Cada documento se convierte en un conjunto de palabras (o 'hits'), donde se graba la palabra y su posición en el documento, una aproximación de su fuente de texto y otra serie de detalles, por medio del clasificador.

El indexador analiza también los enlaces incluidos en cada página Web, información necesaria para calcular el alineamiento de las páginas a la hora de la recuperación de información.

La indexación de la base de datos en un motor de búsqueda, generalmente consiste en una lista de palabras con valor de discriminación asociadas a sus correspondientes documentos, que en este caso son las descripciones de los contenidos de las URL recopiladas. La mayor parte de los motores de búsqueda emplean como estructura de datos una estructura de tabla inversa, basado en la idea general que se muestra en la figura 1.5.

[7]Spam: mensajes no solicitados, habitualmente de tipo publicitario

Tabla de documentos

Id Documento	Contenido
A	.. variedad de frutas como naranjas, peras, manzanas ...
B	... conest tiene una variedad de ventajas ...
C	... la variedad tiene sus ventajas y desventajas ...

Tabla de palabras

Palabra	Aparece en
variedad	A, B, C
frutas	A
naranjas	A
peras	A
manzanas	A
conest	B
tiene	B, C
ventajas	B, C
desventajas	C

Figura 1.5: Ejemplo de tabla inversa

En la práctica una tabla inversa se convierte en una enorme estructura de datos con serios problemas de gestión.

El índice emplea un conjunto de punteros que apuntan a una tabla donde se recogen todas las URL en las que aparece una palabra clave. La manera en la que se ordenan estos punteros depende de un mecanismo interno de ordenación basado, generalmente, en criterios de frecuencias o pesos en el documento. El enorme tamaño de la colección de URL recopiladas por los motores obliga a buscar formas de simplificar al máximo el tamaño de estos índices. En la Tabla 1.2 se presentan algunas de las diversas técnicas empleadas.

Técnicas empleadas por los motores de búsqueda	
Conversión de texto a minúsculas	Se convierten todas las palabras a caracteres en minúscula. Reduciendo así el numero de entradas para un mismo texto (Ejemplo: Puerto-puerto)
Stemming	Aislamiento de la base de la palabra (Por ejemplo, comprensión y comprensivo se reducirían a compren), reduciéndose así el numero de entradas en el índice
Supresión de las palabras vacías	Se suprimen del índice todas aquellas palabras por las que no tiene sentido recuperar información (Artículos, preposiciones, adjetivos o interjecciones por ejemplo)
Compresión de textos	Técnicas de compactación del tamaño del archivo

Tabla 1.2: Algunas técnicas empleadas

Resulta clara la tendencia a disminuir el tamaño del índice, ya que cuando las búsquedas constan de varios términos y uno de ellos es muy frecuente, el motor puede tardar varios segundos en responder, hecho no muy bien considerado por muchos autores y profesionales.

1.4.3. Software y Algoritmo de Búsqueda

Su función es extraer la información cuando se realiza una búsqueda, y para ello hace un recorrido por la base de datos indexada buscando la información solicitada, y entregándola teniendo en cuenta un orden de relevancia.

Es lo que entra en acción cuando el usuario llega a enviar una petición de búsqueda al sistema por medio de la interfaz, para posteriormente devolver una lista con los resultados de la consulta.

Se define junto al tipo de indexación que se aplica a la base de datos.

El ordenamiento de los resultados de la consulta constituye uno de los procesos críticos a la hora de valorar la efectividad de un motor de búsqueda, ya que se trata del orden en el que el motor presenta los resultados a sus usuarios, quienes, como es lógico esperan encontrar los documentos más relevantes con sus necesidades situados entre los primeros. El motor debe ordenar el conjunto de documentos constituyente de la respuesta en función de la relevancia de estos documentos con el tema de la pregunta realizada.

En función del buen funcionamiento de su algoritmo de ordenamiento, el motor será mejor o peor valorado por los usuarios del mismo. Si un motor no discrimina su respuesta en función de la relevancia con la temática-objeto de la pregunta, el usuario encontrará documentos muy relevantes mezclados con otros menos relevantes e incluso con muchos nada relevantes, lo que le obligará a consultar un gran número de los documentos devueltos por el motor, teniendo que visitar muchas pantallas y perdiendo, en consecuencia, un cuantioso tiempo. En esta situación, el usuario terminará por no recurrir a este motor de búsqueda. Si, en cambio, el motor discrimina ese grado de relación, el usuario encontrará entre los primeros documentos a los más relevantes con la temática objeto de la pregunta, por lo que aumentará su grado de satisfacción con el motor y continuará utilizándolo.

Tradicionalmente este procedimiento ha sido uno de los secretos mejor guardados por los responsables de los distintos motores de búsqueda y realmente, no se dispone de una información clara u oficial de cómo los motores lo llevan a cabo, con excepción del motor Google que ha hecho público su algoritmo *PageRank*.

Al igual que ocurre con los criterios de indexación existen dos grandes grupos de algoritmos para el alineamiento, los que emplean variantes del modelo de espacio vectorial o del modelo booleano y los que siguen el principio de extensión de los enlaces. hay tres métodos

englobados en el primer grupo, en adición al clásico esquema TF-IDT[8]: "se denominan Booleano extendido, Vectorial extendido, y Más citado. Los dos primeros son adaptaciones de los algoritmos normales de alineamiento empleados en estos modelos clásicos de recuperación de información para incluir el hecho de la existencia de enlaces entre las páginas Web. El tercero se basa únicamente en los términos incluidos en las páginas que poseen un enlace hacia las páginas de la respuesta" [10].

El segundo grupo de algoritmos aporta una de las mayores diferencias conceptuales sobre el alineamiento: el uso de los enlaces de cada página (tanto los que recibe una página como los que emanan de ella). El número de enlaces que apuntan a una página sirve como una medida de su popularidad y calidad. La presencia de enlaces comunes entre un conjunto de página es también una medida de relación de los temas tratados en ellas. Dentro de esta nueva tipología de técnicas de alineamiento, identificamos tres clases:

- WebQuery: da un alineamiento a las páginas que forman la respuesta a una consulta con base a cómo están conectadas entre ellas. Adicionalmente, extiende el conjunto de páginas de la respuesta a otra serie de páginas altamente conectadas al grupo original de respuestas.

- HITS45: alinea las páginas Web en dos tipos distintos, que guardan una relación de mutua dependencia: autoridades (páginas muy referenciadas desde otras) y hubs (o conectores, páginas desde las que se hace referencia a otras consideradas por el autor de calidad en relación a un tema). Esta idea asume que cuando alguien establece un enlace a una página es porque la considera interesante, y que personas con intereses comunes tienden a referirse a las autoridades sobre un tema dentro de una misma página. Conectores y autoridades son conceptos que se retroalimentan: mejores autoridades son inducidas por enlaces desde buenos conectores y buenos conectores vienen de enlaces desde buenas autoridades.

- PageRank asume que el número de enlaces que a una página le proporcionan tiene mucho que ver con la calidad de la misma, es por ello que este algoritmo se puede resumir de la siguiente manera: "una página A tiene T1....Tn páginas que apuntan a ella por medio de algún enlace (es decir citas). El parámetro d es un factor que se puede fijar entre 0 y 1 (generalmente se fija en 0.85). Sea C(A) el número de enlaces que salen de la página A. Entonces, el PageRank de la página A vendrá dado por la expresión: PR(A) = (1-d) + d(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))". Este cálculo puede realizarse por medio de un algoritmo iterativo y corresponde al vector propio de una matriz normalizada de enlaces en la Web. PageRank está concebido como un modelo del comportamiento del usuario: si se asume que hay un "navegante aleatorio"que pasa de una página a otra sin presionar nunca el botón de "retroceder" y que, eventualmente nunca se aburriera, la probabilidad de que este navegante visitara una página determinada es precisamente su PageRank. Es decir, se trata de un modelo

[8]TF-IDT: Siglas de term frequency – inverse document frequency. Este modelo entiende que los documentos pueden expresarse en función de unos vectores que recogen la frecuencia de aparición de los términos en los documentos. Véase http://en.wikipedia.org/wiki/Tf-idf

basado en los enlaces de las páginas y que pretende representar la forma de trabajar de los usuarios. Otra justificación intuitiva de PageRank es que una página puede tener un alto coeficiente de PageRank si existen muchas páginas que apuntan a ella, o si hay un número algo menor de páginas que apuntan a ella pero que posean, a su vez, un alto nivel de PageRank. Lo normal es que "aquellas páginas muy citadas son páginas que vale la pena consultar y, en cambio, aquellas que sólo posean un enlace son páginas de poco interés para su consulta".

Figura 1.6: Page rank

El page rank en la Wikipedia está definido de una forma sencilla y concreta: "PageRank confía en la naturaleza democrática de la Web utilizando su vasta estructura de enlaces como un indicador del valor de una página en concreto. Google interpreta un enlace de una página A a una página B como un voto, de la página A, para la página B. Pero Google mira más allá del volumen de votos, o enlaces que una página recibe; también analiza la página que emite el voto. Los votos emitidos por las páginas consideradas "importantes" valen más, y ayudan a hacer a otras páginas "importantes"[9].

1.4.4. Interfaz

El estudio de la interfaz debe abordarse bajo dos perspectivas: la interfaz que el sistema dispone para que el usuario exprese sus necesidades de información (interfaz de consulta) y la interfaz de respuesta que dispone el sistema para mostrar al usuario el resultado de su operación de búsqueda.

No todos los sistemas poseen iguales prestaciones en lo relacionado con la recuperación de información. La clásica y más simple interfaz de usuario es la típica caja de formulario Web que se muestra en la figura 1.7. En este formulario el usuario inserta el conjunto de términos integrantes de su frase de búsqueda. Algunas veces, ese formulario le permite insertar alguna restricción a la búsqueda: el idioma de las páginas a recuperar, el tipo de objeto, si se desea emplear la búsqueda por frase literal o "búsqueda exacta", etc.

[9]http://es.wikipedia.org/wiki/PageRank

Figura 1.7: Interfaz simple de buscadores

Aunque generalmente los motores de búsqueda suelen disponer de una interfaz de búsqueda avanzada como la que se muestra en la figura 1.8, en la cual el usuario puede incorporar a su ecuación de búsqueda una serie de parámetros adicionales, tales como: uso de operadores booleanos, búsqueda por frase literal, búsqueda aplicando operadores de adyacencia, búsqueda por términos opcionales (Baeza-Yates los denomina "invitados", son esos términos que podrían aparecer o no en un documento objeto de una consulta), restricciones geográficas, restricciones por tipo de dominio, restricciones por idioma, etc.

Algunos sistemas permiten refinar la búsqueda, es decir, especificar más la pregunta sobre el conjunto de documentos recuperado inicialmente e incluso algunos motores permiten restringir el alcance de la operación de búsqueda a alguna de las partes de los documentos contenidos en sus índices (el caso más común es el título o el texto).

Figura 1.8: Interfaz avanzada de buscadores

Otra de las diferencias, algo más interna y no tan explícita, es la forma en la que un sistema interpreta una relación de varios términos como expresión de consulta sin operadores entre ellos (por ejemplo: "Historia Región Murcia"), es decir, cómo descompone el motor la expresión y construye la ecuación de búsqueda. En este punto existen también diferencias entre los sistemas, unos realizan una búsqueda por proximidad de las palabras de la expresión (es el caso de Overture), otros motores recuperarán documentos donde aparezcan todas las palabras (Google) y otros recuperarán documentos donde al menos aparezca una de las tres palabras de la ecuación (Alta Vista).

La búsqueda por término simple tiene como objeto devolver una colección de documentos donde al menos se pueda encontrar una ocurrencia de ese término, algunos sistemas permiten restringir esa búsqueda a un campo determinado (búsqueda por referencia cualificada). La búsqueda por términos múltiples permite diversas combinaciones basadas en el Álgebra de Boole: intersección de los subconjuntos correspondientes a cada término, unión de estos subconjuntos o exclusión de un subconjunto de otro; algunos sistemas permiten la combinación de los operadores para construir expresiones booleanas complejas.

Las búsquedas basadas en el contexto usan los operadores de proximidad, es decir, loca-

lizan documentos donde los términos integrantes de la ecuación de búsqueda se encuentren situados en la misma frase o en el mismo campo (además de, por supuesto, el mismo documento). El caso más cercano de proximidad es la adyacencia41 (cuando los términos están escritos en un orden determinado, por ejemplo, uno a continuación del otro).

Algunos motores permiten la búsqueda en lenguaje natural, que puede resultar especialmente interesante para aquellos usuarios no experimentados en el uso de un motor específico o en el empleo de los operadores booleanos o basados en el contexto. Estos sistemas interpretan cuestiones del estilo de "¿qué jugador de fútbol es el máximo goleador de la Copa de Europa?" o "¿qué ciudad es la capital de Angola?", devolviendo como resultados un conjunto de documentos que han considerado adecuados con la temática de la pregunta efectuada, tras haber sometido a esta expresión a un procedimiento de análisis del texto, interpretando la necesidad informativa (Alta Vista y Northern Light implementan esta modalidad). Un caso extremo de procesamiento de las expresiones en lenguaje natural es el motor Askjeeves, que llega a simular una "entrevista" con el usuario ya que, tras recibir la cuestión, la interpreta y extrae de su base de conocimientos una serie de cuestiones que traslada al usuario para refinar su exploración en la base de datos y ajustar mejor la respuesta. Por último, algunos sistemas devuelven los documentos por correspondencia con un patrón de caracteres introducido en la interfaz de consulta. Es el caso de aquellos motores que permiten hacer uso del operador de truncamiento, como es el caso de Alta Vista.

1.5. Web Semántica

"La Web Semántica es una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante" [32].

La Web Semántica ha sido impulsada por Tim Berners-Lee, creador de la World Wide Web, y otras personas relacionados con el W3C (World Wide Web Consortium).

La mayoría de los sitios Web están construidos en lenguaje HTML con marcas o etiquetas que se muestran cuando se visualiza el código fuente, pero que permanecen ocultas en la visualización normal de los navegadores, enlaces hacia otras páginas, formatos de letra, color, párrafos, imágenes, vídeos, etc. Los orígenes de la Web se basaron en el carácter abierto y universal de la base de la Web: el lenguaje HTML, y el empleo de archivos ASCII y los gráficos GIF y/o JPG. Esto permite a los buscadores clasificar los documentos HTML de la

red y ponerlos en una página Web a modo de índice o catálogo, que se puede mostrar por medio del navegador.

Gracias a que el lenguaje HTML se ajusta a unas normas estandarizadas, todos los ordenadores pueden reproducir correctamente esos documentos. Sin embargo, el lenguaje HTML se quedaba corto pues, orientado a la presentación de datos, la información que ofrece es muy limitada, no permite describir datos y no es extensible, esto es, únicamente ofrece un pequeño número de etiquetas. El sistema evolucionó y se realizaron algunas mejoras para hacer este lenguaje algo más dinámico con la introducción de otros elementos como DHTML, Javascript, hojas de estilo e, incluso, se añadieron a la Web otros lenguajes que permitieran ofrecer una información más estructurada, como el lenguaje XML, pero hacen falta otros lenguajes que permitan una descripción más detallada del documento y de su contenido, y que faciliten la comunicación entre los ordenadores. Y también hace falta una nueva generación de buscadores más inteligentes que puedan leer y evaluar rápidamente los documentos de Internet.

Así pues, el desarrollo de la Web semántica requiere la utilización de otros lenguajes como el lenguaje estructurado XML (Extensible Markup Language) y el lenguaje RDF (Resource Description Framework) que puedan dotar a cada página, a cada archivo y a cada recursos o contenido de la red, de una lógica y un significado, y que permitan a los ordenadores conocer el significado de la información que manejan con el fin de que esta información pueda no sólo ser presentada en pantalla, sino también que pueda ser integrada y reutilizada. XML ha logrado convertirse hoy en un lenguaje estándar. Se trata de un subconjunto del complejo y sofisticado lenguaje SGML que aporta datos estructurados a la Web y que se ha convertido en la infraestructura preferida para el intercambio de datos. Además, las páginas XML pueden ubicar metadatos, esquemas XML y esquemas RDF, que aportan un mecanismo para que los programas puedan interpretar y comprender documentos con un vocabulario descriptivo.

Para poder explotar la Web semántica, se necesitan lenguajes semánticos más potentes, esto es, lenguajes de marcado capaces de representar el conocimiento basándose en el uso de metadatos y ontologías. Utilizando anotaciones RDF y RDF Schema se pueden presentar algunas facetas sobre conceptos de un dominio del conocimiento y se puede, mediante relaciones taxonómicas, crear una jerarquía de conceptos. Pero se precisan lenguajes de marcado (basados en RDF) con una mayor expresividad y capacidad de razonamiento para representar los conocimientos que contienen las ontologías. Además, estos lenguajes deben ser estandarizados y formalizados para que su uso sea universal, reutilizable y compartido a lo largo y ancho de la Web. Se necesita un lenguaje común basado en Web, con suficiente capacidad expresiva y de razonamiento para representar la semántica de las ontologías. De esta forma, la utilización de lenguajes tales como OWL son una paso más en la consecución de la Web Semántica.

Es necesario, pues, crear una ontología o biblioteca de vocabularios descriptivos/semánticos, definidos en formato RDF y ubicados en la Web para determinar el significado contextual

de una palabra por medio de la consulta a la ontología apropiada. De esta forma, agentes inteligentes y programas autónomos podrían rastrear la Web de forma automática y localizar, exclusivamente, las páginas que se refieran a la palabra buscada con el significado y concepto precisos con el que interpretemos ese término. Por lo tanto, para potenciar el uso de ontologías en la Web, se necesitan aplicaciones específicas de búsqueda de ontologías, que indiquen a los usuarios las ontologías existentes y sus características para utilizarlas en su sistema.

1.5.1. Ejemplos Web Semántica

Dos de los ejemplos más conocidos de aplicación de Web Semántica son RSS y FOAF:

- RSS es un vocabulario RDF basado en XML que permite la catalogación de información (noticias y eventos) de tal manera que sea posible encontrar información precisa adaptada a las preferencias de los usuarios. Los archivos RSS contienen metadatos sobre fuentes de información especificadas por los usuarios cuya función principal es avisar a los usuarios de que los recursos que ellos han seleccionado para formar parte de esa RSS han cambiado sin necesidad de comprobar directamente la página, es decir, notifican de forma automática cualquier cambio que se realice en esos recursos de interés seleccionados. Un ejemplo de la aplicación de RSS se puede encontrar en las Noticias de la Oficina Española del W3C como canal RSS.
- FOAF es un proyecto de Web Semántica, que permite crear páginas Web para describir personas, vínculos entre ellos, y cosas que hacen y crean. Se trata de un vocabulario RDF, que permite tener disponible información personal de forma sencilla y simplificada para que pueda ser procesada, compartida y reutilizada. Dentro de FOAF podemos destacar FOAF-a-Matic, que se trata de una aplicación Javascript que permite crear una descripción FOAF de uno mismo. Con esta descripción, los datos personales serán compartidos en la Web pasando a formar parte de un motor de búsqueda donde será posible descubrir información a cerca de una persona en concreto y de las comunidades de las que es miembro de una forma sencilla y rápida. FOAFNAUT, por su lado, se utiliza para mostrar relaciones de estructuras FOAF con SVG.

Los buscadores semánticos son un ejemplo más de aplicaciones basadas en Web Semántica. El objetivo es satisfacer las expectativas de búsqueda de usuarios que requieren respuestas precisas.

Capítulo 2

Libros Electrónicos

En éste capítulo se habla sobre los documentos digitales, repasando los conceptos de información digital, el libro convencional y una comparación entre los documentos digitales frente a los libros tradicionales. Luego se hace un estudio de los formatos disponibles para los documentos electrónicos, ofreciendo una clasificación de éstos formatos basada en criterios generales y estándares de la informática actual. Por último se da una descripción de los formatos de documentos digitales más importantes.

2.1. Información Digital

Se denomina información digital o electrónica a aquella que se almacena en forma binaria, en un soporte magnético u óptico y que puede consultarse utilizando un computador. La información digital se almacena en unidades lógicas denominadas archivos. La información digital es algo bien amplio e incluye archivos de muchos tipos entre los cuales tenemos los siguientes:

- Archivos de imagen.
- Archivos de audio y video.
- Archivos ejecutables.
- Archivos de data y configuración.
- Archivos de texto plano y enriquecidos.
- Archivos con formatos especiales.
- Archivos de datos comprimidos.

La información digital nos ofrece una serie de ventajas y por pertenecer al contexto de la computación tienen las siguientes características:

- Se almacena en formato binario. La información digital se almacena en un soporte magnético u óptico en forma de ceros y unos.
- Es procesable por la computadora. Dado el tipo de soporte en que se almacena, esta información puede ser accedida mediante programas de computadora y puede ser por tanto procesada informáticamente. Este tipo de procesamiento facilita llevar a cabo de forma rápida y segura operaciones repetitivas sobre la información que antes era necesario llevar a cabo manualmente. Además, ha aumentado enormemente la capacidad que tenemos de acceder, manipular y visualizar la información.
- Como resultado de esta capacidad de procesamiento de la información:
  - Es posible obtener nueva información con valor añadido, por ejemplo mediante la extracción de gráficas o estadísticas de una serie de datos numéricos o mediante la obtención de resúmenes y el contraste de distintas informaciones sobre un mismo tema.
  - Es posible acceder a la información a distancia. A través de las redes de computadoras podemos acceder de forma casi instantánea a información situada en cualquier parte del mundo.
  - Es posible crear documentos que integren todo tipo de información textual, visual y sonora: documentos multimedia.
- La información es reutilizable. Es fácil duplicar la información y transferirla de un tipo de soporte a otro manteniendo un alto grado de fiabilidad en la copia. Además, varios usuarios pueden acceder a la misma copia al mismo tiempo.
- Es interactiva. Algunos documentos digitales pueden ser interactivos, modificándose en función del comportamiento del usuario. Una forma básica de interactividad puede ser por ejemplo, la proporcionada a través de formularios de búsqueda y recuperación de almacenes de información, o bien la utilizada por las enciclopedias infantiles en CD para enseñar a los niños conceptos básicos mediante su interacción con una serie de gráficos y animaciones.
- Es actualizable. La información digital puede ser actualizada por sus autores de forma sencilla y económica, pudiéndose generar versiones actualizadas de la misma con mucha rapidez. Además, su facilidad de distribución en forma de discos ópticos, y sobre todo a través de Internet, permite que las nuevas versiones de los documentos lleguen a sus destinatarios de forma muy rápida.
- Es recuperable. Es fácil generar almacenes de documentos digitales y utilizar mecanismos de búsqueda y recuperación de los mismos. Los modernos buscadores permiten buscar de forma rápida y relativamente eficaz documentos que cumplan una gran variedad de condiciones en repositorios que pueden contener miles de ellos.

2.2. Libros Convencionales

Según la real academia española "Un libro es un conjunto de muchas hojas de papel u otro material semejante que, encuadernadas, forman un volumen"[1].

Es una obra impresa o manuscrita que consta de una serie de hojas (más de 49 según la definición de libro dada por la UNESCO) de papel, pergamino, vitela u otro material, cosida o encuadernada que se reúne en un volumen. Un libro puede tratar sobre cualquier tema.

Hoy día, no obstante, está definición no queda circunscrita al mundo impreso o de los soportes físicos dada la aparición y auge de los nuevos formatos documentales y especialmente de la World Wide Web.

Seis mil años atrás, los sumerios escribían en tablillas de arcilla. Los egipcios lo hicieron después sobre papiros. En el siglo XV de nuestra era, Gutenberg desarrolló la imprenta, y desde entonces y hasta ahora no se han presentado cambios considerables en la constitución básica de lo que es un libro convencional. Sin embargo la computación y las tecnologías que van apareciendo acerca de los materiales alternativos al papel aportaran cambios interesantes el desarrollo evolutivo del libro como lo conocemos hoy en día, por ejemplo la invención del papel electrónico, véase la figura 2.1.

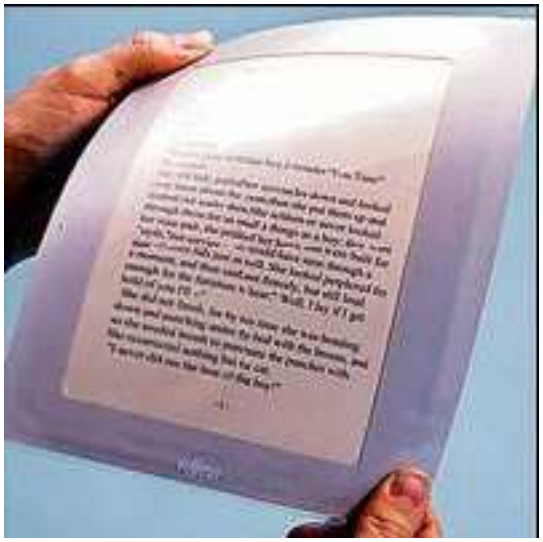

Figura 2.1: Papel electrónico

[1]Concepto de libro en http://buscon.rae.es/draeI/

2.3. Concepto de Libro Electrónico

Un e-book, o libro-e es una versión electrónica o digital de un libro. El término es ambiguo, ya que se refiere tanto a una obra individual en formato digital, como a un dispositivo utilizado para leer libros en formato digital. La mayoría de los usuarios no utiliza el término e-Book en el segundo sentido, y emplean, en cambio, el término más preciso de dispositivo de e-Book. En nuestro contexto se usará el primer sentido.

También puede entenderse a libro electrónico como "libro impreso sometido a un proceso de digitalización". Es decir, se mantiene la idea de libro a modo de referente tradicional, en parte justificable por el hecho de que los primeros e-Books aparecidos en el mercado editorial, tenían este origen. Pero también porque toda nueva tecnología o medio subsume o contiene el que le ha precedido. Esto es, la palabra oral quedó asumida en la escrita "manualmente" (libro manuscrito), ésta en la impresa (libro impreso) que a su vez queda contenida en la "digitalizada" (libro electrónico). En este comportamiento "sucesivo", el libro impreso se ratifica como el referente inmediatamente anterior, y de hecho el e-Book tiende a semejarse a él. Esta idea del libro dependiendo de los diferentes formatos bajo los que se ha realizado históricamente su circulación y consumo, adquiere específicas nomenclaturas (rótulo, códice...). En definitiva, el concepto de libro exige una materialización a la que responden a lo largo de su historia las diversas denominaciones según sus diferentes morfologías.

El libro es considerado el medio por excelencia para la difusión de la cultura. Su implantación y arraigo es muy grande y van muy unidos al papel, el soporte material más utilizado hasta el momento. No obstante, cada vez más, podemos encontrar los libros en formato digital: son los denominados libros electrónicos o e-books. Existen diversas formas de distribución de los libros electrónicos:

- **En disco óptico:** En la actualidad se están distribuyendo en este soporte fundamentalmente obras de referencia, incluyendo diccionarios y enciclopedias, aunque también es posible encontrar por ejemplo libros infantiles interactivos.
- **Consulta en línea:** En este caso se accede al libro por Internet y suele poder leerse de forma gratuita o previo pago de una cantidad determinada. Cada día aumentan los repositorios en Internet en los cuales es posible acceder a una gran cantidad de libros, incluyendo fundamentalmente literatura clásica en diversos idiomas que no posee restricciones en cuanto a derechos de copia o distribución.
- **Descarga a una computadora personal:** En ocasiones, podemos encontrar en Internet, libros almacenados en archivos que están pensados para ser descargados en una computadora personal y ser leídos o impresos posteriormente en el mismo.
- **Descarga a un lector de libros digitales:** Existen en el mercado dispositivos de lectura de libros digitales de tamaño reducido. Es posible conectar estos dispositivos a una computadora y copiar los libros a los mismos para poder leerlos posteriormente.

Los principales problemas de estos dispositivos siguen siendo la baja calidad de visualización en relación al papel y la inexistencia de un formato estándar de almacenamiento de libros digitales.

Gran parte de los libros que podemos encontrar en Internet, se encuentran almacenados como texto puro, esto es, no incluyen ningún tipo de formato, tal como letras en negrita o cursiva o con distintos tamaños y fuentes, sangrado y justificación de párrafos, etc. No obstante, existen otros formatos de almacenamiento que permiten incorporar éstas características, junto con la posibilidad de incluir imágenes y gráficos. Entre ellos podemos destacar el formato PDF (Portable Document Format) o uno más reciente como el formato LIT de Microsoft. En una sección posterior se revisarán los principales formatos de archivos que nos permiten almacenar textos electrónicos.

Por ahora, los libros digitales son un instrumento muy útil para realizar consultas de forma rápida y sencilla, como las que se pueden realizar a un diccionario. Ahora bien, los libros dirigidos al gran público, siguen editándose mediante el sistema tradicional y en papel. No obstante, ya existen experiencias en la distribución de algunas super ventas, a través de la Web.

Entre los tipos de contenidos de documentos electrónicos también tenemos los siguientes:

- Revistas especializadas.
- Prensa.
- Obras de referencia.
- Diccionarios.
- Enciclopedias.
- Directorios.
- Fuentes geográficas.
- Publicaciones oficiales.
- Obras de referencia secundaria.

2.3.1. Documento Digital Frente a Documento en Papel

Para comparar los documentos digitales con los documentos en papel se va a utilizar unos parámetros similares a los propuestos por diversos autores: ergonomía, actualización, densidad, interactividad, durabilidad, autenticidad y visualización.

Con respecto a la ergonomía, una vez que se dispone del documento en cualquiera de los dos formatos, la versión en papel es mucho más fácil de manipular (cargar, mover, ...). Para acceder a los documentos digitales es necesario utilizar una computadora, con un dispositivo de lectura y unos programas adecuados. Obviamente no siempre disponemos de estos elementos, y es mucho más fácil cargar un libro, o artículo en papel y leerlo en cualquier parte. En la actualidad se está investigando en el desarrollo de nuevos soportes de lectura de documentos digitales de pequeño tamaño y fáciles de transportar (ver figura 2.2). Puede que cuando estos aparatos funcionen de forma adecuada y estén disponibles para el gran público, la ergonomía de los documentos digitales iguale a la de los documentos en papel.

Figura 2.2: Nuevas tecnologías para la lectura de libros electrónicos

La durabilidad es una propiedad crítica para garantizar la transmisión de la ciencia y la cultura a largo plazo. Aunque parezca mentira, los modernos sistemas de almacenamiento magnético y óptico son menos durables que los documentos en papel. Se sabe que un documento editado en un buen papel tiene una duración mínima de unos quinientos años, mientras un disco óptico puede durar en condiciones óptimas unos cien. Más aún, se ha comprobado que un uso normal de un disco óptico le garantiza una vida media de entre dos y diez años.

Por otro lado, los documentos digitales están sujetos, no sólo al desgaste físico del soporte, sino a la obsolescencia tecnológica. La evolución de los tipos de soporte y de la tecnología y los programas de acceso a los documentos digitales hace que en pocos años, un soporte que parecía legible a largo plazo deje de ser accesible. Se puede pensar en lo que ha ocurrido con los disquetes de 5,25 pulgadas y lo que pronto ocurrirá con los de 3,5. Esta misma duda podía establecerse respecto a la inminente sustitución de los CDs, y DVDs, pero los distintos fabricantes de dispositivos lectores, garantizan compatibilidad hacia atrás.

El tercer parámetro en el que el papel supera al documento digital es la autenticidad, referida a su resistencia a la falsificación. Es relativamente complicado falsificar un documento en papel, en cambio, no existen muchos impedimentos para duplicar o modificar un documento digital, en ocasiones, de modo indetectable. Esta circunstancia hace que los documentos digitales no tengan validez jurídica. También en este ámbito la tecnología y las leyes están evolucionando, y se están estableciendo sistemas de autentificación que permiten identificar de modo inequívoco la originalidad de un documento digital.

En relación a la visualización de los documentos, la calidad de la misma es superior en el caso del papel, que en el de los documentos digitales. Las pantallas de las computadoras tradicionales con monitores de tubo de rayos catódicos suelen funcionar en base a un rayo de luz que actualiza el contenido de la misma varias veces por segundo y proyecta su contenido hacia los ojos del usuario. Este modo de funcionamiento produce un parpadeo, apenas perceptible, pero que fatiga la vista y resta calidad a la visualización. Se sabe que una frecuencia superior a 120 actualizaciones por segundo (120Hz) elimina el parpadeo y se acerca un poco a la calidad de visualización con el papel en este sentido. En la actualidad las pantallas CRT están desapareciendo.

Además de las dificultades relacionadas con el parpadeo de las pantallas, es necesario tener en cuenta la definición con que se visualizan los documentos. Por definición nos referimos a la cantidad de puntos por unidad de superficie con que se presenta el documento en la pantalla. En último término, tanto las imágenes como el texto que vemos en la pantalla están formadas por una cantidad dada de puntos de distintos colores. Cuanta más definición tenga el documento en pantalla, mejor se verá. Las actuales pantallas de computadoras todavía tienen una definición bastante inferior que la obtenida a partir de la impresión en papel. Mientras las pantallas habituales ofrecen una definición de entre 75 y 90 puntos por pulgada, el papel puede ofrecer entre 300 y 3000 puntos por pulgada.

Por otro lado, los documentos digitales son superiores al papel en los parámetros de actualización, densidad e interactividad. En cuanto a la densidad, nos referimos con este concepto a la cantidad de información que podemos almacenar en un documento digital frente a documento en papel. Tal y como comentamos en el tema anterior, los soportes informáticos, tales como los discos ópticos o magnéticos permiten almacenar una enorme cantidad de información en un espacio muy reducido.

2.4. Formatos de Documentos Electrónicos

En el ámbito de la informática, todos los documentos, y en general toda la información que es manejada mediante computadoras se almacena en forma de archivos. Los archivos son el elemento básico mediante el cual los sistemas operativos almacenan la información. Sistemas Operativos como las distintas versiones de Windows o de UNIX almacenan toda la información en archivos y la organizan en los dispositivos de almacenamiento (discos duros,

disquetes, discos ópticos, etc.).

Cuando hablamos de formato de un archivo informático nos referimos a la forma en que se ha codificado la información en el mismo, es decir, el formato de un archivo es el modo en que se ha traducido a ceros y unos la información que almacena. Este formato debe ser reconocido e interpretado de forma adecuada por los programas que lo manejan para poder visualizar y manipular la información que contienen.

Existen formatos propios de determinadas aplicaciones que corresponden a la forma en que las mismas almacenan la información por defecto. Por ejemplo, los documentos almacenados mediante Microsoft Word suelen tener la extensión .doc. Por otro lado, existen formatos de archivo más genéricos que corresponden a un tipo particular de información y que pueden venir o no dados por un estándar. Por ejemplo, la mayoría de las imágenes que se manejan en Internet se almacenan en formato GIF o JPEG. En el caso de los formatos genéricos, existen distintos programas que pueden leerlos y visualizarlos y que nos pueden permitir modificar el contenido de los archivos. Así, numerosas aplicaciones gráficas nos permiten acceder a los formatos GIF y JPEG o guardar los gráficos que hemos generado o modificado con ellas en estos formatos. Lo anterior no significa que no podamos intentar abrir cualquier tipo de archivo con la mayoría de las aplicaciones.

Lo que ocurre simplemente es que cuando las aplicaciones no reconocen un determinado formato, pueden producir un error o bien mostrarnos la información más extraña imaginable en pantalla, tal y como puede constatarse en la figura 2.3.

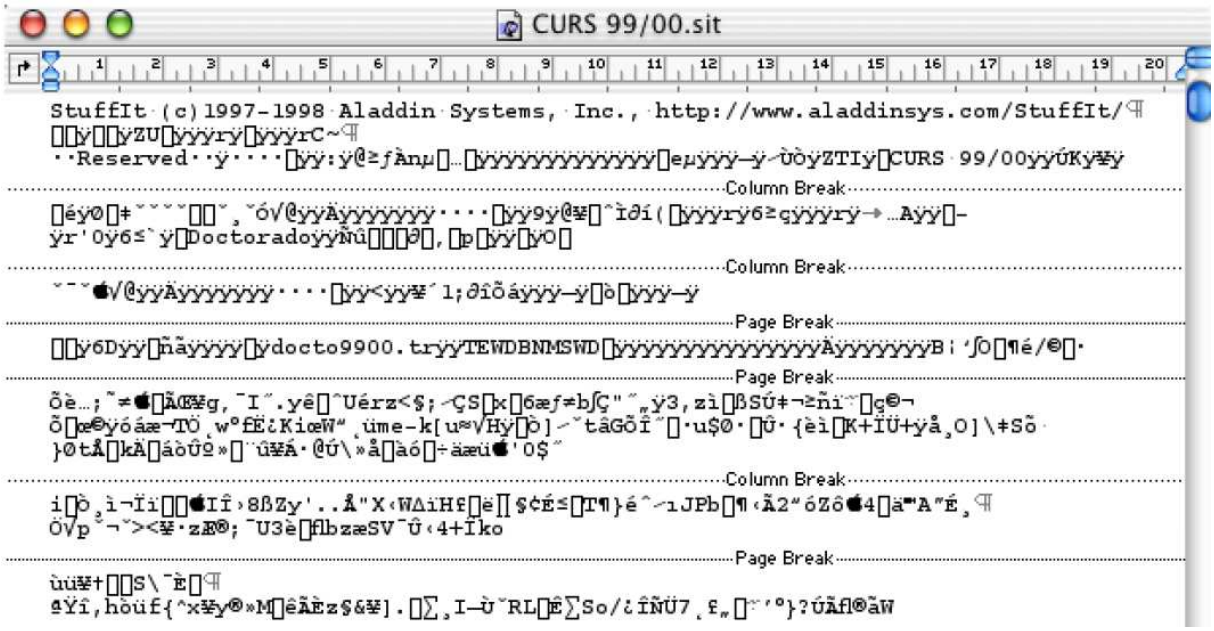

Figura 2.3: Formato ilegible

En este capítulo se van a revisar algunos de los principales formatos informáticos en los que pueden guardarse los documentos electrónicos que básicamente equivale a referirse a ellos como textos electrónicos. En un primer nivel podemos clasificar estos en dos tipos básicos:

El texto puro o texto sin formato corresponde a archivos que sólo almacenan información textual, incluyendo letras, números, signos de puntuación y otros símbolos como paréntesis, interrogantes, etc. Algunos de estos archivos restringen tanto los caracteres que pueden almacenar que no permiten por ejemplo guardar texto con acentos o la letra ñ.

El texto con formato es aquel que no sólo contiene la información textual, sino que además dota al texto de determinadas características que mejoran la presentación global del documento. En este tipo de documentos puede por ejemplo incluirse texto utilizando distintas fuentes (tipo de letra), podemos utilizar texto en negrita o en cursiva, o añadir características tales como el sangrado o la justificación a los párrafos. Más aún, aunque estrictamente hablando no se trata de texto, podemos incluir en la categoría de textos electrónicos con formato aquellos que incorporan imágenes, gráficos o tablas y que pueden ubicarlos en determinadas posiciones del documento.

Algunas de las condiciones deseables en los archivos que guardan texto con formato son las siguientes:

- Que puedan ser visualizables con un visor universal, fácil de conseguir para distintos sistemas operativos y a ser posible gratuito.
- Que conserven el color, formatos y fuentes con los que fueron generados.
- Que puedan incluir imágenes.
- Que se impriman con fidelidad al original.
- Que permitan añadir condiciones de seguridad para impedir su alteración, e incluso su impresión.
- Que ocupen poco espacio en disco.

A continuación se va a revisar la codificación utilizada en los archivos de texto plano y los principales tipos de textos con formato que podemos encontrar y sus ventajas e inconvenientes con respecto a las características anteriores. Entre los textos con formato vamos a diferenciar tres tipos básicos:

- El generado mediante procesadores de texto.
- Los formatos de visualización e impresión.
- Los lenguajes de formato

2.4.1. Texto Plano ASCII (Sin formato)

Para guardar información textual es necesario acordar una codificación que haga corresponder cada carácter almacenado con una combinación de ceros y unos. Existen diversos estándares internacionales que establecen esta correspondencia, el más extendido de los cuales es el código ASCII (American Standard Code for Information Interchange).

La versión básica de este código incluye un máximo de 128 caracteres, de los cuales los 31 primeros no corresponden a caracteres representables en pantalla, sino que se utilizan para almacenar información de control, por ejemplo, para indicar cuando se producen los cambios de párrafo. En este código los números del 0 al 9 se corresponden con los códigos del 48 al 57 y las letras minúsculas quedan codificadas con los valores entre el 97 y el 122. En la versión básica del código no están representadas por ejemplo las letras con acento, la letra ñ, o los símbolos de apertura de interrogación y exclamación (¡, ¿), ni diversos caracteres especiales utilizados por idiomas distintos del inglés. Debido a que esto representa un claro problema fuera de los países del ámbito anglosajón, existe la versión extendida del código ASCII, que permite almacenar hasta 256 caracteres distintos y que incluye todos los de la versión básica en la misma posición y añade los que hemos comentado (p.e. la ñ corresponde al código 164). En la figura 2.4 podemos ver los caracteres del código ASCII extendido representables.

TABLA ASCII EXTENDIDA

http://www.opcionweb.com

Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter
200	80	128	Ç	240	A0	160	á	300	C0	192	└	340	E0	224	Ó
201	81	129	ü	241	A1	161	í	301	C1	193	┴	341	E1	225	ß
202	82	130	é	242	A2	162	ó	302	C2	194	┬	342	E2	226	Ô
203	83	131	â	243	A3	163	ú	303	C3	195	├	343	E3	227	Ò
204	84	132	ä	244	A4	164	ñ	304	C4	196	─	344	E4	228	õ
205	85	133	à	245	A5	165	Ñ	305	C5	197	┼	345	E5	229	Õ
206	86	134	å	246	A6	166	ª	306	C6	198	ã	346	E6	230	µ
207	87	135	ç	247	A7	167	º	307	C7	199	Ã	347	E7	231	þ
210	88	136	ê	250	A8	168	¿	310	C8	200	╚	350	E8	232	Þ
211	89	137	ë	251	A9	169	®	311	C9	201	╔	351	E9	233	Ú
212	8A	138	è	252	AA	170	¬	312	CA	202	╩	352	EA	234	Û
213	8B	139	ï	253	AB	171	½	313	CB	203	╦	353	EB	235	Ù
214	8C	140	î	254	AC	172	¼	314	CC	204	╠	354	EC	236	ý
215	8D	141	ì	255	AD	173	¡	315	CD	205	═	355	ED	237	Ý
216	8E	142	Ä	256	AE	174	«	316	CE	206	╬	356	EE	238	¯
217	8F	143	Å	257	AF	175	»	317	CF	207	¤	357	EF	239	´
220	90	144	É	260	B0	176	░	320	D0	208	ð	360	F0	240	
221	91	145	æ	261	B1	177	▒	321	D1	209	Ð	361	F1	241	±
222	92	146	Æ	262	B2	178	▓	322	D2	210	Ê	362	F2	242	‗
223	93	147	ô	263	B3	179	│	323	D3	211	Ë	363	F3	243	¾
224	94	148	ö	264	B4	180	┤	324	D4	212	È	364	F4	244	¶
225	95	149	ò	265	B5	181	Á	325	D5	213	ı	365	F5	245	§
226	96	150	û	266	B6	182	Â	326	D6	214	Í	366	F6	246	÷
227	97	151	ù	267	B7	183	À	327	D7	215	Î	367	F7	247	¸
230	98	152	ÿ	270	B8	184	©	330	D8	216	Ï	370	F8	248	°
231	99	153	Ö	271	B9	185	╣	331	D9	217	┘	371	F9	249	¨
232	9A	154	Ü	272	BA	186	║	332	DA	218	┌	372	FA	250	·
233	9B	155	ø	273	BB	187	╗	333	DB	219	▮	373	FB	251	¹
234	9C	156	£	274	BC	188	╝	334	DC	220	▄	374	FC	252	³
235	9D	157	Ø	275	BD	189	¢	335	DD	221	¦	375	FD	253	²
236	9E	158	×	276	BE	190	¥	336	DE	222	▌	376	FE	254	■
237	9F	159	ƒ	277	BF	191	┐	337	DF	223	▀	377	FF	255	

Figura 2.4: Tabla ASCII extendida

Con respecto a las ventajas e inconvenientes de los documentos de texto puro:

La principal ventaja del texto puro es que utiliza una codificación estándar de cada símbolo. De este modo, numerosas aplicaciones pueden leer el contenido de este tipo de documentos y nos permiten su modificación. Así, todos los procesadores y editores de texto pueden leer documentos en este formato y la mayoría de ellos ofrecen la posibilidad de guardar los documentos utilizándolo. Esto hace que con el fin de facilitar la máxima difusión de la información y simplificar su acceso, muchos de los documentos que podemos encontrar en Internet se encuentren en formato de texto puro. Así, muchos de los libros electrónicos que podemos encontrar en los principales repositorios de la red, están almacenados en este formato.

Por su propia definición los documentos en texto puro no conservan color, formatos o fuentes porque no permiten incorporarlos, y tampoco permiten incluir imágenes. Al tratarse

de un formato tan simple, la impresión del texto puro conserva una alta fidelidad con el original.

En cuanto a la seguridad, no es posible añadir condiciones para evitar la alteración de los documentos en texto puro.

Finalmente, y como otra ventaja de este tipo de formato, los documentos guardados en el mismo ocupan un espacio muy reducido, dado que sólo guardan los caracteres que contienen y ninguna información adicional sobre su formato.

2.4.2. Generado con Procesadores de Texto

Una parte importante de los documentos de texto que solemos manejar con las computadoras han sido generados utilizando algún procesador de textos. El objetivo de este tipo de aplicaciones es precisamente la creación o modificación de texto con formato. Alguno de los procesadores de textos más conocidos son Microsoft Word y WordPerfect.

Los procesadores de textos más extendidos son comerciales y guardan por defecto la información en un formato que les es propio. Esto significa que tan sólo ese mismo procesador es capaz de leer estos documentos. No obstante, dada la importancia del fácil intercambio de información, todos los procesadores permiten almacenar los documentos en otros formatos que no poseen la restricción anterior. Por ejemplo, podemos almacenar la información como texto puro o en formato RTF (Rich Text Format). El problema de guardar los documentos con un procesador en un formato distinto del que le es propio, es que en muchos casos se pierden parte de sus características. Por ejemplo, si guardamos un documento Word como texto puro, perdemos todo el formato que contiene, incluyendo obviamente cualquier imagen o gráfico.

Otro problema que tienen los formatos propios de los procesadores de texto es la poca consistencia de los mismos entre distintas versiones del programa. Así, distintas versiones de un mismo procesador pueden no reconocer un documento generado por otras o cambiar notablemente su apariencia. Es muy normal que versiones antiguas de un mismo procesador no sean capaces de abrir documentos generados con versiones más recientes. Por ejemplo, Word 6 puede tener bastantes dificultades en abrir y visualizar correctamente un documento creado y guardado con Word XP. No sólo eso, una misma versión de un procesador, pero utilizada sobre dos sistemas operativos distintos (p.e. Windows XP y MacOS X) puede dar lugar a visualizaciones bastante diferentes del mismo documento.

En definitiva, este tipo de documentos no disponen de un visor universal y conservan, aunque no totalmente el color, formato y fuentes del original.

Al igual que ocurre con la visualización, no siempre es posible imprimir este tipo de documentos con fidelidad. El resultado depende en gran medida de la versión del procesador

que usemos para imprimirlo, de los tipos de caracteres disponibles en el sistema operativo y de la impresora que utilicemos.

Los principales procesadores de textos suelen incorporar algún mecanismo para proteger los documentos contra su posterior alteración. Principalmente el uso de contraseñas.
Nuevas tecnologías aplicada a los libros Otro de los inconvenientes de este tipo de formato es que los archivos que lo utilizan suelen ocupar mucho espacio. De hecho, bastante más del que parece lógico para incluir información sobre su contenido y formato.

2.4.3. Formatos de Visualización e Impresión

Existen formatos especialmente creados para la visualización e impresión de calidad de textos con formato. Entre ellos podemos destacar dos muy extendidos: El formato PDF y el formato PostScript.

El formato PDF (Portable Document Format) es un formato muy difundido creado por Adobe y pensado especialmente para la distribución de textos electrónicos con formato. Incorpora todas las características deseables para este tipo de formatos:

- Existe un visor universal de libre distribución (gratuito) que permite visualizar este tipo de documentos en los sistemas operativos más extendidos: el Acrobat Reader.
- El formato está pensado para conservar las fuentes, formatos y colores de los documentos.
- Permite la ubicación precisa de imágenes y los mantiene ante un cambio de sistema operativo.
- Aunque no es estrictamente un formato de impresión, mantiene un alto grado de fidelidad al original en las versiones impresas.
- Las aplicaciones para la generación de documentos en PDF permiten incorporar condiciones de seguridad a los documentos que impiden su posterior alteración o impresión.
- Los documentos en PDF suelen ocupar bastante menos espacio que los equivalentes generados con los procesadores de textos más extendidos.

Nuevas tecnologías aplicada a los libros
El principal inconveniente de los documentos en formato PDF es que no pueden editarse y modificarse. Dado que se trata de un formato pensado para el intercambio y la visualización de documentos, no existen programas pensados para su modificación.

Podemos crear archivos con este formato a partir de procesadores de texto y otras aplicaciones como editores de gráficos u hojas de cálculo. Para ello es necesario disponer de

la aplicación comercial (de pago) Adobe Acrobat. Esta aplicación permite imprimir los documentos sobre una impresora especial que no genera una copia en papel, sino un archivo en formato PDF. Además, permite incorporar a aplicaciones como Word la posibilidad de guardar directamente en este formato mediante una de sus opciones de menú.

El formato PostScript (.ps) fue creado específicamente para permitir la impresión de documentos con alta calidad y fidelidad al original. Este tipo de archivos poseen unas características muy similares al formato PDF [39].

- Existen visores bastante difundidos que permiten visualizar los documentos PostScript. Sin embargo, no se trata de un formato pensado para una visualización de calidad, sino para su impresión, con lo que no siempre podemos esperar buenos resultados en pantalla. El visor más difundido de este tipo de formato es GhostView.
- Permite conservar las fuentes, formatos y colores de los documentos en su versión impresa.
- Permite la ubicación precisa de imágenes.
- Conserva con total precisión el formato al imprimir el documento, ya que es su principal finalidad. No obstante para poder imprimir documentos de este tipo es necesario utilizar impresoras que reconozcan este formato. Este tipo de impresoras se denominan impresoras PostScript y no suelen ser las más vendidas entre usuarios particulares.
- El formato PostScript no incorpora mecanismos para impedir su modificación, aunque, tal y como puede verse en la figura 20, resulta cuanto menos difícil lograrlo accediendo directamente a la información textual almacenada.
- Los documentos en PostScript suelen ser de pequeño tamaño salvo cuando incluyen imágenes o gráficos que ocupan mucha memoria.

```
%!PS-Adobe-2.0
%%Creator: dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
%%Title: Linuxdoc-Ejemplo.dvi
%%Pages: 3
%%PageOrder: Ascend
%%BoundingBox: 0 0 596 842
%%DocumentPaperSizes: A4
%%EndComments
%DVIPSCommandLine: dvips -q -t a4 -o Linuxdoc-Ejemplo.ps
%+ Linuxdoc-Ejemplo.dvi
%DVIPSParameters: dpi=600, comments removed
%DVIPSSource:  TeX output 1998.11.12:0436
%%BeginProcSet: tex.pro
/TeXDict 250 dict def TeXDict begin /N{def}def /B{bind def}N /S{exch}N
/X{S N}B /TR{translate}N /isls false N /vsize 11 72 mul N /hsize 8.5 72
mul N /landplus90{false}def /@rigin{isls{[0 landplus90{1 -1}{-1 1}
ifelse 0 0 0]concat}if 72 Resolution div 72 VResolution div neg scale
isls{landplus90{VResolution 72 div vsize mul 0 exch}{Resolution -72 div
hsize mul 0}ifelse TR}if Resolution VResolution vsize -72 div 1 add mul
TR[matrix currentmatrix{dup dup round sub abs 0.00001 lt{round}if}
forall round exch round exch]setmatrix}N /@landscape{/isls true N}B
/@manualfeed{statusdict /manualfeed true put}B /@copies{/#copies X}B++
/FMat[1 0 0 -1 0 0]N /FBB[0 0 0 0]N /nn 0 N /IE 0 N /ctr 0 N /df-tail{
/nn 8 dict N nn begin /FontType 3 N /FontMatrix fntrx N /FontBBox FBB N
string /base X array /BitMaps X /BuildChar{CharBuilder}N /Encoding IE N
```

Figura 2.5: Comienzo del contenido de un archivo Post Script

Al igual que ocurre con el formato PDF, los archivos en PostScript no pueden modificarse, ni pueden extraerse o copiarse parte de los mismos, tan sólo pueden visualizarse y, fundamentalmente, imprimirse.

Los archivos PostScript suelen generarse a partir de cualquier aplicación que genere un documento susceptible de ser impreso. Si disponemos de una impresora PostScript y del software adecuado para imprimir en la misma, las propias aplicaciones permitirán imprimir sobre un archivo PostScript en lugar de sobre papel.

2.4.4. Lenguajes de Formato

Los lenguajes de formato están pensados para describir el contenido y formato de documentos. El contenido de los archivos de documentos de un lenguaje de formato determinado son instrucciones basadas en texto plano tal como se explico en la subsección 2.4.1.

Mediante una sintaxis preestablecida nos permiten describir todas las características relativas al formato del documento y del texto que contienen, pudiendo incluir aspectos tales como los márgenes, las cabeceras, las características asociadas a los caracteres, a los párrafos, etc. Además, nos permiten describir tablas y ubicar imágenes. La idea es poder describir textos electrónicos con formato totalmente independiente del entorno en el que estemos visualizándolos, de modo que su apariencia no varíe aunque cambiemos de computadora o de sistema operativo.

El más difundido de estos lenguajes de formato en la actualidad es el HTML (HyperText Mark-Up Language), debido a que se usa para la creación de páginas Web.Este lenguaje de formato tiene su origen en el SGML (Standard Generalized Mark-Up Language) que constituye un estándar internacional para definir lenguajes de formato basados en etiquetas. El lenguaje HTML se basa en las normas establecidas por el SGML para dar formato a textos e incorporar enlaces, es decir, para crear documentos hipertexto con formato.

El que el lenguaje esté basado en el uso de etiquetas significa que existen unos textos especiales (etiquetas, o tags en inglés) que se incorporan al documento y que permiten definir las partes de que consta y todo lo relativo al formato de las mismas.

Dado que las páginas Web han ido aumentando su complejidad para responder a las demandas de los usuarios de la Web, el lenguaje utilizado para poder describirlas ha ido evolucionando. De este modo se han definido varias versiones del lenguaje HTML. En la actualidad la versión más reciente es el HTML 4.01. No obstante, este lenguaje sigue teniendo serias limitaciones para adaptarse a las nuevas tecnologías y medios, tales como la telefonía móvil. Lo que se ha hecho es definir una nueva versión de las normas dadas por el SGML denominada XML (eXtensible Mark-up Language), a partir de la cual será posible definir nuevas versiones del HTML, denominadas XHTML.

Además de los lenguajes de formato relacionados con el HTML y con las páginas Web, existe otro lenguaje de formato muy extendido entre los científicos e investigadores, ya que está especialmente pensado para la descripción de artículos científico-técnicos con un formato sofisticado, incluyendo gráficos y fórmulas matemáticas complejas. Este lenguaje se denomina LaTex (ver figura 2.6) que sera tratado más adelante y existen programas que permiten la traducción de los documentos escritos en el mismo a formatos visualizables o imprimibles, tales como el PostScript.

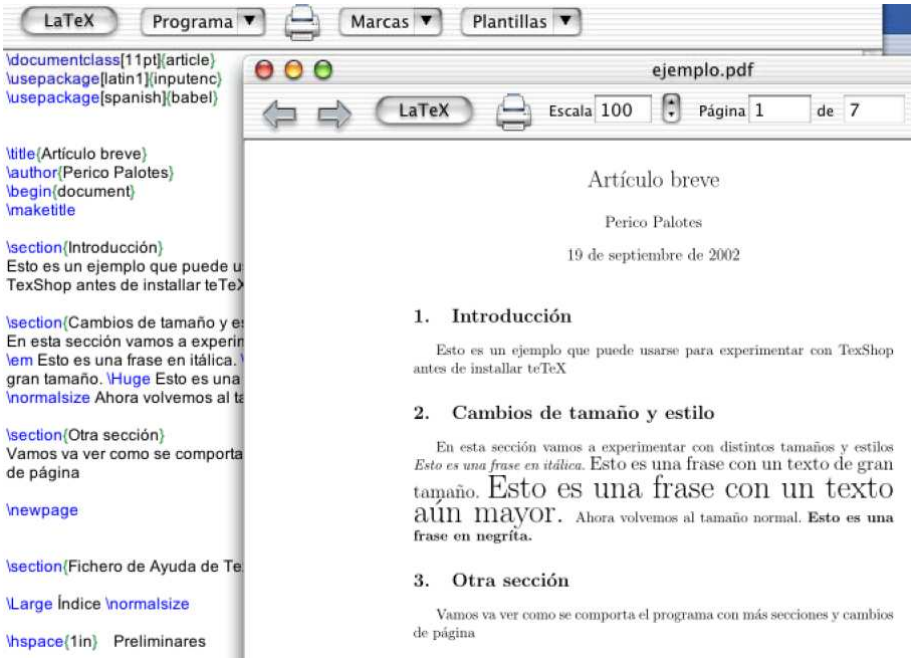

Figura 2.6: LaTex

Las características generales de los documentos definidos mediante lenguajes de formato son las siguientes:

- Al estar pensados para ser visualizados en cualquier entorno, existen programas para distintos sistemas operativos que son capaces de traducir el lenguaje a un documento con formato. Por ejemplo, los navegadores Web están pensados para visualizar documentos escritos en HTML, y las versiones más recientes de los principales procesadores de textos también permiten su visualización.

- Permiten la inclusión de fuentes, formatos y colores en el documento, aunque la variedad de los mismos depende del lenguaje en concreto. Así, el lenguaje HTML es bastante limitado en este sentido, mientras el LaTex es mucho más completo.

- Los lenguajes de formato suelen permitir la ubicación de imágenes en el texto, pero la precisión del resultado obtenido depende de cada lenguaje concreto.

- La calidad de la impresión obtenida depende mucho del lenguaje utilizado. Mientras HTML no garantiza una impresión fidedigna y depende mucho del programa visor utilizado para imprimir, el lenguaje LaTex está pensado para generar versiones imprimibles que no varíen al cambiar el entorno de trabajo.

- Los lenguajes de formato no están pensados para incorporar mecanismos de protección de los documentos obtenidos.

- Los archivos con lenguaje de formato suelen ser de tamaño reducido, ya que contienen tan sólo texto puro. No obstante hay que tener en cuenta que no incorporan las imágenes o gráficos y que éstas deben ser tratadas como archivos adicionales cuyo tamaño debería sumarse al del archivo con el lenguaje de formato.

2.5. Formatos más Populares

Entre los formatos más populares podemos destacar los siguientes:

2.5.1. Portable Document Format (PDF)

Del inglés Portable Document Format, (Formato de Documento Portátil) es un formato de almacenamiento de documentos, desarrollado por la empresa Adobe Systems. Está especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento, determinando todos los detalles de cómo va a quedar, no requiriéndose procesos anteriores de ajuste ni de maquetación. Cada vez se utiliza más también como especificación de visualización, gracias a la gran calidad de las fuentes utilizadas y a las facilidades que ofrece para el manejo del documento, como búsquedas, hiperenlaces, etc [40].

Entre las características de PDF las cuales mencionan ellos mismo tenemos:

- Es una especificación abierta, para la que se han generado herramientas de Software Libre que permiten crear, visualizar o modificar documentos en formato PDF. Un ejemplo es la suite ofimática OpenOffice.org.
- Multiplataforma: se puede ver e imprimir en cualquier plataforma Macintosh, Microsoft Windows, UNIX y múltiples plataformas móviles.
- Mantenimiento de la integridad de la información: los archivos PDF de Adobe tienen el mismo aspecto y muestran la misma información que los archivos originales, como, por ejemplo, texto, dibujos, 3D, gráficos en color, fotos e incluso lógica empresarial, independientemente de la aplicación utilizada para crearlos.
- Puede integrar cualquier combinación de texto, gráficos, imágenes e incluso música.
- Es uno de los formatos más extendidos en Internet para el intercambio de documentos. Por ello es muy utilizado por empresas, gobiernos e instituciones educativas.
- Puede cifrarse para proteger su contenido e incluso firmarlo digitalmente.
- El archivo PDF puede crearse desde varias aplicaciones exportando el archivo, como es el caso de los programas de OpenOffice.org.

2.5.2. Microsoft WORD

Microsoft Word es un procesador de texto creado por Microsoft, y actualmente integrado en la suite ofimática Microsoft Office.

Originalmente desarrollado por Richard Brodie para la computadora de IBM con el sistema operativo DOS en 1983. Se crearon versiones posteriores Apple Macintosh en 1984 y Microsoft Windows en 1989, siendo esta última versión la más difundida en la actualidad, llegando a ser el procesador de texto más popular.

Microsoft Word utiliza un formato nativo cerrado y muy utilizado, comúnmente llamado DOC (utiliza la extensión de archivo .doc). Por la amplísima difusión del Microsoft Word, este formato se ha convertido en estándar de facto con el que pueden transferirse textos con formato o sin formato, o hasta imágenes, siendo preferido por muchos usuarios antes que otras opciones como TXT para el texto sin formato o JPG para gráficos; sin embargo, este formato posee la desventaja de tener un mayor tamaño comparado con algunos otros. Por otro lado, la Organización Internacional para la Estandarización ha elegido el formato OpenDocument como estándar para el intercambio de texto con formato, lo cual ha supuesto una desventaja para el formato .doc. Ahora en el word 2007, tiene un nuevo formato "docx". Es un formato más avanzado y comprime más el documento.

El formato RTF surgió como acuerdo para intercambio de datos entre Microsoft y Apple en los tiempos en que Apple dominaba el mercado de las computadoras personales.

Las primeras versiones del formato .doc de Word derivaban del RTF. Incluso ahora hay programas de Microsoft como Wordpad que usan el RTF como formato nativo. El RTF también tiene extensión .doc al igual que los sucesivos formatos de Word que se han presentado.

RTF también es usado por Word para importar y exportar a formatos implementados por DLLs. Puede considerársele un segundo formato nativo.

2.5.3. Open Document. Text (ODT)

El Formato de Documento Abierto para Aplicaciones Ofimáticas de OASIS (en inglés, OASIS Open Document Format for Office Applications), también referido como OpenDocument u ODF, es un formato de archivo estándar para el almacenamiento de documentos digitales como texto electrónico con formato, hojas de cálculo, memorandos, gráficas y presentaciones. Su desarrollo ha sido encomendado a la organización OASIS y está basado en un esquema XML inicialmente creado por OpenOffice.org [41].

El estándar fue desarrollado públicamente por un grupo de organizaciones, es de acceso libre, y puede ser implementado por cualquiera sin restricción. El formato OpenDocument pretende ofrecer una alternativa abierta a los formatos de documentos propiedad de un fabricante cuyos requisitos de licencia impiden su empleo a diversos competidores.

Open Document es el primer estándar para documentos ofimáticos implementado por distintos competidores, visado por organismos de estandarización independientes y susceptible de ser implementado por cualquier proveedor.

El formato Open Document para procesador de palabras (ODT) es el que cuenta en este contexto.

Como es un estándar abierto, es natural que varias aplicaciones sean las que utilicen este formato siendo el más destacado y principal el `writer` de `Open Office`. Entre los otros programas que manejan el formato ODT aparte de `write` tenemos: AbiWord, AjaxWrite, Google Docs & Spreadsheets, Kword, etc.

Writer es un procesador de textos que forma parte del conjunto de aplicaciones libres de oficina OpenOffice.org. Soporta el formato propietario .doc de Microsoft Word casi en su totalidad, además de otros formatos clásicos de documentos. Puede exportar a archivos PDF nativamente sin usar programas intermedios. Es multiplataforma como la suite ofimática OpenOffice, que lo compone.

2.5.4. HyperText Markup Language (HTML)

Es el lenguaje de formato por el cual están compuestos principalmente los documentos electrónicos de la Web por excelencia: "Las páginas Web". Html permite a la pagina tener formato e hiperenlaces a otras páginas Web. de allí su concepto:

HTML es el acrónimo inglés de HyperText Markup Language, que se traduce al español como Lenguaje de Marcas de Hipertexto. Es un lenguaje de marcas basado en SGML diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web.

Como este lenguaje esta en forma de texto plano, puede ser creado y editado con cualquier editor de textos básico, como puede ser Gedit de linux, el Bloc de Notas y el notepad++ de Windows.

Existen además, otros programas para la realización páginas Web o edición de código HTML, como por ejemplo Microsoft FrontPage, el cual tiene un formato básico parecido al resto de los programas de Office. También existe el famoso Dreamweaver, siendo uno de los más utilizados en el ámbito de diseño y programación Web. Estos programas se les conoce como editores WYSIWYG o What You See Is What You Get (en español: “lo que ves es lo que obtienes”). Esto significa que son editores en los cuales se ve el resultado de lo que se está editando en tiempo real a medida que se va desarrollando el documento.

2.5.5. Microsoft Compiled HTML Help (CHM)

Es un formato propietario de ayuda en línea desarrollado por Microsoft. Se publico por primera vez en 1997 como sucesor del formato Microsoft WinHelp format. Microsoft anuncio que debido a fallos de seguridad no van a seguir usando dicho formato a partir de Windows Vista. El substituto para la ayuda de Windows que proponen es Microsoft Assistance Markup Language.

Un archivo CHM básicamente consiste en la compilación de una serie de páginas Web escritas en un subconjunto del lenguaje html y una tabla de contenidos hiperenlazada. Este formato esta optimizado y fuertemente indexado para su lapida y fácil lectura. Los archivos que la componen están comprimidas por el formato LZX.

2.5.6. LaTeX (TEX)

Este formato consiste en un lenguaje de marcados formado por un gran conjunto de macros de TeX. El archivo fuente consiste en uno o más archivos planos ASCII en donde se hayan las instrucciones Latex.

"LaTeX está basado en el lenguaje de composición de Donald E. Knuth's llamado TeX" [2].

En éste contexto está clasificado como un lenguaje de formato.

TeX es a la vez un lenguaje de composición tipográfica y un lenguaje de programación. Inicialmente inventado para componer documentos matemáticos de manera profesional, ahora se usa en muchas otras áreas. LaTeX también es un lenguaje de programación y de composición tipográfica. De hecho, es una versión simplificada de TeX que permite manipular instrucciones del más alto nivel, como HTML es una versión simplificada de SGML.

Con LATEX podemos crear cualquier tipo de documento, al igual que los procesadores de textos más conocidos, pero la filosofía de trabajo es radicalmente distinta. En lugar de realizar una edición visual del documento, se realiza una descripción estructurada del mismo mediante instrucciones del lenguaje Latex, es decir, se indica qué significa cada elemento del texto, en lugar de cómo debe aparecer impreso. La decisión sobre el cómo la toma después un *procesador de LATEX* de manera automática, liberándonos de ese trabajo.

Un archivo fuente TeX o LaTeX tiene que ser compilado. El resultado de esta compilación es el formato DVI, que puede leerse desde cualquiera plataforma. El archivo DVI producto de la compilación puede ser convertido a PDF o PS.

[2] http://www.latex-project.org/intro.html

2.6. Protección Legal de Documentos Electrónicos

Dadas las características de accesibilidad a través de Internet, es evidente que existe un gran riesgo para los autores y editores de libros electrónicos en frente de reproducciones fraudulentas o copias no autorizadas de dichos libros. En efecto, si cualquier persona pudiera descargar un libro completo a su computadora sin ninguna restricción para su uso posterior, ello facilitaría las reproducciones ilegales. Así por ejemplo podrían reproducirse cuantas copias se quisiera a fin de venderlas, modificarlas, etc, sin necesidad ni siquiera de tener que reescribir el contenido del libro, producir planchas de impresión u otros tantos inconvenientes o trabas que tienen que superar los infractores de derechos de autor de libros físicos.

El derecho de autor o derecho a la propiedad intelectual es una preocupación que existe desde la aparición de las primeras civilizaciones. Desde entonces se han hecho leyes que protegen los derechos de los autores sobre todas las obras del ingenio de carácter creador, ya sean de índole literaria, científica o artística, cualesquiera que sea su género, forma de expresión, mérito o destino.

Garantizar el derecho de autor siempre ha sido un reto, pero con la aparición de Internet el problema se agrava debido a las características de Internet, de los documentos digitales, y a los hábitos de los usuarios. A continuación se mencionas algunos de los mas importantes retos que se deben afrontar para garantizar la propiedad intelectual con respecto a los documentos digitales e Internet.

- Uso de la tecnología para proteger los derechos de los autores sobre sus obras.
- Creación de leyes aplicables, ya que el carácter mundial de la Internet hace que en la transmisión de una obra protegida puedan entrar en juego múltiples ordenamientos jurídicos.
- Definir la responsabilidad civil y penal en caso de infringir la ley.

Las limitaciones físicas de los libros tradicionales hacen más difíciles las reproducciones no autorizadas, pero con la digitalización e Internet estas barreras desaparecen y hacen más vulnerables los derechos de los autores. Las bibliotecas tradicionales pueden lograr su propósito de compartir la información a los interesados sin comprometer, al menos no tan fácilmente, la propiedad intelectual de las obras que posee.

Para lograr un implementación de seguridad sobre los documentos se ha hecho uso de algunos mecanismos, como por ejemplo algunas protecciones y restricciones que permite el formato PDF, pero lamentablemente existe software capaz de romper esa seguridad. Microsoft tiene un formato para documentos denominado LIT, el archivo está cifrado con dos contraseñas, una dependiente del lector y otra de un dispositivo electrónico para su lectura. De esta forma, el archivo, aunque fuese copiado a otro dispositivo, no podría ser leído en él,

pues no se encontrarían las dos claves de forma simultánea. Un solo cliente puede activar su cuenta en dispositivos diferentes, ésta activación se produce a través de cuentas Passport de Microsoft y es obligatorio realizar la compra a través de Internet.

En cuanto a leyes que sean aplicables tenemos que ante tanta diversidad todavía no se ponen de acuerdo en el sector jurídico acerca de cómo deben ser exactamente la leyes para regular las actividades de internet y los derechos de autor. Solo se puede hablar por los momentos de convenios internacionales, algunas leyes o reglamentos que varían mucho en cada país o grupo de naciones como la Unión Europea.

Vemos que se hacen intentos para lograr implementaciones de seguridad, pero aún queda mucho camino por recorrer. Internet es un medio de comunicación global y una plataforma de comercialización de bienes y servicios a gran escala. No posee una estructura definida y constante, lo que le permite evolucionar de muchas maneras, absorbiendo nuevas tecnologías . Nos encontramos ante una nueva realidad que ha superado culturas e idiomas originando una realidad paralela en la que salimos beneficiados pero que también nos ha traído otros problemas.

Con respecto a los buscadores y los documentos digitales hay un asunto curioso que tiene que ver con la parte legal y comercial. Buscadores como Google, Yahoo, o Altavista utilizan la información de las páginas Web para luego ofrecer resultados en las consultas tal como se explicó en el capítulo 1, realizando así una actividad lucrativa, ya que se financian a través de la colocación de publicidad en sus sites, con la reproducción parcial de la obra de otros autores. Por ello, estarían vulnerando derechos de reproducción y sin retribuir ningún beneficio económico a los autores.

Capítulo 3

Búsqueda en Contenido de Archivos

Este capitulo trata sobre las aplicaciones que realizan búsquedas dentro de los documentos electrónicos almacenados en una computadora, la manera en que funcionan, los datos que leen de estos documentos como la metadata.

En este capitulo se muestra que aparte de buscar solo por nombres o contenido se produce todo un proceso de indexación y búsquedas inteligentes. Todo este sistema de buscadores permite obtener resultados inmediatos, acertados y localizados en varios otros puntos de almacenamiento como mensajes de correo electrónico, historiales de Internet , cache, etc.

3.1. Recuperación de Información

La recuperación de información, llamada en inglés Information retrieval (IR), es la ciencia de la búsqueda de información en documentos, búsqueda de los mismos documentos, la búsqueda de metadatos que describan documentos, o, también, la búsqueda en bases de datos , ya sea a través de Internet, intranet, para textos, imágenes, sonido o datos de otras características, de manera pertinente y relevante.

La IR es un estudio interdisciplinar. Cubre tantas disciplinas que eso genera normalmente un conocimiento parcial desde tan solo una u otra perspectiva. Algunas de las disciplinas que la participan de estos estudios son la psicología cognitiva, la arquitectura de la información, diseño de la información, el comportamiento humano hacia la información, la lingüística, la semiótica, informática, biblioteconomía y documentación.

Los buscadores, tales como Google, Lycos y Copernic, son algunas de las aplicaciones más populares de la recuperación de información. Básicamente hay que construir un vocabulario, que es una lista de términos en lenguaje natural, un algoritmo que incluya las reglas lógicas de la búsqueda Tabla de verdad y una valoración de los resultados o cantidad de información lograda o posible. Este motor de búsqueda es pues el que permite plantear una pregunta con no menos de dos términos y mostrar los resultados mínimos.

3.2. Buscadores en Contenido

Es curioso, pero en los últimos años ha sido más fácil encontrar información en la inmensa y caótica Internet, con sus millones de páginas, que un archivo propio situado en algún lugar de la computadora.

Para buscar en la Web, simplemente se tiene que utilizar Google, Yahoo u otro buscador, escribir unas cuantas palabras clave, y se despliegan frente a la pantalla un largo listado de páginas Web que pueden ofrecer la información que se necesita.

Los buscadores de escritorio se empezaron a popularizar desde octubre del 2004, cuando Google, el líder en búsquedas de Internet, presentó la versión de prueba de su herramienta. El interés que generó, pese a que ya existían herramientas similares desde hace algún tiempo, llevó a Yahoo y MSN, sus rivales en Internet, a presentar sus propias aplicaciones para no dejarse ganar en ese terreno.

Pero la idea de crear índices de los archivos del PC no es nueva. De hecho en los años 80 Lotus introdujo una aplicación llamada Magellan, que si bien no fue muy famosa, sí ganó muchos fanáticos, sobre todo entre quienes trabajaban con grandes volúmenes de información. Pero Lotus se concentró más en otros productos, como la suite de oficina SmartSuite y las aplicaciones empresariales Notes y Domino, y abandonó Magellan.

Pequeñas compañías siguieron el camino trazado y crearon interesantes aplicaciones, pero no lograron generar mayor interés, en parte porque no eran gratuitas, y en parte porque hace varios años una computadora personal no podía almacenar muchos documentos, la Web no existía o apenas estaba surgiendo, el correo electrónico era para unos pocos privilegiados y, por tanto, una herramienta de este tipo no era tan necesaria.

Las computadoras hoy en día están llenas de documentos de texto, hojas de cálculo, canciones en formato MP3, presentaciones en PowerPoint o PDF, videos y archivos de todo tipo; y cuando se necesita encontrar alguno de ellos, lograrlo puede ser un verdadero reto con la herramienta de búsquedas del sistema operativo, probablemente no se recuerda el nombre del archivo ni su ubicación. Así, el proceso dura varios minutos, y en algunos casos puede resultar infructuoso.

Por otro lado, si se requiere encontrar mensajes de correo electrónico, las búsquedas del sistema operativo no funcionan, y los programas de correo, como Outlook u Outlook Express, generalmente son muy lentos e ineficientes para encontrar los mensajes.

Pero en los últimos años las cosas empezaron a cambiar. Las compañías de búsquedas crearon aplicaciones gratuitas que permiten examinar en la computadora de manera tan fácil como en la Red, que no se limitan a archivos y carpetas sino que revisan en los mensajes de correo o incluso en los historiales de navegación Web.

Además, no buscan solo con base en los nombres de los archivos, y lo mejor: encuentran lo que se necesita en fracciones de segundo. A estos programas se les empieza a conocer como buscadores de escritorio, por su nombre en inglés (desktop search).

Los tres grandes rivales en el campo de los buscadores Web son los mismos que han tomado este nuevo frente de batalla: Google, Yahoo y MSN (de Microsoft), sus herramientas son gratuitas, ya están disponibles en varios idiomas (incluido el español), son fáciles de usar y, gracias a la dura competencia, prometen seguir mejorando.

Además de los tres clásicos líderes de los buscadores Web, otras relativamente pequeñas compañías de software ya han creado aplicaciones similares, con sus propias particularidades y en algunos casos tan especializadas y útiles para ciertas exigencias, que no todas son gratuitas. AskJeeves, HotBot, Copernic, Blinkx y Filehand son algunas de ellas.

Entre las ventajas de los buscadores de escritorio tenemos: reducen considerablemente el tiempo que los usuarios gastan en nombrar lo mejor posible sus archivos, organizarlos en lugares específicos del disco duro y, sobre todo, en buscarlos. Y además, a esto se suma una mayor productividad, pues se minimizan notablemente los riesgos de tener que repetir el trabajo de documentos antiguos que serían casi imposible de encontrar dependiendo de como se encuantre saturada la computadora.

3.3. Funcionamiento

Lograr éstas búsquedas eficientes y en tiempo récord dentro de la computadora puede resumirse en que los buscadores "memorizan" los contenidos del disco duro, incluso los que se van creando. Se hace uso del mismo razonamiento con el que están hechos los buscadores de páginas Web.

En otras palabras: en lugar de hacer el proceso de buscar los archivos cuando el usuario requiere encontrar un archivo, previamente escanean los archivos de la computadora y crean un índice de ellos a partir de las palabras que hay en ellos. Dependiendo del buscador y de lo que contenga la computadora, la construcción del primer índice puede tardar entre varios minutos o varias horas.

Por ejemplo, mientras con el modelo tradicional (como la herramienta de búsqueda básica de Windows) buscar un documento que incluya una palabra especifica puede tardar entre muchos minutos, y es posible que no consiga respuesta ni tratándose de un elemental archivo ASCII plano, mientras que con aplicaciones como Google Desktop Search u otros buscadores de escritorio el procedimiento tarda menos de pocos segundos y las posibilidades de éxito si son considerables.

Buscar un archivo con una de éstas herramientas es muy sencillo: Generalmente se es-

criben las palabras clave en la casilla del programa, a los pocos segundos aparece el listado de resultados (organizado por fecha, relevancia o formato) y de inmediato se puede dar un clic sobre el archivo elegido para abrirlo.

3.4. Seguridad

No todo es perfecto, tener toda la información a la mano genera riesgos de seguridad. Algunas de éstas herramientas pueden poner archivos personales o confidenciales en manos de alguien que acceda a la computadora y logre vulnerar alguna seguridad. También se han presentado fallas como la de la primera versión de prueba de Google Desktop Search, que indexaba archivos de Word y Excel protegidos y permitía su posterior lectura sin necesidad de digitar las contraseñas. también la versión actual de Google Desktop Search arroja los resultados a través de un servidor Web local para ser visualizados por un Browser, cosa que se presta a muchos ataques, aunque ellos aseguran que esto está previsto y es seguro.

Los desarrolladores están trabajando para resolver estos inconvenientes y aumentar los niveles de seguridad. Hoy en día para muchos de estos buscadores de escritorio los usuarios establezcan exactamente qué quieren tener en los índices y a la vez puedan proteger cierta clase de archivos. Por ahora, se recomienda usar estos buscadores con cautela en las empresas o en las computadoras personales que tengan información crítica o reservada. En algunos casos como el de las entidades gubernamentales de Estados Unidos esta prohibido el uso de éstas herramientas.

3.5. Aplicaciones más Populares

3.5.1. Google Desktop Search

Google Desktop es una aplicación de búsqueda que permite acceder fácilmente a la información almacenada en tu equipo y en la Web. Con esta aplicación, buscar información en los mensajes de correo electrónico, archivos, archivos de música y fotografías es tan fácil como buscar en la Web con Google [1].

El Cuadro de búsqueda rápida de google desktop es la forma más rápida de efectuar búsquedas en la Web y en el escritorio. Se escribe unas pocas palabras o letras en el cuadro y los primeros resultados aparecerán instantáneamente. Para mostrar el cuadro de búsqueda rápida, se pulsa dos veces la tecla Ctrl; para ocultarlo, presiona la misma tecla otras dos veces.

Google Desktop indexa de forma automática prácticamente cualquier tipo de archivo y permite buscar el texto completo de estos. Entre los tipos de archivo que indexa el progra-

[1] http://desktop.google.com/es/features.html

ma, se incluyen los siguientes:

- Gmail.
- Archivos de texto y código fuente.
- Archivos PDF y PS.
- Archivos HTML.
- Correo electrónico de Thunderbird.
- Documentos OpenOffice.org.
- Archivos de imagen y de música.
- Páginas manuales y páginas de información.
- Nombres de archivos y carpetas.

Cuando se efectúa una búsqueda con Desktop, se ve una lista de resultados. Cada resultado, al igual que ocurre con la búsqueda en la Web de Google, incluye el nombre del archivo y un breve fragmento con los términos de búsqueda resaltados. Si se desea, se puede filtrar por un determinado tipo de elemento, por ejemplo, mensajes de correo electrónico. Para ello, se usan los vínculos de la parte superior de la página de resultados de Desktop.

Google Desktop empieza a indexar los archivos del equipo inmediatamente después de haberlo instalado. Esta indexación, que tiene lugar una sola vez, se ha diseñado para que coexista perfectamente con el trabajo diario del usuario, así que se puede continuar trabajando mientras el sistema efectúa la indexación. En función de los archivos y de los otros elementos que se tengan en el equipo, este proceso puede durar varias horas. Cuando finaliza, Desktop asegura de que el índice esté actualizado: agrega los mensajes de correo electrónico que se van recibiendo, los archivos que se actualizan y las páginas Web que se visitan.

Google ofrece una integración con su buscador en Web pensando en qué sucede si la información que se buscaba ya se encuentra en el equipo pero el usuario no se había percatado de ello. Cuando hay resultados útiles en el equipo relacionados con la búsqueda en la Web de Google, Google Desktop incluye estos archivos en los resultados de la búsqueda.

Google Desktop crea copias (instantáneas) en caché de los archivos y de otros elementos cada vez que se modifican, y almacena éstas copias en el disco duro del equipo. Esta función permite usar Desktop para encontrar versiones anteriores de los archivos o localizar los archivos que se han eliminado por error.

3.5.2. Spotlight

Spotlight no es solamente un buscador de escritorio, sino un elemento del sistema operativo Mac OS X de Apple que puede ser aprovechado hasta por cualquier programa; es decir es muy amplio e integrado con el sistema operativo.

Fue introducido en la versión 10.4 del mencionado sistema operativo. Permite al usuario realizar una búsqueda rápida y amplia, incluyendo documentos, fotos, música, preferencias del sistema, e incluso palabras especificas dentro de documentos, además de poder ser especificada con atributos, como fecha de creación, fechas de modificación, tamaño, tipo, ect.

Con Spotlight se puede encontrar cualquier cosa en la computadora, mientras se va escribiendo lo que se quiere. Busca en todo el sistema desde una misma interfaz.

Spotlight para Mac OS X Tiger permite rastrear en un instante todos los archivos y aplicaciones y ver los resultados en cuanto se escribe la primera letra. Y como Spotlight indexa de forma transparente todos los archivos del ordenador en segundo plano, nunca se experimentan retrasos o pérdidas de rendimiento. Y cada vez que se hace un cambio, como añadir un archivo nuevo, recibir un correo electrónico o introducir un nuevo contacto, Spotlight actualiza automáticamente el índice, a fin de mantener los resultados de las búsquedas siempre a la última.

Con Spotlight siempre se encontrará lo que se busca, incluso aunque no se sepa dónde buscar. Una nueva adición permanente a la barra de menús de Tiger, el cómodo campo de búsqueda de Spotlight ofrece resultados al instante, incluyendo no sólo archivos, carpetas y documentos, sino también mensajes de correo, contactos de la agenda, calendarios de iCal, Preferencias del Sistema y demás aplicaciones. Spotlight incluso garantiza que los demás sólo buscan lo que puedan encontrar. Así, los archivos que no se deberían poder ver no aparecen jamás entre los resultados a las búsquedas de Spotlight, manteniendo los datos bajo privacidad.

Cuando se hace una búsqueda con Spotlight, en realidad se está accediendo a un completo índice, actualizado constantemente, capaz de ver todos los metadatos de los archivos, una ficha completa de toda la información almacenada en la Mac, incluyendo el tipo de contenido, autor, historial, formato, tamaño y muchos otros detalles. La mayoría de los documentos, incluyendo documentos Word de Microsoft, imágenes Photoshop y mensajes electrónicos, ya contienen completos metadatos. Y como Spotlight también indexa el contenido, los resultados incluyen hasta lo que aparece dentro de un archivo o documento, y no sólo en su título.

Con Spotlight se encontrarán cosas incluso aunque no se sepan muchos detalles. Por ejemplo, si se recuerda más o menos de cuándo se guardó o recibió algo, se puede utilizar periodos de tiempo relativos como la semana pasada o ayer para encontrarlo. Si no estás seguro del formato de un archivo, se selecciona una categoría como Película, Imagen o Documento. Spotlight muestra una lista de resultados organizados que se puede revisar hasta

encontrar lo que se busca.

Se pueden utilizar palabras descriptivas para obtener unos resultados mucho más precisos, incluso buscando en miles de archivos. Por ejemplo, para encontrar imágenes en formato retrato, se selecciona Imagen y Retrato. Para encontrar todos los archivos de un compañero, se escribe su nombre. Spotlight muestra los documentos que creó o editó, las imágenes que mandó por correo, los mensajes que escribió (y los que se le mandan) y sus datos de contacto. Obtienes los resultados en categorías ordenadas, más fáciles de examinar, revisar y seleccionar.

Gracias a la velocidad y flexibilidad de Spotlight se encuentran infinidad de maneras nuevas de organizar los archivos. Los resultados de una búsqueda y se pueden guardar como una Carpeta inteligente que se actualiza automáticamente a medida que se añaden o eliminan documentos del Mac. Las Carpetas inteligentes contienen varios archivos relacionados entre sí por un criterio de búsqueda en vez de un emplazamiento físico, con lo que un mismo archivo puede aparecer en diferentes Carpetas inteligentes sin moverse de su lugar original. No hace falta duplicar, cambiar o actualizar ningún archivo: las Carpetas inteligentes de Spotlight lo organizan todo para el usuario.

Gracias a la gran variedad de formatos que admite, se puede encontrar cualquier cosa en la Mac, desde texto oculto en un documento PDF hasta una foto tomada con una cámara determinada [42].

3.5.3. Copernic Desktop Search

Copernic Desktop Search es una aplicación rápida, eficiente que consume pocos recursos y que permite encontrar instantáneamente archivos, emails, imágenes, multimedia y más, que estén localizados en la computadora. Es totalmente gratuito, esta disponible solo para la plataforma Windows. permite instalarle plugins. A cambio no realiza búsquedas en Internet (Copernic desarolla otros productos para ese cometido).

Este software, como el Google Desktop, o el Microsoft Desktop Search, indexa los archivos que se tengan en el disco duro. Este índice es personalizable, es decir, se puede recoger en él sólo un tipo determinado de archivos con la información que más interese tener disponible a la hora de realizar búsquedas.

Destaca por su atractiva interfaz, fácil de usar, con búsqueda avanzada (por tamaño, fecha, formato, etc.) y desde la que se abren carpetas o archivos directamente. Ofrece rapidez y ligereza por su integración con otros programas a través de su Deskbar. Devuelve resultados inmediatamente. Para la búsqueda de fotos provee imágenes en miniatura y actualiza periódicamente de forma automática con los elementos nuevos o cambiados.

Permite escoger en que categoría ha de buscar las palabras que se le introduzcan, y tiene

unas pestañas que muestran el número de entradas encontradas en cada categoría. Además permite refinar la búsqueda según la pestaña que esté abierta, por ejemplo si se está en la de correo permite indicar el asunto, de, para, etc. Y si se está en la de archivos de música por ejemplo permite indicar el álbum, el artista, etc. También permite usar operadores booleanos en las palabras a buscar, y tiene un operador de cercanía para las palabras a buscar. Otra particularidad es que permite guardar las búsquedas para reutilizar después.

Una vez mostrado los resultados en formato de lista, permite ordenarlos por diferentes criterios, también en función de la categoría de lo encontrado como en el refinamiento de las búsquedas. Y muestra una pantalla de presentación preliminar del documento o correo electrónico, y deja ejecutar diferentes acciones sobre él, como borrar, abrir, responder, etc. Si por ejemplo se ha puesto dos palabras en la búsqueda permite saltar a donde están individualmente en el documento.

se maneja perfectamente entre los documentos de Office, PDFs y los formatos más habituales de imagen, audio y vídeo. Copernic es capaz de encontrar un archivo multimedia por sus metadatos, es bueno para encontrar imágenes.

Copernic Desktop Search puede también (opcionalmente) indexar el historial de navegación en la Web, así como también contactos y favoritos.

la última versión administra mejor los recursos del sistema, analiza más rápido el correo electrónico y los contactos (además de compatibilidad con el Mozilla Thunderbird), una interfaz mucho más adaptada y una mejora en la funcionabilidad que hacen de esta utilidad un logrado motor de búsqueda capaz de localizar cualquier archivo al instante.

3.5.4. X1

X1 es un software propietario que está disponible para probar o comprar [2]. es un excelente buscador de escritorio y muy completo.

X1 primero indexa todo el contenido de todos los discos duros del sistema. X1 indexa muchísimos tipos de formatos (Excel, Word, PDF, Powerpoint, Emails, Fotos, etc, etc). Este proceso dura varias horas la primera vez y luego las siguientes veces se actualiza en background.

X1 ofrece una sola interfaz que permite buscar a la vez en todos los formatos. Por ejemplo, si uno sabe que alguna vez recibió una información pero no recuerda si fue un e-mail, o un attachment, o un documento Word, etc., X1 busca a la vez en todos los formatos que se elija. No es necesario hacer 7 búsquedas separadas. X1 busca no solo en el título del documento sino que busca en el contenido también.

[2] http://www.x1.com/

X1 busca no solo en los emails sino también dentro de los attachments. Esta funcionalidad es muy útil. X1 posibilita no tener que estar guardando cada vez cada documento o e-mail en cientos de carpetas ya que es fácil encontrar todo después.

Los resultados de X1 aparecen tan rápido como se tipee la búsqueda. En tiempo real. No hay que esperar ni un segundo. El preview panel de resultados es excelente y facilita la búsqueda enormemente.

Como ocurre con cualquier software, hay quienes lo prefieren por encima de otros. hay que recordar que su fuerte desventaja es que no es gratis.

3.5.5. Windows Desktop Search

Como una de las tecnologías de búsqueda de Microsoft, Windows Desktop Search (Búsqueda en el Escritorio de Windows) es un software que ofrece todas las herramientas necesarias para encontrar y recuperar correos electrónicos, documentos y archivos dentro de un equipo o de una red corporativa. Windows Desktop Search se ofrece de forma gratuita junto a su Licencia de Microsoft Windows.

Permite a los usuarios buscar fácilmente en el contenido de las unidades de disco duro y en los mensajes de correo electrónico de sus equipos. Mejora la productividad de los usuarios individuales y empleados ya que les permite buscar de manera rápida y fácil aquello que necesitan.

Con Windows Desktop Search, puede realizar una sola búsqueda para encontrar prácticamente cualquier cosa en su equipo: mensajes de correo electrónico, citas del calendario, fotografías, documentos y mucho más, con la misma facilidad con que buscas en Internet. El punto de entrada principal de la Búsqueda en el escritorio es la Barra de escritorio de Windows, un cuadro de búsqueda que aparece en la barra de tareas de Windows. La Barra de escritorio de Windows le permite buscar en el escritorio e iniciar una búsqueda en Internet o en una intranet. Con la práctica funcionalidad para 'buscar mientras se escribe', puede ver los resultados al instante y con un solo clic, puede obtener la información que busca. También puede iniciar Windows Desktop Search desde un acceso directo que se agrega al menú Inicio.

Con la Búsqueda en el escritorio, puede buscar en el contenido de los tipos de archivo más comunes como por ejemplo archivos de texto, documentos de Word, hojas de cálculo de Excel, fotografías, videos, música y mucho más. Además, el producto presenta la posibilidad de ampliarse, de manera que puede optar por instalar varios complementos, disponibles, como el complemento IFilter de Adobe que también te permite buscar en el contenido de documentos PDF. La documentación de los IFilters se encuentra en MSDN y todo desarrollador puede escribir un IFilter para un tipo propietario de archivo o documento.

Un IFilter es un complemento que habilita al indexador de Windows Desktop Search a

abrir, leer e indexar el contenido de los tipos de archivo nuevos que, de otra forma, no podría indexar. El servicio de índice de búsqueda de Windows original que se entrega con Windows también es compatible con esta tecnología flexible, además de hacerlo con SQL Server y SPS.

Asimismo, la Búsqueda en el escritorio realiza búsquedas en la información de metadatos en los títulos de temas, imágenes y archivos de video, así como en archivos ejecutables de programas.

También puede buscar en el contenido de todos los elementos de Microsoft Office Outlook y Microsoft Outlook Express.

Windows Desktop Search también es compatible con entornos empresariales de Tecnologías de información, ya que permite a los usuarios utilizar todas las capacidades en su entorno laboral, a la vez que se ajusta a las políticas y a la seguridad del administrador interno.

Cuando se instala Windows Desktop Search, se agrega un acceso directo al menú Inicio y a la barra de tareas de todos los usuarios del sistema. La Barra de escritorio de Windows automáticamente queda disponible para todos los usuarios en sus respectivas barras de tareas, salvo que se deshabilite mediante directiva de grupo. Windows Desktop Search se ejecuta automáticamente para cada usuario cada vez que dicho usuario inicia sesión en el equipo. Cada usuario mantiene su propio índice y busca elementos dentro de su propio ámbito.

De forma predeterminada, se indexará el contenido de la carpeta 'Mis Documentos'de los usuarios y del correo electrónico que se almacena de forma predeterminada. Los usuarios pueden ir al menú Opciones de indexación para especificar exactamente las carpetas de Outlook que se indexarán, además de las carpetas locales específicas.

El archivo de índice es ilegible, de manera que si un usuario se cruza con este archivo en su equipo, no podrá leer su contenido en un formato texto claro. Se puede cifrar el archivo de índice. Para cifrar el índice de Windows Desktop Search de manera segura, deberá colocarlo en el Sistema de cifrado de archivos (EFS). La finalidad de esta implementación es contribuir a proteger el contenido del archivo de índice del equipo del usuario y dificultar la intención de piratas informáticos de seguir conductas maliciosas con el archivo de índice de un usuario. El archivo de índice se almacena en C:/Archivos de programa/MSN Toolbar Suite/DS.

Parte III

Marco Aplicativo

Capítulo 4

Adaptación XP

El presenta capítulo trata sobre la programación XP y la aplicación.

Sobre XP se plantea la manera en que se adopta la metodología para la realización del sistema. Sobre la aplicación se explica en qué consiste y cómo se concibe.

4.1. Adaptación del Proceso de Desarrollo XP

4.1.1. Programación Extrema (Extreme programming)

XP (eXtreme Programming) se trata de una metodología ágil en contraposición a las metodologías pesadas como RUP. Se basa en la simplicidad, la comunicación y la retroalimentación o reutilización del código desarrollado. No se enfoca en la documentación sino en los requerimientos comunicados por el cliente.

El objetivo principal que se persigue es la satisfacción del cliente[1], por eso tiene mucha importancia la comunicación con los usuarios o clientes. Esta comunicación se va a soportar principalmente en las *historias de usuario* (UserStories) cuando proviene desde el cliente, y de las *entregas* y versiones parciales del sistema cuando la comunicación es hacia el cliente.

4.1.2. Iteraciones

El desarrollo esta basado o compuesto principalmente por *iteraciones*. La idea de las iteraciones es ir trabajando sobre versiones pequeñas o parciales del sistema hasta llegar al producto final. En el desarrollo de la aplicación se reciben los requerimientos y la retroalimentación progresivamente. En éste sistema en las primeras iteraciones se trata de realizar una arquitectura de sistema que pueda ser utilizada durante el resto del proyecto.

Las iteraciones pueden ser de dos tipos principalmente: por objetivos o por lapsos de tiempo. En el desarrollo de este sistema las iteraciones están basadas en lapsos de tiempo

estimados a una semana. Durante el tiempo fijado para cada iteración se realizan las implementaciones indicadas en las historias de usuario que abarque.

4.1.3. Historias de Usuarios

Las historias de usuario son la técnica utilizada para especificar los requisitos del software, éstas tienen el propósito de describir los requerimientos de los clientes. El contenido de las historias de usuario provienen del cliente.

En cada iteración se lleva a cabo un grupo de historias, es decir, en cada iteración se trabaja sobre uno o mas requerimientos, tal como se puede apreciar en la tabla 4.1.

Iteración 0	Historia de usuario 1 Historia de usuario 2 ...
Iteración 1	... Historia de usuario 5 Historia de usuario 6 ...

Tabla 4.1: Iteraciones e historias de usuario

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en un tiempo corto.

Cabe destacar que se tienen tres variantes de historia de usuario para los requerimientos:

- Nueva.
- Corrección.
- Mejora.

Con respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no un único formato a seguir.

En esta aplicación se utilizará el formato mostrado en la tabla 4.2:

Id	Fecha	Requerimiento	Tipo

Tabla 4.2: Formato de historias de usuario

4.1.4. Adaptación de las Tareas XP

La aplicación de las tareas XP busca lograr con éxito la alta comunicación con el cliente y la agilidad en el proceso de desarrollo.

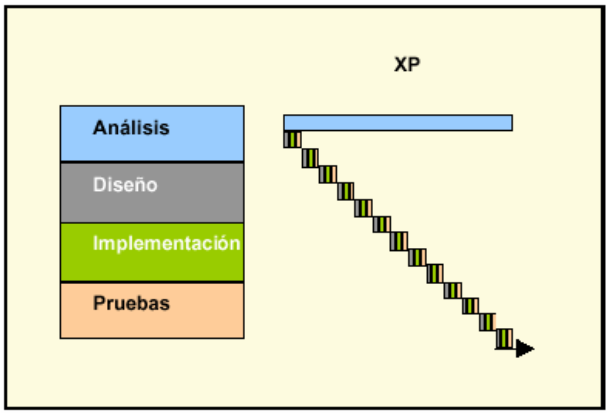

Figura 4.1: Desarrollo en XP

Se tienen cuatro tareas fundamentales (ver figura 4.1) durante las iteraciones las cuales serán adaptadas de la siguiente manera:

- **Planificación**: Para la planificación se utilizan las *historias de usuario* adaptado según ha sido explicado anteriormente.
- **Diseño**: En el diseño se hace las definiciones y diagramas de clases (desarrollo orientado a objetos). También se hacen otros diagramas como el diagrama de tablas de la base datos o sino algún otro diagrama que permita comprender y solucionar el problema.
- **Codificación**: El código de implementación del sistema se irá registrando y sincronizando constantemente en un servidor de control de versiones(*subversion*). Para este Trabajo Especial de Grado se extraerán y mostrarán partes del código fuente que sean esenciales en la comprensión del sistema y la solución a los requerimientos de las historias de usuario.
- **Pruebas**: Las pruebas serán de aceptación en donde el usuario o cliente pone a prueba el sistema y verifica que hayan quedado cubiertos los requerimientos. También se realizaran pruebas unitarias y de funcionalidad.

Es posible que en algunas iteraciones no se tengan desarrolladas todas las cuatro tareas. Esto se debe a las actividades que involucran determinadas historias de usuario durante una iteración.

4.2. Requerimientos Generales del Sistema

La aplicación que se va a realizar es principalmente un motor de búsqueda sobre un repositorio de publicaciones digitales. Los usuarios del sistema deben tener una experiencia similar al uso de los motores de busqueda de internet y de escritorio.

Existen algunas diferencias con respecto a los buscadores ya conocidos debido al contexto y el alcance de la aplicación, ésta se va a hacer para la facultad de Ciencias y solo contempla las publicaciones de los estudiantes por ahora. Aún así se tiene especial cuidado en desarrollar una aplicación incrementable y escalable.

4.2.1. Requerimientos Funcionales

Entre los requerimientos funcionales mas notables y generales del sistema de repositorio tenemos los siguientes:

- Capacidad de almacenar los documentos digitales y su información asociada.
- Hacer busquedas básicas y avanzadas.
- Obtener los documentos según los criterios especificados.
- Ordenar los documentos obtenidos según una relevancia acertada.
- Ofrecer los resultados de la consulta con un buen grado de usabilidad.

4.2.2. Requerimientos No Funcionales

Los requerimientos no funcionales básicamente están determinados por la eficiencia que se necesita para este tipo de sistemas, y son los siguientes.

- Las tecnologías de desarrollo deben ser no propietarias.
- Que funcione sobre una plataforma robusta, eficiente y escalable.
- El sistema debe ser de gran eficiencia y velocidad de respuesta.

Estos requerimientos están cubiertos por las tecnologías referenciadas en la propuesta el principio de este trabajo.

4.2.3. Aspectos Determinates del Sistema

Cabe destacar dos aspectos cruciales del sistema:

1. Que la velocidad de respuesta de la aplicación sea la mas rápida y óptima posible.
2. Que muestre de primero los documentos considerados mas relevantes.

El primer aspecto le otorga mas complejidad al desarrollo del sistema, ya que plantea el reto de conseguir la forma mas eficiente en tiempo y espacio para lograr la implementación de los requerimientos. Esto implica la elaboración de algoritmos y estructuras que van mas allá de los que pudieran surgir del sentido común, se recurre por ejemplo a creación de estructuras auxiliares o redundantes como la tabla inversa de palabras explicada en la sección 1.4.2. Otro reto importante es lograr el equilibrio entre la eficiencia en tiempo y la eficiencia en espacio, ya que la ganancia de una implica la disminución de la otra, por ejemplo un algoritmo que utilice solo el espacio esencial de los archivos con su metadata haría imposible la existencia de los motores de busqueda que deben buscar sobre enormes cantidades de documentos; y por otro lado abusar de estructuras auxiliares, redundancia, e información extra, haría crecer muchísimo la base de datos hasta un punto imposible de implementar.

El segundo aspecto implica hacer un algoritmo elaborado que se encargue de situar a los documentos mas importantes de primero, ya que los usuarios revisan las primeras posiciones de los resultados con mayor expectativa y ganas; de no conseguir allí los documentos necesitados darán por frustrada la busqueda o será indicativo de que el buscador no es bueno. Para lograr un buen posicionamiento se deben seleccionar los criterios a tomar en cuenta de manera muy cuidadosa, con operaciones que pudieran ser complejas o no. Lo importante es que el ordenamiento sea acertado y este matemáticamente justificado. Por ejemplo lo mas lógico es que el ordenamiento sea calculado con valores numéricos que representan el peso de los criterios que se van a tomar en cuenta y la interrelación matemática entre ellos.

4.3. Metáfora del Sistema

Metáfora es la forma de transmitir cómo funciona el sistema. Algunas veces podremos encontrar metáforas en frases sencilla o diagramas. Las metáforas ayudan a cualquier persona a entender el objeto del programa.

Es sistema a desarrollar en general es un repositorio de publicaciones digitales integrado con un buscador web de los mismos, según las necesidades y características tratadas en la Propuesta de Trabajo Especial de Grado.

En la sección 1.4 se mostraron los componentes de un motor de busqueda. El sistema a desarrollar tiene practicamente los mismos componentes de un motor de busquedad de

internet salvo por el componente *araña o robot*. La no presencia de la araña se debe a que nuestro buscador no se desempeña sobre un medio tan extenso e indeterminado como internet, sino que actúa sobre un repositorio digital bien establecido y definido, en donde la información que aporta la araña en los sistemas de internet es obtenida por el sistema cuando se cargan los nuevos documentos. En otras palabras la información se obtiene cuando se carga el documento, y dado este proceso, ya no se necesita de un software que recorra todo el sistema para obtener esa información como sucede en el caso de internet y las arañas.

La mejor metáfora a utilizar para la comprensión del sistema es el diagrama mostrado en la figura 4.2 en el cual se muestran las vistas y los componentes módulos principales del software.

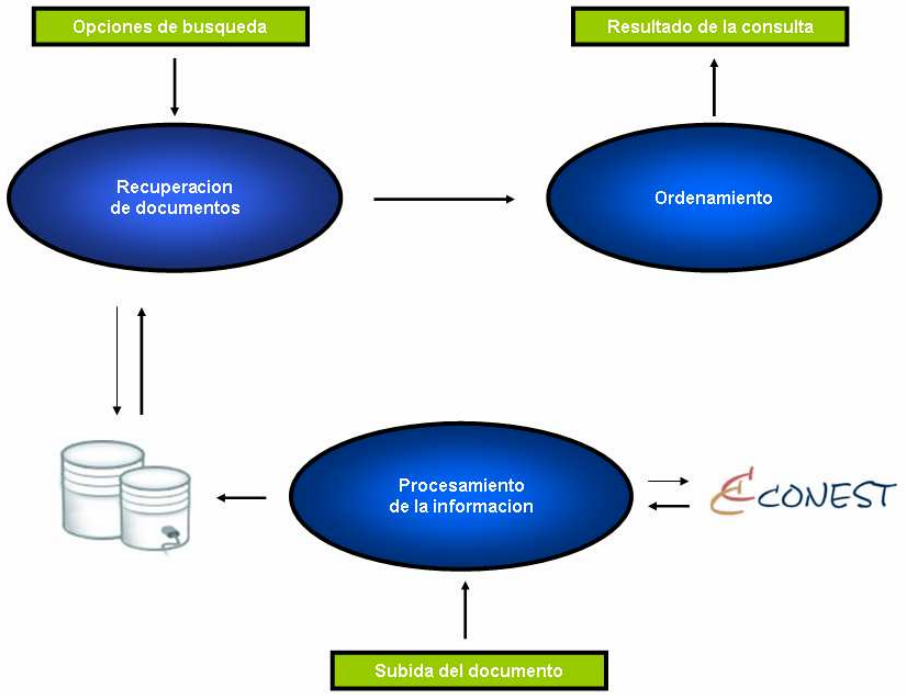

Figura 4.2: Componentes del sistema a desarrollar

En este diagrama las vistas están representadas por los rectángulos verdes, mientras que la persistencia y el sistema CONEST están representados mediantes los iconos correspondientes. Los módulos principales del sistema están representados por los óvalos azules, a su vez se muestran los flujos de datos entre los componentes del sistema. A continuación se explicará qué hace cada modulo del sistema:

- **Procesamiento de la información**: Es la parte en la que se sube el documento al sistema, una ves adquirido el archivo y algunos datos del estudiante, el sistema le pide a CONEST toda la información asociada con el documento y se va llenando la

metadata, luego el archivo binario es sometido a la extracción de texto puro, luego se procesa este texto para la obtención de todas las palabras distintas con sus respectivas frecuencias en el contenido. luego finalmente se termina de calcular toda la metadata en la que también se encuentran los pesos de relevancia. una vez obtenido toda la metadata, se procede a almacenar toda la información en la base de datos llenando archivo binario con metadata y demás campos y estructuras. En este modulo se llevan a cabo las tareas que hacen las arañas en los motores de búsqueda.

- **Recuperación de documentos**: Aquí se está en el momento después de que el usuario consultante establece sus criterios de búsqueda. Se procede a parsear el campo de texto de comandos para extracción y validación de la instrucción y palabras, también se preparan las restricciones en caso de tratarse de busquedas mas avanzadas. Luego estos argumentos obtenidos se trasforman en una llamada a una función que se encarga de comunicar estas busquedas especificadas al manejador de base de datos, para finalmente obtener todos los documentos bajo la forma de una estructura apropiada.

- **Ordenamiento**: En este módulo se aplica un algoritmo matemático de ordenamiento valiéndose de los campos que contienen los pesos numéricos que serán tomados en cuenta tales como frecuencia de palabras, valor numérico de la fecha y otros más. A este algoritmo se le pasa la estructura de documentos encontrados para efectuar el ordenamiento y luego éste devuelve una estructura con el orden definitivo con el que será mostrado al usuario. Este es una de los módulos que menor trabajo pudiera llevar en caso de implementar un algoritmo sencillo y eficaz, pero sin embargo es bastante importante ya que su funcionamiento y desempeño determina la calidad del buscador del repositorio digital. En caso de no ser muy bueno este módulo, el sistema practicamente no sería utilizable.

También la figura 1 en la parte de la propuesta muestra otra metáfora que representa al sistema a desarrollar.

Capítulo 5

Desarrollo de la Aplicación

En el capítulo 4 se trató sobre la adaptación de la aplicación XP para la realización de este sistema. Este capítulo trata sobre el desarrollo de la aplicación (implementación).

El presente capítulo esta dividido en iteraciones, durante las cuales se implementan las soluciones a los requerimientos.

5.1. Iteración 0

En la iteración 0 se definen e identifican las estructuras y el modelo que componen el repositorio digital de publicaciones. Sobre estas estructuras se va a basar el sistema y la aplicación de los algoritmos.

El modelo de datos básicamente consiste en el archivo de documento y una metadata asociada, pero además toma en cuenta la eficiencia, y por lo tanto, la necesidad de otras estructuras y datos para optimizar las consultas.

5.1.1. Planificación

En la tabla 5.1 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
1.00	17/09/2007	Definir las estructuras y las tablas del sistema	Nueva
2.00	17/09/2007	Identificar las tablas a usar de CONEST	Nueva

Tabla 5.1: Historias de usuario. Iteración 0

5.1.2. Diseño

En la figura 5.1 se muestra el diagrama de tablas que conforma las parte de persistencia del sistema. En ella se puede apreciar tanto la data esencial de los documentos como las estructuras de optimización de las consultas. El diagrama expresa de manera sencilla la naturaleza del problema y la solución.

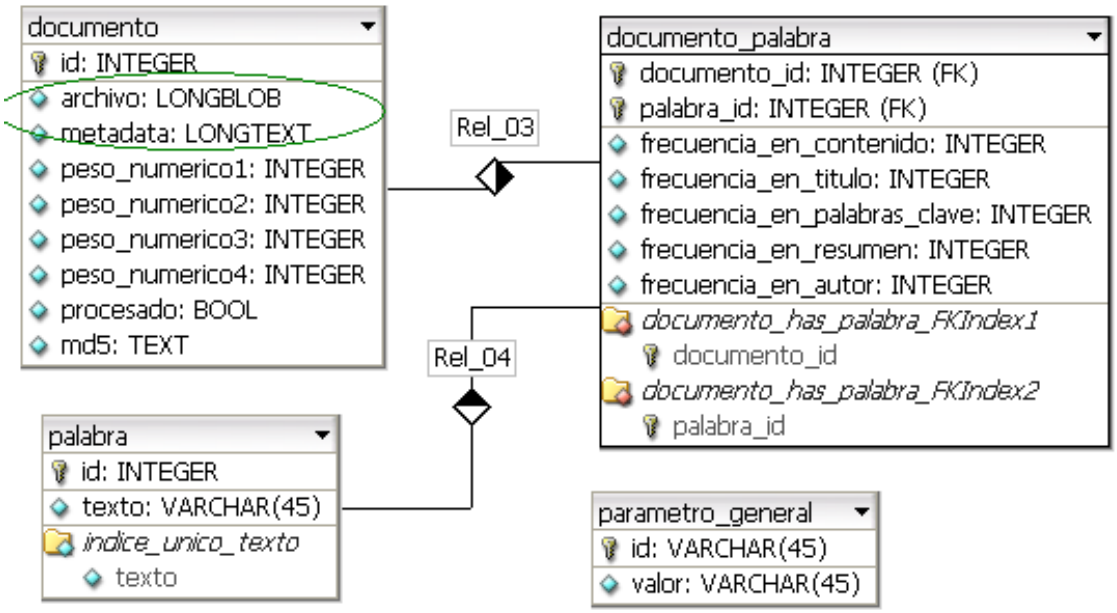

Figura 5.1: Diagrama de tablas del sistema

En la figura 5.1 del diagrama cabe destacar lo señalado en el círculo verde, se trata de la información esencial del sistema, es decir, el archivo de documento y su metadata asociada. Un sistema de busqueda puede funcionar con solamente esta tabla con estos dos campos señaladados, pero seria extremadamente ineficiente durante su desempeño y eficiente en el uso del espacio, tuviera que hacer búsquedas secuenciales a través de los documentos y extraer sus datos durante la ejecución en el mejor de los casos desde la metadata almacenada.

El modelo esencial consiste del archivo de documento y su metadata asociada. A continuación se muestra en qué consiste la metadata de documentos a usar en el sistema. Esta metadata esta en formato XML por los muchos beneficios que aporta, entre ellos la flexibilidad y lo compatible entre sistemas heterogéneos. En la figura 5.2 se muestra la estructura del archivo XML que representa a la metadata.

```
<titulo>valor</titulo>
<titulo_html>valor</titulo_html>
<fecha_publicacion>valor</fecha_publicacion>
<calificacion>valor</calificacion>
<tiene_premio>valor</tiene_premio>
<resumen>valor</resumen>
<resumen_html>valor</resumen_html>
<palabras_clave>valores</palabras_clave>
<nombre_archivo>valor</nombre_archivo>
<extension_formato_archivo>valor</extension_formato_archivo>
<!-- autores  -->
<autor>
  <nombre>valor</nombre>
  <correo>valor</correo>
  <telefonos>valor</telefonos>
  <eficiencia>valor</eficiencia>
  <promedio_general>valor</promedio_general>
  <promedio_ponderado>valor</promedio_ponderado>
</autor>
<autor>
```

Figura 5.2: Parte de la estructura XML de la metadata

En el modulo de procesamiento de la información del documento gran parte de la metadata se obtiene del sistema CONEST. En la figura 5.3 se muestran las tablas del sistema CONEST que están involucradas en el aporte de información necesaria para la metadata.

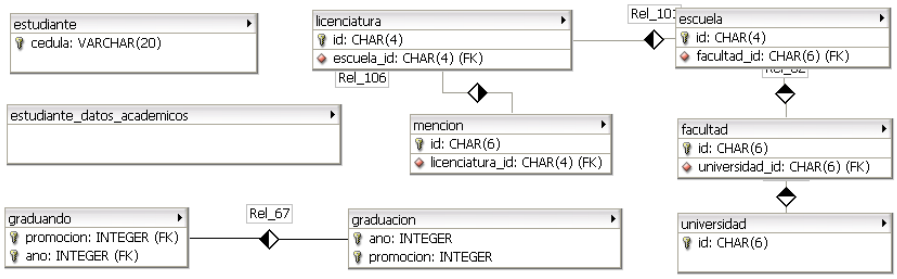

Figura 5.3: Algunas tablas del sistema CONEST involucradas

5.2. Iteración 1

En esta iteración crean las primeras versiones de las interfaces, los primeros algoritmos, y los ajustes a las estructuras de la iteración anterior. En esta parte se comienza con el código y la implementación, por eso se necesita empezar a hacer las interfaces para el desarrollo de las primeras partes del sistema, y el primer objetivo de la implementación es el modulo de carga del documento explicado en la sección 4.3. También está la mejora de la parte del modelo del sistema, ahora el objeto documento tiene atributos y metodos de acceso, y no sólo una metadata XML con rígido acceso, es decir ahora la metadata es muy manipulable y lo de XML queda encapsulado.

5.2.1. Planificación

En la tabla 5.2 se muestran las historias de usuario desarrolladas en esta iteración:

Id	Fecha	Requerimiento	Tipo
3.00	24/09/2007	Hacer las interfaces	Nueva
4.00	24/09/2007	Extracción del texto de los documentos	Nueva
1.01	24/09/2007	Estructuras del sistema	Mejora

Tabla 5.2: Historias de usuario. Iteración 1

5.2.2. Diseño

El modelo del sistema ahora queda representado por el diagrama de clases de la figura 5.4.

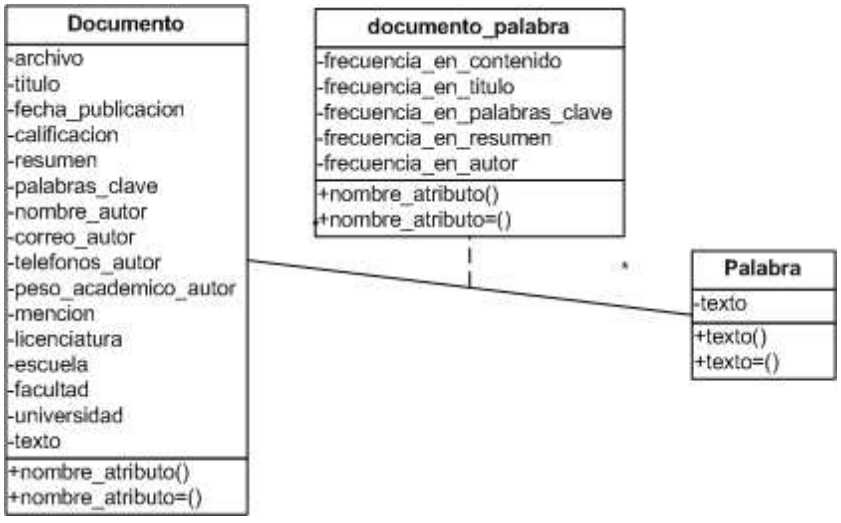

Figura 5.4: Diagrama de clases

Las interfaces siguen un patrón similar debido a que las vistas usan un mismo *Layout*, o sea, todas utilizan una misma plantilla y a su vez una misma hoja de estilos css. en la figura 5.5 se da una muestra de una vista representativa de las paginas web de esta aplicación:

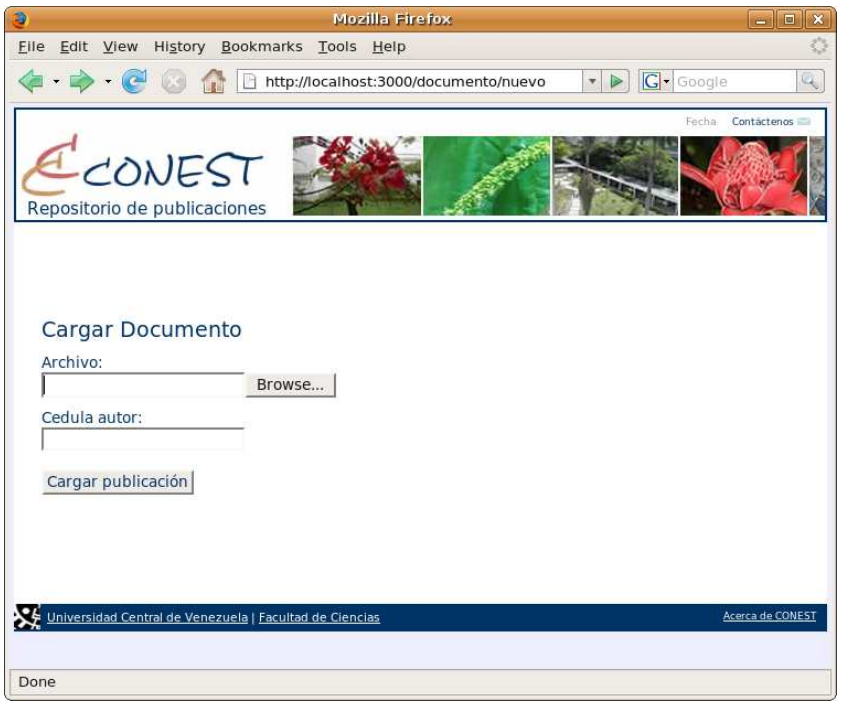

Figura 5.5: Vista de la carga de documentos electrónicos

5.2.3. Codificación

La implementación de la extracción del texto de un archivo (PDF, Word o texto) puede verse el fragmento de código de la figura 5.6.

```
113  def obtener_texto(documento)
114    #recibe al objeto documento con archivo y metadata seteados
115|   extension_formato_archivo = documento.nombre_archivo.split
   (".").last.downcase
116    if (extension_formato_archivo == "pdf")
117      nombre_archivo_binario = "./tmp/#{documento.nombre_archivo}".gsub(/[\s]
   +/,"_")
118      nombre_archivo_texto = "./tmp/#{documento.nombre_archivo}.txt".gsub(/[\s]
   +/,"_")
119      archivo_binario = File.open("#{nombre_archivo_binario}","wb"){|f|f.write
   documento.archivo }
120      system("pdftotext #{nombre_archivo_binario} #{nombre_archivo_texto}")
121      archivo_texto = File.new("#{nombre_archivo_texto}","r")
122      retorno = archivo_texto.read
123      #system("rm #{nombre_archivo_texto}") # TODO activar
124      system("rm #{nombre_archivo_binario}")
125      return retorno
126    end
127    if (extension_formato_archivo == "doc")
128      nombre_archivo_binario = "./tmp/#{documento.nombre_archivo}".gsub(/[\s]
   +/,"_")
129      nombre_archivo_texto = "./tmp/#{documento.nombre_archivo}.txt".gsub(/[\s]
   +/,"_")
130      archivo_binario = File.open("#{nombre_archivo_binario}","wb"){|f|f.write
   documento.archivo }
131      system("antiword #{nombre_archivo_binario} > #{nombre_archivo_texto}")
132      archivo_texto = File.new("#{nombre_archivo_texto}","r")
133      retorno = archivo_texto.read
134      #system("rm #{nombre_archivo_texto}") # TODO activar
135      system("rm #{nombre_archivo_binario}")
136      return retorno
137    end
138    puts "ADVERTENCIA: Tipo de archivo no soportado o sin extencion, se
   devolvio una cadena vacia"
139    return ""
140  end
```

Figura 5.6: Código de extracción de texto de un archivo PDF

El código anterior consiste básicamente en hacer llamadas al sistema. Por ejemplo en la línea 120 se usa un comando que transforma el contenido de un archivo PDF a texto plano, este comando esta disponible luego de instalar una aplicación llamada XPDF, Este comando necesita 2 argumentos que son el nombre del archivo PDF y el nombre del archivo de salida de texto.

El próximo extracto de código (figura 5.7) es una parte de la clase "Documento" en la cual se crean los *getter* y *setter* de los atributos a partir de los *xpath* de la metadata XML.

Este código genera los metodos accesores a partir de los xpath, esto significa que si la estructura XML cambia, sólo sera necesario actualizar la lista de *xpath* y en objeto tendrá los metodos correspondientes, esto solo es posible con el lenguaje que se está utilizando: `Ruby`.

```
17  elementos_xml = ["titulo", "fecha_publicacion", "calificacion"]
18  elementos_xml += ["autor/nombre", "autor/correo", "autor/telefonos", "autor/
  promedio_general", "autor/promedio_ponderado", "autor/eficiencia", "autor/
  tiene_premio"]
19  elementos_xml += ["mencion", "licenciatura", "escuela", "facultad",
  "universidad"]
20  elementos_xml += ["extension_formato_archivo", "nombre_archivo", "texto"]
21  elementos_xml += ["resumen", "palabras_clave"]
22
23  elementos_xml.each do |elemento_xml|
24    xpath_elemento = elemento_xml
25    nombre_metodo_get = elemento_xml.split("/").reverse.join"_"
26    nombre_metodo_set = nombre_metodo_get + "="
27    #
28    define_method(nombre_metodo_get){
29      return "metadata no cargada" if metadata.nil?
30      begin
31        valor_elemento = self.xml.root.elements[xpath_elemento][0] # accede al
  xpath indicado
32      rescue
33        valor_elemento = ""
34        puts "no se tiene el elemento en el xml, se devuelve una cadena vacia"
35      end
36      return "#{valor_elemento}"
37    }
38    define_method(nombre_metodo_set){ |argumento|
39      xml = self.xml
40      begin
41        xml.root.elements[xpath_elemento][0] #para que explote si falta
42        elemento = xml.root.elements[xpath_elemento] # accede al xpath indicado
43      rescue
44        elemento = xml.root.add_element("#{xpath_elemento}")
45      end
46      elemento.text = "#{argumento}"
47      metadata.replace(xml.to_s)
48    }
49  end
```

Figura 5.7: Código de los métodos de acceso a los atributos de la clase “Documento”

5.2.4. Pruebas

Las pruebas de aceptación para la interfaz se hicieron observando una pagina de carga prototipo que utiliza un css común. Esta pagina representa a todas las páginas que se van a realizar durante el desarrollo. Las primeras interfaces cubren la funcionalidad, pero se tiene que ir ajustando la simplicidad y la estética a posterior durante el desarrollo.

Para probar la extracción de texto, se hizo que el código tenga algunas lineas que generan salidas tanto en consola como hacia archivos de texto. Se obtuvieron buenos resultados apreciándose un texto limpio normalizado y fiel a contenido en el documento con formato.

5.3. Iteración 2

En esta iteración se continua con el desarrollo del modulo se carga de documentos. En la iteración anterior se hizo la extracción del texto, y alguno tratamiento de refinamiento, ahora se continua con la obtención de las frecuencias de palabras en el texto. Luego se desarrolla el Web Service que con el que se comunicara el sistema para obtener la mayor parte de la metadata. Finalmente terminar de almacenar toda la parte persistente durante el proceso de carga del documento.

5.3.1. Planificación

En la tabla 5.3 se muestran las historias de usuario desarrolladas en esta iteración:

5.00	01/10/2007	Obtención de las frecuencias de palabras en los textos	Nueva
6.00	01/10/2007	Crear los *Web Services* para la comunicación con CONEST	Nueva
7.00	01/10/2007	Almacenamiento total del objeto documento, palabras y las estructuras auxiliares	Nueva

Tabla 5.3: Historias de usuario. Iteración 2

5.3.2. Diseño

El sistema CONEST ofrece la metadata vía *Web Service* según los parámetros pasados por el sistema. En la figura 5.8 se observa el diagrama de clases del Web Service del lado de CONEST.

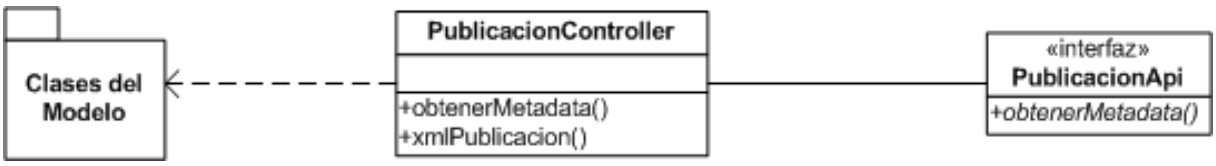

Figura 5.8: Diagrama de clases del Web Service de CONEST que aporta la metadata

5.3.3. Codificación

En el sistema CONEST se encuentra el Web Service, en `Ruby on Rails` se tiene que definir una interfaz a través de un API (figura 5.9) y luego la implementación a través de un controlador (figura 5.10).

```
1
2 class PublicacionApi < ActionWebService::API::Base
3   api_method :obtener_metadata, :expects => [:string], :returns => [:string]
4 end
5
```

Figura 5.9: Api de interfaz del Web Service de CONEST

```
2 class PublicacionController < ApplicationController
3   wsdl_service_name 'Publicacion'
4
5
6   def obtener_metadata(cedula)
7     begin
8       autor = Estudiante.find(cedula)
9       licenciatura = EstudianteEnLicenciatura.find(:first, :conditions =>
  ["estudiante_cedula = ?", autor.cedula], :order =>
  "ano_lectivo_ingreso").licenciatura
10      datos_academicos = EstudianteDatosAcademicos.find(:first, :conditions =>
  ["estudiante_cedula = ? AND licenciatura_id = ?", autor.cedula,
  licenciatura.id])
11      return xml_publicacion(:autor => autor, :licenciatura =>
  licenciatura, :datos_academicos => datos_academicos)
12    rescue
13      puts "Error en el webservice al generar la metadata"
14      return ""
15    end
16  end
```

Figura 5.10: Implementación a través de controlador del Web Service de CONEST

Una vez creado y puesto en funcionamiento el Web Servive de CONEST, el sistema procede a obtener ese servicio, mas especificamente la metadata en formato XML. El código para llamar a ese servicio y obtener la metadata está presente en la línea 6 y en la línea 28 de la figura 5.11, la primera crea un objeto que representa al Web Service “Publicacion” de CONEST, y la otra hace la llamada al método del Web Service que aporta lo que se necesita.

```
2 class DocumentoController < ApplicationController
3
4   CantidadDeDocumentosPorPagina = 5
5
6   web_client_api :publicacion, :soap, "http://localhost:3001/publicacion/api"
7
8
9   def nuevo
10  end
11
12  def index
13    redirect_to(:action => 'busqueda_sencilla')
14  end
15
16  def busqueda_sencilla
17    flash[:desplazable] = false
18  end
19  def busqueda_avanzada
20    flash[:desplazable] = false
21  end
22
23
24  def crear
25    @documento = Documento.new
26    archivo = params[:archivo]
27    begin
28      metadata = publicacion.obtener_metadata(params[:cedula])
29      if metadata.length < 1
30        flash[:mensaje_error] = "CONEST no pudo generar una metadata para el
  documento"
31        redirect_to(:action => "nuevo")
32        return
33      end
34    rescue
35      flash[:mensaje_error] = "Ocurrió algo terrible al obtener la metadata
  desde CONEST"
36      redirect_to(:action => "nuevo")
```

Figura 5.11: Código del lado del sistema para obtener la metadata

La obtención de la frecuencia de cada palabra dentro de un texto se basa en el método de la figura 5.12. Aquí primero se crea un *Hash* (Estructura que colecciona elementos compuestos del tipo clave-valor), luego se obtiene un arreglo de las palabras distintas (línea 200), luego se itera por estas palabras y se cuenta las cantidad de veces que esta presente la palabra actual en la lista de palabras pasadas por argumento, en el Hash se guarda la entrada `palabra_actual =>frecuencia` (línea 202). Luego se eliminan las palabras del arreglo de palabras que sean iguales a las de la palabra actual para ir reduciendo el conjunto para las nuevas búsquedas.

```
198  def obtener_frecuencias(palabras) # recibe un arreglo de palabras de un texto,
   y  devuelve un hash { "palabra" => frecuencia } que indica las repeticiones de
   cada palabra distinta
199    tf = {}
200    palabras_unicas = palabras.uniq
201    palabras_unicas.each{ |palabra_unica|
202      tf[palabra_unica] =  palabras.find_all{ |pal| pal == palabra_unica }.size
203      puts " #{palabra_unica} #{tf[palabra_unica]}" #para ver que todo esta
   saliendo bien
204      palabras.delete_if{ |pal| pal == palabra_unica} #Elimina la palabra para
   reducir el arreglo de palabras
205    }
206    return tf
207  end
```

Figura 5.12: Método para obtener las frecuencias de las palabras en texto

5.3.4. Pruebas

Las pruebas de aceptación para la obtención de las frecuencias de las palabras se hizo colocando lineas de código que generaran salidas a archivos en donde salen las palabras y sus frecuencias de ocurrencias dentro del documento, luego se tomaban aleatoriamente las palabras y se pudo confirmar la veracidad de la salida que generaba el programa al comparar búsquedas en los documentos originales.

Para probar el almacenamiento total del objeto documento junto con las estructuras auxiliares, se hizo sucesivas consultas en la base de datos y querys para verificar que se almacenaban correctamente la metadata, las frecuencias de las palabras y que los archivos binarios se descargaban bien.

Las pruebas de aceptación para la comunicación con CONEST a través de los Web Services se hicieron observando la veracidad de la información de la metadata que devolvía el servidor de CONEST en el XML a partir de los datos parciales que se le enviaban (la cédula).

5.4. Iteración 3

Para la iteración ya se dispone del objeto *Documento* almacenado, por lo tanto se pueden realizar las consultas apoyándose en las estructuras auxiliares. Se usa la metadata solo en el caso de que las búsquedas incluyan frases.

En esta iteración se desarrolla tanto la búsqueda simple como la búsqueda avanzada en donde se especifican varias condiciones.

Las búsquedas implican tener ya resuelto e algoritmo de recuperación de documentos de la base de datos y el ordenamiento. Se llegó a la conclusión de que era necesario llevar a cabo estas dos funcionalidades en una sola pasada a nivel de recuperación de la base de datos, es decir, que la aplicación debe ser capaz de pedirle a la base de datos los documentos ya ordenados, por razones de eficiencia explicados en la sección 4.2.3.

Las búsquedas también implican la realización del manejo de los resultados ya obtenidos y su interfaz. Para el manejo de resultados de la consulta se utiliza un arreglo de claves primarias que se obtiene de la consulta hecha sobre la base de datos. Una vez obtenido ese arreglo de claves primarias, se tiene que desarrollar una funcionalidad bastante importante: *la paginación*. La paginación se encarga de ofrecer los resultados en varias vistas en lugar de traerlo todo en una sola página, luego debe ofrecer navegabilidad a través de esas páginas y traer la información de los objetos que corresponden a la pagina actual; para ello se decidió desarrollar un objeto *Paginador* que estará encargado de administrar la paginación.

5.4.1. Planificación

En la tabla 5.4 se muestran las historias de usuario desarrolladas en esta iteración:

8.00	08/10/2007	Paginación para los resultados de las consultas	Nueva
9.00	08/10/2007	Interfaz para búsqueda sencilla	Nueva
10.00	08/10/2007	Interfaz para búsqueda avanzada	Nueva
10.00	08/10/2007	Recuperación de documentos de la base de datos	Nueva

Tabla 5.4: Historias de usuario. Iteración 3

5.4.2. Diseño

La administración de la paginación la lleva a cabo el objeto *Paginador* mostrado en la página 5.13, éste objeto puede manejar cualquier tipo de objetos, no solo documentos. El paginador se inicializa y posee las claves primarias, la clase de objetos que maneja, el numero de objetos que aporta a cada pagina. además el paginador posee metodos útiles para las vistas como obtener la página actual, obtener los objetos de la página actual, la última

pagina, establecer la página actual, etc.

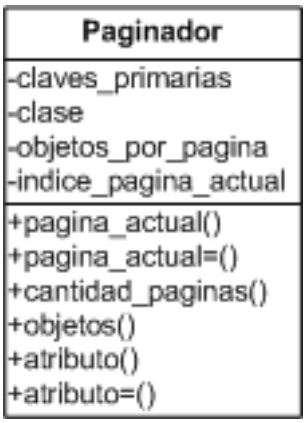

Figura 5.13: Diagrama de la clase *Paginador*

El aspecto de la interfaz de busqueda sencilla puede verse en la figura 5.14. Esta interfaz consta de una linea de comando en la que el usuario introduce lo que quiere. de esta linea de comando se obtienen las palabras y las frases encerradas en doble comillas. luso El sistema prepara y valida estas entradas para invocar al método *buscar* que llevará a cabo la petición a la base de datos.

Figura 5.14: Vista búsqueda sencilla

La interfaz de busqueda avanzada puede verse en la figura 5.15. Esta interfaz consta de varias opciones y varios campos en los cuales el usuario le especifica al sistema lo que quiere

con mas precisión. A diferencia de la busqueda sencilla, el sistema ahora obtiene mas datos y podría tardar mucho más debido a la complejidad que pudiera generar la consulta.

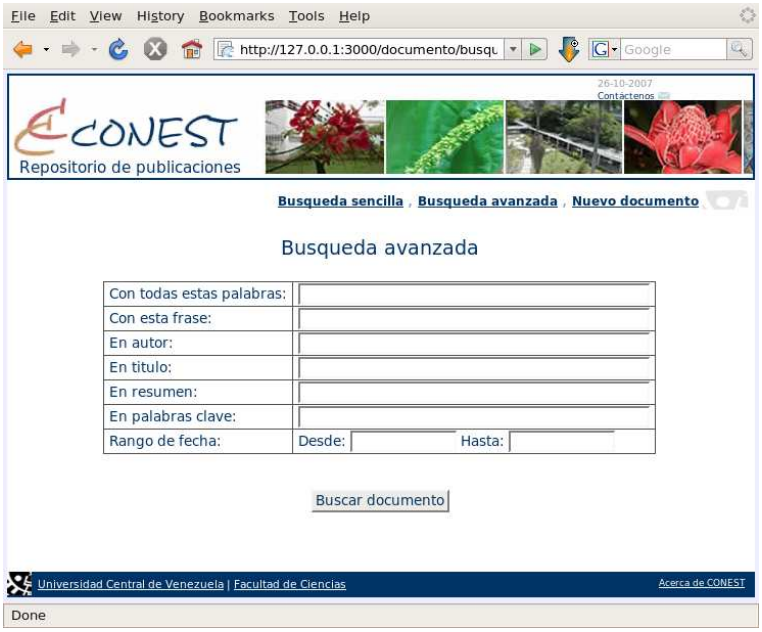

Figura 5.15: Vista búsqueda avanzada

5.4.3. Codificación

Parte de la implementación de la clase paginador se muestra en la figura 5.16, especificamente la inicialización y el método *objetos* que se encarga de devolver los objetos de la pagina actual.

```ruby
class Paginador

  attr_accessor :claves_primarias
  attr_accessor :clase
  attr_accessor :objetos_por_pagina
  attr_accessor :indice_pagina_actual


  def initialize(args)
    self.clase = args[:clase]
    self.claves_primarias = args[:claves_primarias]
    self.objetos_por_pagina = args[:objetos_por_pagina]
    self.indice_pagina_actual = 0
    self.indice_pagina_actual = -1 if !(self.claves_primarias.size > 0)
  end


  def pagina_actual
    self.indice_pagina_actual + 1
  end


  def pagina_actual=(i)
    return if self.indice_pagina_actual == -1
    if i < 1
      self.indice_pagina_actual = 0
    elsif i > self.cantidad_paginas
      self.indice_pagina_actual = self.cantidad_paginas() - 1
    else
      self.indice_pagina_actual =  i - 1
    end
  end


  def cantidad_paginas
    numero = self.claves_primarias.size / self.objetos_por_pagina
    numero += 1 if self.claves_primarias.size % self.objetos_por_pagina > 0
    return numero
  end


  def objetos
    li = indice_pagina_actual * objetos_por_pagina
    cantidad_objetos = objetos_por_pagina
    ls = li + claves_primarias.size % objetos_por_pagina if pagina_actual ==
  cantidad_paginas and claves_primarias.size % objetos_por_pagina > 0
    objs = []
    claves_primarias[li,cantidad_objetos].each{ | pk |
      objs << self.clase.find(pk)
    }
    return objs
  end


end
```

Figura 5.16: Código de la clase paginador

La busqueda sencilla de los documentos parte de las especificaciones de los usuarios a través de la interfaz correspondiente, luego los parametros son validados y preparados por

el método *buscar sencillo* del controlador, luego estos argumentos son enviados al método buscar del *ApplicationController*. La busqueda avanzada opera de la misma forma solo que el método que recibe y prepara los parametros del usuario es el método *buscar avanzado* del controlador.

Por lo tanto lo esencial de la recuperación de los documentos está en el método *buscar* de *ApplicationController* que se muestra en la figura 5.17. Este método obtiene los datos ya preparados y genera el SQL necesario para la recuperación de las claves primarias de los documentos a nivel de base de datos.

Despues de invocar al método buscar, los metodos de *buscar sencillo* y *buscar avanzado* instancian al paginador con las claves primarias recibidas. Luego se redirecciona a la interfaz *resultados de la busqueda* administrada por el paginador.

```
231   def buscar(args)
232     palabras = args[:palabras] # objetos palabra
233     fechas = args[:fechas] # en timestamp
234     frases = args[:frases] # texto simple sin las comillas
235     campos_especificos_palabras = args[:campos_especificos_palabras] # hash con
    el campo como clave y su arreglo de palabras objeto como valor
236     #
237     session[:paginador] = nil
238     establecer_maximos(:cantidad_palabras => palabras.size)
239     #
240     palabras_id = palabras.collect{|pal| pal.id}
241     if palabras_id.size > 0
242       sql_palabras_where = "AND palabra_id IN (#{palabras_id.join(', ')}) "
243     else
244       sql_palabras_where = " "
245     end
246     if fechas
247       sql_fechas_where = "AND peso_numerico1 BETWEEN #{fechas[0]} AND #{fechas
    [1]} "
248     else
249       sql_fechas_where = ""
250     end
251     if frases or frases.size > 0
252       frases.delete_if{|f|f.gsub(/[\s]+/,"").size < 1}
253       sql_frases_where = frases.collect{|frase| "AND metadata LIKE '%#{frase}%'
    "}.join(" ")
254     else
255       sql_frases_where = ""
256     end
257     if !campos_especificos_palabras # si es busqueda sencilla
258       sql_palabras_having = "AND COUNT(*) = #{palabras_id.size} "
259     else
260       sql_palabras_having = ""
261       campos_especificos_palabras.each{ |cp|
262         if cp[1].size > 0
263           puts cp[0]
264           sql_palabras_having += "AND SUM( frecuencia_en_#{cp[0]} > 0 AND
    palabra_id IN ( #{cp[1].collect{|p|p.id}.join(', ') }) ) = #{cp[1].size} "
265         else
266         end
267       }
268     end
269     #
270     sql =  "SELECT documento_id, #{formula_ordenamiento_sql} AS
    campo_ordenamiento "
271     sql += "FROM documento_palabra, documento "
272     sql += "WHERE 1 = 1 "
273     sql += sql_palabras_where
274     sql += sql_fechas_where
275     sql += "AND documento_id = documento.id "
276     sql += sql_frases_where
277     sql += "GROUP BY documento_id "
278     sql += "HAVING 1 = 1 "
279     sql += sql_palabras_having
280     sql += "ORDER BY campo_ordenamiento "
281 puts ("SQL buscar() >>>  " + sql)
282     rs = ejecutar_sql(sql)
283     claves_primarias = []
284     rs.each{|row|
285       claves_primarias << row[0]
286 puts "** El documento #{row[0]} dista del óptimo en #{row[1]}"
287     }
288     return claves_primarias
289   end
290
```

Figura 5.17: Método *buscar*

5.4.4. Pruebas

Las pruebas de aceptación para las interfaces fue la verificación de que las vistas tuvieran los campos correctos en los formularios y que esos datos hayan sido enviados y tratados como se esperaba por la aplicación, siendo la busqueda avanzada la mas sometida a pruebas debido a su complejidad.

Las pruebas de aceptación para la recuperación de documentos de la base de datos se hizo comparando los resultados del algoritmo de busqueda contra consultas SQL sobre la base directamente, llegando a la conclusión de que la aplicación hace las búsquedas correctamente.

Las pruebas de aceptación para la paginación se hizo haciendo consultas que devolvían tanto una numerosa cantidad de documentos como uno solo, y reduciendo los valores de máximos documentos por página. Se ratificó que el sistema pagina de manera correcta y eficiente los documentos recuperados en la vista de resultados.

5.5. Iteración 4

Para esta iteración ya se tiene un buscador que recupera documentos tanto en la busqueda sencilla como en la búsqueda avanzada. Para esta iteración el objetivo principal es otorgarle ordenamiento al resultado de las consultas basado en formulas matemáticas adecuadas.

5.5.1. Planificación

En la tabla 5.5 se muestran las historias de usuario desarrolladas en esta iteración:

11.00	15/10/2007	Ordenamiento de las consultas	Nueva

Tabla 5.5: Historias de usuario. Iteración 4

5.5.2. Diseño

El ordenamiento se hará basado en el modelo matematico de "balance lineal simple" pero antes de abordar esta teoría se mostrará la naturaleza del problema.

Para hacer un ordenamiento se necesitan valores numéricos. Cada documento debe ser representado por un valor numérico y para ello se toma en cuenta más de un criterio. Unos criterios son fijos para el documento en todo momento (por ejemplo el peso académico), y otros criterios son variables en el momento de la consulta (por ejemplo la presencia de palabras de una consulta específica). Además unos criterios son mas importantes que otros.

El modelo de balance lineal simple puede explicarse a través de los siguientes pasos para nuestro caso:

- Se llevan todos los criterios numéricos a un mismo rango de valores acotados, para ello se toma en cuenta el mínimo y el máximo valor para cada tipo de criterio.

$$crn = cota_{min} + \frac{(cota_{max} - cota_{min})(cr - cr_{min})}{cr_{max} - cr_{min}}$$

  Para esta aplicación el rango será [0,1] y asumiendo que el mínimo para todo valor es 0, la fórmula queda de la siguiente manera

$$crn = \frac{cr}{cr_{max}}$$

- Se establece el peso o importancia para cada criterio determinandolo a través de un vector de coeficientes cuyos valores representan la importancia numérica en cada caso.

$$\vec{importancia} = (imp1, imp2, imp3, ...)$$

- Luego cada documento tiene asociado unos valores para todos los criterios normalizados, con estos valores se obtiene un vector para cada documento:

$$\vec{X} = (crn1, crn2, crn3, ...)$$

- Se establece el *documento optimo*, es decir, el que tiene máximo valor en todos los criterios e nuestro caso. Vector del documento óptimo:

$$\vec{optimo} = (1, 1, 1, ...)$$

- El mejor documento es el que este lo mas cerca del documento óptimo. La fórmula de la distancia es la siguiente:

$$\sqrt{(1 - crn1)^2 + (1 - crn2)^2 + (1 - crn3)^2 + ...}$$

- Como el documento tiene diferentes pesos de importancia para los criterios se debe aplicar una modificación a la formula de la distancia para obtener el valor de ordenamiento definitivo:

$$\sqrt{imp1(1 - crn1)^2 + imp2(1 - crn2)^2 + imp3(1 - crn3)^2 + ...}$$

En este sistema habrá que hacer algunas variaciones a la formula original de ordenamiento para optimizarla y reducir los calculos ya que se necesita una respuesta muy rápida y es el manejador de base de datos quien se encargara de ejecutarla.

De la formula de la distancia lo que interesa es su valor numérico para establecer el orden, quiere decir que es valido alterarla si nos sigue dando ese mismo orden.La modificación que se hizo es quitar la raíz cuadrada de la formula original de la distancia, ya que al omitirla se sigue obteniendo el mismo ordenamiento y se ahorra ese cálculo. Otra variación es asumir que el valor mínimo para cualquier peso numérico siempre sera CERO, así se ahorra el cálculo del rango mínimo en las normalizaciones.

Los Criterios utilizados

El criterio mas importante es el peso académico de la tesis dada por la formula:

$$(eficiencia)(promedio\_ponderado)(calificacion\_tesis)$$

cuando se trata de un solo autor[1]. Este peso es muy importante y su valor se puede deducir a partir del hecho de que si un estudiante ha tenido un buen desempeño durante toda su

[1]Esta pendiente de definir una fórmula para dos o mas autores

carrera probablemente habrá realizado un buen trabajo especial de grado.

El otro criterio es el peso numérico calculado en base a las palabras presentes en el documento ya sea en el contenido, el el titulo, en el resumen, en las palabras clave, etc. Este criterio es precalculado parcialmente durante la carga del documento, el valor numérico definitivo depende de la consulta que haga el usuario.

Por cada palabra presente en el documento se tiene información de la frecuencia con que aparece en el contenido (f_c), la frecuencia con que aparece en el título (f_t), la frecuencia con que aparece en el resumen (f_r) y la frecuencia con que aparece en las palabras claves (f_{pc}) entre otras [2]. El peso numérico de las palabras viene dado por la formula:

$$\ln\left(1 + f_c + 4f_t + 4f_r + 4f_{pc}\right)$$

La formula del valor de las palabras otorga mas importancia a las palabras si están en otros lugares aparte del contenido. por otro lado cuando se buscan dos o mas palabras se le da mas importancia al documento que tiene las frecuencias lo mas parecidas posible para cada palabra.

El hecho de que la formula se calcule con la función logaritmo se debe a que en SQL estándar no se dispone de una función multiplicatoria para agrupaciones, pero si de la función `SUM()` que puede ser utilizada basado en la ecuación siguiente: $\ln(xy) = \ln(x) + \ln(y)$. La necesidad de multiplicar las frecuencias de cada palabra en lugar de simplemente sumarlas se debe a que la multiplicación aporta mas valor cuando las palabras tienden a tener la misma frecuencia de aparición, mientras que la suma es menos significativa o representativa.

5.5.3. Codificación

La implementación del diseño anterior se concreta con el código en SQL que indica los cálculos y el ordenamiento que debe realizar el manejador de base de datos. Primero se calculan los máximos para normalizar los criterios (figura 5.19), luego se escribe la formula de ordenamiento en SQL, y al final de la sentencia de busqueda SQL se le indica que ordene por el campo que resulta de los calculos al principio. La formula de ordenamiento SQL es una cadena que se va armando a partir de constantes y variables y se puede obtener con el método *formula_ ordenamiento_ sql* como puede apreciar en la figura 5.18.

[2] algunas frecuencias de palabras como en el título parecen no tener sentido ya que deberla ser siempre 1, pero es válido tomarlo así ya que proporciona homogeneidad en la lógica del código sin sacrificio de recursos

```
1
2 class ApplicationController < ActionController::Base
3   # Pick a unique cookie name to distinguish our session data from others'
4   session :session_key => '_app_tesis_session_id'
5
6   FormulaPesoNumericoPalabras = "SUM(ln(1 + frecuencia_en_contenido +
  4*frecuencia_en_titulo + 4*frecuencia_en_resumen +
  4*frecuencia_en_palabras_clave))"
7
8   def formula_ordenamiento_sql
9     "POW((1 - #{FormulaPesoNumericoPalabras}/ #
  {@maximo_peso_numerico_palabras}),2) " + " + " + "POW((1 - peso_numerico2/#
  {@maximo_peso_numerico2}), 2) "
10  end
11
```

Figura 5.18: Código que obtiene la subcadena de la sentencia SQL para el ordenamiento

```
292   def establecer_maximos(args)
293     @maximo_peso_numerico1 = valor_sql("SELECT MAX(peso_numerico1) FROM
  documento")
294     @maximo_peso_numerico2 = 1 * 20 * 20 * 1.2
295     maximo_frecuencia_en_contenido = valor_sql("SELECT MAX
  ( frecuencia_en_contenido ) FROM documento_palabra")
296     @maximo_peso_numerico_palabras = (args[:cantidad_palabras]).to_i * Math.log
  (13*maximo_frecuencia_en_contenido.to_f + 0.000001)
297   end
```

Figura 5.19: Código para establecer los máximos de los criterios

En la figura 5.20 se muestra la sentencia SQL generada por la aplicación cuando se hizo una búsqueda sencilla de tres palabras.

```
1
2 SELECT
3  documento_id,
4  POW((1 - SUM(ln(1 + frecuencia_en_contenido + 4*frecuencia_en_titulo +
  4*frecuencia_en_resumen + 4*frecuencia_en_palabras_clave))/ 31.350835790291),2)
  + POW((1 - peso_numerico2/480.0), 2)  AS campo_ordenamiento
5 FROM documento_palabra, documento
6 WHERE true
7   AND palabra_id IN (281, 2138, 1864)
8   AND documento_id = documento.id
9 GROUP BY documento_id
10  HAVING true
11    AND COUNT(*) = 3
12 ORDER BY campo_ordenamiento
13
```

Figura 5.20: Código SQL generado por una busqueda sencilla de tres palabras

5.5.4. Pruebas

Las pruebas de aceptación para el ordenamiento de las consultas se se hizo cargando documentos con varias combinaciones de datos como autor, calificaciones, etc; luego comparando que efectivamente las consultas se ordenaban de la manera que se espera, valorando los criterios de formula de palabras y peso académico.

5.6. Iteración 5

En esta iteración se hacen algunos ajustes para que el sistema realice de forma mas segura y precisa algunas funciones.

Anteriormente el sistema ofrecía la carga de los archivos a través de un formulario Web estándar, Pero ésta no es una forma de subir archivos que garantice el éxito de la trasferencia. El riesgo de que la operación termine abruptamente por un error de *timeout* se debe a que hoy en día el tamaño de los archivos puede llegar a ser considerable. Es necesario implementar una forma de subir archivos de gran tamaño con aplicación Web. En este contexto la solución es usar tecnología *Ajax* para que las transferencias largas tengan éxito. Como la aplicación se construye con *Rails*, existen *plugins* de Rails que llevan a cabo este cometido utilizando *Ajax*.

La otra modificación al sistema es la sobreescritura de la clase *String* estándar. Hasta esta iteración el sistema extraía el texto y normalizaba las palabras a minúsculas llamando a funciones que manipulaban las cadenas, generando complejidad y confusión en el código. Ahora la clase *String* tiene metodos para obtener normalizaciones y palabras que antes requerían de un procesamiento externo al objeto texto.

5.6.1. Planificación

En la tabla 5.6 se muestran las historias de usuario desarrolladas en esta iteración:

12.00	22/10/2007	Upload de archivos largos	Modificación
13.00	22/10/2007	Sobreescritura de la clase *String*	Modificación

Tabla 5.6: Historias de usuario. Iteración 5

5.6.2. Diseño

Sobre el diseño de la carga archivos pesados se puede decir que la funcionalidad va situada la aplicación como otro de los controladores, que en la aplicación es *FilesController* que básicamente se dedica a dar soporte a la vista dinámica de la carga de documentos por *Ajax*.

A la clase *String* se le añadieron los métodos que se observan el la figura 5.21.

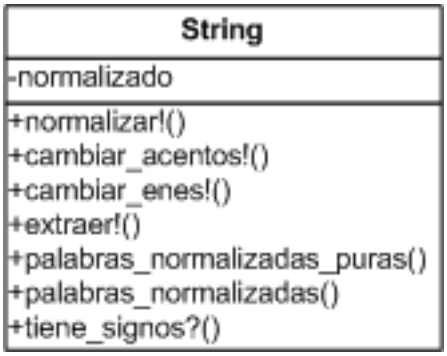

Figura 5.21: Métodos añadidos a la clase *String*

5.6.3. Codificación

Para poder implementar la carga de archivos se instaló un plugin de *Rails*: *Mongrel Upload Progress*. Luego se modifica la vista del formulario de carga para que haga llamadas a la implementación del cargador. El código para el funcionamiento del *Mongrel Upload Progress* esta presente en archivos JavaScript(*upload_progress_javascript.js*), la plantilla o vista que accede a las funcionalidades, y un controlador que maneja la transferencia (ver figura 5.22).

```
1  class FilesController < ApplicationController
2    #funciona con un hash
3    #session[:upload_progress] = { :controlador, :accion, :data }
4    session :off, :only => :progress
5
6    def progress
7      render :update do |page|
8        @status = Mongrel::Uploads.check(params[:upload_id])
9        page.upload_progress.update(@status[:size], @status[:received]) if @status
10     end
11   end
12   alias upload_progress progress
13
14   def upload
15     url = url_for(:controller => session[:upload_progress]
   [:controlador], :action => session[:upload_progress][:accion])
16     session[:upload_progress][:nombre_archivo_temporal] = "./tmp/#{params
   ["upload_id"]}.bin"
17     session[:upload_progress][:nombre_archivo] = params[:data].original_filename
18     File.open(session[:upload_progress][:nombre_archivo_temporal], "wb"){ |f|
19       f.write params[:data].read
20     }
21     render :text => "<script>window.parent.location='#{url}'</script>"
22   end
23
24 end
```

Figura 5.22: *FilesController*

Para reescribir la clase *String* y dotarla de unos métodos muy recurridos por la apli-

cación, basta con desarrollar la clase como si se estuviera codificando desde cero (Figura 5.23). *Ruby* sabe que ya existe una clase con ese nombre y por lo tanto conserva todos los metodos de la versión ya existente. Se obtiene el beneficio de que no estamos obligados a hacer una nueva clase que herede de la estándar, y también se maneja con los literales de comillas dobles o simples, para no estar invocando a constructores de *String* explícitamente.

```
1  class String
2
3    CaracterSigno = "#"
4
5    def normalizar!
6      self.gsub!(/[\s]+/, " ")
7      self.cambiar_acentos!
8      self.cambiar_enes!
9      self.downcase!
10     self.gsub!(/[^a-z\s]+/,CaracterSigno)
11     self.strip!
12     @normalizado = true
13     return self
14   end
15
16
17   def cambiar_acentos!
18     self.gsub!("\303\241".to_s,"a")
19     self.gsub!("\303\251".to_s,"e")
20     self.gsub!("\303\255".to_s,"i")
21     self.gsub!("\303\263".to_s,"o")
22     self.gsub!("\303\272".to_s,"u")
23     self.gsub!("\303\201".to_s,"A")
```

```
39     self.replace(cadena_reemplazo)
40     return self
41   end
```

```
92
93   def tiene_signos?
94     if @normalizado
95       cadena = self
96     else
97       cadena = self.clone
98       cadena.normalizar!
99     end
100    return self.match(/[#{CaracterSigno}]+/)
101  end
102
103
104 end
```

Figura 5.23: Modificación de la clase *String*

5.6.4. Pruebas

Las pruebas de aceptación para la carga de archivos largos se hizo subiendo archivos de considerable tamaño (5MB, 50MB y 400MB), luego que fueran recibidos y correctamente procesados tal como sucedía anteriormente.

Para probar que la sobreescritura de la clase *String* es efectiva, se eliminaron todos los metodos que "purificaban" las cadenas, luego en su lugar se hacia que los propios textos se limpiaran, observándose que no ocurría ninguna anomalía en el proceso de extracción y búsqueda.

5.7. Iteración 6

En esta iteración se comienza a trabajar con atención la parte de interfaz del usuario. Anteriormente todo el desarrollo estaba centrado en los procesos de tratamiento de los datos del documento y de la búsqueda, mientras que la interfaz se basaba en los estilos de CONEST. Ahora se va a realizar las propias combinaciones de colores y estilos, siguiendo un patrón minimalista acorde con el propósito de la aplicación. Para evitar errores por parte del usuario en los campos de fechas, se instalo un plugin de Rails llamado *Google Calendar Date Select* el cual proporciona asistencia gráfica basada en JavaScript.

5.7.1. Planificación

En la tabla 5.7 se muestran las historias de usuario desarrolladas en esta iteración:

14.00	29/10/2007	Desarrollo propio de interfaz y estilos	Nueva

Tabla 5.7: Historias de usuario. Iteración 6

5.7.2. Diseño

Ahora con una interfaz mas ligera y sobria, el usuario experimenta poca sobrecarga de elementos en las paginas Web y se hace más clara y evidente la funcionalidad que ofrece el sistema en cada vista. En la figura 5.24 se puede apreciar el aspecto nuevo en contraste con las interfaces anteriores (ver figura 5.15).

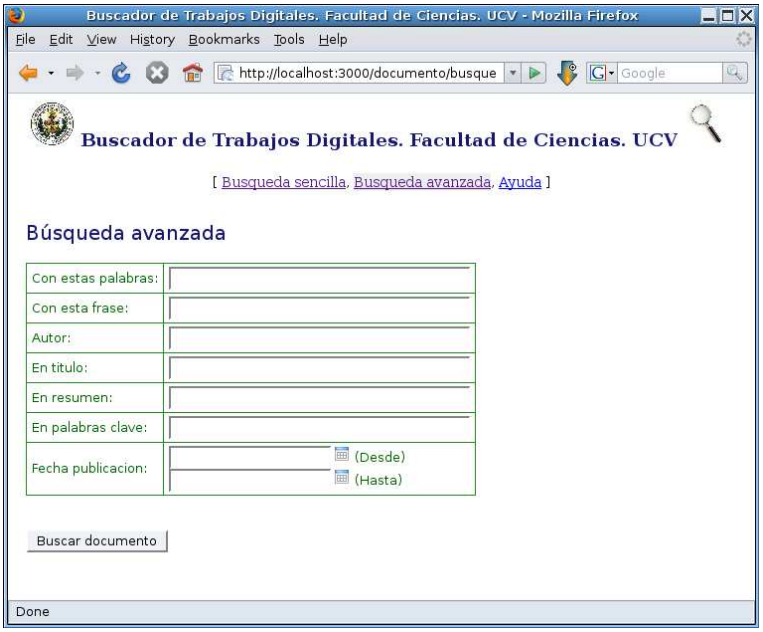

Figura 5.24: Aspecto minimalista de la nueva interfaz (Búsqueda avanzada).

Para los campos de fechas, el plugin *Google Calendar Date Select* proporciona una cómoda usabilidad como se aprecia en la figura 5.25.

Figura 5.25: Google Calendar Date Select

5.7.3. Codificación

La codificación para el desarrollo de la interfaz, implicó retocar o redefinir las plantillas de las vistas presentes en toda la aplicación, pero el código nuevo básicamente yace en la hoja de estilos (*buscador.css*) que aporta gran parte de la apariencia en el sistema. Un extracto del código puede apreciarse en la figura 5.26.

```
130
131 #info_busqueda_avanzada a:visited{
132   font-size: 0.8em;
133   color: Maroon;
134 }
135 #info_busqueda_avanzada a:link{
136   font-size: 0.8em;
137   color: Maroon;
138 }
139
140
141 #paginacion{
142   display: inline;
143   font-size: 1.2em;
144 }
145
146 #error_campo{
147   display: inline;
148   color: Crimson;
149 }
150 #advertencia_campo{
151   display: inline;
152   color: DarkGreen;
153 }
154
```

Figura 5.26: extracto de hoja de estilo *Busqueda.css*

5.7.4. Pruebas

Las pruebas de aceptación para la interfaz se hicieron navegando por todas las páginas de la aplicación, donde se observo que las proporciones eran adecuadas, y la funcionalidad no se vio afectada. En la vista de búsqueda avanzada se probó el desempeño correcto del plugin *Google Calendar Date Select* observándose un buen despliegue del calendario con su estilo *css* propio.

En general la interfaz es menos sobrecargada y ofrece al usuario una fácil usabilidad.

5.8. Iteración 7

En la iteración anterior se enfatizó en la apariencia y el aspecto. Para esta iteración se trabajara sobre la usabilidad del sistema.

Despues de haber analizado y probado varias veces el sistema en general, se llegó a la conclusión de que las búsquedas que hagan los usuarios tienen que ser representadas por un "objeto búsqueda" para lograr usabilidad. Con un objeto búsqueda se tiene mayor modularidad, se logra una buena abstracción y encapsulamiento de las ordenes del usuario. Se puede revisar mejor los datos, mostrar errores específicos incluso en el lugar que fueron hallados, en fin, se tiene mucha más información y hay mas precisión en las respuestas del sistema.

En esta iteración se desarrolla la clase *Busqueda* con metodos como validar y guardar los errores en su estado, el cual puede ser utilizado por la vista que la muestre.

Para evitar errores en el funcionamiento provocados por los usuarios o por no seguir los pasos, se desarrollan una cantidad de *Filtros* cuyo propósito es garantizar una correcta respuesta del sistema.

5.8.1. Planificación

En la tabla 5.8 se muestran las historias de usuario desarrolladas en esta iteración:

15.00	5/11/2007	Modelación de las búsquedas implementando la clase *Busqueda*	Modificación
16.00	5/11/2007	Aplicación de filtros y validaciones en las entradas del usuario	nuevo

Tabla 5.8: Historias de usuario. Iteración 7

5.8.2. Diseño

El diseño de la clase búsqueda puede apreciarse en la figura 5.27.

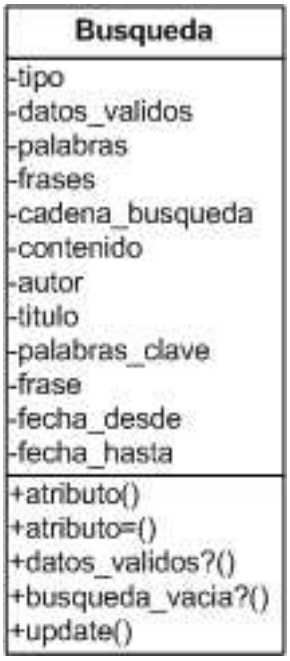

Figura 5.27: Diagrama de clase *Busqueda*

5.8.3. Codificación

En la figura 5.28 se muestra el método *actualizar* de la clase *Busqueda*, el cual se encarga de recibir los valores de los campos del formulario de búsqueda y a partir de allí actualizar su estado y marcar los posibles errores.

```
40  def update(params)
41
42    if @tipo == Sencilla
43      errores = []
44      @cadena_busqueda[:texto] = params[:cadena_busqueda]
45      @frases = @cadena_busqueda[:texto].extraer!(/["][^"]*["]/).uniq
46      @frases.each_with_index{ |f, i|
47        f.delete!("\"")
48        f.normalizar!
49        errores << "error en frase #{i+1}" if f.tiene_signos?
50      }
51      @frases.delete_if{|f| f.size==0}
52      @cadena_busqueda[:texto].normalizar!
53      errores.unshift "error en la cadena de busqueda" if @cadena_busqueda
  [:texto].tiene_signos?
54      @cadena_busqueda[:texto] += (frases.collect{|f| " \"#{f}\""}.join)
55      @cadena_busqueda[:texto].strip!
56      @palabras = @cadena_busqueda[:texto].palabras_normalizadas_puras.uniq
57      if errores.size > 0
58        @datos_validos = false
59        @cadena_busqueda[:error] = errores.join(", ")
60      else
61        @datos_validos = true
62        @cadena_busqueda[:error] = nil
63      end
64
65    elsif @tipo == Avanzada
```

Figura 5.28: Método *update* de la clase *Busqueda*

Los Filtros son procesos que pueden interceptar la las peticiones (`before_filter`) desde el cliente antes de que vayan a dar con un recurso o accesar al método de la lógica propiamente dicho. Puede manipular la petición, hacer verificaciones, desviar la petición a otro proceso, etc. También están los filtros de vuelta (`after_filter`) que pueden interceptar la respuesta y manipularla antes de devolverla al cliente.

En la figura 5.29 se muestra uno de los muchos filtros y verificaciones que están a lo largo de toda la aplicación, éste en particular sirve para interceptar la petición del usuario de realizar la búsqueda sencilla con los parámetros enviados por *post*.

```
233  def pre_buscar_sencillo
234    begin
235      # validacion del objeto busqueda (sencilla)
236      if session[:busqueda].nil? or session[:busqueda].tipo != Busqueda::Sencill
237        raise "Error intencional: el objeto busqueda en la sesion no es correcto
238      end
239      @busqueda = session[:busqueda]
240      @busqueda.update(params)
241      # cadena vacia
242      if !@busqueda.datos_validos?
243        flash[:mensaje_alerta] = @busqueda.cadena_busqueda[:error]
244        redirect_to(:action => "busqueda_sencilla")
245        return
246      end
247
248
249      if @busqueda.busqueda_vacia?
250        flash[:mensaje] = "Por favor coloque algo para saber qué tengo que
   buscar. Gracias"
251        redirect_to(:action => "busqueda_sencilla")
252        return
253      end
254      # bug dame todo
255      if @busqueda.cadena_busqueda[:texto] == "dame todo"
256        claves_primarias_documentos =  self.class.buscar_todos()
257        paginador = Paginador.new(:claves_primarias =>
   claves_primarias_documentos, :clase => Documento, :objetos_por_pagina =>
   CantidadDeDocumentosPorPagina)
258        session[:paginador] = paginador
259        redirect_to(:action => "resultado_consulta")
260      end
261    rescue Exception => excepcion
262      eval cod_post_error_imprevisto
263    end
264  end
```

Figura 5.29: Filtro que intercepta la petición de realizar la búsqueda sencilla

Es importante destacar que en esta aplicación los filtros están rodeados de un bloque de tratamiento de errores (*begin - rescue - end*) donde van a parar todas las excepciones, tanto las producidas por la aplicación como las que que no se tienen pensadas (figura 5.30).

```
329  def <nombre_metodo>
330    begin
331    rescue Exception => excepcion
332      eval cod_post_error_imprevisto
333    end
334  end
```

Figura 5.30: Bloque de tratamiento de errores utilizado en los filtros

En tiempo de ejecución se corre el código de tratamiento del error con la instrucción *eval*. Uno de los códigos dinámicos Ruby que se ejecutan se puede ver en la figura 5.31.

```
21  def cod_post_error_imprevisto
22    %q/
23    # para llamar a este codigo debe existir la variable excepcion
24    puts '.:. Error imprevisto .:.', excepcion, excepcion.backtrace
25    flash[:mensaje_error] = 'Ocurrió algo imprevisto, asegúrese de interactuar
segun la interfaz y seguir los pasos ;)'
26    redirect_to(:action => AccionPrincipal)
27    return
28    /
29  end
```

Figura 5.31: Método que devuelve código Ruby

5.8.4. Pruebas

Las pruebas de aceptación se hicieron realizando diferentes consultas obteniendo los resultados esperados más las mejoras de la interfaz y usabilidad, manejando los errores de entrada de usuario, y respondiendo a otros tipos de errores.

La clase *Busqueda* proporcionó información mas específica para el usuario, señalando los errores específicos, o informando acertadamente cualquier suceso de interés.

5.9. Iteración 8

En este sistema existen procesos que pueden tardar mucho tiempo en ejecutarse, la más destacada de ellas es el proceso de obtención de estructuras auxiliares del documento. Del documento se extraen las palabras, sus frecuencias, fechas, etc; y este proceso suele llevar mucho tiempo. Que se tengan respuestas tardías debido a un proceso muy largo es algo que no es acorde con una aplicación web puramente síncrona donde las respuestas deben ser inmediatas.

Para este tipo de problemas la tecnología Web ya tiene soluciones como *AJAX*, y en la iteración 5 se trato un caso especifica de proceso largo: la carga de archivos pesados. Para esta iteración se usará una solución más general, aplicable a cualquier proceso largo asíncrono que pueda ser monitoreado por una vista web mediante *AJAX*. Para ello se escogió un plugin de Rails llamado *BackgrounDRb*. Este plugin utiliza la integración *Rails-Ajax*.

BackgrounDRb[3], según lo explican sus creadores, es un servidor y administrador de tareas para que éstas sean ejecutadas en segundo plano, mientras que el usuario y la aplicación Web pueden seguir interactuando síncronamente en otras acciones una vez disparada la tarea. Además la aplicación puede comunicarse con el hilo que ejecuta la tarea a través del administrador de tareas para consultar su estado de progreso, estatus, para detenerlo, etc.

5.9.1. Planificación

En la tabla 5.9 se muestran las historias de usuario desarrolladas en esta iteración:

16.00	12/11/2007	Implementación de procesos largos en background	Modificación

Tabla 5.9: Historias de usuario. Iteración 8

5.9.2. Codificación

Básicamente *BackgrounDRb* es un servidor que corre un demonio, y se accede a los hilos de ejecución de tareas (workers) a través de una clase llamada *MiddleMan* (Desde la aplicación Rails). Para cada tarea que se desea realizar con BackgrounDRb se debe implementar una clase (worker) que herede de *BackgrounDRb::Rails* y un método *do_ work()* que es donde estará el código de la tarea a ejecutarse.

En la figura 5.32 se tiene la implementación de un *worker* que realiza la tarea de procesar todos los documentos que estén faltos de procesamiento. En este caso el código llama a una función ya existente que procesa varios documentos en lote. Lo que anteriormente se

[3]http://backgroundrb.rubyforge.org/

hacia llamando directamente al método *procesar_no_procesados()*, ahora se hace a través del worker.

```
1 require RAILS_ROOT + '/app/controllers/application'
2
3 class ProcesarDocumentosWorker < BackgrounDRb::Rails
4
5
6   attr_accessor :status, :progreso, :total, :finalizado
7
8   def do_work(args = nil)
9
10    begin
11      #
12      @progreso = 0; @status = "Iniciando el procesamiento de los documentos
    cargados"
13      ApplicationController.procesar_no_procesados(self)
14      #
15      @finalizado = true
16      MiddleMan.delete_worker(@_job_key) # fino!
17      ActiveRecord::Base.connection.disconnect!
18    rescue Exception => e
19      puts e.inspect, e.backtrace
20    end
21  end
22
23  def parametros
24    return @total, @progreso, @status, @finalizado
25  end
26
27 end
```

Figura 5.32: Implementación de un *worker* que realiza el procesamiento de varios documentos

Cabe señalar que para mantener el código del proceso en su lugar (ApplicationController), no se mudó la implementación y la lógica hacia el worker, sino que este último llama al método, y escribe el estado del proceso en el worker.

En la figura 5.33 se tiene el código del método *procesar_no_procesados()*, el cual ha sido adaptado para establecer el estado del proceso en el worker en el caso de haber sido iniciado por éste.

```
265  def self.procesar_no_procesados(worker = nil)
266    claves_primarias = buscar_no_procesados
267    worker.total = claves_primarias.size if worker # WORKER!
268    claves_primarias.each_with_index{|clave_primaria_documento, i|
269      ejecutar_sql("DELETE FROM documento_palabra WHERE documento_id = #
   {clave_primaria_documento}")
270      worker.progreso = i+1 if worker # WORKER!
271      worker.status = "Comenzando ..." if worker # WORKER!
272      almacenar_estructuras(Documento.find(clave_primaria_documento),
   worker) # WORKER!
273    }
274  end
```

Figura 5.33: Método *procesar_no_procesados* adaptado para aceptar un worker y escribir en él su progreso

Es importante deja claro que BackgrounDRb cuando inicia es servicio aparte realmente, cuyo código reside dentro del mismo proyecto.

5.9.3. Pruebas

Para probar la ejecución de los procesos largos en segundo plano con BackgrounDRb, se hizo una interfaz temporal donde se monitoreaba el progreso de la tareas, además en el servidor de BackgrounDRb, se obtuvieron las mismas salidas por consola que las generadas en el servidor de aplicación principal con una invocación directa a los métodos. Por ejemplo el procesamiento de documentos genera como salida por cónsola, cada palabra con su frecuencia en el documento.

5.10. Iteración 9

Prácticamente desde el comienzo se desarrollaron unas funcionalidades para el mantenimiento del sistema a nivel de desarrollo, como por ejemplo la eliminación de documentos, reprocesamiento, etc. Estas funcionalidades deben estar disponibles para el manejo del sistema por medio del sistema mismo. Esto implica la creación de la figura del administrador y las funcionalidades de mantenimiento y monitoreo.

Mientras que cualquier usuario puede hacer únicamente consultas sencillas o avanzadas. El administrador configura el sistema y puede tomar decisiones que modifiquen su estado.

En esta iteración se agrupan las funcionalidades del administrador en un controlador aparte, es decir, un grupo de acciones correspondientes al rol del administrador por separado.

5.10.1. Planificación

En la tabla 5.10 se muestran las historias de usuario desarrolladas en esta iteración:

17.00	19/11/2007	Funciones y rol del *Administrador* del sistema	Nueva

Tabla 5.10: Historias de usuario. Iteración 9

5.10.2. Diseño

En la aplicación la funcionalidad del administrador esta agrupada toda en un mismo controlador, los métodos de administración se siguen situando el lugar donde fueron creados (*ApplicationController*).

Ahora teniendo la herramienta de procesos largos (BackgrounDRb), las acciones del administrador no invocan directamente a los procesos largos sino que disparan tareas en background a través de la clase *MiddleMan*.

En la tabla 5.11 se muestra la funcionalidad del administrador junto con una breve descripción.

Función	Descripción
Iniciar Sesión	El administrador ingresa su login y password, luego se verifica y se activa la sesión.
Actualizar parámetros	Se establece en el sistema los valores de configuración.
Procesar documentos sin procesar	Calcula las estructuras auxiliares de referencias de palabras y frecuencias. Esto aplica a los documentos que se encuentran es estado de "no procesados", por ejemplo los nuevos documentos cargados sin procesamiento.
Desprocesar todos los documentos	Elimina estructuras auxiliares no pertenecientes a los documentos como tal, conservando a éstos en el sistema.
Eliminar un documento	Esta opción esta disponible para cada documento cuando el administrador esta en la vista de resultados de una búsqueda.
Eliminar todos los documentos	Elimina por completo todos los documentos y las estructuras asociadas. También elimina toda palabra existente.
Cerrar sesión	Sale de la sesión y las vistas y funciones del administrador dejan de estar disponibles en todo el sistema.

Tabla 5.11: Formato de historias de usuario

En la figura 5.34 se aprecia la vista con las funciones disponibles para el administrador. Es importante señalar que el administrador debe iniciar sesión mediante login y password.

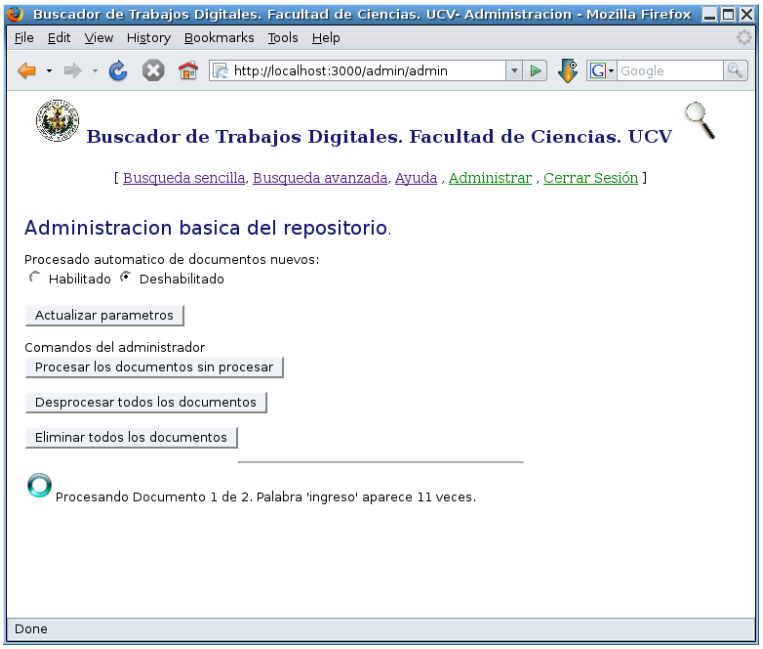

Figura 5.34: Vista de las funciones del administrador

En esta vista del administrador aparecen los procesos largos en segundo plano en caso de estar llevándose a cabo.

5.10.3. Codificación

En la figura 5.35 se tiene un extracto de código del *AdminController* donde se implementa la función de procesar los documentos no procesados.

```
95  def procesar_no_procesados #con worker
96    #self.class.procesar_no_procesados
97    session[:jobkey_procesar_documentos] = MiddleMan.new_worker :class
   => :procesar_documentos_worker, :args => nil
98    nuevo_mensaje "Se envió la orden de procesar los documentos."
99    eval cod_post_comando_administrador
100 end
```

Figura 5.35: Extracto de código de *AdminController*

Lo anterior inicia al worker *procesar_ documentos_ worker* que a su vez llama al método *procesar_ no_ procesados* (ver figuras 5.32 y 5.33 de la iteración anterior). Luego desde éste ultimo método existe un ciclo principal en cuyo interior se llama a uno de los primeros metodos desarrollados en este proyecto desde sus comienzos: *almacenar_ estructuras()*. La implementación del método *almacenar_ estructuras()* se puede ver en las figuras 5.36 y 5.37, al cual también se le hizo adaptaciones para aceptar workers de BackgrounDRb.

```
27  def self.almacenar_estructuras(documento, worker = nil)
28    # pesos numericos estaticos
29    documento.peso_numerico1 = Time.mktime *documento.fecha.scan(/[\d]
  +/).reverse
30    documento.peso_numerico2 = documento.eficiencia.to_f *
  documento.promedio_ponderado.to_f * documento.nota.to_f
31    documento.peso_numerico2 *= 1.2 if documento.tiene_premio == "si"
32    # obtencion de texto y palabras
33    texto_contenido = obtener_texto(documento).normalizar!
34    documento.texto = texto_contenido
35    palabras_contenido = texto_contenido.palabras_normalizadas_puras
36    worker.status = "cantidad de palabras: #{palabras_contenido}" if worker
  # WORKER!
37    # prosesamiento de frecuencias y tablas externas al documento
38    tfs_contenido = obtener_frecuencias(palabras_contenido, worker)  #
  WORKER!
39    # TODO quitar el registro de archivo de frecuencias
40 archivo_frecuencias = File.open("tmp/frecuencias de #
  {documento.titulo}.txt","w")
41 tfs_contenido.each{|tf|archivo_frecuencias.puts "#{tf[1]} #{tf[0]}"}
42 archivo_frecuencias.close
43    tfs_titulo = obtener_frecuencias
  ( documento.titulo.palabras_normalizadas_puras )
44    tfs_palabras_clave = obtener_frecuencias
  (documento.palabras_clave.palabras_normalizadas_puras)
45    tfs_resumen = obtener_frecuencias
  (documento.resumen.palabras_normalizadas_puras)
46    tfs_autor = obtener_frecuencias
  (documento.autor.palabras_normalizadas_puras)
47    # sera un hash de hashes  { palabra => { tipo_frecuencia =>
  valor1, ... }, ...    }
48    worker.status = "Finalizado el calculo de frecuencias" if worker #
  WORKER!
```

Figura 5.36: Método *almacenar_ estructuras* (primera parte)

```
49    tfs = {}
50    tfs_contenido.each{|tf| tfs[tf[0]]= { "frecuencia_en_contenido" => tf
  [1] }  }
51    tfs_titulo.each{|tf|
52      if tfs[tf[0]]
53        tfs[tf[0]]["frecuencia_en_titulo"] = tf[1]
54      else
55        tfs[tf[0]]= { "frecuencia_en_titulo" => tf[1] }
56      end
57    }
58    tfs_palabras_clave.each{|tf|
59      if tfs[tf[0]]
60        tfs[tf[0]]["frecuencia_en_palabras_clave"] = tf[1]
61      else
62        tfs[tf[0]]= { "frecuencia_en_palabras_clave" => tf[1] }
63      end
64    }
65    tfs_resumen.each{|tf|
66      if tfs[tf[0]]
67        tfs[tf[0]]["frecuencia_en_resumen"] = tf[1]
68      else
69        tfs[tf[0]]= { "frecuencia_en_resumen" => tf[1] }
70      end
71    }
72    tfs_autor.each{|tf|
73      if tfs[tf[0]]
74        tfs[tf[0]]["frecuencia_en_autor"] = tf[1]
75      else
76        tfs[tf[0]]= { "frecuencia_en_autor" => tf[1] }
77      end
78    }
79    # procesamiento general de todas las frecuencias
80    worker.status = "Almacenando estructuras auxiliares" if worker # WORKER!
81    tfs.each{ |tf|
82      palabra = Palabra.find(:first, :conditions => ["texto = ?", tf[0] ] )
83      if palabra.nil?
84        palabra = Palabra.new
85        palabra.texto = tf[0]
86        palabra.save
87        worker.status = "Nueva palabra: \'#{palabra.texto}\'" if worker #
  WORKER!
88      end
89      insertar_documento_palabra(documento, palabra, tf[1])
90      worker.status = "Asociando \'#{palabra.texto}\'" if worker # WORKER!
91    }
92    # finalizacion del procesamiento
93    documento.procesado = true
94    documento.save
95  end
```

Figura 5.37: Método *almacenar_ estructuras* (segunda parte)

5.10.4. Pruebas

Para probar las funciones del administrador, Se probaron cada una de ellas por medio de sus vistas , verificando si se obtenían los mismos resultados que anteriormente daba el sistema, y en efecto los resultados eran los mismos.

Se probó intentar acceder a las funcionalidades del administrador como usuario visitante, y el sistema impidió todos los intentos, redirigiendo la petición a la vista de acceso a través de login y password.

5.11. Iteración 10

En esta iteración se termina de definir la forma en que se agregan nuevos documentos al sistema. El sistema CONEST aporta la información relacionada al documento, y la aplicación le notifica a éste el recibimiento del documento por parte del autor.

Anteriormente los documentos se cargaban totalmente por medio de la aplicación que pedía datos por Web Service según las entradas de un usuario con permisos.

Ahora la aplicación se entiende como un subsistema de CONEST, que por razones técnicas y lógicas debe permanecer como un sistema interdependiente. La información inicial o metadata asociada se obtiene de CONEST para quienes aprobaron su trabajo especial de grado, luego los datos y el control se pasan a la aplicación la cual notifica a CONEST luego de almacenar el archivo.

En esta iteración también se desarrolla el tratamiento que la aplicación tiene sobre las palabras que son muy comunes a todos los documentos y que se repiten muchísimas veces dentro de ellos. Entre estas palabras tenemos las preposiciones, artículos, pronombres, conjunciones, entre otras.

5.11.1. Planificación

En la tabla 5.12 se muestran las historias de usuario desarrolladas en esta iteración:

18.00	26/11/2007	Reestructuración en la manera de cargar los nuevos documentos	Modificación
5.01	26/11/2007	Frecuencias de términos para palabras irrelevantes	Modificación

Tabla 5.12: Historias de usuario. Iteración 10

5.11.2. Diseño

El ciclo de carga de documentos nuevos comienza en CONEST (Estudiantes) en donde se le ofrece a cada graduando la opción de subir su Trabajo Especial de Grado (T.E.G.).

Se puede ver de manera sencilla la carga de nuevos documentos, a través de los siguientes pasos:

- En la vista principal de CONEST del estudiante graduando aparece un link el cual se dirige hacia la aplicación llevando los datos encriptados.

- La aplicación recibe los datos recibidos de CONEST (estudiantes) desencriptando y validando su integridad.
- La aplicación pide subir el archivo
- La aplicación pide establecer el título, el resumen y las palabras clave
- La aplicación comunica a CONEST (administración) que el documento ha sido subido por el autor.

Los datos que recibe la aplicación son la calificación del trabajo, la licenciatura y los datos personales y académicos del autor entre otros.

en la figura 5.38 puede apreciarse la manera cómo sucede una carga de documento.

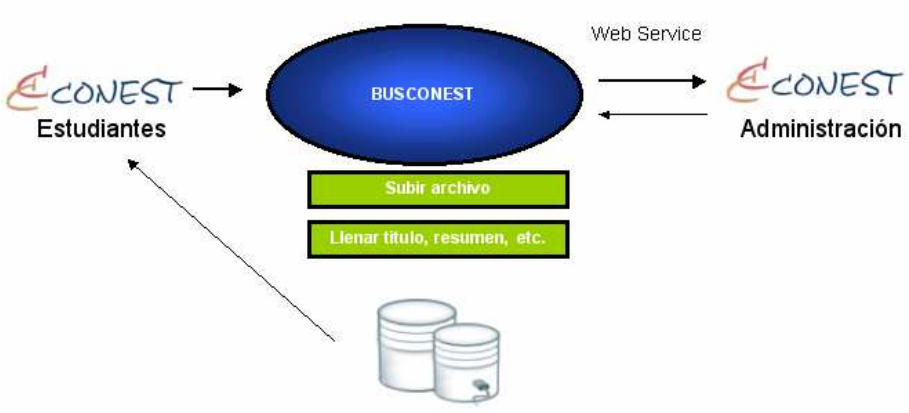

Figura 5.38: Carga de documentos nuevos

Es muy importante destacar que durante la carga del documento nuevo, se hicieron esfuerzos para que el usuario graduando no sintiera el cambio de contexto de las diferentes aplicaciones. Para ello se utilizaron unas plantillas adaptadas para que funcionaran en la aplicación y ofrecieran una interfaz consistente e incluso los enlaces funcionales como regresar a principal y cerrar sesión de CONEST.

La figura 5.39 muestra la vista principal de CONEST (Estudiante) y en la figura 5.40 se muestra la vista a donde lleva el enlace (en la aplicación).

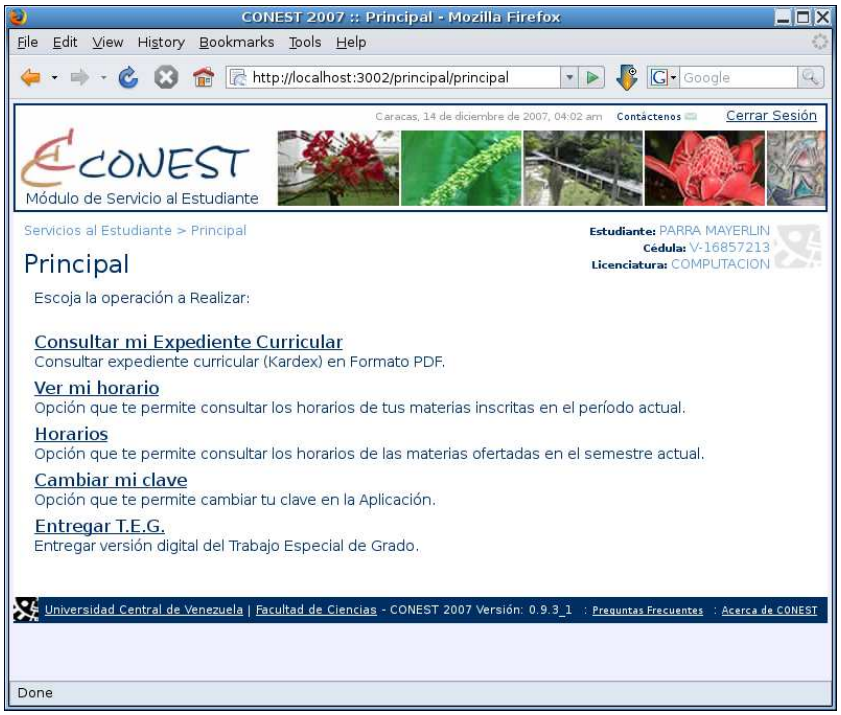

Figura 5.39: Vista principal de CONEST (Estudiante)

Figura 5.40: Destino del enlace para subir el documento (vista de la aplicación)

5.11.3. Codificación

Tal como se explicó en el diseño, el ingreso de un nuevo documento comienza en la vista principal de CONEST estudiantes, desde aquí se envía la data por *POST* en un campo oculto hacia la aplicación en la acción *nuevo* del controlador *Documento*. el código que procesa dicha acción esta en la figura 5.41.

```
62  def nuevo
63    begin
64      session[:documento] = Documento.new
65      @documento = session[:documento]
66      llave_conest = params[:data]
67      cadena_descifrada = llave_conest.tr(CadenaTraduccionB,CadenaTraduccionA)
68      parametros_recibidos = Hash[*(cadena_descifrada.split("!"))]
69      #
70      raise "Error intencional: No coincide la clave proveniente de conest" if
  parametros_recibidos[:integridad.to_s] != "integridad"
71      campos =  [:fecha_publicacion, :calificacion, :tiene_premio]
72      campos +=
  [:mencion, :licenciatura_id, :licenciatura, :escuela, :facultad, :universidad]
73      campos.each{ |campo|
74        # usa los metodos del objeto documento
75        @documento.send "#{campo.to_s}=", parametros_recibidos[campo.to_s]
76      }
77      # genera directamente el xml
78      xml = @documento.xml
79      xml_autor = REXML::Element.new "autor"
80      xml.root.add xml_autor
81      campos_autor =
  [:nombre, :cedula, :correo, :telefonos, :promedio_general, :promedio_ponderado,
```

Figura 5.41: Código de la aplicación que recibe los nuevos datos en *documento/nuevo*

En la figura anterior se muestra cómo el sistema desencripta y valida los datos luego hace otras verificaciones por Web Service.

Los Webservices de CONEST ahora han cambiado con respecto de la iteración 2 (ver figuras 5.8, 5.9, 5.10 y 5.11).

En la figura 5.42 se muestra en API del Web Service.

```
class PublicacionesApi < ActionWebService::API::Base
  api_method :teg_marcar_subido, :expects => [:string, :string], :returns => [:bool]
  api_method :teg_subido, :expects => [:string, :string], :returns => [:bool]
  api_method :tiene_planilla, :expects => [:string, :string], :returns => [:bool]
  api_method :fecha_publicacion_teg, :expects => [:string, :string], :returns => [:string]
  api_method :ok, :expects => [], :returns => [:bool]
end
```

Figura 5.42: API del Web Service de comunicación con CONEST

En la figura 5.43 se muestra una parte de la implementación del API del Web Service del lado de CONEST. Con estos métodos del Web Service la aplicación puede registrar que un documento ya ha sido recibido, puede saber si un documento ya se encuentra subido, y otras funciones mas.

```
class PublicacionesController < ApplicationController
  wsdl_service_name 'Publicaciones'

  def teg_marcar_subido(estudiante_cedula, licenciatura_id)
    planilla_individual = PlanillaIndividual.find(:first, :conditions => ["estudiante_cedula = ? AND licenciatura_id = ?", estudiante_cedula, licenciatura_id])
    if planilla_individual
      planilla_individual.documento_subido = true
      planilla_individual.save
      return true
    else
      return false
    end
  end

  def teg_subido(estudiante_cedula, licenciatura_id)
    planilla_individual = PlanillaIndividual.find(:first, :conditions => ["estudiante_cedula = ? AND licenciatura_id = ?", estudiante_cedula, licenciatura_id])
    if planilla_individual
      return planilla_individual.documento_subido
    else
      return false
    end
  end
```

Figura 5.43: *PublicacionesController*: la implementación del API del Web Service

Para el tratamiento de palabras muy comunes se hizo una modificación al método de obtener frecuencias. Ahora se tiene un listado de palabras muy comunes e irrelevantes en el modelo *Palabra*, si la palabra pertenece a éste conjunto, automáticamente su frecuencia se establece en uno (1) sin importar la cantidad de ocurrencias real. En la figura 5.44 puede apreciarse el tratamiento.

```
  def self.obtener_frecuencias(palabras, worker = nil) # recibe un arreglo
de palabras de un texto, y  devuelve un hash { "palabra" => frecuencia } que
indica las repeticiones de cada palabra distinta
    tf = {}
    palabras_unicas = palabras.uniq
    palabras_unicas.each{ |palabra_unica|
      if Palabra.omitidas.include? palabra_unica
        tf[palabra_unica] =  1
      else
        tf[palabra_unica] =  palabras.find_all{ |pal| pal ==
palabra_unica }.size
      end
        worker.status = "Palabra \'#{palabra_unica}\' aparece #{tf
[palabra_unica]} veces. " if worker # WORKER!
        puts " #{palabra_unica} #{tf[palabra_unica]}" #para ver que todo
esta saliendo bien
      palabras.delete_if{ |pal| pal == palabra_unica} #Elimina la palabra
para reducir el arreglo de palabras
    }
    return tf
  end
```

Figura 5.44: Asignación de la frecuencia de cada palabra en el método *obtener_frecuencias*

5.11.4. Pruebas

Para probar la integración con CONEST, se hicieron ingresos a los sistemas de CONEST, administración y estudiantes, y desde allí se probaron los enlaces añadidos, observando que la interfaz de usuario se mantenía casi intacta incluso conservando las funcionalidades cuando se cambiaba el contexto de la aplicación.

Se intento que varios estudiantes ingresaran y trataran de subir documentos mas de una vez, pero el sistema no se los permitió. Se verificó que los nuevos datos sean consistentes con CONEST respecto a lo que finalmente se almacenaba en el objeto *Documento*. Se revisó que quedara registrado correctamente la subida del documento. También se verificó que la vista del estudiante en CONEST quedara afectada y ya no se mostrara el enlace a subir el nuevo documento.

Las funcionalidades como cerrar sesión o ir a la pagina principal ofrecidas por la aplicación para simular el contexto de CONEST se probaron repetidas veces, obteniéndose el comportamiento y respuesta esperados.

Para las el requerimiento de tratamiento de las palabras irrelevantes durante el procesamiento de los nuevos documentos, se reviso la salida por consola del servidor en los que se va listando cada palabra y su frecuencia asociada, pudiéndose demostrar que a todas las palabras listadas como irrelevantes se les asignó el valor uno (1).

Pruebas unitarias del objeto *Documento*

A lo largo de todas las iteraciones anteriores, la clase *Documento* ha sufrido cambios en su estructura. En esta última iteración se sometió a este tipo de objetos a las pruebas unitarias que ofrece el framework Rails.

Las pruebas unitarias de Rails son utilizadas para probar una clase del Modelo, es decir, se prueban los métodos de la clase. Principalmente consiste en implementar unos métodos de prueba que realizan aserciones (*asserts*) o afirmaciones sobre el código según los datos cargados en una base de datos de prueba que se llena a partir de unos *fixtures*.

Los *fixtures* son registros de prueba, y constituyen los datos con los cuales la prueba va a trabajar. Los fixtures son definidos en formato YAML [4].

Para las pruebas unitarias que se realizaron, se llenó el *fixture* de la clase *documento.yml* con dos objetos *Documento* con id y metadata xml. En el primer objetos se colocó un sólo autor y en el segundo se colocaron dos elementos "autor".

En la figura 5.45 se muestra el código del test unitario de la clase *Documento*.

[4]YAML es un lenguaje muy sencillo que permite describir datos como XML, pero con una sintaxis mucho más sencilla (http://www.yaml.org)

```ruby
require File.dirname(__FILE__) + '/../test_helper'

class DocumentoTest < Test::Unit::TestCase
  fixtures :documento

  # Replace this with your real tests.
  def test_truth
    assert true
  end

  def test_consistencia_xml
    doc = Documento.find 1
    assert_equal "Trabajo de Tesis General de Juan", doc.titulo
    assert_equal "El resumen es una muestra del contenido del documento", doc.resumen
    assert_equal "no", doc.tiene_premio
    assert_equal doc.promedio_ponderado_autor1.to_f, doc.promedio_ponderado
  end

  def test_consistencia_2_autores
    doc = Documento.find 2
    assert_equal 2, doc.cantidad_autores
    assert_equal REXML::Document.new(doc.metadata).root.elements["count(//autor)"], doc.cantidad_autores
    assert_equal "JOSE RODRIGUEZ", doc.nombre_autor1
    assert_equal "ANA FLOR", doc.nombre_autor2
    assert_equal "10.51", doc.promedio_ponderado_autor1
    assert_equal "18.51", doc.promedio_ponderado_autor2
    assert_equal "14.51", doc.promedio_ponderado.to_s
  end

  def test_cambios_correctos
    atributos = ["titulo", "resumen", "palabras_clave"]
    doc = Documento.find 1
    doc_anterior = Documento.new
    doc_anterior.update_attributes(Hash[*atributos.collect{|actual| [:"#{actual}", doc.send(actual)] }.flatten])
    #puts doc.inspect
    atributos.each{ |atributo|
      doc.send("#{atributo}=", "un cambio")
      assert_not_equal doc_anterior.send(atributo), doc.send(atributo)
    }

  end

end
```

Figura 5.45: Test unitario de la clase *Documento* (*documento_ test.rb*)

Para llevar a cabo la prueba se llama al comando `ruby test/unit/documento_test.rb` desde el directorio raíz del proyecto Rails. Al ejecutar el test unitario no hubo novedad.

Parte IV

Conclusiones

Conclusiones

La investigación en el marco teórico y el desarrollo realizado en este Trabajo Especial de Grado dieron como resultado una aplicación llamada BUSCONEST, la cual es un repositorio de documentos digitales y a su vez una aplicación Web que permite hacer búsquedas sobre éste, al estilo de los grandes buscadores de la Word Wide Web y buscadores de contenido en general.

El logro de los objetivos propuestos se debe a las bases del marco teórico y a la adaptación de la programación extrema (XP). El uso de una modelación práctica en el diseño permitió una clara comprensión de requerimientos para la implementación. Una relativa libertad en el uso de los artefactos durante el desarrollo permitió un desarrollo objetivo, desapegado de dogmas y sobrecarga informativa.

La intención de este Trabajo Especial de Grado fue realizar un sistema interesante y productivo en la práctica. La aplicación es un subsistema de otro principal (CONEST) y sin embargo tiene tal grado de independencia y versatilidad que se permite funcionar por si solo al aplicarle pocos cambios.

En el desarrollo se utilizó un lenguaje dinámico e interpretado (`Ruby`) sobre un framework destacado en la actualidad (`Rails`), ésto condujo a obtener una serie de experiencias y nuevas visiones en la programación de aplicaciones.

Se experimentó con una idea acerca del dinamismo del objeto principal de la aplicación (el documento) en cuanto a su modelado. La idea mencionada consiste en darle al modelado del documento una libertad para poseer datos. Los atributos considerados para ser parte de un documento son muy variados y sujetos a cambios, por ello en lugar de implementar varios, se utilizó una forma estándar y práctica de colocar en un solo campo una agrupación de atributos, el campo consiste de una metadata con formato XML del cual se obtienen unos métodos de acceso. Se observó que resultaba muy útil en la programación orientada a objetos del código, pero sin embargo es necesario también tener algunos atributos apartados y redundantes por razones de optimización en las búsquedas.

En cuanto a la robustez del programa, la aplicación permite una respuesta más dinámica y precisa ante los errores posibles e inesperados. Los buenos resultados de esta solución sugirió explotarla y aplicarla más en toda la aplicación.

La aplicación maneja cantidades de objetos susceptibles de ser numerosos: los documentos del resultado de una consulta. Para ello se implementó una solución fuera del estándar propuesto; consiste en almacenar en una lista en memoria sólo los números de identificación de los documentos encontrados y hacer una manejo paginado de ellos en las vistas correspondientes. En cada página se recuperan los objetos según los identificadores pertenecientes a esa página parcial de resultados. Al probar y poner en práctica la solución resultó tener un desempeño correcto y esperado.

La aplicación BUSCONEST al principio era concebida como un sistema totalmente autónomo, que pedía opcionalmente información a CONEST. Al avanzar el proceso de desarrollo, un requerimiento cambió la naturaleza de la aplicación, convirtiéndola en un subsistema de CONEST. Sin embargo la solución pudo implementarse sin complicaciones y sin restricciones sobre el requerimiento, debido a la estructura del programa y a las prácticas de programación empleadas. Por esto, BUSCONEST tiene propiedades tanto de subsistema como de sistema autónomo.

BUSCONEST se basa en estándares para comunicaciones entre aplicaciones Web sencillas y muy exitosas. Por un lado el empleo de Web Services y por otro el simple uso del protocolo HTTP. Si CONEST cambiara radicalmente su implementación o plataforma, la aplicación seguiría funcionando de la misma forma sin cambiarle nada.

CONEST cuando tiene que cargar un nuevo documento o cuando quiere administrar el repositorio, accede a la aplicación mediante unos enlaces que transmiten datos. Luego en BUSCONEST se validan esos datos y se accede a las vistas, las cuales simulan las interfaces de CONEST con una alta fidelidad que hasta se ofrecen sus funcionalidades. Varias de la validaciones que realiza la aplicación es a través de los Web Services de CONEST.

Es muy importante destacar la capacidad de expansión de la aplicación a nivel de desarrollo, ya que su diseño y estructura de software se hicieron pensando en la adaptación a los cambios basándose en modelos fundamentales producto del trabajo de investigación. La incrementabilidad se garantiza porque el sistema permite agregar más código y estructuras sin que se vean afectadas las funciones y estructuras ya existentes en gran medida. Por ejemplo agregar un atributo y método nuevo al objeto documento implica solamente agregar el nombre de este en una linea de código, se puede agregar mas formatos soportados insertando nuevas lineas en un sólo método, etc. El sistema esta desarrollado en módulos y pensado en que va a evolucionar con cambios necesarios.

La realización de este Trabajo Especial de Grado ha aportado una experiencia muy aleccionadora acerca de cómo mantener el equilibrio en los aspectos de la programación, la flexibilidad del sistema frente a la correcta definición de las estructuras y métodos básicos y que no deberían cambiar mucho, la teoría y prácticas exitosas frente a la innovación. Muchas de las soluciones innovadoras o espontáneas del desarrollo de este sistema resultaron en aciertos recompensados, así como también en algunas dificultades técnicas.

Esta aplicación tiene características especiales que no son parte de los antecedentes y las bases teóricas, por ejemplo La búsqueda avanzada ofrece opciones específicas y confiables como la fecha, el autor, y muchas más que pueden añadírseles. También posee algunos elementos de Web 2.0 que pasan desapercibidos como la carga de archivos de gran tamaño y los procesos en background monitoreados usando tecnología *Ajax*.

Con respecto a la adaptación XP realizada para este trabajo, algunas de las prácticas no pudieron realizarse al pie de la letra. Por ejemplo la programación en parejas no se pudo realizar exactamente como lo dictan los cánones ya que un sólo desarrollador estuvo involucrado, sin embargo las funciones del segundo programador pudieron ser realizadas mediante las revisiones posteriores, el tutor revisaba, corregía y hacía sugerencias sobre la codificación. Debido al tipo de aplicación, no se requería de mucha presencia del "cliente", ya que la aplicación tiene unas características bien conocidas y definidas en el trabajo de investigación, y las particularidades del sistema se iban revisando y corrigiendo con cada entrega, caso que no sucedería con un sistema mas específico y especial. En cada sistema habrá una serie de condiciones que hacen que ciertas prácticas de programación sean aplicadas o adaptadas en mayor o menor grado.

En el desarrollo de la aplicación se utilizaron una cierta cantidad de tecnologías, cada una de ellas muy beneficiosa pero que bajo una buena integración, aporta muchos mas beneficios tanto al usuario como al desarrollador y a la aplicación misma. Una aplicación con buenas bases, que sea incrementable y que sea mantenible, posiblemente le espera una vida larga.

Recomendaciones

BUSCONEST es un sistema repositorio y buscador de documentos digitales. Se ofrecen opciones de búsquedas simples y avanzadas. Maneja tres formatos de documentos: WORD, PDF y texto plano, los cuales cubren casi el total de los casos.

La aplicación estuvo definida desde el comienzo como un aplicación Web al estilo de los buscadores comúnmente utilizados.

Debido a la estructura de la implementación del sistema y a la amplitud de las nuevas necesidades y tecnologías, se pueden mejorar o añadir nuevas funcionalidades y características:

- Soporte para más formatos de documentos. Existen otros formatos populares, por ejemplo Open Document (ODT). Basta con agregar el tratamiento en uno de los métodos.
- Independencia total de la plataforma. Actualmente la aplicación esta desarrollada para funcionar sobre Linux. Se hacen llamadas a unos comandos del shell de Linux y de algunos programas instalados como `pdftotext`. Como todo lo utilizado es de código abierto, pudieran accederse a los códigos fuentes y pasar lo que sea necesario a la aplicación, incluso en su lenguaje C original. Ruby se combina bien con lenguaje C ya que su versión oficial está implementada en en este lenguaje y además permite insertar código fuente de C.
- Agregar clasificaciones a los documentos. Actualmente se manejan documentos y se hacen búsquedas avanzadas pero no se dispone de un sistema de discriminación entre ellos. Los documentos pudieran clasificarse según varios criterios como el formato, el tipo de documento académico (ej: pasantías y seminarios).
- Nuevas interfaces de comunicación. Por ahora la aplicación es enteramente una aplicación Web basado en el protocolo HTTP. Pero mediante tecnología XSLT aplicada sobre la metadata XML se podrían ofrecer otras salidas y otros protocolos como WAP o salidas RSS Feeds.

Bibliografía

[1] Sánchez Jorge. *Tópicos para la Elaboración de Prototipo de Repositorio de Documentos Académicos de la Facultad de Ciencias UCV*. Universidad Central de Venezuela, 2007.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[3] Kent Beck, Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Professional, 2001.

[4] Scott W. Ambler. *Agile Modeling: Effective Practices for EXtreme Programming and the Unified Process*. J. Wiley, 2002.

[5] Jeffries R., Anderson A., Hendrickson C. *Extreme Programming Installed*. Addison-Wesley Professional, 2001.

[6] David Heinemeier. *Agile Web Development With Rails*. Pragmatic Bookshelf, 2006.

[7] David A. Black. *Ruby for Rails: Ruby Techniques for Rails Developers*. 2006.

[8] Bruce Duyshart. *The Digital Document: A Reference for Architects, Engineers and Design Professionals*. Architectural Press, 1997.

[9] Michael W. Berry, Murray Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, 2005.

[10] Baeza-Yates R. *Modern Information Retrieval*. Addison-Wesley, 1999.

[11] David Plà Santamaría. *Localización de información específica en la web*. Editorial de la Universidad Politécnica de Valencia, 2005.

[12] Hossein Bidgoli. *The Internet Encyclopedia*. John Wiley and Sons, 2004.

[13] http://www.extremeprogramming.org

[14] http://www.agilealliance.org

[15] ://www.martinfowler.com/articles/designDead.html

[16] http://www.willydev.net/descargas/prev/ExplicaXP.pdf

[17] http://www.willydev.net/descargas/masyxp.pdf

[18] http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf

[19] http://www.agilemodeling.com

[20] http://homepages.mty.itesm.mx/al792774/programacionExtrema/ProgramacionExtrema.doc

[21] http://www.programacionextrema.org/

[22] http://oness.sourceforge.net/proyecto/html/ch05.html

[23] http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[24] http://www.hipertexto.info/documentos/busq_rec.htm

[25] http://manuales.ojobuscador.com/historia/

[26] http://www.unlu.edu.ar/~tyr/tyr/TYR-motor/manso-motor.pdf

[27] http://sci2s.ugr.es/publications/ficheros/tesis_ difinitiva.pdf

[28] http://eprints.rclis.org/archive/00001772/01/uso.pdf

[29] http://www.eduteka.org/pdfdir/BuscadoresBasico.pdf

[30] http://www.thesmokesellers.com/?p=819

[31] http://www.hipertexto.info/documentos/web_semantica.htm

[32] http://www.w3c.es/Divulgacion/Guiasbreves/WebSemantica

[33] http://www.spip.net/es_article3188.html

[34] http://www.pisitoenmadrid.com/blog/2007/04/como-funciona-un-motor-de-busqueda/

[35] http://www.aspefam.org.pe/ciem/recursos/busquedamotor.htm

[36] http://ingenierias.uanl.mx/32/32_editorial.pdf

[37] http://latorredehercules.blogia.com/2004/121904-razones-para-no-usar-el-formato-.doc.php

[38] http://www.openformats.org/esShowAll

[39] http://www.adobe.com/products/postscript/

[40] http://www.adobe.com/es/pdf/

[41] http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.html.zip

[42] http://www.apple.com/macosx/features/spotlight/

[43] http://www.copernic.com/en/products/index.html

[44] http://desktop.google.com/es/linux/gettingstarted.html

[45] http://www.microsoft.com/latam/windows/desktopsearch/default.mspx

[46] http://www.x1.com/products/

[47] http://www.rubyonrails.org.es/

[48] http://www.ruby-lang.org/es/about/