UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUEL A DE COMPUTACIÓN

DESARROLLO DE UNA APLICACIÓN WEB QUE PERMITE DETECTAR SIMILITUDES ENTRE DOCUMENTOS DIGITALES

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela
por los bachilleres:

Diego Flores G. C.I. 14.200.556
Anja C. Klopp N. C.I. 15.665.008

Tutores:
Prof. Jossie Zambrano
Prof. Sergio Rivas

Noviembre de 2008

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

ACTA

Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado titulado **"Desarrollo de una aplicación Web que permite detectar similitudes entre documentos digitales"** y presentado por los Brs. Diego Flores García, de Cédula de Identidad 14.200.556 y Anja Cristina Klopp Nowacka, de Cédula de Identidad 15.665.008 a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído como fue dicho trabajo, por cada uno de los miembros del jurado, se fijó el día __ de _____ de _____, a las _____ horas, para que los autores lo defendieran en forma pública, lo que estos hicieron en la Sala ___ de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobar con la nota de _____ puntos.

En fe de lo cual se levanta la presente Acta, en Caracas el día ____ de _____ de _____.

_____	_____
Prof. Jossie Zambrano (Tutora)	Prof. Sergio Rivas (Tutor)
_____	_____
Prof. Eliezer Correra (Jurado)	Prof. Damaris Barrantes (Jurado)
_____	_____
Prof. Adrián Bottini (Suplente)	Prof. Mayerling Márquez (Suplente)

Agradecimientos

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

DESARROLLO DE UNA APLICACIÓN WEB QUE PERMITE DETECTAR SIMILITUDES ENTRE DOCUMENTOS DIGITALES

RESUMEN

El presente Trabajo Especial de Grado tiene como objetivo desarrollar una aplicación Web que permita detectar similitudes entre documentos digitales, implementando filtros sencillos y avanzados basados en las características de los trabajos especiales de grado tales como: licenciatura, mención, fecha de presentación y similitud del título, resumen y palabras clave, adoptando la técnica Check para el desarrollo del motor de comparación del sistema. Esta aplicación se desarrolló para ser utilizada por los docentes de la Facultad de Ciencias de la Universidad Central de Venezuela.

Palabras Clave: Similitud, originalidad, copia, plagio, comparación, semejanza, cotejo, documentos, técnica Check.

Autores:
Diego Flores García.
diegolaz@gmail.com

Anja Cristina Klopp Nowacka.
anjaklopp@gmail.com

Tutores:
Prof. Jossie Zambrano.
jossie.zambrano@gmail.com

Prof. Sergio Rivas.
sergiorivas@gmail.com

Fecha: 22 de Octubre de 2008

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Introducción

Al momento de realizar consultas en distintas fuentes de información para desarrollar un trabajo de investigación, bien sea para adquirir nuevos conocimientos o para reforzar los ya adquiridos, se debe tener sumo cuidado de no violar los derechos de autor. Estas violaciones, conocidas también como plagio, ocurren cuando la persona que realiza el trabajo incluye ideas, conceptos, aportes, etc. de otros autores asumiéndolas como propias. Esto es, o bien copiando la información literalmente, o haciendo un parafraseo de ésta, sin hacer las referencias bibliográficas correspondientes en ambos casos.

En el ámbito académico, la copia de documentos es un problema común. Detectar estas copias requiere de tiempo y esfuerzos considerables por parte de las personas. Sin embargo, hoy en día, los grandes avances tecnológicos en el mundo de la informática permiten el desarrollo e implementación de herramientas computacionales que puedan detectar copias entre documentos.

En la Facultad de Ciencias de la Universidad Central de Venezuela existe actualmente una aplicación Web llamada BUSCONEST, la cual es un repositorio y buscador de documentos digitales de índole académico que puede ser consultada desde cualquier lugar.

El objetivo de este Trabajo Especial de Grado es desarrollar una aplicación Web, para ser utilizada por los docentes de la Facultad de Ciencias, que permita comparar los contenidos entre documentos digitales de índole académico, presentados por los estudiantes de Pregrado de esta Facultad, con la finalidad de determinar el grado de similitud con otros existentes en el repositorio.

Este trabajo está dividido en tres capítulos:

En el primer capítulo se expone el planteamiento del problema de este Trabajo Especial de Grado. Se detalla el objetivo general y los objetivos específicos, así como también la justificación, el alcance y las tecnologías.

En el segundo capítulo se explican las bases conceptuales involucradas en el desarrollo de este trabajo. Se abarca el concepto de documento digital junto a una técnica de detección de copias entre los mismos, y se describen las aplicaciones Web bajo la arquitectura cliente servidor, el sistema manejador de base de datos MySQL y el lenguaje de programación Ruby junto al entorno de desarrollo Rails. Finalmente, se hace referencia al sistema de automatización de procesos de control de estudios llamado CONEST y al repositorio y buscador de documentos digitales BUSCONEST.

En el Tercer capítulo se describe el procedimiento llevado a cabo para la comparación de documentos, se expone brevemente la metodología de desarrollo de software RAD (*Rapid Application Development*), y se explican detalladamente los pasos llevados a cabo para la elaboración de una aplicación Web que permita comparar contenidos para establecer similitudes entre documentos digitales, basándose en una adaptación de la metodología RAD.

Por último, se presentan las conclusiones y recomendaciones para trabajos futuros sobre la aplicación propuesta en este trabajo, y las referencias bibliográficas consultadas en esta investigación.

CAPITULO I – PLANTEAMIENTO DEL PROBLEMA

A continuación se presenta el contexto del problema el cual está relacionado con la copia de documentos digitales, específicamente, trabajos de investigación presentados por los estudiantes de la Facultad de Ciencias de la Universidad Central de Venezuela, y se plantea una solución computacional para comparar el contenido de los documentos y establecer similitudes entre ellos.

1. Situación Actual

La División de Control de Estudios de la Facultad de Ciencias de la UCV se encarga de administrar y gestionar las actividades relacionadas a los procesos administrativos y académicos de los estudiantes de pregrado de esta Facultad. Algunos de los procesos llevados a cabo por esta división son la inscripción, calificación de estudiantes, asignación de horarios y aulas, gestión de docentes, generación de reportes e informes, entre otros.

Esta división se encuentra automatizando los procesos haciendo uso de las tecnologías en el área de informática, para facilitar y agilizar la ejecución de las tareas, y mejorar los servicios ofrecidos a los usuarios pertenecientes a la comunidad de la Facultad de Ciencias. Este proyecto de automatización consiste en una aplicación Web llamada CONEST la cual se encuentra operativa desde Febrero de 2007 y sigue en desarrollo mediante la participación activa de estudiantes, docentes y personal administrativo perteneciente a esta comunidad.

Debido a los beneficios que provee el software, como facilitar y agilizar las actividades llevadas a cabo por distintos trabajadores, y a la integración positiva de CONEST, se ha incentivado el desarrollo de soluciones computacionales para las diversas actividades llevadas a cabo en la Facultad de Ciencias. Una de de las iniciativas enmarcadas en el proyecto de desarrollo de CONEST fue la elaboración de un repositorio y buscador de documento digitales de índole académico llamado BUSCONEST. Los estudiantes de la Facultad de Ciencias de la UCV deben realizar un trabajo de investigación al final de sus estudios para optar al título de Licenciado. Estos trabajos se encuentran almacenados físicamente en la Biblioteca Alonso Gamero y en la Biblioteca

Central de la UCV. Contar con un repositorio y buscador de documentos digitales permite almacenar no solo estos trabajos de investigación, sino también otros documentos académicos, de manera que se pueda consultar desde cualquier lugar. Esta aplicación se encuentra vinculada a CONEST porque permite establecer una relación entre los datos académicos de los autores y el contenido del documento presentado.

A medida que pasa el tiempo, no sólo se desean proveer soluciones computacionales que automatizan las diversas actividades llevadas a cabo en la Facultad de Ciencias, sino que también se empiezan a incluir nuevas herramientas que proveen servicios para la comunidad que la conforma, siendo la Escuela de Computación una parte primordial de este proyecto, aportando nuevas líneas de investigación que permitan el desarrollo de nuevas herramientas como es el caso de la propuesta planteada en este Trabajo Especial de Grado.

Al momento de realizar un Trabajo Especial de Grado, se deben consultar en distintas fuentes de información las teorías, investigaciones y aportes necesarios que apoyen o sirvan de base para lo que se quiere desarrollar. Sin embargo, estas consultas pueden conducir a violaciones de derechos de autor cuando se incluyen porciones de texto de otros autores en el trabajo y no se citan debidamente las fuentes de donde fueron extraídas, sino que son consideradas como propias del que realiza el trabajo de investigación. Este tipo de acciones se conoce como plagio, el cual se puede clasificar en los siguientes tipos:

Copiar literalmente párrafos, frases o citas de otros autores sin colocar las referencias o hacerlo de manera incorrecta.

> En general, una referencia está citada correctamente cuando se identifica la porción de texto, gráfico, tabla, etc., que fue extraída de alguna fuente en particular, indicando en detalle tanto el autor como la fuente de la misma. Existen formatos sobre el uso de referencias y citas bibliográficas. Entre los más comunes se pueden mencionar (Microsoft Office Online):

Derecho de autor: “El autor de una obra tiene sobre ella - por el sólo hecho de haberla creado - un derecho de propiedad, exclusivo y oponible a todos, denominado derecho de autor.” (Sociedad Chilena del Derecho de Autor)

- APA
- Chicago
- GB7714
- ISO 690 – Primer elemento y fecha
- ISO 690 – Referencia numérica
- MLA
- SIST02
- Turbian

Parafrasear un texto, es decir, copiar un texto cambiando sólo unas cuantas palabras o alterando el orden de las frases, sin colocar las referencias. Un ejemplo de parafraseo se plantea a continuación.

Texto original:

Al momento de realizar un trabajo de investigación, se debe consultar sobre distintas fuentes de información las teorías, investigaciones y aportes necesarios que apoyen o sirvan de base para lo que se quiere exponer.

Texto parafraseado:

Cuando se desarrolla un trabajo de investigación, se consultan diversas fuentes para apoyar, mediante teorías, investigaciones y aportes, lo que se desea exponer.

Este ejemplo es considerado copia porque:

- Sólo se cambiaron algunas palabras.
- No se menciona la fuente.

Si bien incurrir en este tipo de actividad ha existido siempre, hoy en día es mucho más factible gracias a la distribución, adquisición y copia de información de manera casi inmediata que proveen los grandes avances tecnológicos en el área de la informática, como almacenamiento de datos, redes, sistemas de bases de datos, técnicas de procesamiento, entre otros.

En este trabajo se enfoca la problemática de copia de documentos entre los trabajos de investigación presentados por los estudiantes de la Facultad de Ciencias, los cuales estarán disponibles en un repositorio de documentos digitales localizado en la base de datos del sistema de almacenamiento y búsqueda de documentos BUSCONEST. Para ello, se propone desarrollar una aplicación Web que permita a los docentes comparar contenidos y detectar similitudes entre documentos digitales que tienen características comunes. Estos son: autor(es), tutor(es), licenciatura, mención, fecha de presentación, título, resumen y palabras clave.

En la Figura 1 se muestra la propuesta planteada en este Trabajo Especial de Grado.

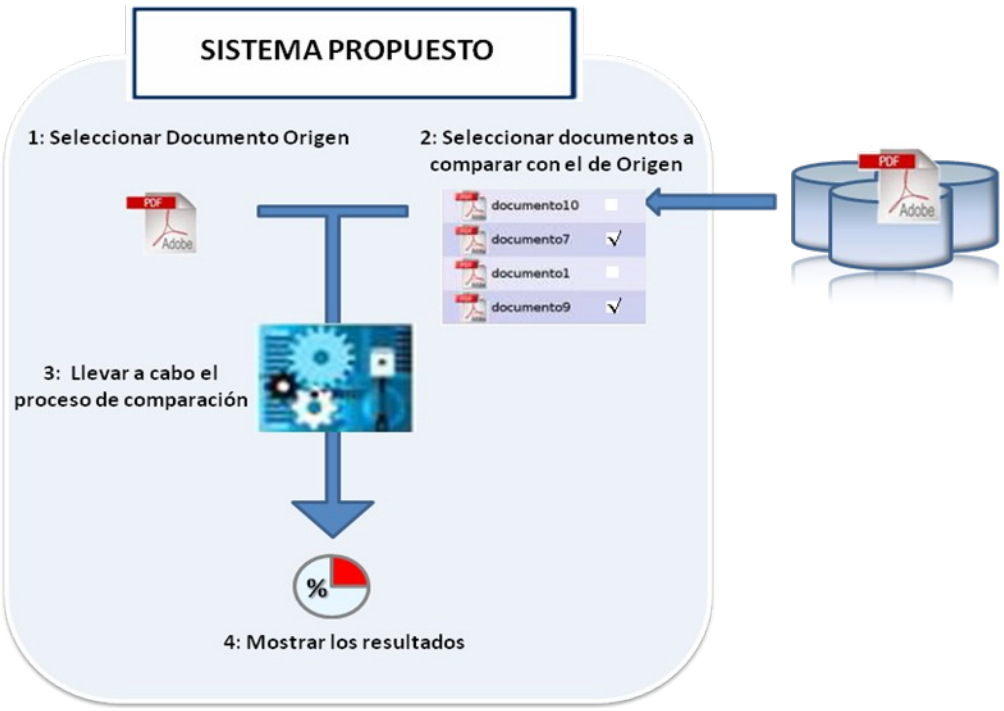


Figura 1, Propuesta de un Comparador de Documentos Digitales.

2. Objetivo General

Desarrollar una aplicación Web para ser utilizada por los docentes pertenecientes a la comunidad de la Facultad de Ciencias de la UCV, que permita, dado un documento origen, comparar su contenido con un conjunto de documentos almacenados en el repositorio BUSCONEST, con la finalidad de detectar las similitudes existentes entre ellos.

3. Objetivos Específicos

Analizar las técnicas de detección de copia de documentos digitales para establecer la que mejor se ajuste a la solución del problema.

Estudiar tanto los modelos de datos como las funcionalidades de las aplicaciones CONEST y BUSCONEST para determinar la integración con el sistema propuesto.

Adaptar e implementar la metodología de desarrollo de software RAD en la elaboración de la aplicación propuesta.

Desarrollar el modelo de datos propio de la aplicación.

Desarrollar el motor de comparación de documentos adaptando la técnica seleccionada para resolver el problema.

Incorporar una funcionalidad al Módulo de Servicio al Docente de CONEST que permita realizar la comparación de documentos digitales de índole académico.

Desarrollar las interfaces de usuario, siguiendo los mismos lineamientos y estándares de interfaz de CONEST.

Realizar pruebas de verificación de código y funcionamiento.

Realizar pruebas de aceptación a los usuarios.

4. Justificación

Cuando se está desarrollando un trabajo de investigación, siempre será necesario consultar distintas fuentes de información. El problema se presenta cuando al adquirir o reforzar los conocimientos de algún tema específico que contribuyan a la realización del trabajo, se parafrasee párrafos o frases o se copie porciones de texto sin hacer las referencias pertinentes, violando los derechos de autor.

Tomando en cuenta este aspecto, desarrollar una aplicación Web que compare el contenido entre los documentos digitales y detecte sus semejanzas, permitir al docente contar con una herramienta para prevenir posibles violaciones de derecho de autor de otros trabajos almacenados.

5. Alcance

El desarrollo de una aplicación Web que permita comparar contenidos y detectar similitudes entre documentos digitales, estará orientado únicamente a prestar un servicio a los docentes de la Facultad de Ciencias de la Universidad Central de Venezuela.

Los docentes sólo podrán realizar comparaciones entre trabajos académicos que se encuentran almacenados en el repositorio de documentos BUSCONEST. Para ello, el docente seleccionará tanto el documento origen (aquel que se comparará contra un conjunto de documentos) como aquellos con los que se realizará la comparación. La selección de estos se llevará a cabo de la siguiente manera:

El docente selecciona el documento origen bien sea a partir de un archivo cargado por él, o a partir de una consulta realizada mediante el buscador de documentos BUSCONEST. Una vez establecido el origen, se despliega un conjunto de documentos que cumplen con condiciones establecidas mediante la implementación de filtros. En estos se incluyen licenciatura, mención, fecha de presentación, y similitud en título, resumen y palabras clave con respecto al origen. El docente selecciona de este conjunto, aquellos documentos con los que desea realizar la comparación.

Es importante tomar en cuenta que los documentos a comparar sólo serán aquellos que además, de lo señalado anteriormente, tengan formato PDF.

A continuación, se resumen los alcances planteados:

La aplicación sólo podrá ser accedida por los docentes que se encuentren registrados en CONEST.

La aplicación debe comparar y detectar similitudes entre documentos digitales con formato PDF.

La aplicación realizará la comparación del documento origen, bien sea seleccionado a partir de una búsqueda en BUSCONEST o suministrado por el usuario, contra un conjunto de documentos que se encuentran previamente almacenados en BUSCONEST.

La aplicación debe proveer un mecanismo de filtrado tanto básico como avanzado, para desplegarle al usuario un conjunto de documentos candidatos a la comparación que cumplan con estos filtros.

6. Tecnologías Utilizadas en el Desarrollo

Debido a que la aplicación está enmarcada en el concepto de desarrollo Web que apoyan la gestión académica de la Facultad de ciencias y brinda un servicio a los docentes de esta Facultad a través del sistema CONEST, se siguieron las mismas convenciones y tecnologías con las que este sistema fue desarrollado. Estas son:

- Lenguaje de programación Ruby versión 1.8.6, el cual es un lenguaje multiplataforma, interpretado y orientado a objetos, cuya implementación es distribuida bajo una licencia de software libre.

- Entorno de desarrollo Rails versión 1.2.6, el cual es un framework de código abierto para el desarrollo de aplicaciones Web.

- Sistema manejador de base de datos MySQL versión 5.0, utilizado para definir, almacenar y procesar los datos.

- Sistema manejador de versiones Subversion 1.4, el cual permite administrar las versiones de cada iteración en el desarrollo del sistema.

Las tres primeras serán ampliadas en el marco conceptual.

CAPITULO II - MARCO CONCEPTUAL

En este capítulo se abarcan las bases conceptuales necesarias para el desarrollo de este Trabajo Especial de Grado. El mismo se encuentra dividido en seis secciones:

En la primera sección se introduce el concepto de documento digital, y se plantea una técnica de detección de copias.

En la segunda sección se explican brevemente las aplicaciones Web y la definición de la arquitectura Cliente Servidor.

En la tercera sección se introduce al sistema manejador de base de datos MySQL, y se exponen algunas de las características principales.

En la cuarta sección, se explica brevemente el entorno de desarrollo de aplicaciones Web conocido como Ruby on Rails.

En la quinta sección, se hace referencia al sistema CONEST y se describen los módulos que lo componen.

Finalmente, en la sexta sección, se describe el sistema BUSCONEST.

7. Documentos Digitales

Gracias al desarrollo y a los avances en las tecnologías informáticas, se ha introducido el concepto de documento digital. En esta sección se definen los documentos digitales y se plantea una técnica utilizada para la detección de copias.

2.1.1. Definición de Documento Digital

Se define como documento a “cualquier unidad significativa de información que haya sido registrada en un soporte que permita su almacenamiento y posterior recuperación” (MSINFO Sistemas de Información). Es un “diploma, carta, relación u otro escrito que ilustra acerca de un hecho, principalmente histórico” (LAROUSSE, 1997), en general, “cualquier otra cosa que sirve para ilustrar o comprobar algo” (LAROUSSE,

1997). Son documentos, por ejemplo, los libros, las revistas, los informes, las facturas, etc.

La información digital se puede definir como “todo aquello que está representado mediante ceros y unos dentro de una computadora” (Biblioteca Digital Universitaria de la DGSCA). Algunos ejemplos de información digital incluyen textos, videos, imágenes, sonidos, entre otros, los cuales están representados y codificados en distintos formatos.

Uniendo estas definiciones, se puede decir que un documento digital es un documento cuya información está representada mediante ceros y unos, y que puede ser visualizada a través de un dispositivo informático. Estos son almacenados en diversos formatos, dependiendo de la aplicación con la que fueron creados. Dentro de los más comunes, se encuentra el formato PDF, acrónimo de Formato de Documento Portable (*Portable Document Format*), creado por Adobe Systems, el cual “permite obtener y visualizar información desde cualquier aplicación y en cualquier sistema informático, así como compartirla con prácticamente cualquier persona en cualquier sitio” (Adobe). Es un formato de almacenamiento de documentos que conserva todas las fuentes, formatos, colores y gráficos con los que fueron creados, manteniendo la presentación final del mismo.

2.1.2. Técnicas para Detectar la Copia de Documentos Digitales

En el trabajo de seminario “Estudio de Técnicas de Medición de Similitud entre Documentos Digitales” (Klopp N, 2008) se plantearon distintas técnicas para la detección de copia de documentos, entre las cuales se escogió el sistema Check que sirvió de apoyo para desarrollar el motor de comparación de la aplicación.

A continuación se explican brevemente las técnicas estudiadas junto con las principales desventajas de cada una.

Técnica: Búsqueda de Patrones Exactos en un Texto

Esta técnica plantea buscar las coincidencias de un patrón dentro de un texto. Un patrón se define como una cadena sobre un alfabeto ∑ donde:

Alfabeto: es el conjunto finito no vacío de símbolos, por ejemplo, formado por las letras del abecedario. ∑ = { a, b, c, d, ..., z}.

Cadena: es cualquier sucesión finita de símbolos del alfabeto Σ , por ejemplo, las palabras formadas como combinación del alfabeto definido previamente (persona, bags, book, etc.).

La búsqueda de patrones dentro de un texto se define como (INAOE, Instituto Nacional de Astrofísica, Óptica y Electrónica):

Dado un patrón P = p1...pm y un texto T = t1...tn, donde p1…pm y t1…tn son secuencias de caracteres sobre el alfabeto finito ∑, encontrar todas las coincidencias de P en T.

En esta técnica, se utiliza una ventana de búsqueda del tamaño del patrón, la cual se desliza de izquierda a derecha, o viceversa, a lo largo del texto, y dentro de la cual se busca el patrón. El esquema general se muestra en la Figura 2.

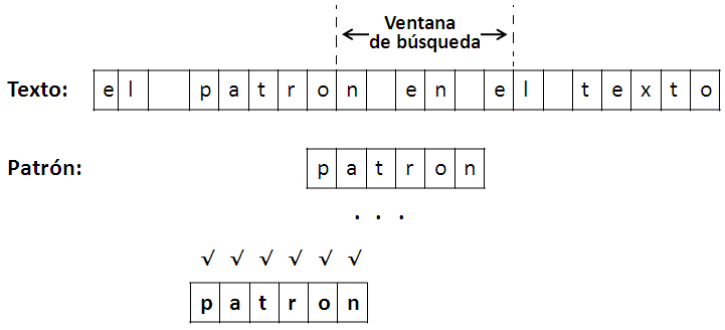


Figura 2, Búsqueda de Patrones Exactos en un Texto

Desventajas:

Una gran limitante de esta técnica consiste en que sólo busca coincidencias idénticas del patrón, por lo que se ve limitada para realizar comparaciones entre textos que comparten similitudes, sin llegar a ser copias exactas, como en el caso del parafraseo.

Técnica: Detección de Documentos Similares usando Términos Importantes (Cooper, Coden, & Brown, 2002)

En esta técnica se extraen los términos de los documentos a comparar y se le asignan un nivel de importancia el cual denominaremos como IQ. Un término que aparece en apenas unos cuantos documentos es altamente selectivo y tendrá un alto IQ. Si, por lo contrario, aparece en muchos documentos, es menos selectivo y tendrá un IQ bajo.

Se buscan los términos más significantes de todos los documentos a comparar (los que tienen un IQ alto) y se procede a hacer la comparación entre cada par de documentos (por ejemplo, sean d1, d2 y d3 los documentos a comprar, la comparación se realiza entre d1 con d2, d1 con d3, y d2 con d3). Para ello, se consultan aquellos términos que se encuentren en ambos documentos y, si la cantidad de términos resultantes de esta consulta es mayor que cierto umbral preestablecido, se identifica los documentos como similares. El esquema general se muestra en la Figura 3.

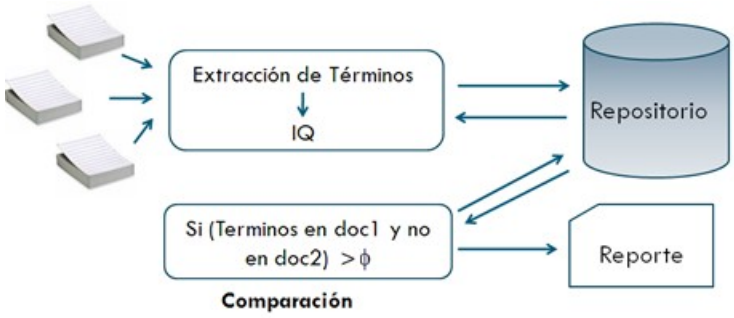


Figura 3, Detectando Documentos Similares Usando Términos Importantes

Desventajas

En esta técnica, para cada conjunto de documentos a comparar se deben extraer los términos para determinar su nivel de IQ. A medida que se incluyan o se excluyan documentos para la comparación, este nivel de IQ debe ser calculado de nuevo, generado una gran cantidad de consultas a la base de datos lo cual no es adecuado por el elevado costo de ejecución que estas tienen, y las cuales se pueden ver aún más comprometidas si los documentos son muy extensos.

Además, para que esta técnica funcione, los documentos a comparar deben ser de tamaño similar. Sin embargo, ésta no determina con claridad la copia entre estos, sino que da conocimiento acerca de las palabras comunes existentes en ambos documentos.

Técnica: Mecanismos de detección de copia para documentos digitales (Brin, Davis, & García-Molina, 1995.)

En esta técnica los documentos son llevados a texto plano y son divididos en secuencias de unidades consecutivas llamados *chunk*, donde las unidades pueden ser secciones, párrafos, sentencias, palabras y caracteres. Por ejemplo, se puede establecer como *chunk* una secuencia de cinco palabras, de dos sentencias, una palabra individual, etc.

A cada *chunk* se le asigna un código *hash*[1], los cuales son almacenados en la base de datos junto al documento en los que aparecen.

Cuando un documento va a ser comparado, se lleva igualmente a texto plano y se procede realizar la extracción de *chunks*. De la misma manera, se asigna un código *hash* y se busca en la base de datos. Si existe, es contada como una similitud. Si la cantidad de similitudes es mayor a un umbral preestablecido, los documentos se consideran copiados.

En la Figura 4 se muestra el esquema general de esta técnica.

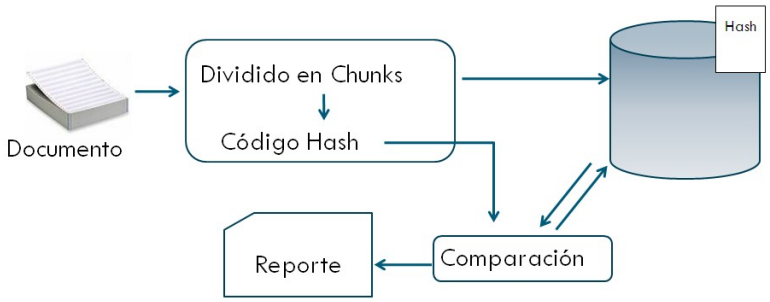


Figura 4, Mecanismos de detección de copia para documentos digitales

Desventajas

[1] Es una clave que se genera como resultado de una función hash, y que sirve para representar de manera unívoca a un documento, registro, archivo, datos, etc.

Sólo aquellos *chunks* que sean iguales tendrán el mismo código *hash*, por lo que sólo de detectarán porciones de textos idénticos, quedando limitado la comparación entre textos que comparten similitudes, sin llegar a ser copias exactas.

A continuación se explica la técnica seleccionada para el desarrollo de este Trabajo Especial de Grado y se presentan sus principales ventajas.

Técnica: Check, Sistema de Detección de Plagio de Documentos. (Si, Leong, & Lau, 1997)

Esta técnica se basa en una comparación de términos presentes en los documentos para determinar la similitud entre éstos. Por cada documento, sección, sub-sección o párrafo, se extraen los términos relevantes (es decir, aquellos que por lo general dan significado al documento como sustantivos, verbos, adjetivos y adverbios) que lo conforman y se les asigna un peso, correspondiente a la frecuencia con la que este aparece, dividido entre la cantidad total de términos. Se crean dos vectores, uno de términos y otro de pesos, los cuales serán usados como parámetros en una función que retorna un estimador de similitud.

A continuación se describe la arquitectura de esta técnica, explicando en detalle su funcionamiento.

Arquitectura del Sistema Check

Check está compuesto de tres módulos principales:

1. **Módulo de parseo de documento**

   Este módulo construye una estructura de indexación interna, la cual llamaremos de aquí en adelante característica estructural o SC, para cada documento que va a ser registrado y comparado.

2. **Módulo de registro de documento**

   Este módulo se encarga de registrar los documentos en una base de datos. Estos registros pueden ser dos tipos:

- Registro de documentos considerados originales, los cuales se almacenan en la base de datos junto a una estructura de indexación interna del documento, sin invocar al módulo de comparación, y
- Registro normal de documentos, los cuales se comparan con los documentos registrados mediante el módulo de comparación, para detectar posibles copias, antes de ser almacenados en la base de datos.

En ambos casos, se invoca al módulo de parseo, encargado de crear la estructura de indexación interna del documento que se utilizará para llevar a cabo la comparación.

3. Módulo de comparación de documento

En este módulo se comparan las características estructurales de un documento de entrada con las de los registrados en la base de datos para detectar posibles copias. Si los documentos son distintos, el documento de entrada es almacenado en la base de datos junto a su estructura de indexación. En caso contrario, el documento no se almacena y se le alerta al usuario sobre esta similitud.

A continuación se describe el diseño y la implementación de los tres módulos del sistema Check:

1. **Módulo de Parseo de Documento**

Está compuesto por los sub-módulos de reconocimiento de documento, extracción de palabras clave y generación de características estructurales.

Sub-módulo Reconocimiento de documento

Este sub-módulo convierte un documento de entrada en un texto plano ASCII. Luego, se crea una estructura llamada árbol del documento, el cual reensambla la estructura de éste. Cada documento puede ser visto en múltiples niveles de abstracción, los cuales incluyen el texto en sí (que llamaremos raíz del árbol), sus secciones, subsecciones (que llamaremos nodos) y párrafos (los cuales denominaremos como hojas). Los párrafos representan el nivel más bajo de la abstracción. En la Figura 5 se

muestra un ejemplo de un documento junto a su árbol correspondiente, compuesto por tres capítulos o secciones, que a su vez están formados por subsecciones y párrafos.

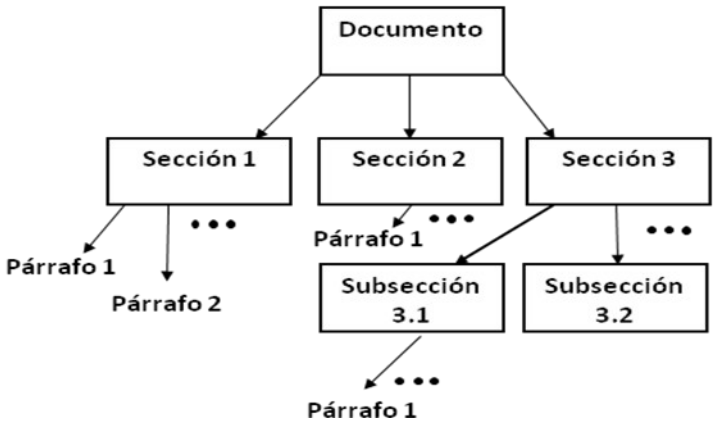


Figura 5, Ejemplo de una Estructura de Árbol de un Documento

Sub-módulo Extracción de palabras clave:

En este sub-módulo, se identifican y extraen las palabras clave que mejor describan la semántica del documento. En general, cada palabra puede ser categorizada en dos tipos de clases: abierta o cerrada. Las palabras de una clase abierta incluyen sustantivos, verbos, adjetivos, y adverbios que usualmente dan significado al texto. Las palabras de clase cerrada incluyen pronombres, preposiciones, conjunciones e intersecciones que, por lo general, no añaden significado al documento, por lo que no son consideradas palabras clave.

El módulo de extracción de palabras clave determina si las palabras pertenecen a la clase abierta o cerrada. Por cada nivel de abstracción del documento, bien sea el documento en sí, una sección, subsección o párrafo, se extraen las palabras identificadas como clase abierta.

Sub-módulo Generación de la característica estructural:

La característica estructural de un documento, el cual denotaremos como A, es creado uniendo el árbol de A con sus palabras clave respectivas, las cuales llamaremos K_A. Por cada nodo, denotado como *ni*, en el árbol del documento, se asigna un lote de palabras clave como sigue:

Si *ni* es la raíz del árbol del documento, las palabras clave del documento A serán K_A. Cada palabra clave de K_A estará asociado a un peso definido por:

$$W\,K_{A,a} = \frac{ocurrencia\ de\ a\ en\ A}{\sum_{x \in K_A} \{ocurrencia\ de\ x\ en\ A\}}$$

Si *ni* reensambla la sección *i* del documento A, las palabras clave K_i asignadas a *ni* contendrán solo aquellas de K_A que aparezcan en la sección *i* del documento. Cada palabra clave de K_i tendrá igualmente un peso asociado definido por:

$$W\,K_{i,a} = \frac{\text{ocurrencia de a en la sección i}}{\sum_{x \in K_i} \{\text{ocurrencia de x en la sección i}\}}$$

Si *ni* reensambla una subsección o párrafos del documento A, las palabras clave y sus pesos asignados a *ni* se calcularán de manera similar al paso 2.

2. **Módulo de Registro y comparación de documentos:**

El módulo de comparación de documentos compara la característica estructural, la cual denotaremos como SC de aquí en adelante, de un documento entrante, con la SC de cada documento registrado en la base de datos.

El módulo de registro almacena un documento entrante al sistema junto a su SC en la base de datos para permitir comparaciones futuras. Un usuario puede introducir un documento en el sistema, e indicar que éste es original. En este caso, el sistema simplemente llama al módulo de registro. Por otro lado, un usuario puede solicitar que el sistema compare un documento antes de registrarlo. En este caso, el sistema llama al

módulo de comparación para evaluar la originalidad del mismo. Si no hay similitudes entre el documento entrante y los registrados, el sistema llama al módulo de registro para su almacenamiento. En caso contrario, se le reporta al usuario la similitud.

Para explicar cómo las características estructurales de dos documentos son comparadas, denotaremos la característica estructural de un documento entrante A como SC_A y la de un documento registrado B como SC_B. La comparación entre SC_A y SC_B se lleva a cabo de la siguiente manera:

Empezando por la raíz de ambas SC (correspondiente al nivel 0), la similitud entre el lote de palabras clave, y por lo tanto la similitud entre los dos nodos raíces, es determinada. Primero se denota el lote de palabras clave del documento A como un vector

$$V_A = [a_{A1}, a_{A2}, a_{A3}, ..., a_{|VA|}]$$

y sus pesos asociados como otro vector :

$$W_A = [W_{A1}, W_{A2}, W_{A3}, ..., W_{|VA|}]$$

donde a_{Ai} y W_{Ai} representan la i-ésima palabra clave y su peso en el documento A respectivamente.

De la misma manera, se definen los vectores:

$$V_B = [b_{B1}, b_{B2}, b_{B3}, ..., b_{|VB|}]$$

y

$$W_B = [W_{B1}, W_{B2}, W_{B3}, ..., W_{|VB|}]$$

donde b_{Bi} y W_{Bi} representan la i-ésima palabra clave y su peso en el documento B respectivamente.

La similitud entre V_A y V_B puede ser determinada siguiendo estos dos pasos: normalización y cálculo del coseno del ángulo entre vectores.

Paso 1 – Normalización

Como los vectores V_A y V_B pueden contener diferente número de elementos, deben ser normalizados al mismo tamaño. Además, ambos vectores, que tienen términos simbólicos, deben ser llevados a vectores numéricos para realizar la comparación.

Se genera un vector de referencia denotado como R a partir de la unión de los elementos de los vectores V_A y V_B sin repetición. Es decir,

$R = ((V_A \cup V_B) - (V_A \cap V_B)) \cup (V_A \cap V_B)$

Quedando el vector

$R = [R_1, R_2, R_3, R\ldots, R_{|R|}] <= |V_A| + |V_B|$

Se denota el vector normalizado correspondiente a V_A y V_B como

$X_A = [X_{A1}, X_{A2}, ..., X_{A|R|}]$

y

$X_B = [X_{B1}, X_{B2}, ..., X_{B|R|}]$

donde X_{Ai} y X_{Bi} son valores entre el intervalo [0,1] el cual representa el peso de la palabra clave i-ésima de R, a_{Ri}, dentro del contexto de A y B respectivamente.

Formalmente, es definido como:

$$x_{A,i} = \begin{cases} 0 & \text{if } a_{R,i} \notin V_A \\ w_{A,j} & \text{if } a_{R,i} = a_{A,j} \in V_A \end{cases}$$

$$x_{B,i} = \begin{cases} 0 & \text{if } b_{R,i} \notin V_B \\ w_{B,j} & \text{if } b_{R,i} = b_{B,j} \in V_B \end{cases}$$

Paso 2 – Cálculo del Coseno del Ángulo

Con los dos vectores normalizados X_A y X_B, la similitud entre V_A y V_B, denotada como $S(V_A,V_B)$, puede ser determinada calculando el coseno del ángulo entre los vectores X_A y X_B. El coseno entre vectores se calcula de la siguiente manera:

$$S(V_A, V_B) = \frac{\sum_{i=1}^{|R|} x_{A,i} \cdot x_{B,i}}{\sqrt{\sum_{i=1}^{|R|} x_{A,i}^2 \cdot \sum_{i=1}^{|R|} x_{B,i}^2}}$$

Un valor alto de $S(V_A, V_B)$ indica una alta similitud entre los dos nodos.

Si $S(V_A, V_B) < \varepsilon\ 0$, donde $\varepsilon\ 0$ representa un umbral de similitudes para el nivel 0, o nodo raíz, los documentos A y B son considerados de temas distintos. En este caso, no tiene sentido hacer futuras comparaciones. El módulo de comparación evalúa entonces el documento A con otros documentos registrados.

Por otro lado, si $S(V_A, V_B) >= \varepsilon\ 0$, se considera que los documentos A y B describen temas similares. En este caso la comparación entre la SC de los documentos A y B será procesada al siguiente nivel. Por cada nivel, que denotaremos con la letra L, donde nivel L > 0, la comparación entre las SC de los nodos es similar al paso uno 1. Dependiendo del umbral de similitud del nivel L, ε L, la comparación será procesada para los hijos de los nodos.

Este procedimiento es repetido por cada nivel de las SCs de los documentos A y B hasta que se determine que no existe similitud, o hasta que los nodos hojas de las SCs sean alcanzados (es decir, que dos párrafos de dos documentos son altamente similares).

En la Figura 6 y la Figura 7 se muestra el funcionamiento general del registro de documentos originales y normales respectivamente del sistema Check, y los módulos y sub-módulos que lo conforman.

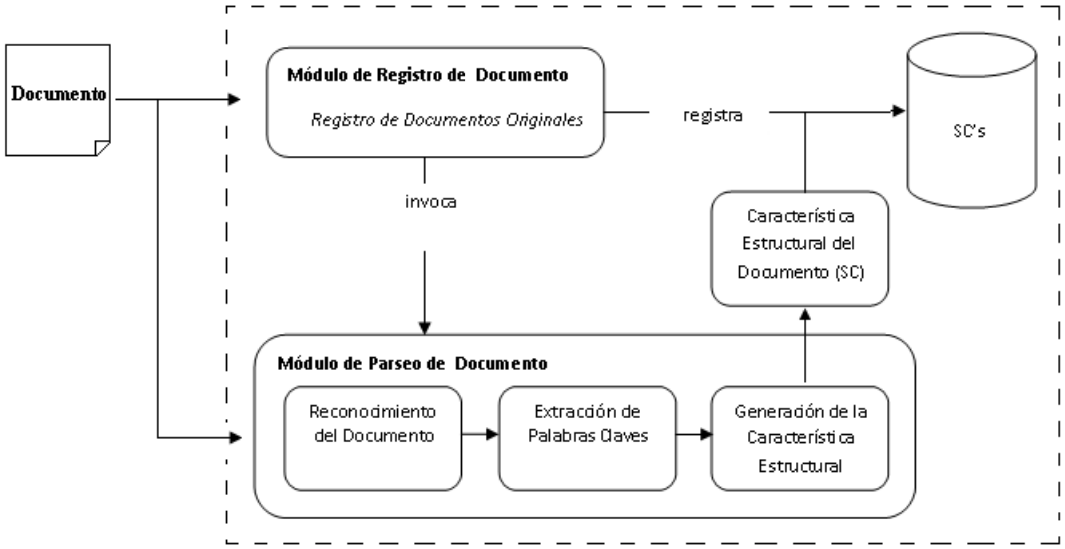


Figura 6, Registro de un Documento Original

Cuando va a ser registrado un documento, considerado original, el sistema invoca, a través del módulo de registro, al módulo de parseo. En este último se lleva el documento a un formato de texto plano, se extraen sus palabras clave, y se genera la característica estructural del mismo, la cual se almacena en base de datos junto al documento. Este proceso se puede visualizar en la Figura 6.

Si el documento a registrar no es considerado original, se realiza una comparación antes de su almacenamiento. Para ello, el módulo de registro invoca al de comparación, quien coteja la característica estructural del documento obtenida mediante el módulo de parseo, contra las características estructurales de aquellos contra los que se realizará la comparación, las cuales ya se encuentran almacenadas. Si se detecta alguna similitud, se reporta al usuario. En caso contrario, se almacena el documento junto a su característica estructural. Este proceso se puede visualizar en la Figura 7.

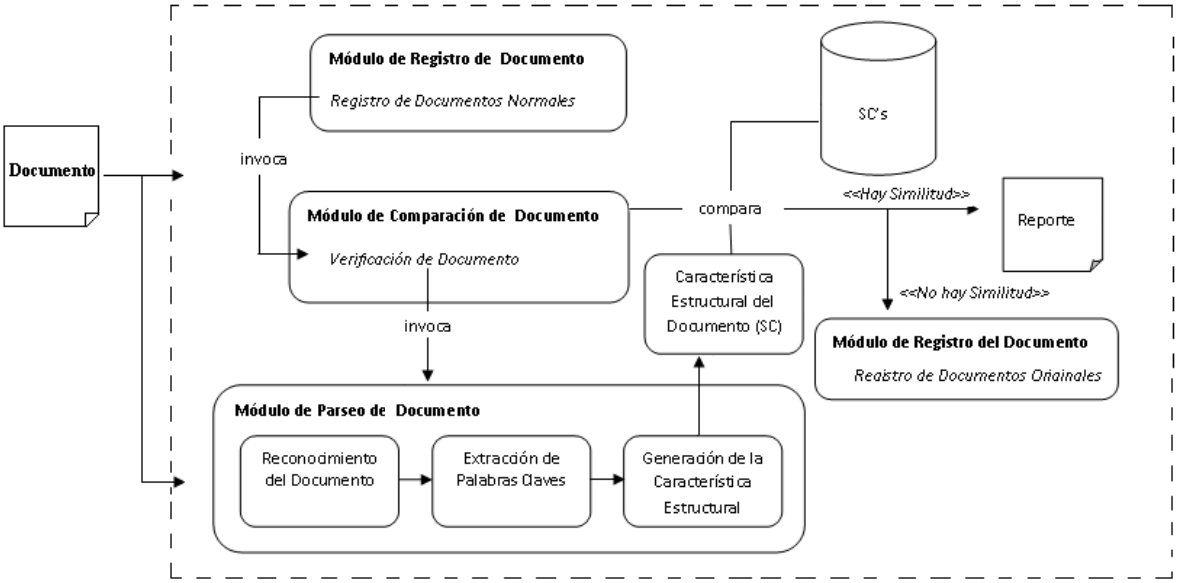


Figura 7, Registro de un Documento Normal

Ventajas

Las principales ventajas de esta técnica se plantean a continuación:

- Permite detectar no sólo las copias entre documentos sino también las similitudes.
- La comparación de los documentos se realiza entre secciones.
- Se almacenan las secciones o estructuras de los documentos junto a sus palabras claves y pesos calculados. Contar con esta información permite realizar comparaciones futuras de los documentos sin tener que realizar de nuevo las extracciones de secciones y palabras clave asociadas, y el cálculo de pesos.

En la Tabla 1 se muestra un cuadro comparativo entre las técnicas planteadas anteriormente.

Técnica \ Acciones	Compara secciones del documento	Compara documentos de distintos tamaño	Detecta similitudes	Detecta copias
Búsqueda de Patrones Exactos en un Texto	Si	Si	No	Si
Detección de Documentos Similares usando Términos Importantes	No	No	Si	No
Mecanismos de detección de copia para documentos digitales	Si	Si	No	Si
Check: Sistema de Detección de Plagio de Documentos	Si	Si	Si	Si

Tabla 1, Cuadro Comparativo entre las Técnicas de Detección de Copias de Documentos

8. Aplicaciones Web – Arquitectura Cliente Servidor

Las aplicaciones Web son aquellas que se ejecutan en un servidor Web y que pueden ser accedidas, por lo general, a través de una red como Internet o una intranet, mediante un navegador. Estas aplicaciones funcionan bajo una arquitectura cliente servidor, donde el cliente es el ente que accede al servidor mediante peticiones realizadas por este, y el servidor, quien las procesa.

Una de las ventajas más significativas que han tenido este tipo de aplicaciones, y que ha incrementado la popularidad de las mismas, es la facilidad con la que pueden ser actualizadas y mantenidas sin necesidad de distribuirlas e instalarlas en muchas computadoras, no requerir de equipos de gran capacidad del lado del cliente, y poder ejecutar la aplicación desde cualquier computador que pueda comunicarse con un servidor, a través de un navegador web. Además, este tipo de aplicaciones pueden funcionar en cualquier plataforma que posea navegadores estándar.

Uno de los beneficios más significativos que tienen los avances tecnológicos en el área de la informática, junto con la existencia de documentos digitales, es el poder dar acceso y distribuir la información a muchas personas de manera casi inmediata. Para que esto sea posible, es necesario contar con una arquitectura donde los documentos estén

almacenados en un repositorio y los usuarios puedan acceder a los mismos. Este tipo de arquitectura es conocida como la Arquitectura Cliente Servidor.

2.2.1. Definición de la Arquitectura Cliente Servidor

La arquitectura cliente servidor se puede definir como aquella en donde “el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas se divide entre dos o más procesos que cooperan entre sí.” (Universidad de los Andes)

Esta arquitectura consiste básicamente de un cliente, quien realiza peticiones, y un servidor, encargado de recibirlas y procesarlas. En estas arquitecturas, los procesos se dividen de manera lógica entre clientes y servidores los cuales, por lo general, se encuentran en distintas máquina. Usualmente, los servidores se encargan del procesamiento de datos, aplicaciones y manejo de la información o recursos, mientras que los clientes se encargan de la interacción con el usuario. La comunicación entre ambos se hace, por lo general, mediante una red como Internet o una Intranet.

En la Figura 8 se muestra el funcionamiento general de esta arquitectura: El cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y éste envía uno o varios mensajes con la respuesta (provee el servicio).

Figura 8, Arquitectura Cliente Servidor

2.2.2. Componentes de la Arquitectura Cliente Servidor

Los componentes que conforman la arquitectura cliente servidor son los que se han estado mencionando en el punto anterior. Estos son los clientes y los servidores.

A continuación se describen estos componentes, junto a las principales actividades que realizan cada uno de estos.

Cliente

Es el ente encargado de la interacción con el usuario: recibe las peticiones formuladas por éste, las procesa de tal manera que cumplan ciertas especificaciones para que el servidor las pueda ejecutar, y las envía al servidor. Una vez procesadas, el cliente recibe la respuesta de la petición, y la despliega al usuario, manteniendo la transparencia en cuanto a la ubicación de los datos o aplicaciones.

En general, el cliente se encarga de recibir las peticiones del usuario, hacer validaciones locales, procesar la lógica de la aplicación, recibir las respuestas del servidor y desplegar los resultados.

Servidor

Es el encargado de proveer un servicio y responder las solicitudes de los múltiples clientes que hacen peticiones de algún recurso administrado por él. En general, se encarga de recibir las peticiones del cliente, procesarlas, realizar validaciones, controlar el acceso concurrente a las bases de datos compartidas y enviar las respuestas al cliente.

2.2.3. Ventajas de la Arquitectura Cliente Servidor

El servidor puede controlar los accesos, recursos e integridad de los datos evitando que clientes no autorizados o defectuosos dañen el sistema.

Las aplicaciones pueden ser actualizadas y mantenidas sin necesidad de distribuirlas e instalarlas en muchas computadoras: Sólo será necesarios realizar los cambios en el lado del servidor.

Escalabilidad: Se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).

Disminución de costos en los equipos clientes. Debido a que los servidores son los encargados de procesar las peticiones, las estaciones clientes pueden ser computadoras que no requieran de gran capacidad.

Multiplataforma: Estas aplicaciones pueden ser accedidas desde cualquier cliente que tenga un navegador Web, sin importar el sistema operativo con el que trabajen.

2.2.4. Desventajas de la Arquitectura Cliente Servidor

Si el servidor recibe una gran cantidad de peticiones simultaneas, el tráfico de la red puede verse afectado.

Si el servidor se encuentra inoperativo, las peticiones de los clientes se ven comprometidas al no poder ser procesadas.

9. Sistemas Manejadores de Base de Datos

El sistema manejador de base de datos es “un programa o conjunto de aplicaciones para almacenar, manipular y recuperar información en una BD” (Universidad Distrital Francisco José de Caldas). “Es un software que controla la organización, el almacenamiento, la recuperación, seguridad e integridad de la información en una Base de Datos”. (Sociedad de Estudiantes de Ciencia de la Computación)

En general, un sistema manejador de base de datos es un conjunto de programas que se encargan de definir, procesar y administrar las bases de datos. El SMBD sirve de interfaz entre la base de datos y los usuarios y aplicaciones que lo utilizan. Se compone de un lenguaje de definición de datos: DDL (*Data Definition Language*) y de un lenguaje de manipulación de datos: DML (*Data Manipulation Language*).

El lenguaje de definición de datos DDL permite crear, modificar y eliminar las estructuras donde se van a almacenar y organizar los datos.

El lenguaje de manipulación de datos DML permite llevar a cabo las tareas de creación, consulta, modificación o eliminación de los datos.

Entre las principales funciones del SMBD se tienen:

- Definición y manipulación de los datos y estructuras de la base de datos.
- Garantizar la integridad y seguridad de los datos.
- Proveer mecanismos de recuperación y respaldo.

- Controlar la concurrencia.

En la Figura 9, se muestra que tanto los usuarios como los programas de aplicación pueden interactuar con la base de datos, gracias a la interfaz que provee el SMBD y el lenguaje DML que permite la manipulación de los datos.

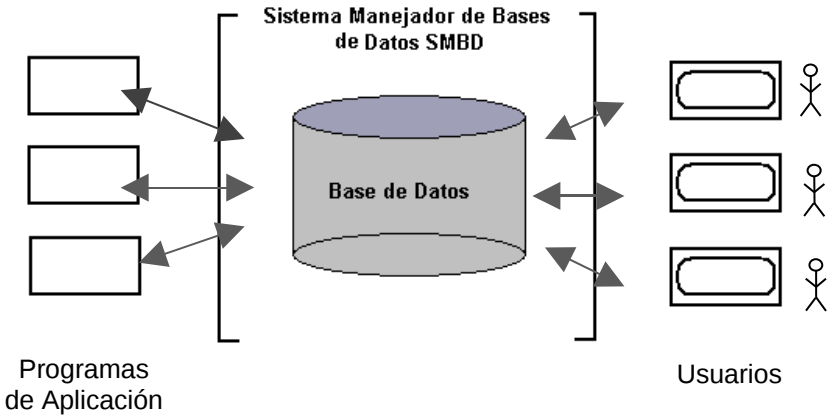


Figura 9, Sistema Manejador de Base de Datos (SMBD)

Actualmente, existen una gran cantidad de sistemas manejadores de bases de datos tanto propietarios como de software libre. Con el arribo de Internet, este último se ha consolidado como alternativa económicamente sostenible, lo que quizá representa uno de los mayores beneficios.

Dentro de los sistemas manejadores de bases de datos de software libre más comunes se encuentran PostgreSQL y MySQL. En este trabajo se detalla MySQL, ya que los documentos digitales se encuentran almacenados en un SMBD MySQL.

2.3.1. Sistema Manejador de Base de Datos MySQL

MySQL es un sistema manejador de bases de datos relacional, multihilo y multiusuario desarrollado, distribuido y soportado por la compañía MySQL AB, la cual creó este sistema con un esquema de doble licencia: por un lado ofrece este SMBD como un producto de código abierto bajo la GNU GPL (*General Public License*) y por otro, lo ofrece bajo una licencia comercial estándar de MySQL AB. (MySql)

"El software MySQL® proporciona un servidor de base de datos SQL (*Structured Query Language*) muy rápido, multi-threaded, multi usuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido. MySQL es una marca registrada de MySQL AB." (MySql)

Entre las características más importantes de este software se tienen: (MySql)

- Interioridades y portabilidad
    - Escrito en C y en C++
    - Probado con un amplio rango de compiladores diferentes.
    - Multiplataforma. Funciona en plataformas como AIX, BSD, FreeBSD, HP-UX, GNU/Linux, Mac OS X, NetBSD, Novell Netware, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, Windows 95/98/NT/2000/XP/Vista y otras versiones de Windows.
    - APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
    - Uso completo de multi-hillos mediante hilos del kernel, aprovechando así la ventaja de usar multiprocesamiento con varios procesadores.
    - Relativamente sencillo de añadir otro sistema de almacenamiento. Esto es útil si desea añadir una interfaz SQL para una base de datos propia.
    - Un sistema de reserva de memoria muy rápido basado en threads.
    - Joins muy rápidos usando un multi-join de un paso optimizado.
    - Las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible.
    - El servidor está disponible como un programa separado para usar en un entorno de red cliente/servidor. También está disponible como biblioteca y

puede ser incrustado en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible.

- Tipos de columnas
    - Diversos tipos de datos permitidos: enteros con/sin signo de 1, 2, 3, 4, y 8 bytes de longitud, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM, y tipos espaciales OpenGIS.
    - Registros de longitud fija y longitud variable.
- Sentencias y funciones
    - Soporte completo para operadores y funciones en las cláusulas de consultas SELECT y WHERE. Por ejemplo:
    - Soporte completo para las cláusulas SQL GROUP BY y ORDER BY. Soporte de funciones de agrupación (COUNT(), COUNT(DISTINCT ...), AVG(), STD(), SUM(), MAX(), MIN(), y GROUP_CONCAT()).
    - Soporte para LEFT OUTER JOIN y RIGHT OUTER JOIN cumpliendo estándares de sintaxis SQL y ODBC.
    - Soporte para alias en tablas y columnas como lo requiere el estándar SQL.
    - El comando específico de MySQL SHOW puede usarse para obtener información acerca de la base de datos, el motor de base de datos, tablas e índices. El comando EXPLAIN puede usarse para determinar cómo el optimizador resuelve una consulta.
    - Los nombres de funciones no colisionan con los nombres de tabla o columna. Por ejemplo, ABS es un nombre válido de columna. La única restricción es que para una llamada a una función, no se permiten espacios entre el nombre de función y el '(' a continuación.
    - Puede mezclar tablas de distintas bases de datos en la misma consulta.

- Seguridad
    - Tiene un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.
- Escalabilidad y límites
    - Soporte a grandes bases de datos.
    - Se permiten hasta 64 índices por tabla. Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas. Un índice puede usar prefijos de una columna para los tipos de columna CHAR, VARCHAR, BLOB, o TEXT.
- Conectividad
    - Los clientes se pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma.
    - La interfaz para el conector ODBC (MyODBC) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (Open Database Connectivity).
    - La interfaz para el conector J MySQL proporciona soporte para clientes Java que usen conexiones JDBC.
- Localización
    - El servidor puede proporcionar mensajes de error a los clientes en muchos idiomas.
    - Soporte completo para distintos conjuntos de caracteres, incluyendo latin1 (ISO-8859-1), german, big5, ujis, y más.
    - Todos los datos se guardan en el conjunto de caracteres elegido.

- Clientes y herramientas
    - MySQL server tiene soporte para comandos SQL para chequear, optimizar, y reparar tablas.
    - Todos los programas MySQL pueden invocarse con las opciones --help o -? para obtener asistencia en línea.

10. Framework Ruby on Rails

Antes de describir el framework Ruby on Rails, se debe introducir a al concepto tanto de Ruby como de Rails por separado.

Ruby

Ruby es un lenguaje de programación multiplataforma, interpretado, flexible y orientado a objetos creado por el japonés Yukihiro "matz" Matsumoto. Su implementación oficial es distribuida bajo una licencia de software libre. (Ruby)

Rails

Antes de definir Rails, se debe explicar el concepto de framework. “Los *frameworks* son parte fundamental en la ingeniería del software, ya que promueven la reutilización del código del diseño y el código fuente.” (Solano Murillo & Coles) Son un “conjunto de Apis y herramientas destinadas a la construcción de un determinado tipo de aplicaciones” (Sourceforge).

Rails es un *framework* de código abierto para el desarrollo de aplicaciones web, basado en el patrón de diseño Modelo Vista Controlador (MVC). Fue creado por David Heinemeier Hansson y liberado por primera vez al público en julio del 2004.

2.4.1. Ruby on Rails

"Ruby on Rails es un entorno de desarrollo web de código abierto que está optimizado para satisfacción de los programadores y de la productividad". (Ruby on Rails)

Ruby on Rails, también conocido como RoR, es un *framework* de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

2.4.2. Patrón MVC (Modelo Vista Controlador) en Ruby on Rails

El patrón Modelo Vista Controlador es un patrón de arquitectura de software que separa los datos (modelo) de las interfaces de usuario (vista) y de la lógica de control (controlador) en el desarrollo de una aplicación.

Surge con la finalidad de mejorar la reusabilidad de las aplicaciones de manera que se desacople la vista del modelo. De esta forma, las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos y viceversa.

En la Figura 10 se puede observar la lógica del patrón MCV en RoR.

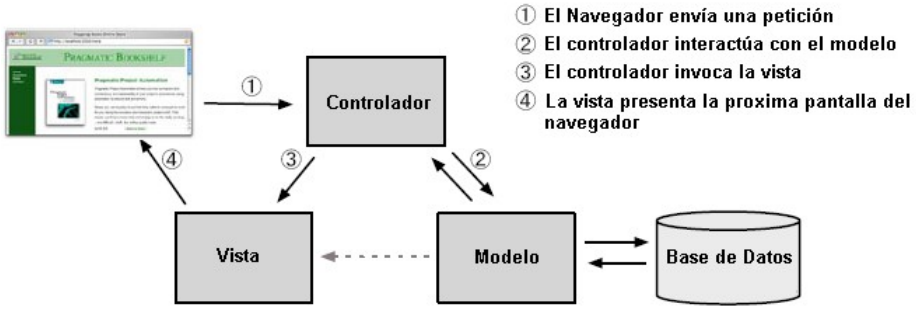


Figura 10, Patrón MVC en RoR

2.4.3. Componentes del patrón MVC en Ruby on Rails

Los componentes que forman el patrón MVC son el Modelo, la Vista y el Controlador. Estos son explicados a continuación:

Modelo

El modelo especifica los datos con el que el sistema opera y las manipulaciones que se van a hacer sobre estos. Es el componente responsable de mantener el estado de los datos, que, por lo general, se almacenan en una base de datos.

En las aplicaciones web orientadas a objetos desarrolladas en RoR y que trabajan sobre bases de datos, generalmente el Modelo consiste en las clases que representan a las tablas de *Active Record*. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::*Base*, y el programa definirá automáticamente qué tabla usar y qué columnas tiene.

Vista

La vista es la lógica de visualización, o cómo se muestran los datos procesados por las clases del Controlador. Con frecuencia en las aplicaciones web desarrolladas con RoR la vista se crea con cantidad mínima de código incluido en HTML. Por defecto, RoR gestiona las vistas usando Ruby embebido en archivos HTML. Estos archivos tienen la extensión rhtml.

En Ruby on Rails las vistas de la aplicación son gestionadas por *ActionView*.

Controlador

El Controlador responde a la interacción con el usuario a través de las interfaces (vista) e invoca a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas.

La implementación del Controlador es manejada por *ApplicationController*, que pertenece a *ActionPack* de Rails.

RoR presenta las mismas ventajas que las provistas por el lenguaje Ruby y el framework Rails. Entre estas, se pueden destacar:

Ventajas de Ruby:

- Lenguaje Orientado a Objetos.
- Fácil de aprender y entender.

- Enriquecido con un alto conjunto de librerías.
- Distribuido bajo una licencia software libre de código abierto.
- Reduce la escritura de código.

Ventajas de Rails:

- Integra una api de conexión a base de datos compatibles con mysql, post-grees, oracle, entre otras.
- Relaciona Objetos con Base de Datos, lo que permite que se consulten los objetos por sus atributos y no por consultas mediante querys.
- Sigue el paradigma MVC lo que permite un desacoplamiento de la lógica de control, los datos y las interfaces de la aplicación, permitiendo la construc-ción.
- Su funcionalidad es expandible por medio de plugins que proveen nuevas librerías y funciones adicionales a rails.
- Su servidor es ligero y de rápido despliegue.

11. CONEST

CONEST es un proyecto que nació con el objetivo de automatizar los procesos que se ejecutan en la División de Control de Estudios de la Facultad de Ciencias de la UCV haciendo uso de los avances tecnológicos en el área de informática junto a la participación de docentes y estudiantes de la Escuela de Computación.

Actualmente, este sistema se encuentra en constante mantenimiento y ampliando sus servicios, mediante la participación activa de estudiantes, docentes y personal administrativo de la comunidad que conforma la Facultad de Ciencias.

Desde el punto de vista académico, CONEST permite el aprendizaje por parte de los estudiantes involucrados en el proceso de desarrollo, dándole a éstos experiencia en cuanto al análisis, diseño y construcción de aplicaciones Web en ambientes y

circunstancias reales, mediante la realización de sus pasantías, seminarios y Trabajos Especiales de Grado.

Actualmente CONEST se encuentra en su versión 0.8.5_7. y está conformado por seis módulos aplicativos, de los cuales cuatro se encuentran en producción. Éstos últimos son Módulo de Administración, Módulo de Servicio al Docente, Módulo de Servicio al Estudiantes y Módulo de Ingreso.

A continuación se describen brevemente los módulos que conforman al sistema CONEST y que se encuentran en producción:

Módulo de Administración

Es el módulo encargado de gestionar algunas de las actividades académicas y administrativas que se llevan a cabo en la División de Control de Estudios (DCE) de la Facultad de Ciencias de la UCV. Desde este módulo se pueden realizar actividades de gestión referentes a los procesos de calificación, inscripción, asignación de planta física y grado entre otras.

Módulo de Servicio al Docente

Este módulo provee servicios aproximadamente a 500 docentes que dictan clases de pregrado en las Licenciaturas de Biología, Computación, Física, Geoquímica, Matemática y Química pertenecientes a la Facultad de Ciencias, mediante el cual pueden calificar estudiantes cursantes de las materias que dictan, consultar el listado de estudiantes inscritos o preinscritos en estas, acceder a los historiales académicos de los estudiantes, consultar los horarios de las materias dictadas, actualizar sus datos personales, entre otras.

Módulo de Servicio al Estudiante

Este módulo provee servicio aproximadamente a 3400 estudiantes inscritos en las Licenciaturas de Biología, Computación, Física, Geoquímica, Matemática y Química de la Facultad de Ciencias, desde el cual pueden acceder a su historial académico, realizar su inscripción en un período académico (siempre y cuando ésta no se encuentre limitada por particularidades especiales como reglamento de permanencia, deudas por parte de los

estudiantes ante dependencias internas, entre otros), visualizar los horarios de las materias inscritas, consultar el comprobante de inscripción. Además en este módulo se despliegan las noticias relevantes sobre el semestre.

Módulo de Ingreso

Este módulo se encarga de gestionar y administrar las inscripciones de los estudiantes que ingresan por primera vez a la Facultad. Los estudiantes suministran, a través de éste módulo, sus datos académicos y personales.

CONEST tiene una gran ventaja y es que ha ido automatizando la compleja administración y gestión de actividades académicas vinculadas a la DCE, agilizando los procesos y reduciendo los tiempos de respuestas. Además, se logró la centralización de la información mediante la integración de las actividades donde interactúan diariamente docentes y estudiantes. Desde el punto de vista académico, permite a sus usuarios (en especial a los estudiantes de Computación) contribuir a nuevas soluciones desde sus actividades académicas como pasantías, seminarios y Trabajos Especiales de Grado.

En la Figura 11 se observan los módulos que componen a CONEST y que se encuentran en producción.

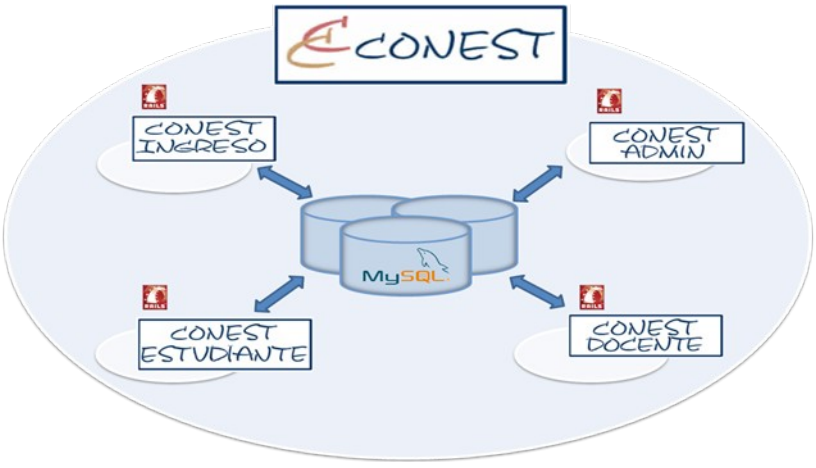


Figura 11, Módulos de CONEST en Producción

Los módulos que aún no se encuentran en producción son CONEST SERVICIO y BUSCONEST. El primero se encarga de gestionar y administrar las actividades y solicitudes que se llevan a cabo en las distintas escuelas y dependencias de la Facultad.

El segundo, es un buscador y repositorio de documentos digitales de índole académico presentados por los estudiantes de pregrado de la Facultad de Ciencias.

12. BUSCONEST: Repositorio y buscador de Documentos Digitales.

BUSCONEST es un sistema Web de almacenamiento, registro y búsqueda de documentos digitales, desarrollado con el fin de facilitar la consulta y acceso a documentos académicos pertenecientes a los estudiantes de pregrado de la Facultad de Ciencias de la UCV.

BUSCONEST es una aplicación de acceso libre a cualquier comunidad, ofreciendo las siguientes ventajas:

Recopilación de un repositorio de documentos digitales de índole académico.

Permite el acceso al repositorio a través de la Web.

Permite realizar consultas implementando búsquedas sencillas y avanzadas vinculadas a los índices académicos de sus autores.

Permite la descarga de los documentos consultados, en caso de que éstos tengan el permiso de ser publicados.

CAPITULO III - MARCO APLICATIVO

En este capítulo se describe la metodología rápida de aplicaciones RAD (*Rapid Application Development*) y se explica cómo fueron adaptadas tanto esta metodología como la técnica Check ajustadas al contexto de este Trabajo Especial de Grado. Por último, se detallan todas las fases de la metodología llevadas a cabo en la elaboración de una aplicación Web que permita comparar y detectar similitudes entre documentos digitales.

13. Metodología de desarrollo utilizada.

A continuación se describe la metodología de desarrollo rápido de aplicaciones, conocida como RAD, la cual será adaptada e implementada en la elaboración de la aplicación Web propuesta en este Trabajo Especial de Grado.

3.1.1. Desarrollo Rápido de Aplicaciones (RAD)

Cuando surge un proyecto para elaborar algún software, se debe llevar a cabo una serie de pasos y procedimientos que permitan el desarrollo del mismo. A esto se le conoce como metodología de desarrollo de software.

El desarrollo de software se hacía mediante una rigurosa definición de roles, actividades involucradas, estructura del proyecto, interfaces, artefactos que se deben producir, y las herramientas a usar dirigidos por una extensa documentación detallada que luego dará paso al proceso de codificación. Este esquema o metodología “tradicional” no resulta muy adecuado para muchos de los proyectos actuales, donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Es por ello que surgen las metodologías ágiles como una alternativa a estos procesos de desarrollo de software tradicionales. Estas metodologías se diferencian de las tradicionales principalmente en que se pone más énfasis en la adaptabilidad que en la previsibilidad. Son metodologías que buscan desarrollar software rápidamente y responder a los cambios que puedan surgir a lo largo

del proyecto en vez de intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después para controlar los cambios en los requisitos.

El desarrollo rápido de aplicaciones (RAD *Rapid Application Development*) es una metodología de desarrollo de software diseñado para facilitar y acelerar la creación de aplicaciones.

Los actores involucrados en esta metodología son los clientes o usuarios, quienes definen los objetivos y requerimientos del sistema o aplicación, y evalúan los resultados; y los analistas o desarrolladores quienes se encargan de llevar a cabo el proceso de diseño y construcción del sistema.

Fases de la Metodología RAD

La metodología RAD se basa en 3 etapas o fases descritas a continuación:

Fase 1 - Planteamiento de los requerimientos o Planificación

En esta fase los usuarios determinan cuales serán las funcionalidades del sistema. Los usuarios y analistas deben reunirse para establecer cuáles serán los objetivos y requerimientos se deben cumplir.

Fase 2 - Construcción

En la fase de construcción se plantean los prototipos a construir. Se incluyen los modelos de negocio, datos y procesos. La construcción de la aplicación consiste en una serie de pasos donde los usuarios tienen la oportunidad de afirmar los requisitos y evaluar los resultados. Las pruebas del sistema se llevan a cabo durante esta fase. También se crea la documentación y las instrucciones necesarias para manejar la nueva aplicación, rutinas y procedimientos para operar el mismo. Debe existir la comunicación entre usuarios y equipo de desarrolladores para lograr la retroalimentación y mejorar los módulos diseñados basados en las respuestas de los usuarios. Esta fase se realiza de manera iterativa hasta lograr la construcción del sistema final.

Fase 3 - Implantación

Se realiza la integración del sistema y la entrega a los usuarios. En esta etapa se adiestran los usuarios y se lleva a cabo la retroalimentación de los mismos.

En la Figura 12 se muestra una comparación de las metodologías tradicionales contra la de RAD junto a las etapas que contempla esta metodología.

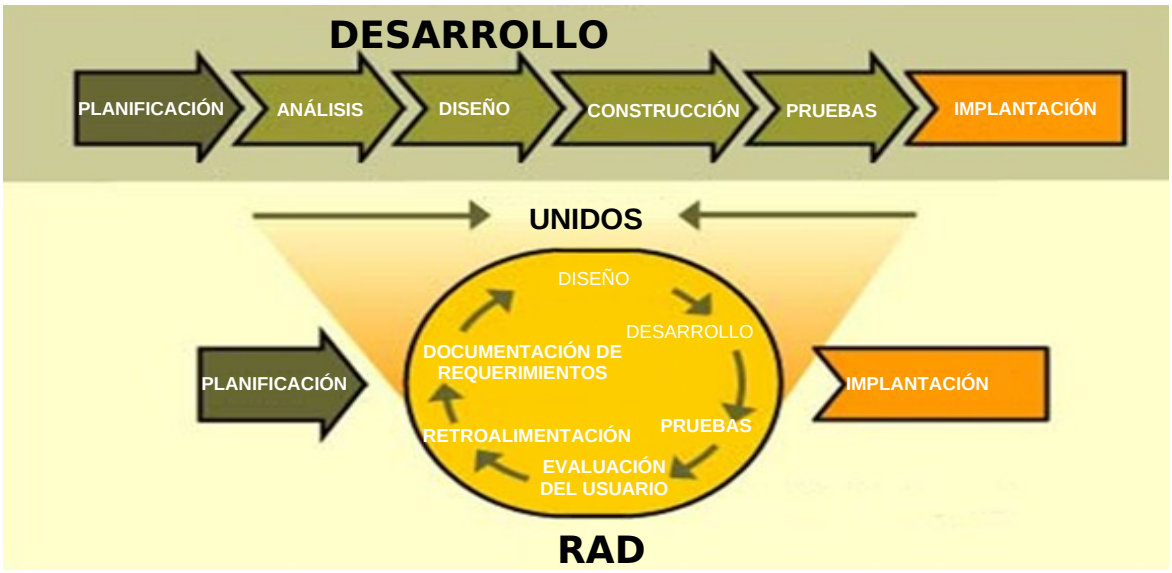


Figura 12, Metodología Tradicional Vs. RAD

3.1.2. Adaptación de la Metodología RAD

Como se mencionó anteriormente, RAD provee una gran flexibilidad antes los posibles cambios que pudieran surgir a lo largo del proyecto. Esta flexibilidad fue una de las motivaciones por las que se deseó utilizar este tipo de metodología, además de la gran iteración con el cliente permitida en RAD, necesarias para la evolución del proyecto mediante aportes y modificaciones solicitadas por este.

A continuación se describen las fases que se plantearon para la implementación de la metodología.

Fase 1 - Planificación

En esta fase se definen, a groso modo, los requerimientos funcionales que deberá cumplir el software. Para ello, el cliente especifica cuál será el uso del sistema, los usuarios a los que estará dirigido, y las operaciones que éstos podrán realizar.

Fase 2 - Construcción

En esta fase se lleva a cabo el proceso de desarrollo del sistema. Esto se logra mediante la implementación continua de iteraciones, caracterizadas por la definición de los requerimientos y retroalimentación por parte del cliente. La idea es que el desarrollo del sistema se lleve a cabo mediante iteraciones, en donde en cada una de estas se incluyan bien sean nuevos requerimientos, cambios o mejoras de unos ya implementados.

Desarrollar el sistema mediante iteraciones, tiene la gran ventaja de permitir entregas parciales del sistema, logrando una buena comunicación con el cliente, y por lo tanto, permitiendo una retroalimentación optima para la modificación, inclusión o mejora de requerimientos en caso de ser necesario.

Las iteraciones se realizan las veces necesarias hasta lograr la construcción total del sistema, satisfaciendo las exigencias del cliente.

Por cada iteración, se llevaron a cabo los siguientes pasos:

- Diseño. Se definen los requerimientos y se modelan sus implementaciones. Aquí se define ¿qué se debe hacer?

- Desarrollo de los requerimientos. Se describe la implementación que da solución a los requerimientos. Aquí se define ¿cómo se hizo?

- Pruebas. Se aplican para comprobar el correcto funcionamiento de los requerimientos desarrollados en la iteración.

- Cambios a implementar. Cuando culmina una iteración, se hace una entrega parcial al cliente, de manera que existe una retroalimentación por parte del éste, quien define los cambios o mejoras que deben implementarse en caso de que sea necesario.

Fase 3 – Implantación

En esta fase se integra el sistema en su totalidad y se realiza la entrega final al cliente, mostrando la funcionalidad de la misma.

14. Adaptación de la Técnica Check

Para la elaboración de la aplicación Web planteada en la propuesta de este Trabajo Especial de Grado, se adaptó la técnica de detección de copias de documentos digitales Check, la cual fue explicada en el marco conceptual. Para ello, se decidió tomar en cuenta:

- Extracción de palabras clave: Sólo se tomarán en cuenta las palabras clave que son las que, por lo general, dan significado al texto, como sustantivos verbos, adjetivos y adverbios.

- Asignación de pesos para cada palabra clave: Por cada palabra clave, se asigna un peso correspondiente a la frecuencia que esta aparece dentro de la sección, entre la cantidad total de palabras de la misma. Se consideró como sección a los párrafos.

- Generación de los vectores normalizados y de referencia: Se construye un vector de referencia uniendo todas las palabras clave de dos párrafos a comparar, sin repetición. Una vez construido este vector, se procede a elaborar un vector normalizado para cada párrafo. Este vector se crea asignando, por cada palabra del vector de referencia, el peso que esta tiene dentro del párrafo. Si la palabra no existe en el párrafo, se asigna el valor 0. Más adelante, esto será explicado mediante un ejemplo.

- Cálculo del coseno del ángulo entre vectores para determinar la similitud: Se calcula la similitud de los dos vectores normalizados correspondientes a cada párrafo mediante el cálculo del coseno del ángulo entre estos vectores. La fórmula para calcular el coseno del ángulo entre vectores se muestra a continuación:

$$S(V_A,\ V_B) = \frac{\sum_{i=1}^{|R|} x_{A,i} \cdot x_{B,i}}{\sqrt{\sum_{i=1}^{|R|} x_{A,i}^2 \cdot \sum_{i=1}^{|R|} x_{B,i}^2}}$$

Figura 13, Fórmula para el Cálculo del Ángulo entre vectores de dos Vectores

Donde:

S(VA, VB) es un valor comprendido entre 0 y 1, el cual corresponde a la similitud existente entre dos párrafos denotados por A y B, siendo 0 el valor que toma cuando no existen coincidencias entre ambos párrafos y 1, cuando ambos párrafos son iguales.

|R| es el tamaño dado por el vector de referencia,

X_{Ai} y X_{Bi} son los pesos, determinados por la posición i, de cada vector normalizado perteneciente a ambos párrafos.

Funcionamiento de la Adaptación de la Técnica Check

A continuación se explica, mediante un ejemplo, el funcionamiento de la adaptación que se hizo de la técnica Check para ser utilizada en el desarrollo de la aplicación Web planteada en este trabajo, detallando cada uno de los pasos que se llevaron a cabo.

Dado un documento a comparar, el cual denotaremos de aquí en adelante como dA, y un documento contra el que se desea realizar la comparación, el cual denotaremos como dB, se llevan a cabo los siguientes pasos:

Paso 1 - Extracción de palabras clave de los párrafos

Para todos los párrafos de dA y dB, se extraen las palabras clave (aquellas que por lo general no agregan significado al documento, como artículos, preposiciones, conectores, etc.). Sean pA y pB dos párrafos en particular de dA y dB respectivamente, los cuales tienen el siguiente texto:

pA = “La metodología RAD es una metodología de desarrollo rápido de aplicaciones que provee una gran flexibilidad en el desarrollo de software”

pB = “La metodología de desarrollo rápido de aplicaciones RAD es una metodología que brinda una gran adaptabilidad en la construcción de éstas”

Se extraen las palabras clave, denotadas por VA y VB, y se le asignan los pesos correspondientes, denotados como WA y WB, para los párrafos pA y pB respectivamente:

VA = [metodología, rad, desarrollo, rápido, aplicaciones, provee, flexibilidad, software]
WA = [2/10, 1/10, 2/10, 1/10, 1/10, 1/10, 1/10, 1/10]

VB = [metodología, desarrollo, rápido, aplicaciones, rad, brinda, adaptabilidad, construcción]
WB = [2/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9]

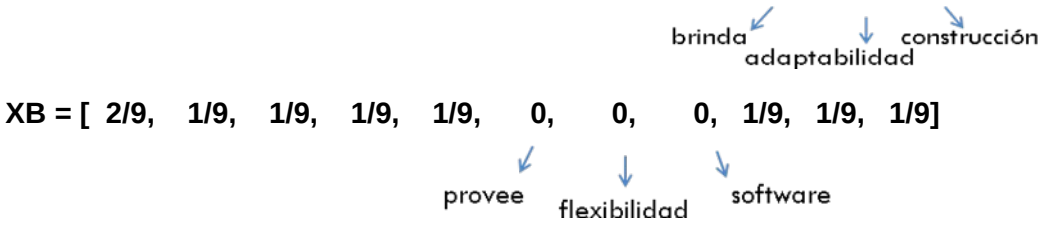
Paso 2 - Comparación de los párrafos

Se genera el vector de referencia, dentado como R, uniendo las palabras clave de ambos párrafos:

R = [metodología, rad, desarrollo, rápido, aplicaciones, provee, flexibilidad, software, brinda, adaptabilidad, construcción]

Se generan los vectores normalizados, denotados por XA y XB. Para ello, se buscan las palabras del vector R dentro del vector de palabras clave de ambos párrafos, y se asigna el peso asociado dentro del mismo. Por ejemplo, la palabra metodología aparece en VA con un peso igual a 2/10 y en VB, con un peso de 2/9. Ambos valores se asignan en la misma posición (en este caso, la primera posición del vector) de los vectores normalizados correspondientes. Continuando con las palabras del vector de referencia, rad aparece en ambos párrafos con un peso de 1/10 y 1/9. Igualmente, esta entrada se almacena en ambos vectores normalizados, que en este caso, corresponden a la segunda posición. Si la palabra del vector de referencia no aparece en alguno de los vectores, en su vector normalizado se coloca el valor 0. Estos vectores se muestran a continuación con todos los valores establecidos.

XA = [2/10, 1/10, 2/10, 1/10, 1/10, 1/10, 1/10, 1/10, 0, 0, 0]



Una vez construidos los vectores normalizados, se lleva a cabo el cálculo del coseno del ángulo entre vectores utilizando la fórmula expuesta en la Figura 13.

En la Figura 14 se muestra el cálculo del coseno del ángulo entre vectores para los vectores XA y XB.

$$\frac{(2/10 \times 2/9) + (1/10 \times 1/9) + (2/10 \times 1/9) + \ldots}{\sqrt{((2/10)^2 + (1/10)^2 + (2/10)^2 + \ldots + 0^2) \times ((2/9)^2 + (1/9)^2 + (1/9)^2 + \ldots + (1/9)^2)}} = 0{,}72524\ldots$$

Figura 14, Cálculo del Coseno del Ángulo entre vectores para dos Vectores dados

Al comparar estos párrafos, la coincidencia obtenida fue de 0,72524 ó 72,524% de similitud.

15. Fases del Proyecto

A continuación se detallan las fases llevadas a cabo para el desarrollo de una aplicación Web que permita comparar contenidos y detectar similitudes entre documentos digitales.

3.3.1. Fase de Planificación

Se desea realizar una aplicación Web que permita comparar los contenidos de los documentos digitales para detectar sus similitudes. Para ello, se tiene en cuenta que se encuentran almacenados en un repositorio de datos.

Esta aplicación será utilizada por los docentes de la Facultad de Ciencias, quienes podrán seleccionar tanto un documento origen como los documentos con los que desea realizar la comparación.

Una vez identificado el objetivo de la aplicación, se procede con la fase de construcción. Para la implementación de esta fase, se adapta la metodología de la siguiente manera:

3.3.2. Fase de Construcción:

En esta fase se lleva a cabo la implementación de todas las iteraciones necesarias para la elaboración de la aplicación.

ITERACIÓN 1

En esta iteración se crea una primera versión del modelo de datos y se desarrolla el motor de comparación del sistema. Para ello se plantean dos requerimientos:

- Crear el modelo de datos.
- Desarrollar el motor de comparación de los documentos.

Diseño:

Para crear el modelo de datos y desarrollar el motor de comparación, se debe tener conocimiento acerca de la estructura del documento. Para ello, se toma en cuenta que un documento se encuentra compuesto por varias páginas. Cada una de éstas posee varios párrafos, y éstos, a su vez, tienen una cantidad de líneas y palabras asociadas. Además, se debe tener una estructura para almacenar las palabras clave junto a sus pesos correspondientes. Finalmente, se considera almacenar el resultado de las comparaciones para poder desplegarlos una vez culminada la comparación.

Teniendo una clara idea de los datos a almacenar, se procede con el diseño.

Se definen las tablas de base de datos que almacenan la siguiente información: párrafos pertenecientes a un documento junto a la cantidad de líneas y palabras asociadas, y palabras clave de cada párrafo junto a sus pesos.

En la Figura 15 se muestra el modelo de datos diseñado para este requerimiento.

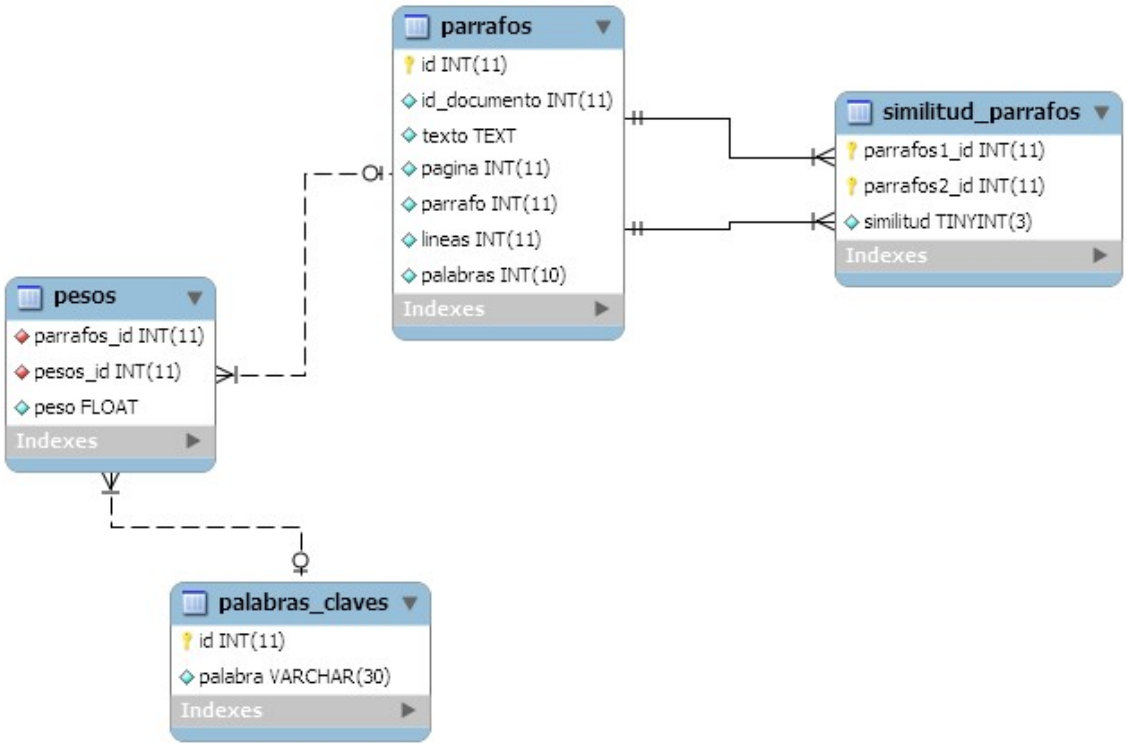


Figura 15, Modelo de Datos de la Aplicación

A continuación se describen las tablas involucradas en el modelo de datos:

parrafos: Almacena todos los párrafos asociados a un documento en particular, en este se almacena el id del párrafo, un id de documento que en este caso será un valor aleatorio, el texto del párrafo, la página del documento al cual pertenece, la cantidad de líneas que este tiene y el numero de palabras .

palabras_claves: Almacena todas las palabras clave pertenecientes a un párrafo en particular.

pesos: Almacena el peso con la que aparecen las palabras clave en un párrafo en particular.

similitud_parrafos: Almacena el nivel de similitud entre dos párrafos. Esta tabla almacena los id de los párrafos comparados y el nivel de similitud que estos tienen.

Una vez definido el modelo de datos, se procede a diseñar el motor de comparación, el cual debe poder cotejar dos documentos extrayendo y almacenando en la base de datos la información necesaria de cada uno de estos para llevar a cabo el proceso de comparación y visualización de resultados.

En la Figura 16 se muestra el diagrama de Modelo Objeto del Análisis del motor de comparación.

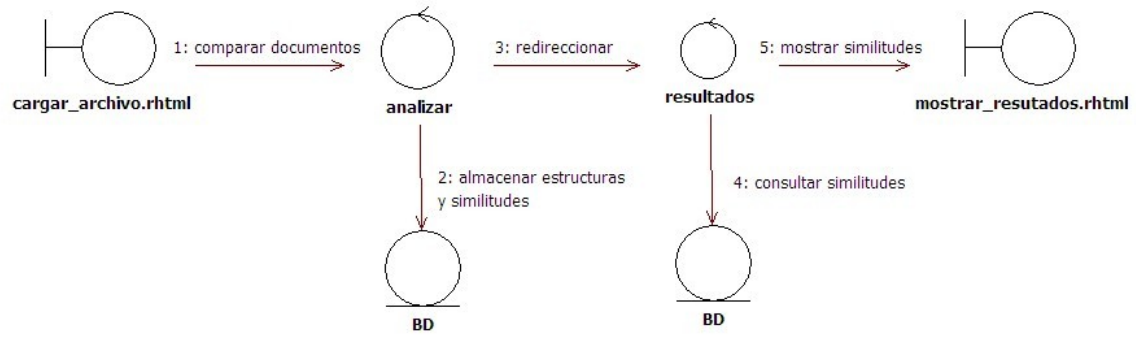


Figura 16, Modelo Objeto del Análisis

En la Figura 17 se muestra el diagrama se secuencia para el motor de comparación.

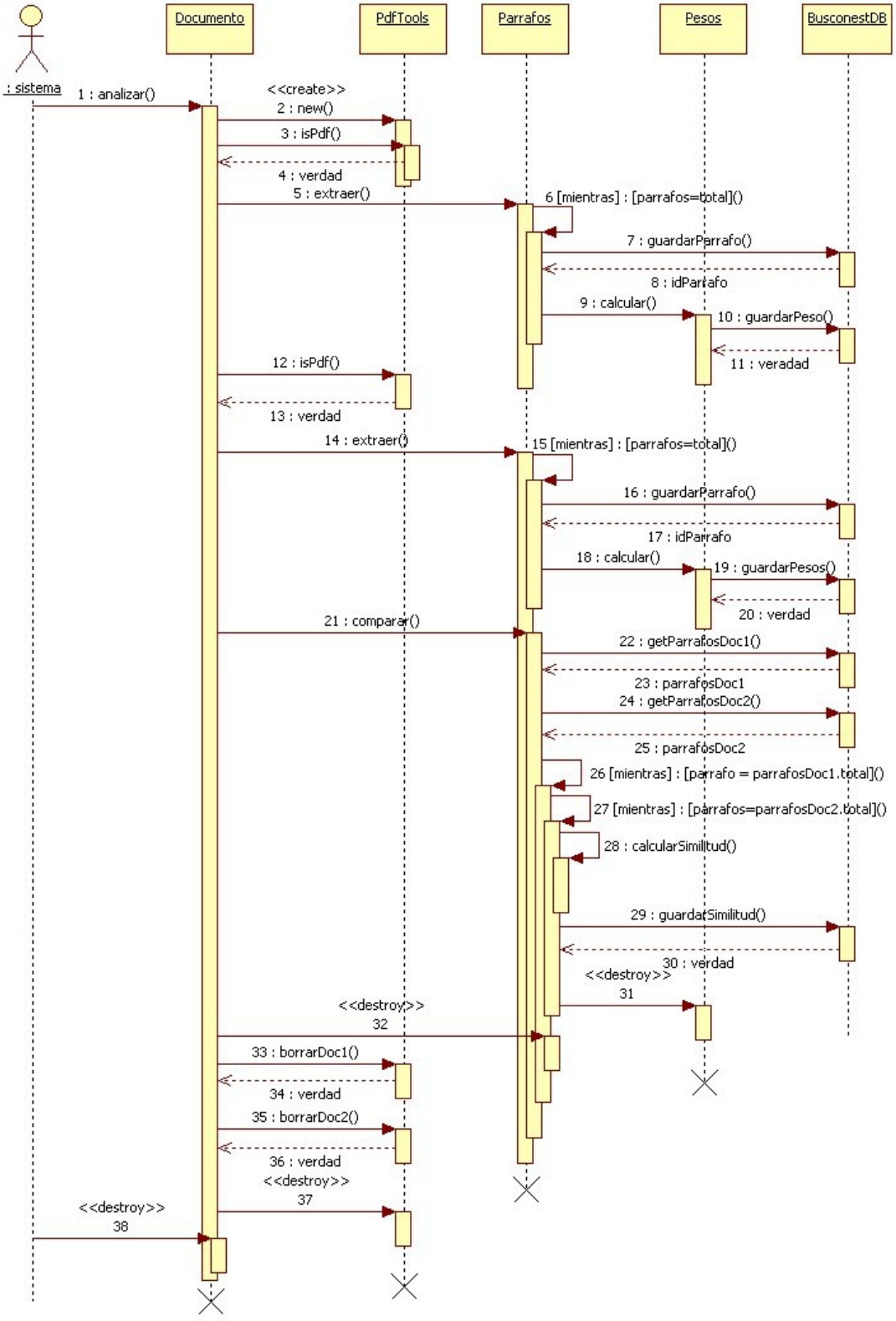


Figura 17, Diagrama de Secuencia del Comparador

Desarrollo

A continuación se describe el desarrollo llevado a cabo para cada uno de los requerimientos

Requerimiento 1 - Crear el modelo de datos

En las Figura 18,Figura 19,Figura 20,Figura 21 y Figura 22 se muestran los scripts utilizados para la creación de las tablas que conforman la base de datos.

```
DROP TABLE IF EXISTS `palabra`;
SET character_set_client = utf8;
CREATE TABLE `palabra` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `texto` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `indice_unico_texto` (`texto`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figura 18, Script utilizado para crear la tabla 'palabra'

```
DROP TABLE IF EXISTS `palabras_claves`;
SET character_set_client = utf8;
CREATE TABLE `palabras_claves` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `palabra` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `indice_palabra_clave` (`palabra`)
) ENGINE=InnoDB AUTO_INCREMENT=4768 DEFAULT CHARSET=utf8;
```

Figura 19, Script utilizado para crear la tabla 'palabras_claves'

```
DROP TABLE IF EXISTS `parrafos`;
SET character_set_client = utf8;
CREATE TABLE `parrafos` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `documento_id` int(11) unsigned NOT NULL,
  `texto` text NOT NULL,
  `pagina` int(11) unsigned NOT NULL,
  `parrafo` int(11) unsigned NOT NULL,
  `lineas` int(11) unsigned NOT NULL,
  `palabras` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_parrafo_documento` (`documento_id`),
) ENGINE=InnoDB AUTO_INCREMENT=3779 DEFAULT CHARSET=utf8;
```

Figura 20, Script utilizado para crear la tabla 'parrafos'

```
DROP TABLE IF EXISTS `pesos`;
SET character_set_client = utf8;
CREATE TABLE `pesos` (
  `parrafos_id` int(11) DEFAULT NULL,
  `palabras_id` int(11) DEFAULT NULL,
  `peso` float DEFAULT NULL,
  KEY `parrafos_id` (`parrafos_id`),
  KEY `palabras_id` (`palabras_id`),
  CONSTRAINT `pesos_ibfk_1` FOREIGN KEY (`parrafos_id`)
  REFERENCES `parrafos` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `pesos_ibfk_2` FOREIGN KEY (`palabras_id`)
  REFERENCES `palabras_claves` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figura 21, Script utilizado para crear la tabla 'pesos'

```
DROP TABLE IF EXISTS `similitud_parrafos`;
SET character_set_client = utf8;
CREATE TABLE `similitud_parrafos` (
  `parrafo1_id` int(11) NOT NULL,
  `parrafo2_id` int(11) NOT NULL,
  `similitud` float DEFAULT NULL,
  PRIMARY KEY (`parrafo1_id`,`parrafo2_id`),
  KEY `fk_similitud_parrafos_parrafo2` (`parrafo2_id`),
  CONSTRAINT `fk_similitud_parrafos_parrafo1` FOREIGN KEY (`parrafo1_id`)
  REFERENCES `parrafos` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_similitud_parrafos_parrafo2` FOREIGN KEY (`parrafo2_id`)
  REFERENCES `parrafos` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figura 22, Script utilizado para crear la tabla 'similitud_parrafos'

Requerimiento 2 – Desarrollar el motor de comparación de los documentos

Para llevar a cabo la comparación de documentos, se deben extraer y almacenar las estructuras de los mismos.

Se crea una clase ***PdfTools*** usada como librería que implementa el uso de las aplicaciones unix ***pdftotext*** , ***file*** e ***iconv***. La implementación de esta librería se muestra en la Figura 23.

```ruby
class PdfTools
  def pages(file)
    text = String.new
    cmd = "pdfinfo #{RAILS_ROOT}/#{file}"
    IO.popen(cmd) { |io|
      text=io.read
      text=text.split
    }
    flag = false
    text.each{|i|
      if flag
        return i.to_i
      end
      if i=="Pages:"
        flag=true
      end
    }
    return 0
  end
  def pdftotext(file,page)
    text=String.new
    cmd="pdftotext -layout -nopgbrk -f #{page} -l #{page} #{RAILS_ROOT}/#{file} -"
    IO.popen(cmd ){|io|
      text=io.read
    }
    return text
  end
  def ispdf(file)
    text=String.new
    IO.popen("file #{RAILS_ROOT}/#{file}") { |io|
      text=io.read
      text= text.split
    }
    if text[1]=="PDF"
      return true
    else
      return false
    end
  end
  def to_utf8(string)
    return Iconv.iconv("UTF-8","Latin1", string).join
  end
end
```

Figura 23, Implementación de la librería PdfTools

Pdftotext es una aplicación unix de consola que transforma documentos en formato PDF a documentos en formato de texto planos conocidos como TXT.

File es otra aplicación de consola que permite capturar información e indicar el tipo de documento de un archivo, por ejemplo, si es PDF, ascii, documento Microsof, etc.

Iconv es una api de Rails que también existe como una aplicación de consola en los sistemas unix, el cual permite transformar codificaciones de caracteres. Por ejemplo, llevar un documento en formato texto plano que se encuentra en la codificación Latin1 a la codificación UTF-8.

Para llevar a cabo las validaciones y pasos a seguir para la extracción de las estructuras de los documentos se implementa el método **analizar.** Este método recibe como parámetros las instancias de dos documentos, los cuales son validados mediante la librería ***PdfTools*** para comprobar si estos son de formato PDF. Después de ser validados, se procede a invocar, para cada documento, al método **procesar**, el cual se encarga de extraer todas las estructuras del documento. Estas son: párrafos, junto al número de líneas y palabras que lo conforman, número de página al cual pertenece y número de párrafo de la página; palabras clave y el peso de estas. En este método, se implementa la aplicación ***Pdftotext*** y la api ***Iconv. Pdftotext*** permite extraer cada una de las páginas de un documento con formato texto plano, con codificación de caracteres por defecto en Latin1. Uno de los estándares definidos en CONEST, es que este sistema trabaja con la codificación de caracteres UTF-8. Esta condición condujo a la implementación de la api ***Iconv*** para realizar la transformación de la codificación Latin1 a la UTF-8.

Finalizada la extracción de las estructuras, se invoca al método **comparar**, encargado de calcular y almacenar en la base de datos las similitudes de los documentos.

En la Figura 24 se muestra el método **analizar**.

```
def analizar(doc1,doc2)
    #creando archivos temporales
    tools= PdfTools.new
    open(pdf1path,"w"){|f| f<<doc1.read}
    if tools.ispdf(pdf1path)
        #Procesar: Extrae los parrafos y asigna los pesos a las palabras
        idDocumento = procesar(pdf1path)
                    idDocumentos.delete(idDocumento)
        idDocumentos.each { |idDocumento2|
            open(pdf2path,"w"){|f| f<<doc2.read}
            if tools.ispdf(pdf2path)
                idDocumento2 =procesar(pdf2path)
                #Compara los parrafos y almacena los resultados en base de datos.
                comparar(idDocumento, idDocumento2)
            else
                p "NO ES PDF:: #{pdf2path}"
                File.delete(pdf2path)
            end
        }
    else
        p "NO ES PDF:: #{pdf1path}"
        File.delete(pdf1path)
    end
end
```

Figura 24, Función analizar

Por cada párrafo del documento, se extraen las palabras clave. Estas se almacenan en la base de datos junto a su peso correspondiente, el cual es calculado dividiendo la frecuencia con la que aparece la palabra dentro del párrafo entre el número total de palabras del mismo. Estos procedimientos se muestran en la Figura 25.

```
parrafo.each_line { |line|
  #por cada linea del parrafo
  #llevar a minusculas
  line = String.normalizar(line)
  #extraer las palabras de la linea
  words = line.split(/[^áÁéÉíÍóÓúÚüÜñÑa-zA-Z]/)
  #eliminar las palabras vacias u omitidas
  words.delete_if{ |word| word.size == 0 || String.omitidas.index(word)!=nil}
  words.each { |w|
    #cont almacena la cantidad de palabras que hay en la linea
    cont = cont+1
    #si la palabra ya se habia encontrado en la linea
    if h.has_key?(w)
      #se incrementa el numero de veces que ha aparecido en la linea
      h[w] = h[w] + 1
    else
      #cada palabra nueva se almacena con un 1 indicando que es la primera coincidencia que se registra
      h[w] = 1
    end
  }
}
#por cada palabra registrada en el hash h.
#h almacena las palabras encontradas en el parrafo junto al numero de veces que aparecen
h.each_key { |key|
  #se calcula el peso dividiendo la frecuencia de la palabra entre el total de palabras del parrafo
  peso = h[key].to_f/cont.to_f
  id_pal= PalabrasClaves.existe(key)
  #si la palabra no ha sido almacenada en la base de datos se agrega.
  if id_pal==nil
    PalabrasClaves.agregar(key)
    id_pal = PalabrasClaves.existe(key)

  end
  agregar(id, id_pal, peso)
  #Retorna el numero de palabras claves
  return cont
}
```

Figura 25, Extraer y almacenar las palabras clave de un párrafo

Las palabras omitidas para la extracción de palabras claves se muestran en la Figura 26, las cuales fueron reutilizadas del sistema BUSCONEST. Contar con una lista de palabras ofrece la ventaja de que estas puedan expandirse en un futuro en caso de que sea necesario.

```
def self.omitidas(otras_mas = [])
  pronombres = []
  pronombres += %w[me te se nos os lo los la las le les yo tu vos el ella ellas ello ellos usted ustedes nosotros
      nosotras vosotros vosotras mi ti si] #personales
  pronombres += %w[nos os se del con el etc ] # reciprocos
  pronombres += %w[este estos esta estas ese esos esa esas aquel aquella aquellos aquellas] # demostrativos
  pronombres += %w[mi mio mios mia mias tu tuyo tuyos tuya tuyas su suyo suyos suya suyas nuestro nuestros nuestra
      nuestras vuestro vuestros vuestra vuestras] # posesivos
  pronombres += %w[el que cual cuales quien quienes cuyo cuya cuyos cuyas] # relativos
  pronombres += %w[donde cuando] # interogativos
  pronombres += %w[alguno algunos alguna algunas varios varias alguien nadie otro otros otra otras cualquier
      cualquiera] # indefinidos
  articulos = []
  articulos += %w[el la lo  los  las un unos una unas]
  adverbios = []
  adverbios += %w[ahora ayer anteayer hoy manana antes anoche aun cuando despues entonces jamas luego mientras
      nunca primero siempre tarde todavia ya] # de tiempo
  adverbios += %w[aqui alli alla aca fuera abajo delante adelante alrededor arriba atras cerca debajo donde encima
      enfrente fuera lejos] # de lugar
  adverbios += %w[asi asimismo bien mal casi como despacio rapido lento deprisa] # de modo
  adverbios += %w[no nunca tampoco jamas] # de negacion
  adverbios += %w[si claro exacto efectivamente ciertamente seguramente justo ya tambien] # de afirmacion
  adverbios += %w[casi cuanto demasiado mas menos mucho poco todo solo mitad tan tanto muy] # de cantidad
  adverbios += %w[quizas acaso probable tal vez etc] # duda
  conjunciones = []
  conjunciones += %w[y ]
  preposiciones = []
  preposiciones += %w[a ante bajo con contra de desde en entre hacia hasta para por segun sin so sobre tras
      durante mediante]
  vocales = []
  vocales += %w[a b c d e f g h i j k l m n ñ o p q r s t u v w x y z á é í ó ú ü]
  return pronombres + articulos + adverbios + conjunciones + preposiciones + vocales
end
```

Figura 26, Palabras omitidas para la comparación

Una vez almacenadas las estructuras de los documentos, se lleva a cabo la comparación de los mismos. Para ello, se obtienen las palabras clave de cada párrafo perteneciente al documento, y se generan el vector de referencia y los dos vectores normalizados (explicados anteriormente en la técnica Check) y se procede a realizar la comparación. El resultado será almacenado en la tabla similitud_parrafos de la base de datos.

En la Figura 27 se muestra una parte del código que permite crear los vectores de referencia y normalizados.

```
#Funcion vectorial para el calculo
def self.similitud (v1, v2)
  referencia = Array.new
  p1 = Hash.new
  p2 = Hash.new
  #v1 y v2 contienen las palabras y pesos de cada parrafo pertenciente a los documentos a comparar
  v1.each { |peso|
      #en p1 se almacenan las palabras con el peso
      p1[peso.id_palabra] = peso.peso
      #Se almacenan las palabras del parrafo en el vector referencia.
      referencia << peso.id_palabra
  }

  v2.each { |peso|
      #en p2 se almacenan las palabras con el peso
      p2[peso.id_palabra] = peso.peso
      #Se almacenan las palabras del parrafo en el vector referencia.
      referencia << peso.id_palabra
  }
  #Se eliminan las palabras repetidas del vector referencia
  referencia.uniq
  #para almacenar los vectores normalizados
  norm_p1 = Array.new
  norm_p2 = Array.new

  #Se normalizan los vectores
  referencia.each { |i|
  #si las palabras almacenadas en el vector de refernecia estan en p1
    if p1.has_key?(i)
      #se almacenan los pesos en el vector  norm_p1
      norm_p1 << (p1[i].to_f)
    else
      norm_p1 << 0.to_f
    end
  }

  referencia.each { |i|
  #si las palabras almacenadas en el vector de refernecia estan en p2
    if p2.has_key?(i)
    #se almacenan los pesos en el vector norm_p2
      norm_p2 << (p2[i].to_f)
    else
      norm_p2 << 0.to_f
    end
  }
```

Figura 27, Creación de los vectores de referencia y normalizados

La función similitud recibe dos duplas de la tabla pesos, correspondiente a los dos párrafos a comparar. Estas duplas contienen el id de las palabras clave de cada párrafo del documento junto a sus pesos asociados. Se crea un vector de referencia uniendo todas las palabras clave de cada párrafo, sin repetición. Luego, se crean los vectores normalizados que contendrán los pesos asociados a cada palabra clave del párrafo. En caso de que la palabra no exista, se le asigna 0 (cero). Las posiciones de los vectores normalizados corresponden con las posiciones de los vectores que almacenan las palabras clave.

Una vez definidos los vectores para la comparación, se procede a realizar el cálculo del coseno del ángulo entre vectores, explicado en la técnica Check.

En la Figura 28 se muestra el código que calcula este coseno, determinando la similitud de los párrafos.

```
while i < norm_p1.size
  #nominador producto cruz de vectores
  acum = (norm_p1[i]*norm_p2[i])+acum
  #denominador producto cruz de vectores
  acum_h1 = (norm_p1[i]*norm_p1[i]) + acum_h1
  acum_h2 = (norm_p2[i]*norm_p2[i]) + acum_h2
  i = i+1
end

raiz=Math.sqrt(acum_h1*acum_h2)

if(raiz>0)
  return acum/raiz
else
  return 0
end
```

Figura 28, Porción de código que calcula la similitud de los párrafos

Pruebas

Se crearon dos interfaces sencillas. Una desde donde el usuario pudiera cargar dos documentos almacenados en el disco local con formato PDF, y otra que permitiera visualizar el resultado de la comparación. El nivel de similitud se expresa mediante una escala comprendida entre los valores que van del 0 al 1 (0, 0.1, 0.134, 0.2, …1), siendo 0

el valor correspondiente a párrafos totalmente distintos y 1 cuando son iguales. En este resultado se visualizan las comparaciones realizadas por cada par de párrafos de ambos documentos. Por ejemplo, si el párrafo i-ésimo del primer documento es igual al párrafo j-ésimo del segundo, se muestran ambos documentos seguidos de un número 1.

Una vez obtenido el resultado, se procedió a verificar con los documentos originales que los párrafos similares eran realmente los que arrojaba el sistema, y que los datos almacenados correspondían con las estructuras de los documentos.

En la Figura 29 se puede presenciar la secuencia e interfaces que se usaron en esta primera iteración para comprobar el funcionamiento del motor de comparación.

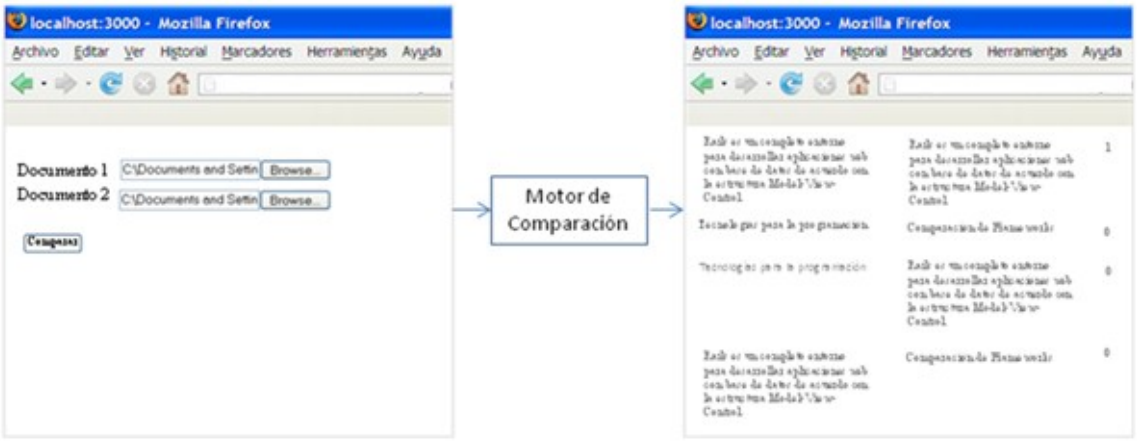


Figura 29, Interfaces utilizadas para probar el motor de comparación

Cambios a implementar

La aplicación debe integrarse al sistema BUSCONEST de manera que el usuario pueda buscar un documento en particular y compararlo contra un conjunto de documentos (almacenados en el repositorio de este sistema) que cumplan ciertas características en común. Estas características son definidas mediante filtros sencillos o avanzados. Estos son: Licenciatura, mención, período(s) académico(s) y similitud en título, palabras clave o resumen.

Debido a que la comparación se tarda un tiempo considerable, se debe implementar un sistema de correo en donde se le notifique al usuario cuando la solicitud haya sido procesada, de manera que pueda ingresar al sistema y visualizar el resultado

de la comparación. Este resultado debe ser mostrado sin romper la estructura de los documentos y deberá ser intuitivo para que el usuario pueda identificar aquellos párrafos que sean similares.

ITERACIÓN 2

En esta iteración se integra la aplicación a BUSCONEST y se implementan tanto los filtros sencillos y avanzados de selección de documentos, como el sistema notificación mediante el envío de correo electrónico. Además, se realiza una adaptación al modelo de datos. Los requerimientos a cumplir se resumen en:

- Integrar la aplicación con BUSCONEST.
- Implementar filtros sencillos y avanzados.
- Implementar un mecanismo de notificación por correo electrónico.

Diseño

Los usuarios deben poder realizar comparaciones de documentos, seleccionando a través de una consulta llevada a cabo en BUSCONEST, el documento origen a comparar. Una vez seleccionado, el usuario debe visualizar una lista de documentos que cumplen un conjunto de características establecidas mediante filtros sencillos. De esta lista, el usuario selecciona aquellos con los que quiere realizar la comparación. Si el usuario no está satisfecho con esta lista preliminar de documentos candidatos para la comparación, debe tener la opción de establecer otras características mediante un formulario de filtros avanzados.

Los filtros sencillos contemplan similitud en título, resumen y palabras clave de documentos que pertenecen a la misma licenciatura y mención que el seleccionado como origen.

En los filtros avanzados, se contemplan documentos que cumplan con ciertas características definidas por el usuario como licenciatura, mención, período(s) académico(s) y similitud en título, resumen y/o palabras clave. De estos, debe seleccionar, por lo menos, la licenciatura y el(los) período(s) académico(s).

Dado que los documentos a comparar se encuentran en un repositorio de datos, la tabla en donde son almacenados debe integrarse con la aplicación. Por otro lado, es importante almacenar información acerca de los documentos que han sido analizados, de manera de evitar que se generen de nuevo las estructuras de dichos documentos, así como evitar realizar comparaciones que ya fueron realizadas.

Además, se deben agregar nuevos campos a la tabla de documentos para implementar los filtros. Es necesario obtener aquellos documentos que cumplan con las condiciones establecidas, como la licenciatura, mención y período académico de presentación. Si bien estos datos se encuentran en una estructura XML almacenada en la base de datos, la consulta a la misma es muy lenta debido a que se deben acceder a cada una de las duplas, para posteriormente obtener del XML los datos necesarios. Debido a esta desventaja, se agregaron nuevos campos a la tabla documento de manera de filtrar estas búsquedas por medio de consultas directas.

Las tablas a integrar son documento, a la que se le agregarán los campos mencionados anteriormente, y parámetro_general**.** Ambas tablas son utilizadas por BUSCONEST. Además, se deben crear las tablas documento_estatus y documentos_analizados para almacenar la información de los documentos que han sido comparados. El modelo adaptado se muestra en la Figura 30, donde las tablas coloreadas en rojo representan a las existentes en BUSCONEST.

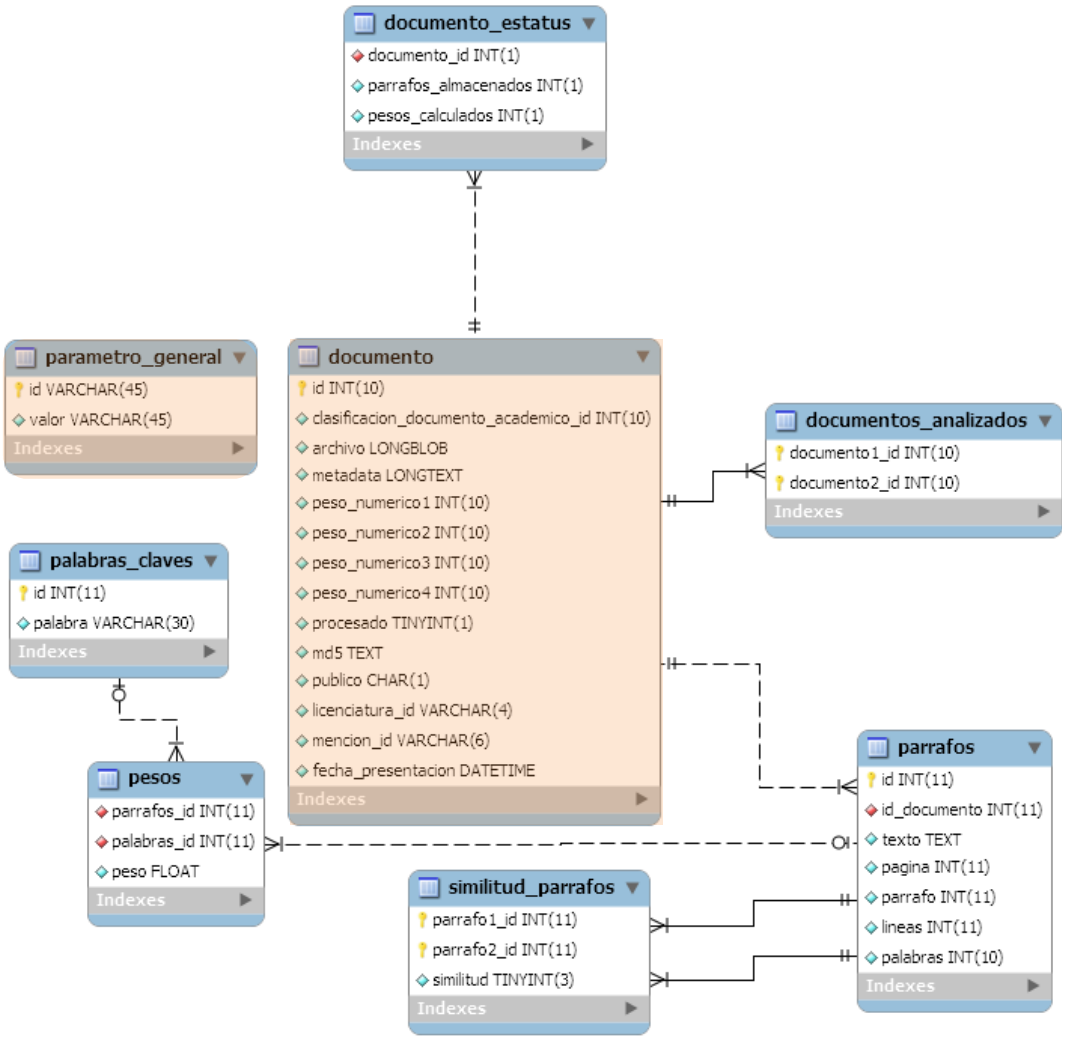


Figura 30, Modelo de datos

A continuación se describen las tablas incorporadas:

documento: Esta tabla almacena el archivo digital, un XML con los datos asociados del documento (como autor, tutor, licenciatura, mención, fecha de publicación, título, resumen, palabras clave, formato del documento, etc.), e información utilizada para el motor de búsqueda de BUSCONEST. De estos, son necesarios el archivo digital y el XML.

parámetro_general: esta tabla almacena información interna del sistema, la cual puede ser modificada por el administrador del mismo. Uno de los datos almacenados en

esta tabla, necesarios para la aplicación planteada, es la dirección del sistema CONEST ADMIN, la cual será utilizada para la comunicación a través de los web services de este sistema, que permita obtener información necesaria para la implementación de los filtros.

documento_estatus: esta tabla almacena información para indicar si se extrajeron todas las estructuras del documento y si se calcularon los pesos de cada una de las palabras clave de cada párrafo. Esta información se utiliza para evitar que los párrafos sean extraídos de nuevo.

documentos_analizados: esta tabla almacena información acerca de los documentos que ya fueron comparados. Esto evita realizar comparaciones sobre documentos que ya han sido cotejados.

Cuando el usuario envía una petición de comparación de documentos, el sistema debe registrar la solicitud y enviar un correo electrónico cuando esta haya sido procesada.

En la Figura 31 se muestra el flujo que sigue el sistema una vez realizada la solicitud de comparación de documento.

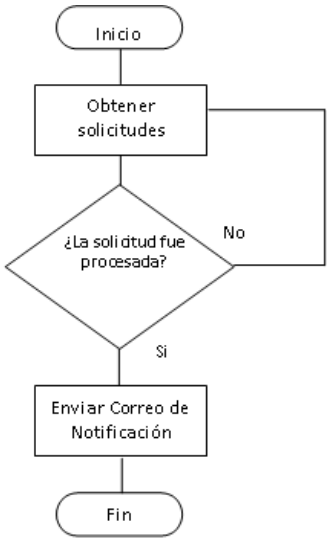


Figura 31, Flujo para el envío del correo de notificación

En la Figura 32 se muestra el diagrama Modelo Objeto del Análisis para la selección de documento origen y documentos a comparar junto a la implementación de los filtros.

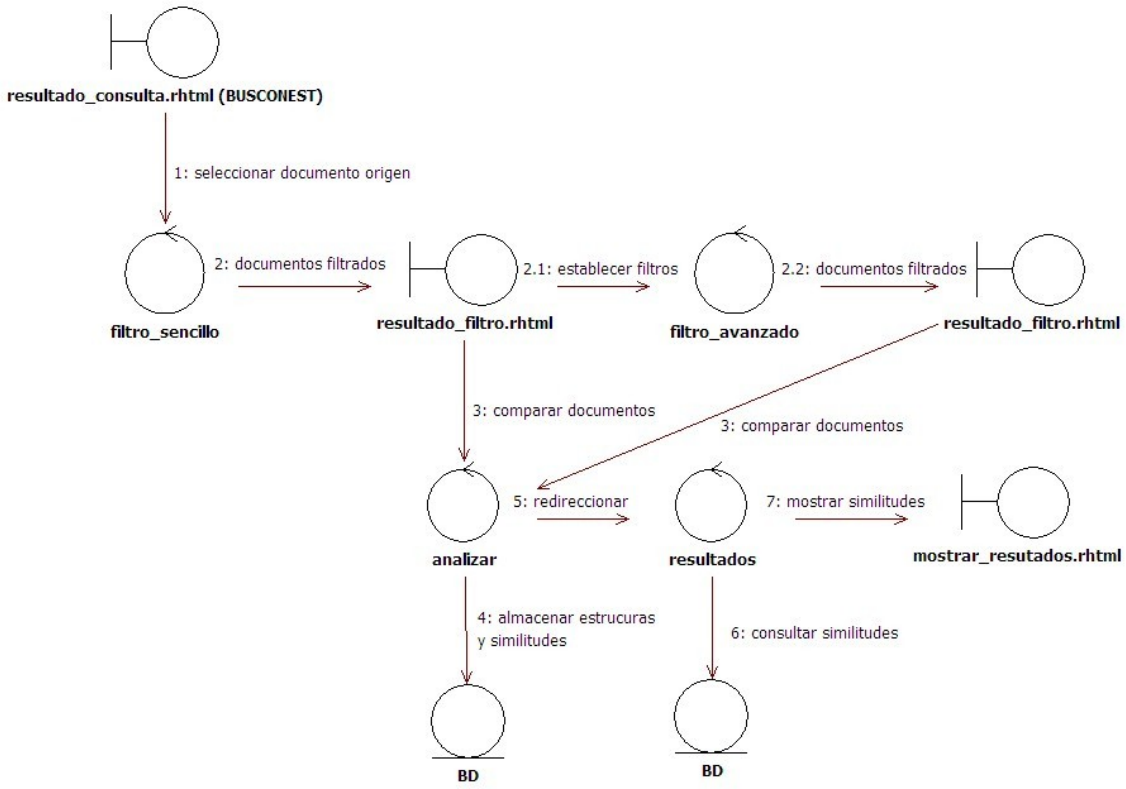


Figura 32, Modelo Objeto del Análisis

Desarrollo

A continuación se describe el desarrollo llevado a cabo para cada uno de los requerimientos.

Requerimiento 1 - Integrar la aplicación con BUSCONEST

Para que el usuario pueda comparar un documento consultado mediante el buscador BUSCONEST, se debe crear un enlace desde el resultado de consultas de documentos. Se considera que éste sólo estará disponible para aquellos documentos que tengan formato PDF. Además, este enlace debe enviar el id de documento seleccionado. En la Figura 33 se muestra el código incluido en la vista de resultados de consultas de documentos de BUSCONEST.

```
<%if documento.extension_formato_archivo == 'pdf' %>
    <%= link_to('[Usar como documento origen para la comparación]',
                "http://localhost:3004/similitud/filtrar_busqueda?id=#{documento.id}" ,
                {:style => "{color: blue;font-size:11px}" }) %>
<%end%>
```

Figura 33, Enlace que permite seleccionar un documento origen

Dado que se utiliza el repositorio de documentos digitales de BUSCONEST, las bases de datos (tanto la de BUSCONEST como la de la aplicación planteada) se integran en una sola.

Requerimiento 2 - Implementar filtros sencillos y avanzados

Para implementar los filtros sencillos, se deben obtener los documentos que tengan la misma licenciatura y mención que la del documento origen.

Una vez obtenido los documentos, se deben comparar tanto los títulos, como los resúmenes y palabras clave de cada documento contra el de origen. Aquellos que tengan una similitud mayor que cierto umbral predefinido, serán entonces los elegidos para desplegar al usuario.

En la Figura 34 se muestra una porción de código que permite realizar estas búsquedas mediante filtros sencillos.

```
#se buscan los documentos que pertenzecan a la misma licenciatura y mencion que la del documento a comparar
doc = Documento.find_by_sql("select id from documento where id not like #{@doc_origen_id} and
                             id_licenciatura = '#{@licenciatura}' and
                             id_mencion = '#{mencion}'")
doc.each { |id_doc|
 documento = Documento.find(id_doc)
 #si los documentos tienen una similitud en titulo, resumen y palabras claves mayor o igual a la predefinida en el umbral de similitud se toman en cuenta
 if
 (ApplicationController.similitud_cadenas(documento.titulo_html, @titulo)>= UmbralSimilitud::UMBRAL_TITULO)
 and
 (ApplicationController.similitud_cadenas(documento.palabras_clave, palabras)>= UmbralSimilitud::UMBRAL_PALABRAS)
 and
 ( ApplicationController.similitud_cadenas(documento.resumen, resumen)>= UmbralSimilitud::UMBRAL_RESUMEN)

 @documentos << documento
  end
}
```

Figura 34, Búsqueda mediante filtros sencillos

Cuando el usuario selecciona la búsqueda mediante filtros avanzados, éste puede seleccionar los criterios para aplicar los filtros de selección de documentos. Estos son

licenciatura, mención, período(s) académico(s) y/o similitud en título, resumen y/o palabras clave. Para ello, se debe establecer un formulario desde donde el usuario pueda elegir los filtros a implementar.

Para construir el formulario de filtros avanzados, se deben consultar las licenciaturas que conforman la Facultad de Ciencias, así como las menciones que en estas se ofrecen, y los períodos académicos registrados. Esta información se encuentra almacenada en la base de datos de CONEST, por lo que se deben desarrollar Servicios Web en el sistema de manera que se comunique con CONEST ADMIN para obtener estos datos.

En la Figura 35 se muestra la integración del sistema con CONEST ADMIN y BUSCONEST.

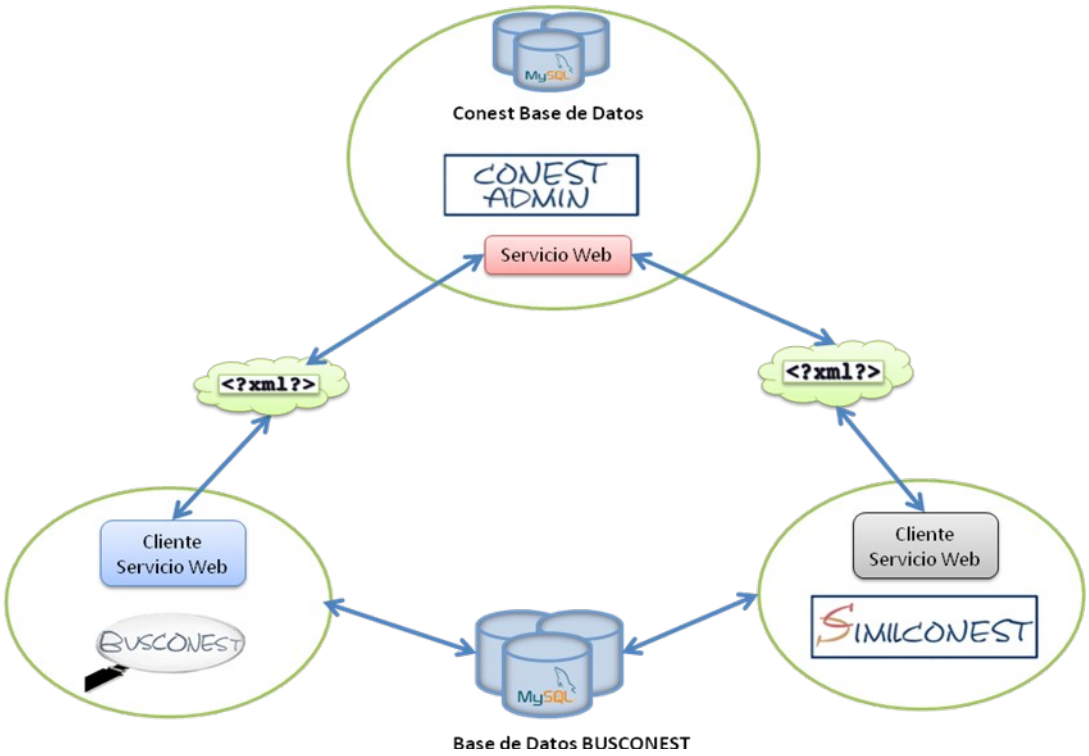


Figura 35, Integración con CONEST ADMIN y BUSCONEST

Las funciones implementadas se muestran en la Figura 36.

```ruby
class PublicacionesController < ApplicationController
  wsdl_service_name 'Publicaciones'
  def get_licenciaturas
    licenciatura = String.new
    licenciatura = ''
    lic = Licenciatura.find(:all, :order => "id ASC")
    lic.each{|l|
      licenciatura += l.nombre_corto.to_s
      licenciatura += '-' + l.id.to_s
      licenciatura += ','

    }
    return licenciatura
  end
  def get_menciones(licenciatura_id)
    menc = Mencion.find(:all, :conditions=> "licenciatura_id = '#{licenciatura_id}'")
    menciones = String.new
    menc.each { |m|
      menciones += m.descripcion.to_s
      menciones += '-' + m.id.to_s
      menciones += ','
    }
    return menciones
  end
  def get_periodos
      periodos = String.new
      periodos = ''
      per = PeriodoAcademico.find(:all,
                                  :conditions => "periodo_academico != 'I'",
                                  :order => "ano_lectivo ASC, periodo_academico")
      per.each{|p|
        periodos += p.periodo_academico.to_s
        periodos +='-' + p.ano_lectivo.to_s
        periodos += ','
      }
    return periodos
  end
  def get_periodo_inicio (periodo)
    per = PeriodoAcademico.find(:first, :conditions => "descripcion = '#{periodo}'")
    periodo_inicio = String.new
    periodo_inicio =  per.fecha_inicio.to_s
    return periodo_inicio
  end
  def get_periodo_fin (periodo)
    per = PeriodoAcademico.find(:first, :conditions => "descripcion = '#{periodo}'")
    periodo_fin = String.new
    periodo_fin = per.fecha_fin.to_s
    return periodo_fin
  end
```

Figura 36, Métodos en los Servicios Web de Conest Admin

A continuación se describen las funciones definidas en el Web Services

get_licenciaturas: esta función retorna todas las licenciaturas que se imparten en la Facultad de Ciencias.

get_menciones: esta función retorna todas las menciones disponibles para una licenciatura en particular.

get_periodos: esta función retorna todos los períodos académicos almacenados en la base de datos, excluyendo los períodos intensivos.

get_periodo_inicio: esta función retorna la fecha en que comienza un período académico dado.

get_periodo_fin: esta función retorna la fecha en que finaliza un período académico dado.

Una vez seleccionados los filtros, se procede a obtener de la base de datos aquellos documentos que cumplan con estos.

En la Figura 37 se muestra una porción de código que permite realizar la búsqueda avanzada.

```
#Se buscan todos los documentos almacenados q no sean el de origen
sql="select id from documento where id not like #{@doc_origen_id} #{sql}"
docs_ids =Documento.find_by_sql(sql)
docs_ids.each { |doc_id|
#Si el documento tiene coincidencias en el titulo
if (params_titulo == "1")
if( ApplicationController.similitud_cadenas(documento.titulo_html,doc_origen.titulo_html)
>= UmbralSimilitud::UMBRAL_TITULO )
doc_coinc_titulo << documento.id
end
end
#Si el documento tiene coincidencias en las palabras claves
if(params_palabras == "1")
if( ApplicationController.similitud_cadenas(documento.palabras_clave, doc_origen.palabras_clave)
>= UmbralSimilitud::UMBRAL_PALABRAS )
doc_coinc_palabras << documento.id
end
end
#Si el documento tiene coincidencias en resumen
if(params_resumen == "1")
if( ApplicationController.similitud_cadenas(documento.resumen, doc_origen.resumen)
>= UmbralSimilitud::UMBRAL_RESUMEN )
doc_coinc_resumen << documento.id
end
end
}
#se almacenan los documentos de la misma licenciatura mencion y periodo
documentos = docs_ids
#si se especifican las similitudes en titulo, palabras claves o resumen, se almcacenan los documentos que coincidan con los almacenados previamente
if(params_titulo == "1")
  documentos = documentos & doc_coinc_titulo
end
if(params_palabras == "1")
  documentos = documentos & doc_coinc_palabras
end
if(params_resumen == "1")
  documentos = documentos & doc_coinc_resumen
end
documentos.each{ |doc|
  @documentos << Documento.find(doc)
}
```

Figura 37, Realizar búsqueda mediante filtros avanzados

Los umbrales establecidos para la comparación de títulos, resúmenes y palabras clave son almacenados en la tabla parametro_general de la base de datos, de manera que el administrador pueda modificarlos.

Adicional a la implementación de filtros, se crearon dos tablas, documento_estatus y documentos_analizados, utilizadas para almacenar la información de los documentos que han sido comparados. Con estas nuevas tablas, se puede validar el estado de la comparación de alguna solicitud realizada por el usuario, de manera que si

ocurre alguna interrupción en el proceso de comparación, esta pueda retomarse sin necesidad de crear de nuevo las estructuras de los documentos que ya fueron creadas.

En la Figura 38 se muestra el *script* utilizado para la creación de las tablas documento_estatus y documentos_analizados.

```
CREATE  TABLE IF NOT EXISTS `documento_estatus` (
  `documento_id` INT(1) UNSIGNED NOT NULL ,
  `parrafos_almacenados` INT(1) NOT NULL ,
  `pesos_calculados` INT(1) NOT NULL ,
  INDEX `fk_estatus_documento` (`documento_id` ASC) ,
  CONSTRAINT `documento_estatus_ibfk_1`
    FOREIGN KEY (`documento_id` )
    REFERENCES `mydb`.`documento` (`id` )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

CREATE  TABLE IF NOT EXISTS `documentos_analizados` (
  `documento1_id` INT(10) UNSIGNED NOT NULL ,
  `documento2_id` INT(10) UNSIGNED NOT NULL ,
  PRIMARY KEY (`documento1_id`, `documento2_id`) ,
  INDEX `fk_ana_doc` (`documento1_id` ASC, `documento2_id` ASC) ,
  CONSTRAINT `fk_ana_doc`
    FOREIGN KEY (`documento1_id` , `documento2_id` )
    REFERENCES `mydb`.`documento` ()
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 38, Creación de tablas 'documento_estatus' y 'documentos_analizados'

Requerimiento 3 - Implementar un mecanismo de notificación por correo electrónico

Se desarrolla un sistema que notifique al usuario, por correo electrónico, cuando las solicitudes hayan finalizado, es decir, cuando se han culminado todas las comparaciones de documentos que la solicitud tiene. En el correo, se muestran los resultados de la comparación indicando porcentaje de similitud y cantidad de párrafos similares.

La notificación por medio de correo electrónico se implementa con la api de Rails llamado *ActionMailer*. Esta api provee las funcionalidades que permiten enviar un correo

mediante un servidor smtp (*Send Mail Transfer Protocol*). *ActionMailer* provee los mecanismos para configurar el destinatario del correo electrónico, así como el formato o plantilla del cuerpo del correo.

En la Figura 39 se muestra el código que implementa el envío del correo electrónico.

```
def self.enviar_email(correo, info)
    pg = ParametroGeneral.find("host_conest_docente")
    AnalizadorMailer.deliver_estatus_message("",[correo],pg.valor,info)
end

class AnalizadorMailer < ActionMailer::Base
    def estatus_message(nombre,correo,url,info )
        @subject    = 'Simconest - Solicitud Activada'
        @body       = {:nombre=>nombre,:email=>correo,:url=>url,:info=>info}
        @recipients = correo
        @from       = 'server@simconest.com'
        @sent_on    = Time.now
    end
end
```

Figura 39, Código implementado para el envío de correo electrónico

Pruebas

Se almacenaron varios documentos en la base de datos, y se procedió a realizar diversas consultas mediante la implementación de ambos filtros. Se pudo observar que los documentos desplegados coincidían con los parámetros establecidos.

Además se realizaron varias comparaciones para verificar que los correos estaban siendo enviados una vez que la solicitud era procesada.

Cambios a implementar

El usuario debe poder seleccionar un nivel de comparación de documentos, que corresponderá con la exactitud de similitud que existirá entre los párrafos.

El usuario debe poder consultar, cuando lo desee, el historial de solicitudes que ha realizado así como también los resultados arrojados por estas. Además, debe poder eliminar una comparación si así lo desea.

Finalmente, se encontró un problema en cuanto al rendimiento del servidor, por lo que se planteó el desarrollo de un servidor de trabajos (*Job Server*).

ITERACION 3

En esta iteración se implementan los niveles de comparación y se desarrolla el módulo que permite a los usuarios visualizar su historial de solicitudes, junto a los resultados generados. Además, se implementa un servidor de trabajos (*Job Server*) de manera que múltiples usuarios puedan solicitar n cantidad de comparaciones sin afectar el rendimiento del sistema. Los requerimientos a desarrollar son:

- Implementar niveles de comparación.
- Permitir al usuario visualizar su historial de solicitudes.
- Implementación del servidor de trabajos (Job Server).

Diseño

Cuando el usuario solicita una comparación de documentos, debe poder establecer el nivel de similitud con el que serán procesados. Este nivel hace referencia a que tan parecidos son los párrafos del documento. Para ello, se van a establecer cuatro niveles: similitud baja, media, alta y muy alta. Estos valores se almacenan en la base de datos por lo que se debe crear una tabla nivel_comparacion. Ésta contiene el texto que describe el tipo de nivel, y un valor entre 0 y 1 utilizado como umbral de similitud, siendo 0 el valor que toma cuando no existen coincidencias, y 1 cuando los párrafos son iguales.

El usuario debe poder visualizar su historial de solicitudes, y el estado en que se encuentran. Estos son: procesando documento y finalizado. Por cada solicitud en el historial, se debe mostrar el resultado de todas las comparaciones que fueron llevadas a cabo. Para ello, se deben almacenar en la base de datos todas las solicitudes realizadas

por un usuario en particular, para lo que se debe incorporar al modelo de datos la tabla solicitudes.

En la Figura 40 y la Figura 41 se observan respectivamente tanto el diagrama de Modelo Objeto del Análisis para la consulta del historial de solicitudes, como el modelo de datos con las tablas nivel_comparacion y solicitudes (las tablas en rojo corresponden a las existentes en BUSONEST).

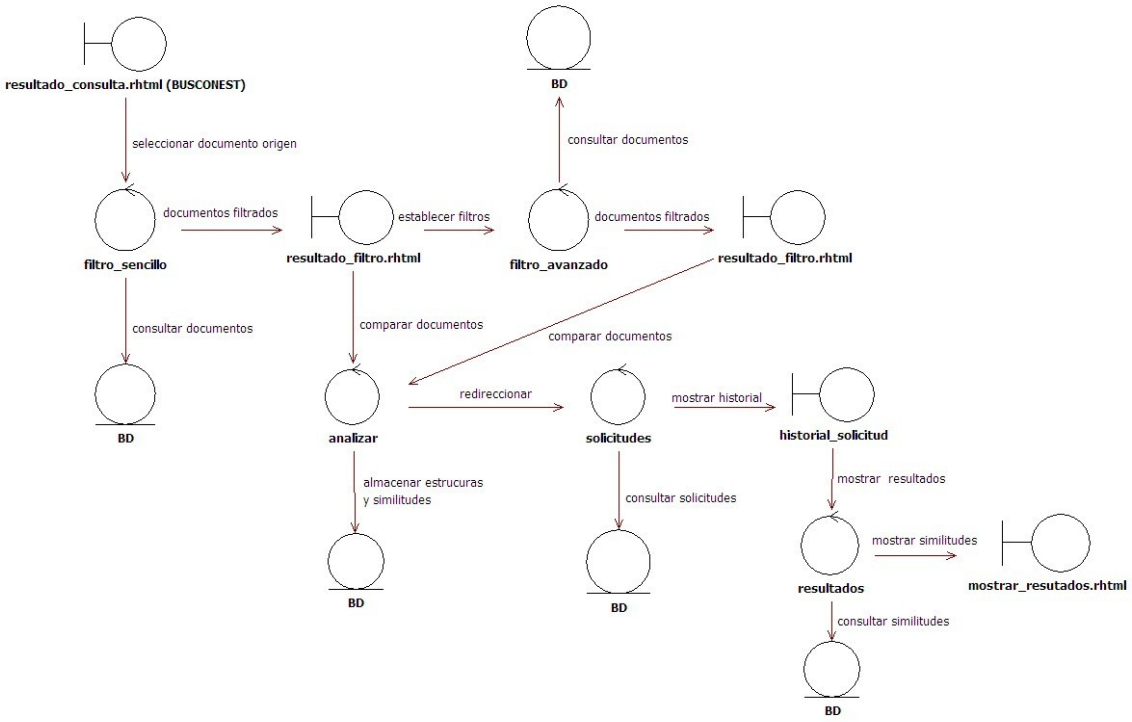


Figura 40, Modelo Objeto del Análisis

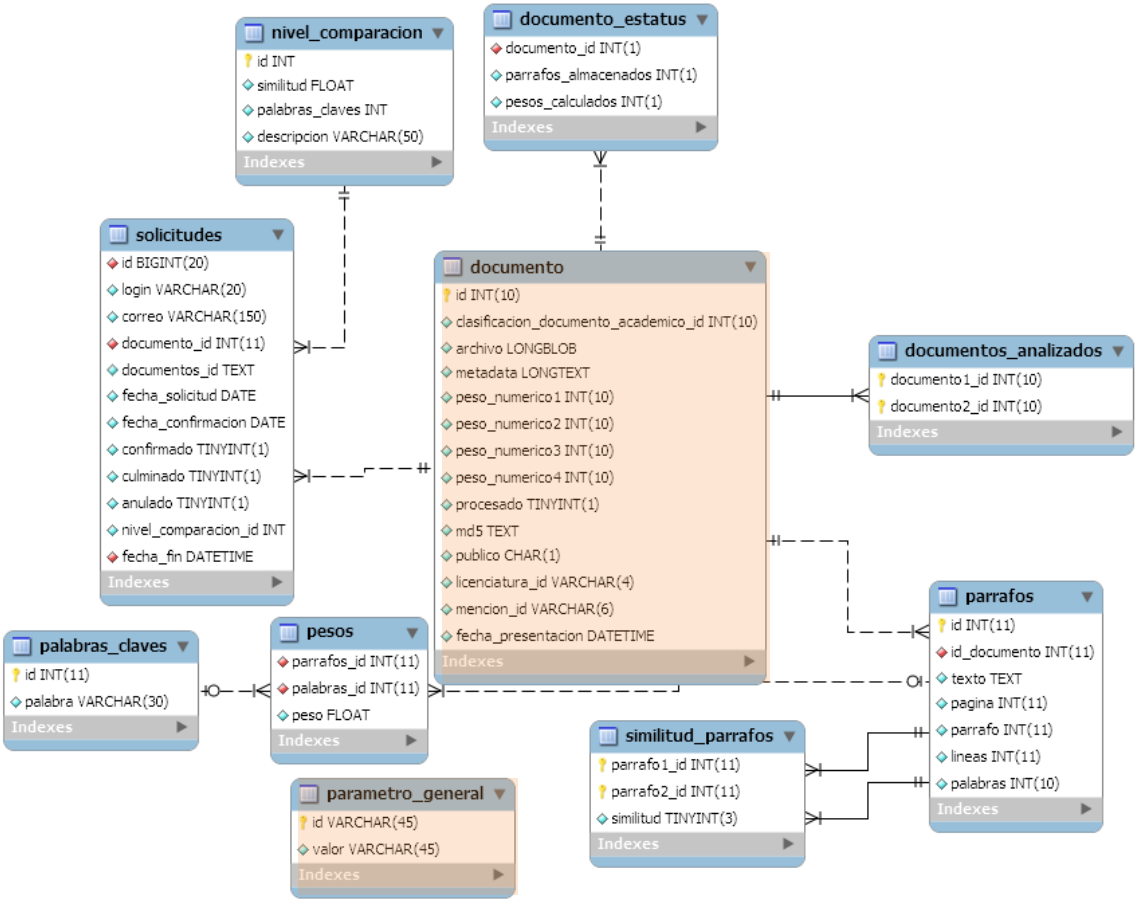


Figura 41, Modelo de Datos

Por último, se plantea desarrollar un servidor de trabajos (*Job Server)* como solución al problema de rendimiento del sistema: cuando n cantidad de usuarios solicitan la comparación de varios documentos digitales, se invocan n veces la función analizar. Como esta función genera un alto procesamiento, se ralentiza el sistema, pudiéndose causar el colapso del servidor.

Desarrollo

A continuación se describe el desarrollo llevado a cabo para cada uno de los requerimientos

Requerimiento 1 – Implementar niveles de comparación

Se crea la tabla nivel_comparacion, la cual almacena el valor de los umbrales de cada uno de los niveles establecidos

En la Figura 42 se puede visualizar el script utilizado para la creación de la tabla.

```
CREATE  TABLE IF NOT EXISTS `nivel_comparacion` (
  `id` INT NOT NULL ,
  `similitud` FLOAT NULL ,
  `palabras_claves` INT NULL ,
  `descripcion` VARCHAR(50) NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB;
```

Figura 42, Script utilizado para la creación de la tabla 'nivel_comparacion'

En cuanto a la integración de este nuevo requerimiento al sistema, se ingresa una opción por medio de la cual el usuario pueda seleccionar el nivel de similitud deseado. En un principio, el sistema comparaba párrafos cuyo umbral de similitud era mayor que uno establecido, de manera fija, dentro del código. Para este requerimiento, lo que cambia fue la lógica de comparación, donde el valor ya no está definido en el código, sino que es obtenido de la base de datos a partir de la selección de nivel que el usuario establece.

Requerimiento 2 - Permitir al usuario visualizar su historial de solicitudes

En la Figura 43 se muestra el script utilizado para la creación de la tabla solicitudes.

```
CREATE  TABLE IF NOT EXISTS `solicitudes` (
  `id` BIGINT(20) UNSIGNED NOT NULL ,
  `login` VARCHAR(20) NOT NULL ,
  `correo` VARCHAR(150) NOT NULL ,
  `documento_id` INT(11) NOT NULL ,
  `documentos_id` TEXT NOT NULL ,
  `fecha_solicitud` DATE NOT NULL ,
  `fecha_confirmacion` DATE NULL DEFAULT NULL ,
  `confirmado` TINYINT(1) UNSIGNED NOT NULL DEFAULT '0' ,
  `culminado` TINYINT(1) UNSIGNED NOT NULL DEFAULT '0' ,
  `anulado` TINYINT(1) UNSIGNED NOT NULL DEFAULT '0' ,
  `nivel_comparacion_id` INT UNSIGNED NOT NULL DEFAULT '0' ,
  `fecha_fin` DATETIME NULL ,
  INDEX `fk_solicitud_nivel` (`fecha_fin` ASC) ,
  INDEX `fk_sol_doc` (`documento_id` ASC, `id` ASC) ,
  CONSTRAINT `fk_solicitud_nivel`
    FOREIGN KEY (`fecha_fin` )
    REFERENCES `mydb`.`nivel_comparacion` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_sol_doc`
    FOREIGN KEY (`documento_id` , `id` )
    REFERENCES `mydb`.`documento` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 43, Script utilizado para crear la tabla 'solicitudes'

En la Figura 44 se muestra una porción de código implementada para buscar las solicitudes de un usuario en particular.

```
def mostrar_solicitudes
  @soliciudes_pages, @solicitudes = paginate  :solicitudess,
                                              :per_page => 15 ,
                                              :conditions=>"login=#{session[:login]}",
                                              :order=>"fecha_solicitud desc , fecha_confirmacion desc "
end
```

Figura 44, Buscar las solicitudes realizadas por un usuario

Las solicitudes que puede realizar un usuario son limitadas a n veces por mes. Este valor es establecido en la tabla parametro_general.

Requerimiento 3 – Implementación del servidor de trabajos (Job Server)

Con la implementación de las solicitudes en el sistema, se pude obtener el orden secuencial en el cual los usuarios solicitan la comparación entre documentos.

Teniendo el orden de las solicitudes, se puede integrar un servicio de trabajos (Job Server) llamado backgroundrb, el cual permite ejecutar tareas que abarcan un largo tiempo de procesamiento en el sistema; sin comprometer el rendimiento del mismo.

Con la implementación del backgroundrb, se desarrolla un trabajo (job) que captura las solicitudes de los usuarios y manda a comparar uno a uno los documentos en el mismo orden en el que fueron solicitados. Este desarrollo permite que el sistema no ejecute n cantidad de comparaciones por n cantidad de usuarios.

En la Figura 45 se muestra el código del trabajo que ejecuta el servidor backgroundrb.

```
class AnalizarWorker < BackgrounDRb::Rails
  def do_work(args)
      while true
          begin
            #Obtiene la lista de solicitudes ordenado desde la fecha mas antigua a la mas actual.
            sol = Solicitudes.find(:first,
                                   :conditions=>"confirmado=1 and culminado=0 and anulado=0",
                                   :order=>"fecha_confirmacion asc")
            if sol
              #Invoca la función analizar pasando por parametros el documento origen , y los documentos a comparar
              ApplicationController.analizar(sol.id_documento,sol.id_documentos.split(','));
              #Finalizado el proceso de comparación actualiza el estatus de la solicitud.
              sol.culminado=1
              sol.fecha_fin=Time.now
              sol.save
              begin
                #Envia un correo electrónico al usuario indicando que ha culminado la comparación de los documentos.
                ApplicationController.enviar_email(sol.correo,sol.id);
              rescue Exception => e
                puts "ERROR (Correo Electronico):"+e.inspect
              end
            end
          rescue Exception => e
            puts "ERROR (Base de Datos):"+e.inspect
          end
          sleep 5.seconds
      end
  end
end
```

Figura 45, Código del trabajo que ejecuta el servidor backgroundrb.

Pruebas del requerimiento

Se comparan documentos con distintos niveles de comparación observándose que los resultados arrojados se adaptaban al establecido. Además, se realizaron varias solicitudes simultáneas con distintos usuarios, y se pudo comprobar que efectivamente el rendimiento del sistema no se veía afectado.

Cambios a implementar

Las comparaciones entre documentos sólo pueden ser solicitadas por los docentes de la Facultad de Ciencias, por lo que la aplicación se debe integrar con el sistema CONEST DOCENTE. Además, se deben implementar mecanismos de autenticación.

El docente debe poder suministrar un documento de su preferencia para compararlo con los almacenados en el repositorio.

ITERACIÓN 4

En esta iteración se llevó a cabo la integración con el sistema CONEST DOCENTE para lo cual se implementaron los mecanismos de autenticación, y se desarrolló una funcionalidad que permite al usuario realizar comparaciones a partir de un documento suministrado.

- Integración con el sistema CONEST DOCENTE
- Permitir al usuario suministrar un documento de su preferencia

Diseño

El docente hace uso de la aplicación a través del módulo de servicio al docente de CONEST, donde puede dirigirse tanto a BUSCONEST para buscar un documento y seleccionarlo como documento origen para la comparación, como ingresar directamente a la aplicación para visualizar su historial de solicitudes o realizar una comparación de documentos a través de uno suministrado por él.

Dado que los documentos a comparar se encuentran almacenados en un repositorio, el documento origen debe ser registrado en el caso de ser mediante carga manual. Se planta un mecanismo de depuración de la base de datos para evitar llenarla de información no deseada, en el cual se eliminan todos los documentos, cargados manualmente, que tengan treinta días registrados a partir de la finalización de su comparación. Después de transcurrido este tiempo, el usuario no podrá visualizar los resultados. Sin embargo, el registro de la tabla solicitudes no es eliminado, de manera de tener algún histórico de que esa comparación fue realizada.

En la Figura 46 se muestra el Modelo Objeto del Análisis con la integración al módulo docente de CONEST.

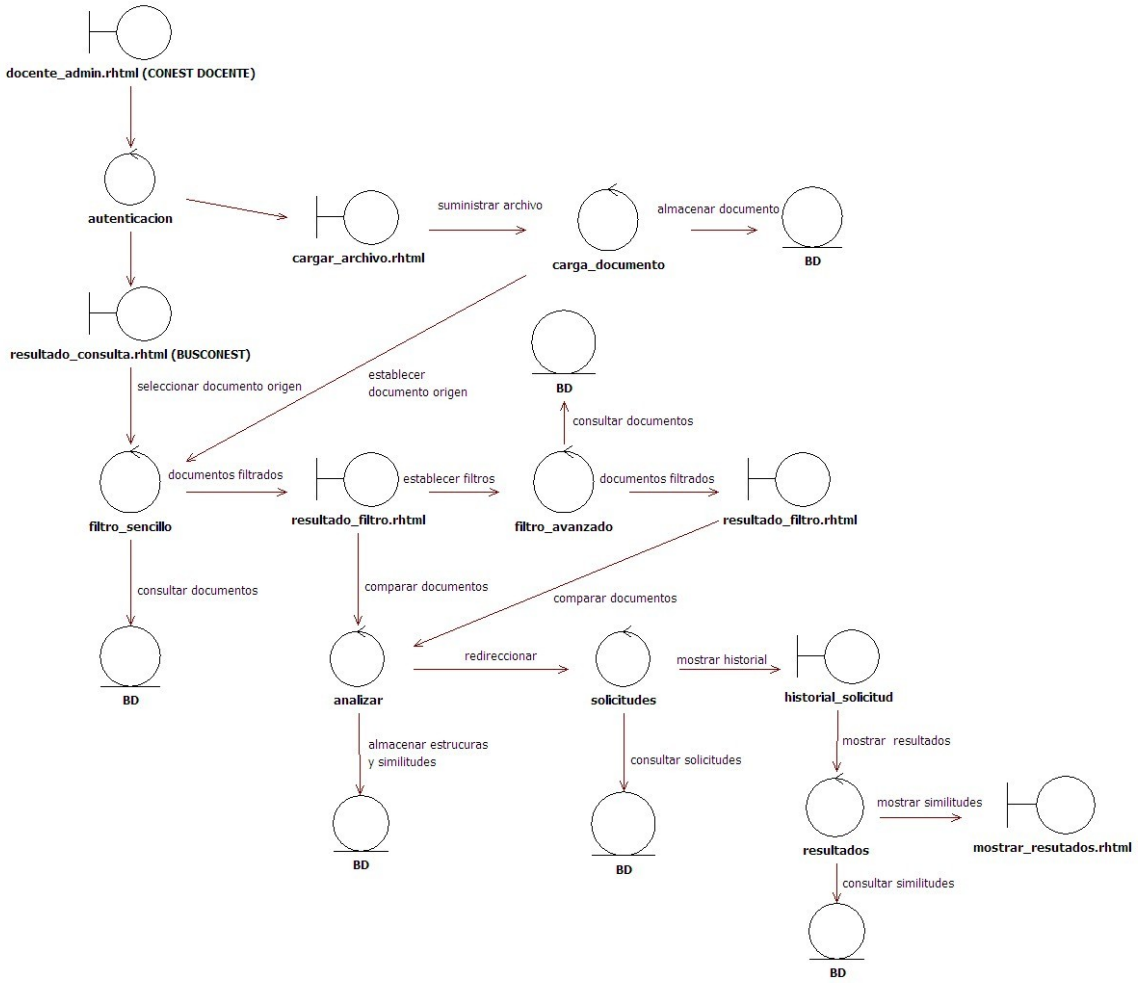


Figura 46, Modelo Objeto del Análisis

Para tener conocimiento acerca de cuáles documentos fueron cargados manualmente, se creó un nuevo tipo de clasificación de documento en la tabla clasificacion_documento_academico, por lo que esta fue integrada con la aplicación.

En la Figura 47 se puede observar el modelo de datos con la integración de esta tabla, donde las coloreadas en rojo representan las tablas utilizadas por BUCONEST.

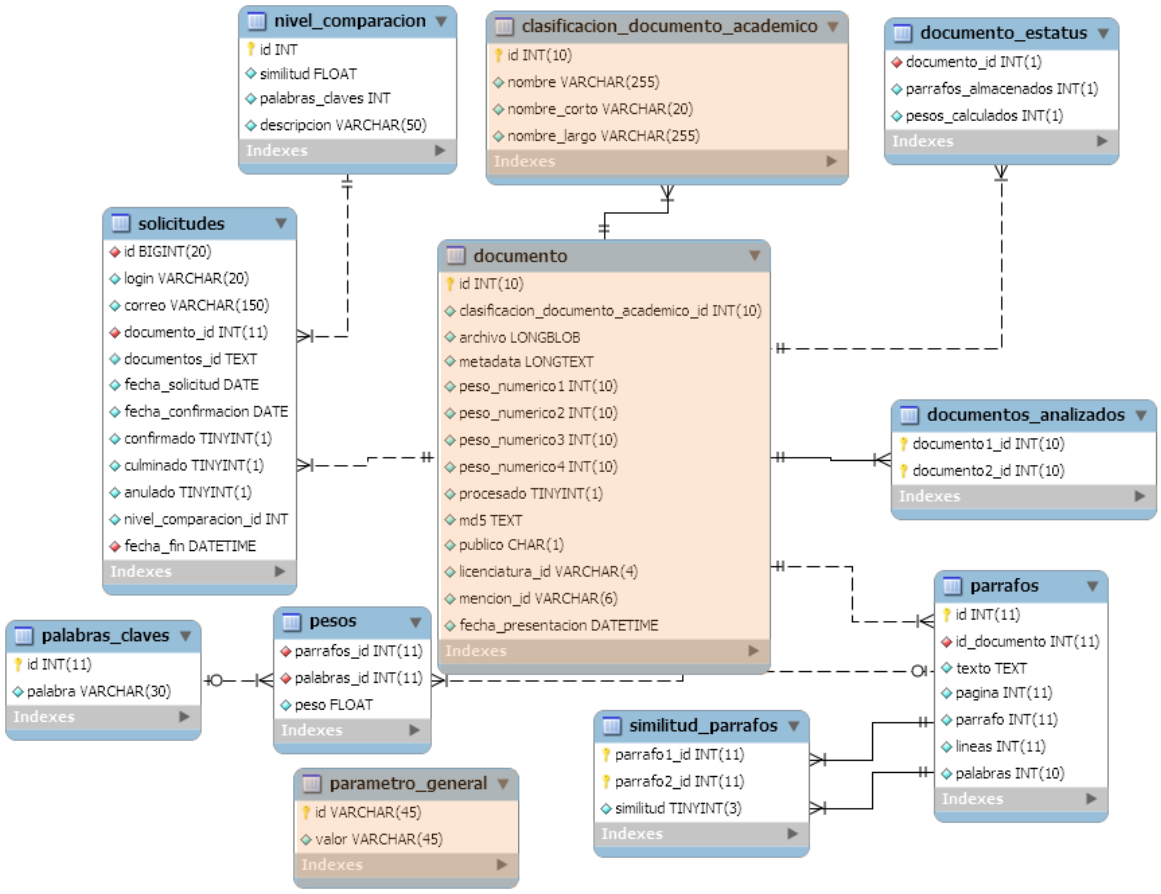


Figura 47, Modelo de Datos

Cuando se le envía al usuario el correo electrónico que indica que la solicitud ha finalizado, se le debe notificar que la misma será eliminada al cabo de treinta días, en caso de que haya sido mediante la carga manual de un documento.

Desarrollo

A continuación se describe el desarrollo llevado a cabo para cada uno de los requerimientos

Requerimiento 1 – Integración con el sistema CONEST DOCENTE

Se crean los enlaces tanto a BUSCONEST como a la aplicación, a través del Módulo de Servicio al Docente de CONEST. Estos enlaces envían información codificada que permite validar si un docente ya ha sido autenticado. Por motivos de seguridad, los códigos asociados a este requerimiento no son mostrados en esta sección.

En la Figura 48 se muestra el paso de la llave de autenticación entre BUSCONEST y CONEST DOCENTE con el sistema propuesto.

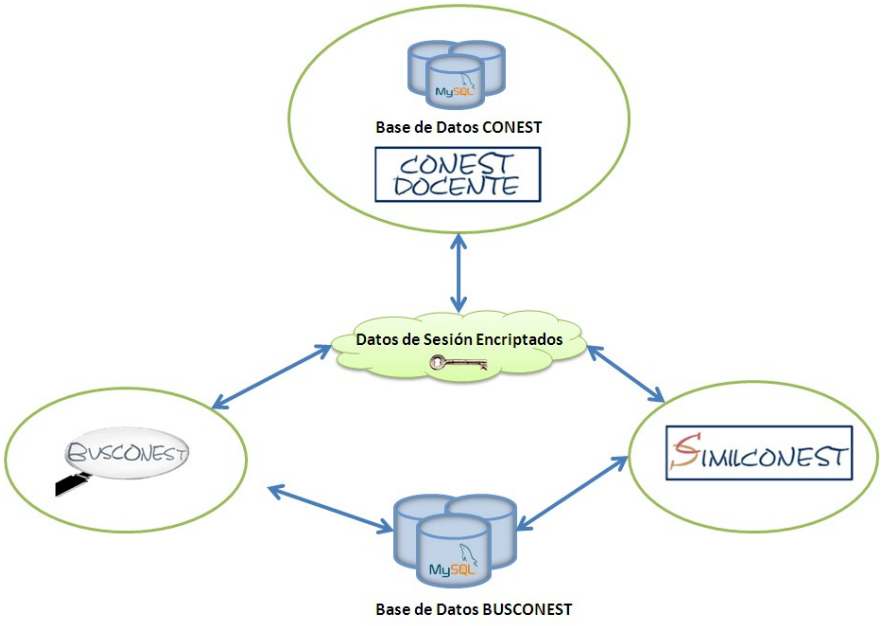


Figura 48, Autenticación entre CONEST DOCENTE BUSCONEST

Requerimiento 2 – Permitir al usuario suministrar un documento de su preferencia.

Para la implementación de este requerimiento, se crea una interfaz desde donde el usuario pueda ingresar un documento PDF de su preferencia. Una vez suministrado el archivo, se lleva a cabo el proceso de validación mediante la implementación de la librería

PdfTools como se puede observar en la Figura 49, con el fin de verificar que el documento de entrada es un archivo con formato PDF.

```
tools = PdfTools.new
num = rand(10000000000)
pdfpath="./tmp/#{num}"
#open(pdf1path,"w"){|f| f<<doc1.archivo}
open(pdfpath,"w"){|f| f<<params[:documento].read}
archivo = File.new( pdfpath,"r")
if tools.ispdf(pdfpath)
   documento = Documento.new
   documento.clasificacion_documento_academico_id = 6
   documento.titulo_html = "#{params[:documento].original_filename}"
   documento.estudiante_cedula = 123
   documento.archivo = archivo.read
   documento.save
   session[:id_documento] = documento.id
   File.delete(pdfpath)
   redirect_to :action=> 'filtrar_documentos'
   else
    File.delete(pdfpath)
    flash[:mensaje_error]='Los documentos deben tener formato PDF'
    redirect_to :action=>'cargar_documento'
end
```

Figura 49, Código para validar que un archivo de entrada tenga formato PDF

El proceso de comparación entre el documento origen y los seleccionados se lleva a cabo de la misma manera que la explicada en las iteraciones anteriores.

3.3.3. Fase de Implantación:

Una vez finalizadas las fases de planificación y construcción, se lleva a cabo la fase de implantación. El sistema es integrado y construido en su totalidad cumpliendo con los requerimientos planteados. En esta fase se hace la entrega total del sistema al cliente y se le muestra el funcionamiento de la misma.

A continuación se muestran las interfaces pertenecientes al sistema:

Menú Principal

En la Figura 50 se muestra el menú principal del sistema mediante el cual se puede visualizar el historial de comparación, comparar documentos a partir de uno seleccionado por el buscador BUSCONEST y comparar documentos a partir de uno suministrado por el docente.

Figura 50, Menú Principal

Historial de Comparación

En las siguientes Figuras se muestra la secuencia llevada a cabo para visualizar el historial de comparación.

La Figura 51 muestra todas las solicitudes que ha realizado un docente en particular. En ella se muestran el estado de la solicitud (el cual puede ser procesando, comparando documento o finalizado), la fecha en que fue realizada, el origen (si el documento a comparar fue seleccionado mediante el buscador BUSCONEST o mediante una carga manual del documento) y la acción, la cual permite consultar los resultados de una comparación o eliminar una solicitud que haya finalizado.

Figura 51, Historial de Comparación

Cuando el docente selecciona un documento del historial de comparación, se le despliega la interfaz de la Figura 52. En esta, se muestran los resultados indicando los documentos con los que se comparó, la cantidad de párrafos similares y el porcentaje de similitud.

Figura 52, Visualización de una solicitud en particular

El docente puede seleccionar de los resultados de la solicitud, un documento en particular para ver en detalle las similitudes entre los párrafos. Esto se muestra en la Figura 53. Los párrafos similares se muestran subrayados en color azul. Si el docente se posiciona sobre uno de los párrafos del documento origen, se muestra su similar en el documento comparado.

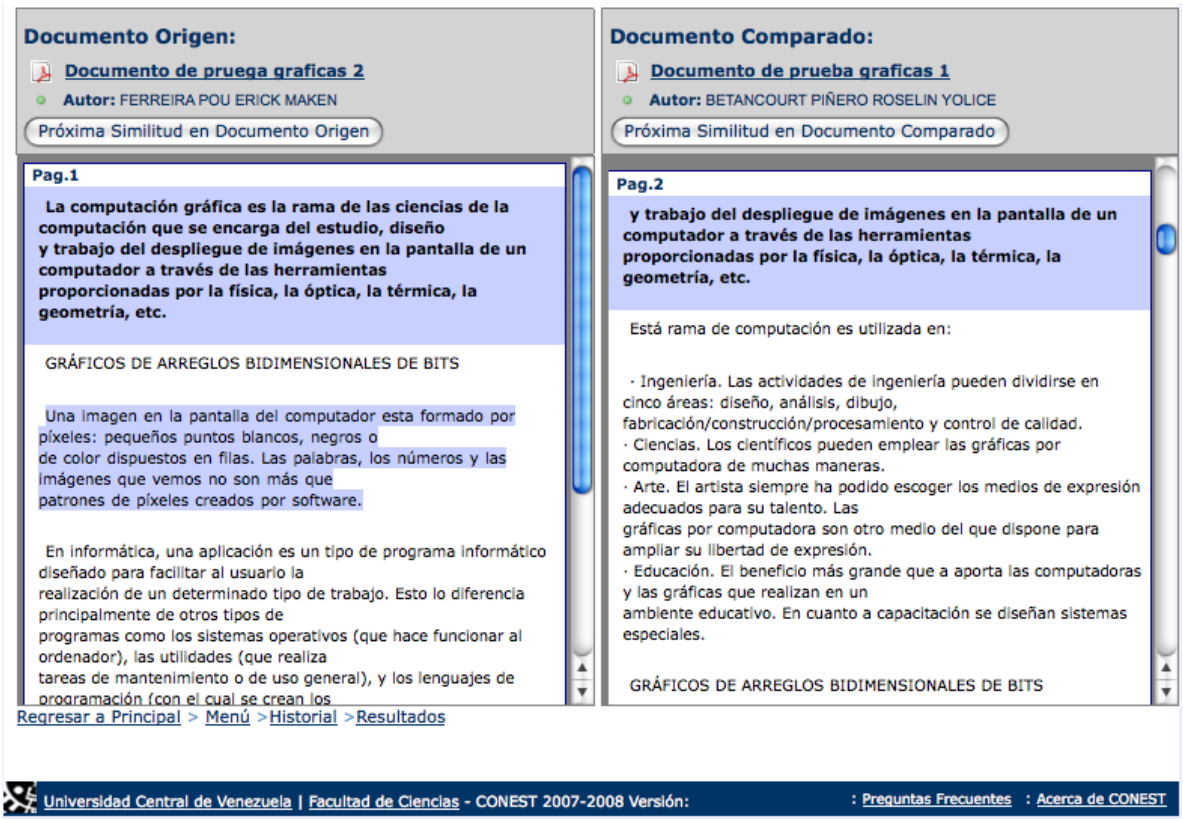


Figura 53, Resultado de la comparación

Comparación de documentos a partir de uno seleccionado mediante el buscador BSCONEST

En la Figura 54, se muestra BUSCONEST, mediante el cual el docente realiza una búsqueda de documento.

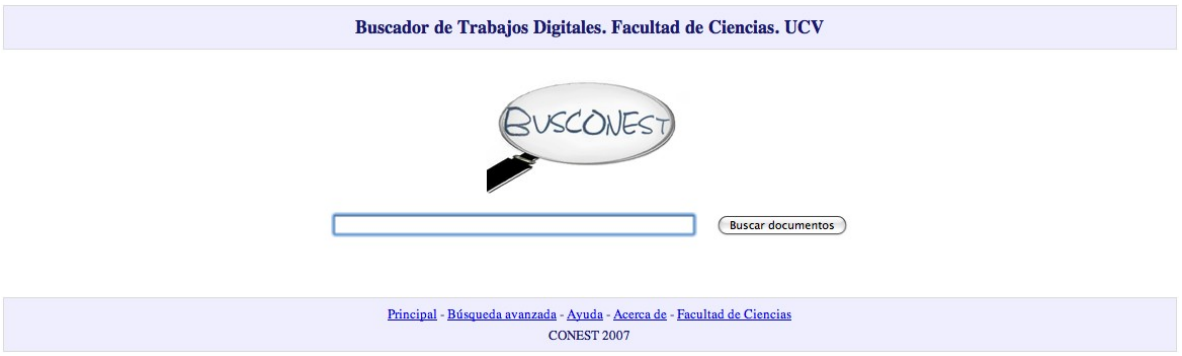


Figura 54, BUSCONEST

En la Figura 55 se muestra el resultado de la búsqueda de documentos junto a un enlace que permite seleccionarlo como documento origen para la comparación.

Figura 55, Resultado BUSCONEST

Una vez seleccionado el documento origen, se despliega al usuario un conjunto de documentos candidatos a la comparación mediante la implementación de filtros básicos. Si el docente no está satisfecho con estos resultados, puede cambiar las condiciones del filtro. Esto se muestra en la Figura 56.

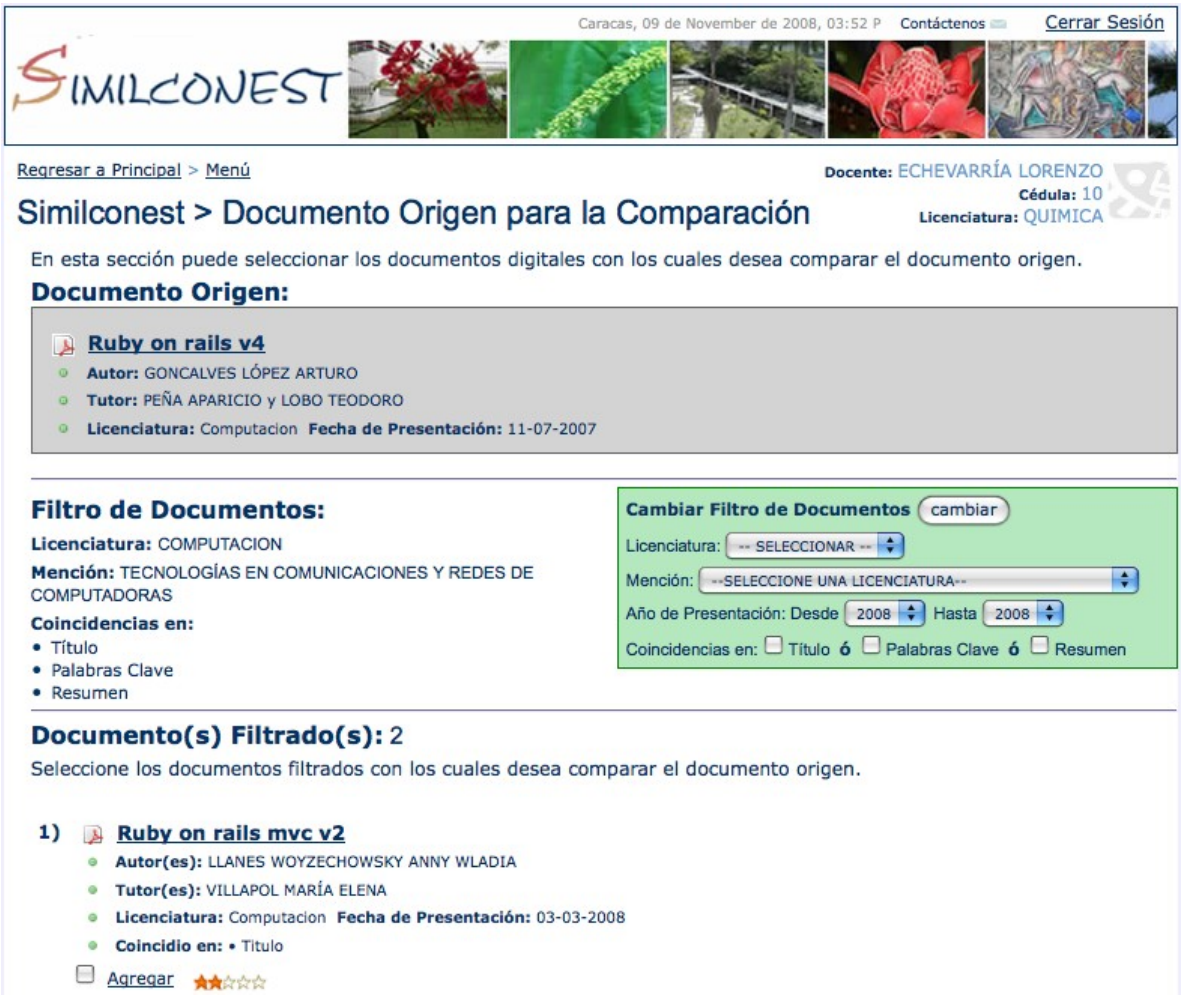


Figura 56, Documento origen con los candidatos a la comparación.

Una vez seleccionados los documentos a comparar con el origen, se muestra la solicitud realizada en la interfaz de historial de comparación. (Figura 51)

Comparación de documentos a partir de uno suministrado por el docente.

En la Figura 57 se muestra la interfaz correspondiente a la carga manual de un archivo suministrado por un docente, donde ingresa el documento y sus datos asociados.

Figura 57, Carga del Documento Digital

Conclusiones

Como resultado final de este Trabajo Especial de Grado se obtuvo una aplicación web que permite comparar los contenidos de documentos digitales de índole académico con la finalidad de detectar posibles similitudes existentes entre estos.

La herramienta cumple con todos los requerimientos que fueron planteados en este Trabajo Especial de Grado, así como también con las modificaciones que fueron surgiendo a lo largo de la vida del proyecto. Se logró una exitosa integración tanto el Módulo de Servicio al Docente de CONEST como al repositorio y buscador de documentos académicos BUSCONEST, permitiendo así la incorporación de una nueva funcionalidad provista para los docentes como una herramienta de detección de similitudes entre documentos académicos, siguiendo los mismos lineamientos y estándares de interfaz de CONEST.

Para el desarrollo del motor de comparación se estudiaron distintas técnicas de detección de copias de documentos digitales, de las cuales se implementó una adaptación realizada a la técnica Check, la cual permitió realizar las comparaciones de los contenidos de documentos detectando las similitudes existentes y permitiendo conocer el porcentaje de semejanza entre estos, arrojando los resultados deseados.

La metodología RAD facilitó considerablemente el desarrollo de la aplicación, gracias a la gran flexibilidad que esta provee para adaptarse a los cambios, así como la posibilidad de estar en constante contacto con el cliente a través de entregas parciales del sistema, permitiéndose una retroalimentación necesaria para la evolución del mismo sin la necesidad de realizar cambios que pudieran comprometer el desarrollo total del sistema.

En cuanto a las pruebas realizas, pudimos concluir que la técnica empleada para la detección de similitudes arroja unos resultados con mayor nivel de certeza para umbrales de similitud establecidos en valores mayores a 0,7 aproximadamente. Con valores menores a éste, se pudo observar que en los párrafos que tenían una gran diferencia de tamaños, se podían presentar palabras comunes a ambos, pero que no necesariamente representaban una similitud entre estos.

En cuanto al rendimiento del servidor, éste pudo ser mejorado con la implementación del servidor de trabajos backgroundrb, el cual captura las solicitudes de los usuarios y compara uno a uno los documentos en el mismo orden en el que fueron solicitados. Si el proceso de extracción y almacenamiento de estructuras del documento es interrumpido, se puede retomar a partir del lugar en que ocurrió esta interrupción evitando realizar operaciones que ya fueron ejecutadas.

Recomendaciones

Contando con una aplicación Web desarrollada para comparar contenidos y detectar similitudes entre documentos digitales, se pueden hacer las siguientes recomendaciones para trabajos futuros a realizar sobre la aplicación, a fin de mejorar o añadir funcionalidades a la aplicación:

- Dar soporte para otros formatos de documentos. Actualmente el repositorio de documentos cuenta con soporte para documentos PDF, WORD y de texto plano. De éstos, los documentos WORD no están soportados en la aplicación, por lo que pudieran ser incorporados.

- Desarrollar un administrador de parámetros desde donde el administrador del sistema pueda establecer los valores de cada uno, como por ejemplo, cantidad de solicitudes de comparación que puede realizar un docente al mes, umbrales de similitud entre los párrafos para cada uno de los niveles establecidos (similitud baja, media, alta y muy alta), etc.

- Desarrollar un generador de reportes para llevar estadísticas referentes a las comparaciones realizadas, resultados arrojados, etc.

- Adaptar la aplicación para ser utilizada por otros usuarios.

- Desarrollar un módulo que pueda descargarse y ejecutarse en el lado del cliente para realizar comparaciones entre documentos de cualquier índole.

Bibliografía

Adobe. http://www.adobe.com/es/products/acrobat/adobepdf.html [Consulta: Febrero de 2008]

Beck, K. (2000). *Extreme Programming Explained.* Addison Wesley.

BerliOS. http://openthes-es.berlios.de/ [Consulta: Octubre de 2008]

Biblioteca Digital Universitaria de la DGSCA. http://www.bibliodgsca.unam.mx/tesis/tes7cllg/sec_16.htm [Consulta: Febrero de 2008]

Brin, S., Davis, J., & García-Molina, H. (1995.). Copy detection mechanisms for digital documents. New York, NY, USA.

CASEMaker Inc. http://www.casemaker.com/download/products/totem/rad_wp.pdf [Consulta: Octubre de 2008]

Cooper, J. W., Coden, A. R., & Brown, E. W. (2002). Detecting Similar Documents Using Salient Terms. NY, USA.

Fernandez, O. (2008). *The Rails Way.* Addison-Wesley.

Flanagan, D., & Yukihiro, M. (2008). *The Ruby Programming Language.* O'Reilly.

INAOE, Instituto Nacional de Astrofísica, Óptica y Electrónica. http://ccc.inaoep.mx/Reportes/CCC-04-007.pdf [Consulta: Febrero de 2008]

Klopp N, A. C. (2008). *Estudio de Técnicas de Medición de Similitud entre Documentos Digitales.* Caracas.

LAROUSSE. (1997). *El pequeño Larousse ilustrado.* México: Larousse.

Microsoft Office Online. http://office.microsoft.com/es-es/word/HA102109193082.aspx [Consulta: Octubre de 2008]

MSINFO Sistemas de Información. http://www.msinfo.info/propuestas/documentos/documentos_digitales.html [Consulta: Febrero de 2008]

MySql. http://dev.mysql.com/doc/refman/5.0/es/introduction.html [Consulta: Agosto de 2008]

MySql. http://dev.mysql.com/doc/refman/5.0/es/features.html [Consulta: Agosto de 2008]

Ruby. http://www.ruby-lang.org/es/about/ [Consulta: Febrero de 2008]

Ruby on Rails. http://www.rubyonrails.org.es/ [Consulta: Febrero de 2008]

Sánchez E, J. I. (2008). *Elaboración de un Prototipo de Buscador de Documentos Académicos de la Facultad de Ciencias.* Universidad Central de Venezuela.

Si, A., Leong, H. V., & Lau, R. W. (1997). *Chek a Document Plagiarism Detection System.* New York, USA.

Sociedad Chilena del Derecho de Autor. http://www.scd.cl/socios.html [Consulta: Octubre de 2008]

Sociedad de Estudiantes de Ciencia de la Computación. http://seccperu.org/ededc/files/SISTEMA%20MANEJADOR%20DE%20BASE%20DE%20DATOS.pdf [Consulta: Octubre de 2008]

Solano Murillo, R., & Coles, E. http://www.di-mare.com/adolfo/cursos/2007-2/pp-RubyOnRails.pdf [Consulta: Septiembre de 2008]

Sourceforge. http://oness.sourceforge.net/proyecto/html/go01.html [Consulta: Agosto de 2008]

Szyperski, C. (1998). *Component Software - Beyond Object-Oriented Programming.* Addison Wesley.

UDLAP. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf [Consulta: Enero de 2008]

Universidad de los Andes. http://agamenon.uniandes.edu.co/~revista/articulos/cliser.html [Consulta: Septiembre de 2008]

Universidad Distrital Francisco José de Caldas. http://www.udistrital.edu.co/comunidad/profesores/rfranco/bd.htm [Consulta: Marzo de 2008]