Instituto Tecnológico y de Estudios Superiores de Monterrey

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

TESIS

MANEJO DE INFORMACIÓN EN BIBLIOTECAS DIGITALES PERSONALES.

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADÉMICO DE:

MAESTRÍA EN CIENCIAS EN TECNOLOGÍAS DE LA INFORMACIÓN

POR

LUIS ALBERTO HURTADO ALVARADO

MONTERREY, N.L. MAYO 2006

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

Los miembros del comité de tesis recomendamos que la presente propuesta sea aceptada para desarrollar el proyecto de tesis que es requisito parcial para obtener el grado de **Maestro en Ciencias de Tecnología Informática**.

Comité de Tesis

Dr. Juan Carlos Lavariega Jarquín
Asesor Principal

Dr. David A. Garza Salazar
Sinodal

Dra. Lorena Gómez Martínez
Sinodal

Dr. David A. Garza Salazar
Director del Programa de Posgrado en Electrónica, Computación, Información y Comunicaciones
Mayo de 2006

MANEJO DE INFORMACIÓN Y MODELO DE DATOS PARA BIBLIOTECAS DIGITALES PERSONALES EN AMBIENTES MÓVILES

por

LUIS ALBERTO HURTADO ALVARADO

TESIS

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

Este trabajo es requisito parcial para obtener el grado de Maestro en Ciencias en Tecnología Informática

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

MAYO 2006

Agradecimientos

A mi Asesor
Por su confianza y apoyo
incondicional en todo momento.

Al grupo de Doctores PDLib
Ha sido un honor trabajar al
lado de personas talentosas e
inspiradoras como ustedes,
nunca cambien.

A mis amigos y compañeros PDLib
Gracias por su amistad,
Francisco son invaluables tus consejos,
el vos " sin ti no somos nada..."

A ti Señor Jesucristo
Gracias porque sin tí nada de esto sería posible,
eres el motor que mueve mi vida.

Dedicatoria

A mis padres,
Gracias por su apoyo,
ustedes representan,
el mejor ejemplo a seguir.

A mis Hermanos,
Gracias por creer en mí.

A mi Esposa,
Por las interminables noches de desvelo,
esta va por tí amor.

A mi pequeña Grace,
Eres mi principal motivación.

Resumen

El manejo de información constituye una pieza fundamental en el desarrollo de la sociedad. Una sociedad informada tomará mejores decisiones que una que no lo esta, y debido a la gran cantidad de información que se genera actualmente en el siglo XXI se requieren de herramientas sólidas que permitan el almacenamiento, indexamiento y búsquedas de esta información.

En esta tesis presentamos una herramienta para el manejo de información en bibliotecas digitales personales llamada DSAPI (Data Storage API), la cuál se desarrolló siguiendo patrones de escalabilidad, transparencia y cuantificación de procesos.

La estrategia para el manejo de información utilizada por el DSAPI es el acoplamiento de una base de datos con una herramienta de indexación para manejar el almacenamiento, indexamiento, actualizaciones y búsquedas sobre documentos.

Además de esto, se requieren de mecanismos que ayuden a difundir ó buscar información de terceros a nivel mundial.

Por esta razón también presentamos la implementación de un protocolo de difusión de información conocido como OAI (Open Initiative Archive), el cual permite a las bibliotecas digitales compartir información a través de arquitecturas heterogéneas.

Finalmente, se incluyen los códigos fuentes de todo el software tratado en esta Tesis.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En las últimas décadas las ciencias computacionales han logrado un avance muy importante, impactando no solo a las grandes empresas y centros internacionales, sino también a los hogares e individuos en general, traspasando fronteras, lenguajes y costumbres. Esto, sumado a la proliferación de la telefonía inalámbrica cada vez más compleja, y de pequeños dispositivos electrónicos de mano (PDAs), ha ocasionado que las necesidades de información que se tienen sobre estos dispositivos superen a la tecnología existente.

En México, con el apoyo del gobierno; las instituciones educativas realizan investigaciones y proyectos enfocados al cómputo móvil, y cada vez más, se localizan establecimientos comerciales, líneas áreas, hoteles, e instituciones públicas y privadas que cuentan con puntos de acceso inalámbricos para el cómputo móvil.

Entre las instituciones mexicanas que están contribuyendo al desarrollo de esta temática se encuentra el Consejo Nacional de Ciencia y Tecnología, la Universidad De Las Américas en Puebla y el Instituto Tecnológico de Estudios Superiores de Monterrey; en este último, se esta desarrollando el proyecto PDLib[33] (The Personal Digital Library Project); cuyo objetivo es que el usuario móvil contando con una arquitectura de software adecuada pueda acceder a los servicios de una biblioteca digital, una biblioteca digital es una colección organizada de documentos digitales que ofrece diversos servicios a los usuarios.

Las bibliotecas digitales en un inicio fueron destinadas para usuarios es-

pecializados como grandes empresas y agencias gubernamentales, manejando exclusivamente información textual, sin embargo debido a su utilidad, hoy en día esta tecnología es utilizada no solamente por especialistas en computación e investigadores, sino también por el público en general con cualquier grado de educación [2], manejando todo tipo de formatos digitales y servicios, desde imágenes hasta multimedia.

Para responder a esta demanda se requieren bibliotecas digitales personales que ofrezcan servicios de envío, búsqueda y recuperación de documentos permitiendo su difusión a nivel mundial, debido a esto proponemos el manejo de información de una biblioteca digital personal basado en un componente que integra una base de datos con una herramienta de indexación proveyendo escalabilidad, transparencia y fácil adaptación a cambios, y la implementación de un protocolo de difusión de información que permite la comunicación de bibliotecas digitales con arquitectura heterogénea.

1.1. Planteamiento del Problema

En la década pasada se realizaron grandes avances en el área informática y de telecomunicaciones, como Internet II, tecnología inalámbrica, microelectrónica, cómputo móvil, etc. Estos avances a su vez se reflejaron en un aumento de información digitalizada entre universidades, instituciones y población en general, manteniendo esta información en bibliotecas digitales.

Estas bibliotecas digitales a menudo trabajaban de forma aislada con documentos cuya información era de tipo textual y difícilmente sufrían cambios en el transcurso de su existencia, además de esto, el tiempo de integración de los documentos ó sus cambios a la biblioteca digital no sucedía de manera inmediata; dificultando debido a esta problemática el manejo de documentos personales.

Los documentos personales contienen no solamente información textual, sino también información en forma de imágenes, audio y video; cambiando constantemente durante el transcurso de su existencia. Además de esto, ciertos documentos personales requieren ser compartidos con determinadas personas ó publicados para que sean visibles a cualquiera que lo solicite. De igual forma, las bibliotecas digitales deben permitir la navegación ó búsqueda

de documentos públicos en otras bibliotecas digitales con arquitecturas heterogéneas.

Debido a estos nuevos requerimientos, fue necesario crear una nueva arquitectura para el manejo de información de bibliotecas digitales que permita escalabilidad a nuevas tecnologías y adaptabilidad a cambios de la información, así como facilidades en la comunicación y distribución de información.

Esta arquitectura para bibliotecas digitales deberá solucionar la problemática expuesta y responder a cada una de las siguientes interrogantes:

- *¿Que elementos y/o módulos se requieren para soportar las funcionalidades de documentos personales en las bibliotecas digitales?*
  Es necesario definir aquellos módulos que permitirán trabajar con documentos cambiantes, y que respondan inmediatamente a las actividades de los usuarios.

- *¿Estos módulos responden instantáneamente a los cambios de los documentos personales?*
  Es necesario la aplicación de políticas que permitan tener los documentos siempre disponibles y que cuando se realicen búsquedas sobre información en constante cambio se responda con información válida.

- *¿El manejo de información en bibliotecas digitales personales es una actividad que involucra mucho procesamiento, son estos módulos escalables?*
  Es necesario que los módulos para el manejo de información tengan las capacidades de escalabilidad y transparencia en su arquitectura y que no funcionen como cajas negras.

- *¿Cuál es la interacción entre estos módulos?*
  Estos módulos se especializan en determinadas tareas, por lo tanto requieren de la colaboración entre sí para ofrecer una funcionalidad completa. Esta interacción o colaboración entre los módulos conlleva a la especificación de políticas y reglamentos de comunicación, las cuales garanticen que la información que se transmite entre ellos llegue a quien la solicita.

- *¿Cuales son los objetos de datos primarios que va a procesar el sistema?*
  Es necesario especificar las estructuras que contendrán la información primaria del sistema de datos, para que tales estructuras sean utilizadas en todos los módulos del sistema, facilitando la comunicación, estandarización y reutilización de software.

- *¿Cuál es la composición de cada objeto de datos y que atributos describen al objeto?*
  Cada objeto de datos posee atributos y métodos que realizan operaciones sobre estos atributos, se debe aplicar una nomenclatura estándar a estos métodos y atributos, así como definir su grado de privacidad dentro del sistema.

- *¿Donde residirán los objetos de datos?*
  Una vez definidas las estructuras de los objetos del sistema, se requiere de un modelo de datos que guarde permanentemente esta información. Este modelo de datos debe ser flexible, adaptable a los cambios y escalable.

- *¿Cuál es la relación entre los objetos y los procesos que los transforman?*
  Se debe dejar perfectamente establecido los procesos que actúan sobre los objetos de datos y como los afectan, así como las reglas que existen en el manejo de información.

- *¿Cuáles serán las políticas para la publicación de documentos?*
  Se debén de establecer los permisos necesarios para la publicación de documentos y las reglas de su funcionamiento.

- *¿Cuáles serán las políticas para la navegación en documentos de bibliotecas digitales foráneas?*
  Se establecerán los mecanismos para la navegación en bibliotecas digitales heterogéneas así como sus alcances y límites.

Las respuestas a estas preguntas se definen en el manejo de información y el modelo de datos para bibliotecas digitales personales, lo cual es el objetivo de esta tesis.

1.2. Organización de la Tesis

El presente trabajo de tesis esta organizado en 5 capítulos, el capítulo 2 describe los antecedentes, el estado del arte y los trabajos relacionados; en el capítulo 3 se describen el modelo de datos y el manejo de información en bibliotecas digitales personales, así como la arquitectura general del proyecto PDLib; en capítulo 4 se describe a detalle la arquitectura del servidor de datos y en el capítulo 5 están las conclusiones y el trabajo futuro esperado del proyecto PDLib.

Capítulo 2

Antecedentes

En este capítulo se presentan los alcances y delimitaciones de las bibliotecas digitales, los conceptos de modelos de datos, sus principales componentes y la importancia de su construcción, así como el uso de herramientas IR (Information Retrieval) en los proyectos actuales de bibliotecas digitales.

También se presentan los trabajos existentes de bibliotecas digitales, su arquitectura, limitaciones y desventajas.

2.1. Bibliotecas Digitales

Algunas décadas atrás la información estaba concentrada en edificios, como bibliotecas públicas o museos, complicando el acceso a la información y el tiempo de búsqueda. Sin embargo, la tecnología digital ha permitido el uso de la computación para la utilización eficiente de esas bibliotecas, y el surgimiento de una nueva disciplina: bibliotecas digitales.

Una biblioteca digital es una colección organizada de documentos digitales que ofrece diversos servicios a los usuarios, como envío, clasificación, búsqueda, recuperación y administración. Facilitando actividades de estudio e investigación colaborativa entre usuarios distribuidos geográficamente[33].

En la figura 2.1 se presenta un caso de uso para los usuarios de las bibliotecas digitales. En la gráfica podemos ver que las bibliotecas digitales

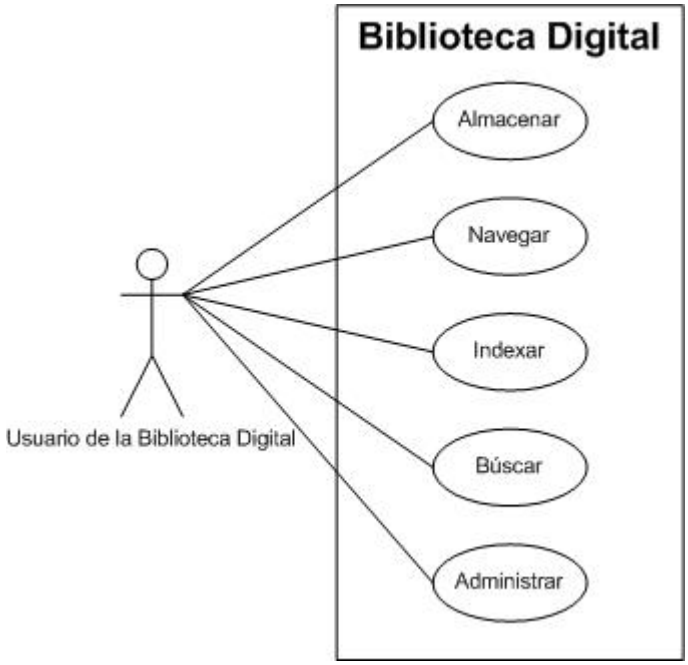

Figura 2.1: Servicios de una biblioteca digital

proveen los siguientes servicios (figura 2.1):

- **Almacenar.** Las bibliotecas digitales son capaces de almacenar grandes cantidades de información en diferentes formatos, siendo ligeros y fáciles de manejar los textuales como ASCII, LaTex, HTML, SGML y PostScript; y pesados formatos multimedia, como audio digital y video, ya que requieren más espacio y procesamiento.

  Las bibliotecas digitales usan diferentes mecanismos para el almacenamiento, como puede ser el sistema de archivos del sistema operativo o mediante Sistemas Gestores de Bases de Datos (DBMSs) con soporte para objetos binarios (BLOBs), lo cual permite mantener dentro de la base de datos los documentos, agregando escalabilidad y una mejor organización a la biblioteca digital.

  Con el objetivo de optimizar espacio en disco duro, las bibliotecas digitales también utilizan técnicas como la compresión para el almacenamiento digital, consiguiendo reducir el tamaño de los

documentos alrededor de un veinticinco por ciento del documento original[41].

- **Navegar.** La interfaz de usuario es uno de los componentes más importantes de las bibliotecas digitales, ya que incorpora la interacción con el usuario y el despliegue de grandes cantidades de datos eficientemente.

  Estudios recientes han indicado que los usuarios prefieren formas o construcciones sencillas para realizar búsquedas de información dentro de las bibliotecas digitales[4]. Por esta razón, es importante contar con una interfaz que se adecué a las necesidades de los usuarios sin sacrificar usabilidad.

  Se han realizado diversas investigaciones en este campo, como la de visualización de resultados en dos dimensiones[35], la cual permite que miles de documentos pueden ser mostrados al mismo tiempo.

- **Indexar.** El indexamiento permite que las bibliotecas digitales puedan obtener rápidamente resultados en las consultas. Este puede realizarse de diferentes modos: incremental o completo. El incremental permite que los índices se actualicen conforme se agregan, editan o eliminan documentos, actualizando solo la porción del índice afectada, mientras que el completo requiere reindexar toda la biblioteca por cada modificación de ésta, a menudo los reindexamientos completos consumen gran cantidad de tiempo.

  El indexamiento puede realizarse sobre la información almacenada en las bibliotecas digitales, sobre los metadatos de ésta, o en ambos; dependiendo del tipo seleccionado y del formato de la información será la complejidad de este indexamiento.

  El formato textual es más ligero de indexar, sin embargo aquellos formatos como audio o video digital, requieren de procesamiento extra para obtener partes indexables del mismo, requiriendo estos procesos un esfuerzo computacional mayor[2].

- **Buscar.** Las bibliotecas digitales usan diversas técnicas para la recuperación de información: búsquedas en metadatos, búsquedas en texto completo y búsquedas en contenidos para otros tipos de formatos.

La efectividad de la recuperación de información puede ser medida en términos de porcentajes de información relevante y no útil. Por esta razón, se utilizan algunas técnicas con el objetivo de regresar información valiosa, como la extracción automática de los metadatos antes de almacenar los documentos, o la utilización de agentes para definir el perfil de los usuarios y filtrar información no importante para ellos.

Entregar contenidos. Dependiendo del tamaño del contenido localizado, se puede optar por diferentes medios para la entrega del contenido. Si es relativamente pequeño se puede regresar por el mismo medio con el que se realizó la consulta, o si es grande, como los archivos multimedia, se pueden regresar por otros medios dedicados a este fin. También se toma en cuenta la cantidad de usuarios que puede soportar la biblioteca digital y los tiempos de tráfico.

Presentar resultados. Dependiendo del dispositivo físico mediante el cual se accede a la biblioteca digital, es la presentación de resultados. Los dispositivos generalmente se clasifican en ligeros y pesados [10], siendo estos últimos aparatos como computadoras personales, laptops, notebooks. Mientras que los dispositivos ligeros son aquellos que el usuario puede portar y transportar fácilmente a cualquier lugar donde se desplace, en esta clasificación se encuentran aparatos como PDAs y celulares; y es en esta clasificación donde se encuentran los algoritmos más complejos para la presentación de resultados, debido a las limitaciones inherentes de los dispositivos ligeros.

- **Administrar.** Las bibliotecas digitales permiten mantener diferentes niveles de seguridad independientes para cada documento o colección de documentos, así mismo, copias de los documentos en diferentes formatos y versiones.

De igual forma, las bibliotecas digitales comparten ciertas características fundamentales en el manejo de información. Entre estas características tenemos:

- *Acceso Universal.* Las bibliotecas digitales requieren adaptarse a tecnologías como cómputo móvil, de tal forma que puedan prestar

sus servicios a través de los diferentes dispositivos que existen como PCs, PDAs, laptops y teléfonos celulares [1] proporcionando un acceso universal a la información.

- *Compartir información con otras bibliotecas digitales, formando una sola biblioteca digital mundial [9].* Para brindar un mejor servicio las bibliotecas digitales necesitan estar comunicadas entre sí, de tal forma que una búsqueda pueda ser realizada en diversas bibliotecas digitales con arquitecturas heterogéneas a través de protocolos especializados como OAI *(veáse sección 3.3).*

- *Precisión en los resultados de las búsquedas.* Uno de los retos mas importantes de las bibliotecas digitales es encontrar información relevante [43] en las búsquedas. Debido a la gran cantidad de información que puede contener una biblioteca digital, esta característica ahorra tiempo en la navegación de los resultados.

- *Detección y tratamiento de copias en los documentos.* Las copias no controladas representan un problema grave para las bibliotecas digitales, ya sea por derechos de autor o por desperdicio de espacio en disco [5]. Por esta razón, es necesario implementar mecanismos de control de copias que ayuden a aminorar este problema
.

Para entender su construcción y el manejo de información dentro de las bibliotecas digitales es necesario primeramente comprender su modelo de datos, el cuál nos mostrará los objetos de datos de la biblioteca digital, así como las operaciones que se realizan sobre estos. A continuación describimos las características de los modelos de datos.

2.2. Modelo de datos

Mediante los modelos de datos representamos los objetos y sus interacciones dentro de un sistema computacional, existen diversas definiciones de lo que es un modelo de datos, aquí presentamos solamente dos de ellas:

- Un modelo de datos es un sistema formal y abstracto que permite definir los datos y su tratamiento de acuerdo con reglas y convenios predefinidos [40].
- Un modelo de datos es entendido como cualquier clase formalmente definible de estructura de datos, la cual puede ser usada como la base para el desarrollo de aplicaciones de procesamiento de datos [23].

Un modelo de datos posee tres componentes principales:

- *Estructuras de datos.* Es la colección de objetos abstractos que representan a los datos.
- *Operadores.* Conjunto de operadores con reglas definidas que permiten manipular a las estructuras de datos.
- *Reglas de Integridad.* Colección de conceptos y reglas que permite validar a los datos del modelo.

Los modelos de datos tienen dos objetivos:

- *Formalización.* Se definen formalmente las estructuras permitidas y las restricciones.
- *Diseño.* El modelo resultante se considerará un elemento básico para el desarrollo de la metodología de diseño de otros sistemas similares.

Dentro del modelo de datos los objetos del sistema serán manipulados usando reglas predefinidas y utilizando exclusivamente los operadores del sistema.

Existen diversas categorías de modelos de datos [36]:

1. Modelos lógicos basados en objetos.
2. Modelos lógicos basados en registros.
3. Modelos físicos de datos.
4. Modelo Objeto-Relacional.

Los ***modelos lógicos basados en objetos*** se usan para describir los datos en el nivel conceptual y de visión, permitiendo representar los datos como se captan en el mundo real. Usan los conceptos de entidades u objetos y representan las relaciones entre ellos. Sus principales ponentes son el modelo entidad-relación (E-R) y el modelo orientado a objetos.

El modelo E-R tiene tres conceptos básicos [6]:

1. *Entidad.* Una entidad es un objeto en el mundo real que es distinguible de otros objetos.

2. *Relaciones.* Una relación es una asociación entre varias entidades.

3. *Atributos.* Los atributos son las propiedades descriptivas de las entidades.

El modelo orientado a objetos es una adaptación de la programación orientada a objetos, implementando conceptos como encapsulación, instancias, etc.

Los ***modelos lógicos basados en registros*** se utilizan para describir datos en los niveles conceptual y físico. Utilizan registros e instancias para representar la realidad, así como las relaciones que existen entre estos.

Los modelos más aceptados de este tipo son:

1. *Modelo Relacional.* Propuesto por Codd a finales de los sesenta[7], proponiendo un modelo de datos basado en la teoría de las relaciones, donde los datos se estructuran lógicamente en forma de relaciones (tablas) independientemente de su estructura física.

   El elemento básico en el modelo relacional es la relación y se puede representar como una tabla. La tabla se encuentra compuesta por una serie de atributos (columnas) que representan a propiedades de la misma y por tuplas (filas) que consiste en los renglones de la tabla. Los atributos poseen un dominio, que es el conjunto de los valores legales que puede adoptar. Cada tupla puede tener uno o mas atributos principales llamados claves, con los cuales se diferencia de las demás tuplas.

   El modelo relacional ha sido el mas popular desde su propuesta, adoptado por la mayoría de los sistemas de bases de datos existentes.

2. *Modelo Jerárquico.* El modelo de datos jerárquico consiste en una serie de registros que se conectan entre sí por medio de enlaces. Cada registro es una colección de campos que contiene un único valor. Un enlace es una asociación entre dos registros.

   El modelo de datos jerárquico se representa a través de diagramas de estructura de árbol en orden descendente formando una estructura jerárquica, de ahí su nombre.

3. *Modelo de Red.* El modelo de datos de red consiste de una colección de registros los cuales se encuentran conectados entre sí a través de enlaces.

   En una estructura de datos en red los nodos hijos pueden tener mas de un padre, a diferencia del modelo jerárquico que cada hijo solo puede tener un padre.

El ***modelo físico de datos*** se utiliza para describir los datos en el nivel más bajo, capturando aspectos de la implementación.

Básicamente se distingue el modelo unificador:

- *Modelo unificador.* El modelo unificador considera a las bases de datos como grandes redes de archivos interconectados entre sí, que puede ser descompuestos básicamente en una lista de archivos primitivos y de enlaces. Estos archivos contienen información en forma de registros y manejan estructuras como hash, indexamiento secuencial, árboles B+ y archivos desordenados.

  Un enlace es una estructura que conecta los registros de dos diferentes archivos, estos enlaces son en forma de arreglos de punteros, listas invertidas, listas circulares y multilistas celulares [3].

  Un punto importante de este modelo es que permite estimar la eficiencia de las operaciones efectuadas sobre la base de datos.

El ***modelo Objeto-Relacional*** fue recientemente desarrollado y combina el modelo de datos orientado a objetos con el modelo relacional.

En esta tesis se describe el modelo de datos utilizando el modelo E-R (veáse figura 3.10)

Otra de las herramientas utilizadas en esta tesis la constituye el indexador, el cuál se describe en la siguiente sección.

2.3. Indexadores

En el ambiente de bibliotecas digitales el texto de los documentos contenidos puede llegar a ser de miles de Terabytes, una búsqueda sobre ellos podría llevar varias horas e incluso días, resultando impráctico para los usuarios de la misma; la solución a este problema se encuentra en el indexamiento previo de esta información.

El indexamiento siempre ha existido en la historia de la humanidad, por ejemplo textos antiguos como la Biblia o el Corán, requerían de índices o concordancias para ubicar determinadas palabras a lo largo del documento.

El indexamiento consiste en crear uno o varios archivos con referencias a documentos, basándose en las palabras contenidas en los mismos, nombrados a menudo archivos de índices invertidos[38], los cuales contienen tuplas que incluyen al menos la palabra a buscar y el identificador del documento $<$ palabra, id_document $>$, un ejemplo de un archivo invertido básico se encuentra en la figura 2.2, donde se indica que la palabra *Java* aparece en el documento 4 y la palabra *Managing* aparece en los documentos identificados como 20 y 22.

Word	Document
Index	3,5
Java	4
Linux	14
Lucene	16
Managing	20,22
.....	.....

Figura 2.2: Archivo invertido básico

No todas las palabras son indexadas, existen algunas de uso común que no tiene caso indexarlas (llamadas stopwords), ya que están presentes en todos

los documentos; a excepción de estas palabras, todo el contenido textual es indexado.

El contenido no textual que necesite ser indexado, como puede ser en el caso de imágenes y archivos multimedia, se requiere de módulos con los cuales sea posible extraer información indexable de esos formatos y así poder conformar un archivo de índices con ellos.

En las operaciones de búsquedas son consultados los archivos de índices, cada indexador es responsable de la administración de estos archivos, reduciendo la complejidad de la búsqueda a un problema de ordenamiento. Los indexadores ocupan diversas técnicas para mantener el archivo de índices, siendo los árboles B los más utilizados, presentando complejidades limitadas superiormente por O(log n). Estas búsquedas pueden ser booleanas u ordenadas por relevancia, con tiempos de respuesta relativamente bajos.

2.3.1. Partes Fundamentales

En las bibliotecas digitales el indexador constituye una parte medular, responsable de la creación y mantenimiento de los índices, respondiendo rápidamente a las peticiones de búsquedas de los usuarios.

La selección del indexador también influye en el modelo de datos, ya que actualmente sistemas manejadores de base de datos, como MYSQL permite mantener los índices dentro del mismo modelo de datos, mientras que un indexador externo requiere de la modificación del modelo de datos para permitir combinar la información de los índices con la información almacenada en la base de datos.

Las partes fundamentales de un indexador[15][42] son:

1. *Actualización de los índices.* Las principales estrategias para la actualización de los índices son: in-place update, index merging, and complete re-build [20].

   La primer estrategia (in-place update) consiste en ir modificando los archivos de índices cada vez que un documento es agregado, editado o eliminado. Sin embargo, un documento puede contener alrededor de cientos de distintos términos, involucrando varios accesos a disco para realizar las actualizaciones.

La segunda estrategia (index merging) consiste en indexar las colecciones dividiéndolas en bloques, construyendo un índice para cada bloque y después combinando.

La tercera estrategia (complete re-build) consiste en reconstruir todo el índice cada vez que llega un nuevo documento, esta estrategia es muy cara, sin embargo muchos sistemas de indexamiento lo han adoptado por su sencillez.

2. *Almacenamiento.* El almacenamiento de la información y de los índices juega un papel relevante en los indexadores, ya que el índice puede ser casi igual al tamaño de la información, es necesario utilizar algún tipo de compresión sin perder *performance* al momento de realizar las búsquedas.

3. *Plataforma.* Las prestaciones de los sistemas operativos sobre los cuales opera el indexador.

4. *Formatos de Archivos.* El tipo de formato de archivos soportado por el indexador.

5. *Metadatos.* Se refiere al tratamiento que le dan los indexadores a los documentos, tratándolos simplemente como cadenas de texto ó como objetos que pueden contener propiedades inmersas (metadatos).

6. *Procesamiento de Stop-words.* El uso de Stop-words (palabras que se encuentran en todos los documentos) que son ignoradas por los indexadores, permite un índice más compacto y eficiente.

7. *Stemming.* Algunos mecanismos de búsqueda completan las palabras de acuerdo a su raíz etimológica, con el objetivo de buscar palabras similares.

8. *Queries.* La complejidad permitida en las expresiones de búsqueda es un punto importante en los indexadores, como expresiones booleanas y búsquedas por relevancia.

9. *Concurrencia.* Se refiere a permitir la utilización de los índices simultáneamente, mientras se realizan otras búsquedas o se actualiza el índice.

10. *Lenguajes Soportados.* Para operaciones como el stemmer y las Stopwords se debe permitir definir el idioma sobre el cual el indexador estará operando.

11. *Orígenes de Datos.* Es el tipo de información con la que trabaja un indexador, por ejemplo algunos indexadores trabajan exclusivamente con páginas Web, con archivos de datos o con ambos.

2.3.2. Tecnologías de indexamiento

A continuación presentamos una serie de tecnologías de indexamiento que fueron consideradas y evaluadas como parte de esta tesis.

MG (Managing Gigabytes)

MG [42] es un sistema de código libre para indexamiento y búsqueda de información textual, desarrollado inicialmente por Ian Witen(Profesor de Ciencias Computacionales en la Universidad de Waikato, Nueva Zelanda), Tim C. Bell(Universidad de Canterbury), Alistair Moffat(Universidad de Melbourne), Justin Zobel(RMIT) y otros colaboradores.

MG opera bajo sistemas UNIX, trabajando sobre *conjuntos* de documentos llamados bases de datos, permitiendo realizar operaciones independientes sobre esos conjuntos, favoreciendo con esto al rendimiento general del sistema.

El sistema MG puede ser dividido en tres partes fundamentales [42]:

- Consultas.
- Indexamiento.
- Almacenamiento.

Consultas. Para realizar las búsquedas MG se apoya en los archivos de índices que previamente generó. Estos índices llamados *archivos invertidos* contienen una lista de referencias a los documentos basándose en los términos contenidos en ellos. Por ejemplo, en la figura 2.3 se observa que del *Source Text* se genera un archivo de índices invertido en la cual el término *Gee*

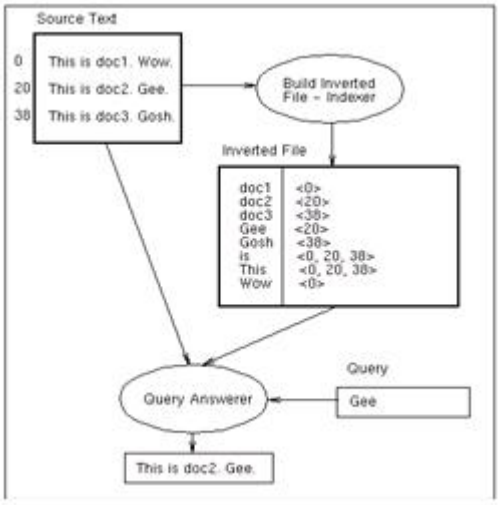

Figura 2.3: MG. Ejemplo de Archivo Invertido

aparece en el documento con identificador 20 y cuando se genera una búsqueda de la palabra *Gee* el documento referenciado es devuelto.

Existen tres tipo de búsquedas soportadas por MG:

- *Booleanas.* La búsqueda booleana consiste en utilizar frases de búsqueda combinadas con operadores booleanos, los documentos devueltos son todos aquellos que cumplan con la expresión de búsqueda.
- *Por Relevancia.* En la búsqueda por relevancia se especifica la expresión de búsqueda y se obtienen los documentos más relevantes de acuerdo con las palabras especificadas.
- *Por el número de documento.* En esta búsqueda se especifica el número del documento que se requiere como resultado.

Las consultas son realizadas por un programa llamado *mgquery*.

Cuando el programa *mgquery* inicia lee un archivo nombrado *.mgrc* el cual contiene una serie de comandos que especifican opciones por default.

Los atributos de las palabras dentro del diccionario pueden contener claves como < f_t, F_t, I_t > donde f_t es el número de documentos en los cuales aparece el término buscado, F_t es el número de ocurrencias del término

dentro de la colección de documentos, e I_t se refiere al puntero del archivo invertido.

Indexamiento. En la indexación MG se ayuda de diccionarios. Los diccionarios se encargan de mantener listas de palabras indexadas y algunos otros atributos acerca de las palabras. Los diccionarios y los archivos invertidos son guardados en diferentes lugares, ya que hay operaciones en las que no es necesario leer el archivo invertido, porque el diccionario es suficiente para responder a la petición.

Almacenamiento. Uno de los puntos fuertes de MG es que mantiene un tamaño de índices muy reducido, ocupando compresión de tipo Huffman.

Java Search Engine

Java Search Engine[19] es un motor de búsqueda generalmente utilizado en sitios Web. Creado para que los usuarios puedan encontrar cualquier documento que busquen en un determinado sitio Web.

Java Search Engine generalmente es utilizado como API de Java desde JSPs, servlets y EJB. Los resultados pueden ser guardados en XML para posteriormente ser transformados en HTML usando hojas de estilo XSLT.

Al estar construido completamente en Java, puede operar en cualquier sistema operativo. El software es libre de uso, sin embargo su código fuente no se encuentra disponible.

Puede indexar sobre texto plano, HTML, PDF y MS Word. Para la indexación soporta la actualización incremental sin detener su funcionalidad de búsquedas.

Tiene características JDBC para comunicarse con bases de datos, esto le permite guardar el archivo de índices en la base de datos cuando es requerido.

MySQL

Algunos sistemas gestores de bases de datos (DBMSs) han incorporado recientemente características de búsquedas en texto completo, este es el caso de MySQL [14].

A partir de la versión 3.23.23, esta disponible la funcionalidad de

indexamiento y búsqueda en texto completo. Un índice de texto completo en MySQL es un índice de tipo FULLTEX, usado exclusivamente con tablas de tipo MyISAM y basado en atributos CHAR, VARCHAR o TEXT.

Un ejemplo del uso de esta funcionalidad es el siguiente:

Consulta 1:
MYSQL> CREATE TABLE articles (id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
-> title VARCHAR(200),
-> body TEXT,
-> FULLTEXT (title,body)
->);

Por medio de la palabra reservada FULLTEXT se define el indexamiento de los campos *title* y *body*.

Para realizar las búsquedas en texto completo se utiliza la función MATCH(), la cual retorna los resultados ordenados por relevancia. El texto a buscar es pasado como argumento a la función AGAINST ('Argumento').

Consulta 2:
MYSQL> SELECT * FROM articles
-> WHERE MATCH (title,body) AGAINST ('database');

En esta consulta (consulta 2) se realiza una búsqueda de texto completo con la expresión *database*.

Consulta 3:
MYSQL> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root') AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root');

Esta consulta (consulta 3) retorna los resultados por relevancia con la instrucción *MATCH* y los ordena en orden decreciente por relevancia.

Las búsquedas booleanas también están soportadas usando el modificador

IN BOOLEAN MODE, ejemplo:

Consulta 4:
MYSQL> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);

Donde la consulta regresa todos aquellos valores donde aparece el término *MySQL*, pero que no contienen el termino *YourSQL*.

Los operadores soportados por MySQL para las búsquedas booleanas son:

- *+* Indica que el término debe aparecer en cada uno de los resultados devueltos.
- *-* Indica que el término no debe aparecer en ninguno de los resultados devueltos.
- *(no operador)* Operador por default cuando no es colocado algún modificador (+, -), índica que la palabra es opcional.
- *> <* Estos operadores son usados para cambiar el orden de relevancia en el que aparecen los documentos, el operador > incrementa la contribución de la palabra en la relevancia y el operador < la decrementa.
- *( )* Los paréntesis son utilizados para agrupar palabras dentro de sub-expresiones.
- *˜* Este operador actúa como un operador de negación, disminuyendo la participación de la palabra en la relevancia de los resultados devueltos, sin llegar a omitirla como el operador -.
- *** Este operador sirve para truncar las palabras en las expresiones, permitiendo búsquedas con solo una parte de la palabra a buscar.
- *"* Este operador delimita la búsqueda de frases.

MySQL también soporta el uso de Stop_words para el indexamiento:

- Todas aquellas palabras con tamaño menor a 4 caracteres son ignoradas.
- Las palabras contenidas en la lista de Stop-words son ignoradas, y esta lista puede ser personalizada.

La longitud mínima y máxima de las palabras a indexar puede ser modificada a través de las variables del sistema: ft_min_word_len y ft_max_word_len. Así como el archivo de Stop_words está definido por la variable ft_stopword_file.

Existen algunas restricciones dentro de MySQL para la búsqueda en texto completo:

- El soporte existe solamente en tablas de tipo MyISAM.
- El conjunto de caracteres UNICODE no está soportado.
- Todas las columnas en un índice FULLTEX debe tener el mismo conjunto de caracteres.
- El orden de los parámetros de la función MATCH debe coincidir exactamente con los que se definieron en la declaración de la tabla, a menos que MATCH se encuentre en modo booleano.
- El argumento pasado a AGAINST() debe ser exclusivamente de tipo cadena.

Jakarta Lucene

Creada por Doug Cutting (desarrollador del sistema de búsqueda V-Twin para Apple y arquitecto de Excite) y mantenida por miles de usuarios en la Internet, Lucene es una herramienta open source basada en Java para indexamiento y búsqueda de texto.

La arquitectura de Lucene consiste en 4 partes fundamentales (Figura 2.4):

Document.
En Lucene un documento es una secuencia de campos, un campo es un par

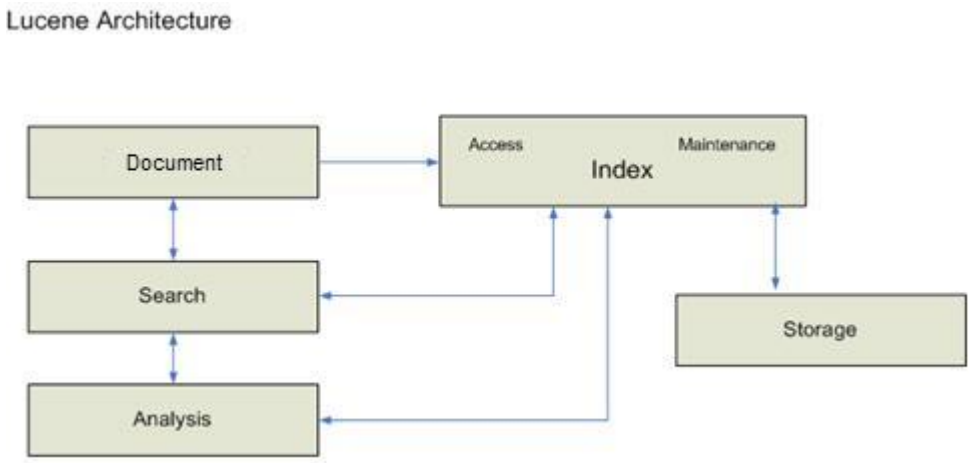

Figura 2.4: Arquitectura de Lucene

<name, value> donde name es el nombre del campo y value su valor en texto plano. Los campos pueden ser almacenados, indexados o analizados individualmente, Existen 4 formas de agregar los campos a los documentos:

- *Field.Keyword* El dato es almacenado e indexado pero no tokenizado, es muy usado para los datos que serán almacenados y que no sufrirán cambios en el transcurso de su permanencia, como las fechas.
- *Field.text* El dato es almacenado, indexado y tokenizado. Es el que se ocupa para la parte indexable de información contenida en los documentos.
- *Field.UnStored* El dato no es almacenado pero es indexado y tokenizado.
- *Field.UnIndexed* Los datos son almacenados, pero no son indexados ni tokenizados. Es usado a menudo como los campos que se desean que verse en una búsqueda, pero no son parte del criterio de la búsqueda

Un ejemplo de los campos definidos de un documento es el que se encuentra en la tabla 2.1.

Analysis.

Field	stored	indexed	Analyzed
Path	Yes	Yes	Yes
Modified	Yes	Yes	No
Content	No	Yes	Yes

Cuadro 2.1: Indexamiento de campos en Lucene

Encargada de tokenizar los documentos, extrayendo todos los tokens indexables y aislándolos del resto, para ser procesados por el indexador.

El objetivo del *analysis* consiste en tener un archivo de palabras buscables, el cual no este contaminado de basura para dejar el camino libre al indexador.

En esta fase de análisis tienen su participación diversas clases del paquete *org.apache.lucene.analysis*:

- *WhitespaceAnalyzer.*
- *SimpleAnalyzer.*
- *StopAnalyzer.*
- *StandardAnalyzer.*
- *SnowballAnalyzer.*
- *Analyzer.*

Index.
Usado para mantener índices, con capacidad para crear índices nuevos o agregar documentos a un índice existente. Los índices son almacenados en el file system, dentro de un directorio, sin embargo se puede adaptar para almacenarse en bases de datos. El indexamiento en Lucene esta basado en árboles B [13].

Search.
Se encarga de interpretar las sentencias de búsquedas, así como de procesarlas y regresar los resultados. Lucene soporta una gran variedad de búsquedas como AND, OR, NOT, búsquedas difusas, búsquedas por proximidad, búsquedas con comodines y búsquedas por rango.

2.4. Otras tecnologías relacionadas

2.4.1. HtDig

HtDig es un sistema completo de indexamiento y búsqueda para un dominio o una intranet [16].

Para su funcionamiento HtDig realiza tres tareas principales:

- *Exploración.* La exploración consiste en el primer paso para las búsquedas, formando una base de datos con todos los documentos a indexar. Cada documento es visitado, examinado y las palabras son extraídas y almacenadas. Al final de este proceso al menos se crean dos archivos, el primero es una lista de palabras y el segundo contiene la ubicación de los documentos con la información acerca de estos.

  Esta fase es realizada por un programa llamado htdig.

- *Combinación.* Los datos extraídos en la fase anterior deben ser convertidos para que el motor de búsqueda pueda entenderlos. La combinación consiste de dos procesos:

  1. Convertir el archivo general de la fase anterior en archivos de índices separados.
  2. Actualizar la información cambiante de los documentos en los archivos de índices.

  Esta fase es realizada por un programa llamado htmerge.

- *Búsquedas.* La información procesada en los pasos previos esta lista para ser utilizada para las búsquedas a través de programas CGIs. Esta fase es realizada por un programa llamado htsearch.

Para el indexamiento de formatos diferentes a los nativos (Texto plano, HTML) se requieren de extractores de texto, por ejemplo, para la extracción de texto en archivos de formato PDF se usa el programa *acroread* que se encarga de extraer datos indexables para HtDig. HtDig también permite indexar el contenido de las bases de datos, siempre y cuando sea accesible su contenido vía Web.

Entre las características relevantes de HtDig se encuentran:

- *Exploración en intranets.* HtDig puede buscar información que se encuentre en varios servidores localizados en una intranet.
- *Código fuente libre.*
- *Búsquedas de expresiones booleanas.*
- *Resultados de búsquedas configurables a través de plantillas.*
- *Búsquedas difusas.* Las búsquedas pueden ser realizadas de forma exacta, a través de la raíz de la palabra, sinónimos, ignorando acentos, subcadenas y prefijos.
- *Búsquedas nativas en archivos de texto y páginas HTML.*
- *Metadatos pueden ser agregados a documentos.*
- *Notificaciones por emails de documentos expirados.*
- *Uso de Colecciones.*
- *Las profundidades de las búsquedas pueden ser limitadas.*

Entre los usuarios de HtDig se encuentran principalmente Web sites e intranets.

2.4.2. Dienst

Dienst es un sistema para configurar un conjunto de servicios individuales ejecutándose sobre servidores distribuidos que cooperan proveyendo servicios de una biblioteca digital. La arquitectura abierta del sistema Dienst permite combinar los servicios de maneras flexibles y aumentar los servicios existentes.

Dienst esta formado por:

- Una arquitectura conceptual para librerías digitales distribuidas.
- Un protocolo para la comunicación en la arquitectura.
- Un sistema de software que implementa el protocolo.

El sistema Dienst fue creado originalmente para el proyecto de Reportes Técnicos sobre Ciencias Computacionales, que en colaboración con DARPA, crearon una biblioteca digital para estos reportes. La arquitectura Dienst consiste en:

- Un modelo de documento que incorpora nombramiento único, múltiples versiones, estructuración lógica y difusión de contenidos.
- Un conjunto de servicios individuales bajo protocolos abiertos.
- Un mecanismo para combinar servicios y contenido en múltiples colecciones.
- Un mecanismo para distribuir colecciones entre regiones que faciliten la distribución global.

Modelo de documento

El modelo de documento de Dienst posee características que permiten almacenar los documentos en múltiples formatos como texto, imágenes, audio, video y permite difundir su contenido en múltiples variaciones.

Las características del modelo de documento son:

- Nombres únicos, para esto usan manejadores compuestos por el nombramiento de autoridad del documento y una cadena de identificación para esa autoridad.
- Múltiples versiones.
- Estructuración lógica de los documentos basados en:
  - Múltiples tipos de metadatos que pueden ser asociados con todo el documento o con alguna parte de este
  - Múltiples vistas de un documento. Como por ejemplo un composición musical puede tener el archivo de audio y el archivo midi.
  - Composición jerárquica de las vistas de los documentos divididos en secciones, capítulos y páginas.

- Múltiples tipos de contenido como MIME, mecanismos de encapsulación y compresión.

Estructura del servicio

La arquitectura Dienst se encuentra construida bajo la noción de servicios individuales que de manera conjunta crean una biblioteca digital distribuida, encontrándose los servicios y recursos en diferentes lugares.

Entre las características del servicio Dienst se encuentran:

- Almacenamiento.
- Acceso a los recursos.
- Posibilidad de agregar nuevos recursos.
- Descubrimiento y búsqueda de recursos.
- Control de usuarios.

Para proporcionar las características mencionadas Dienst se apoya de los siguientes módulos (Figura 2.5):

- Servicio de Repositorio para almacenar documentos digitales.
- Servicio de indexación.
- Servicio mediador de consultas que distribuye las consultas a los servidores de índices apropiados.
- Servicio de información sobre el estado de los servicios.
- Servicio de colecciones para informar sobre la forma lógica de las colecciones.
- Servicio de registro para mantener el control de usuarios.

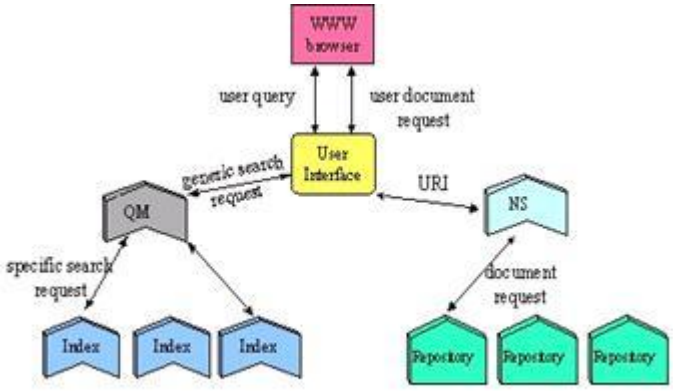

Figura 2.5: Servicios de Dienst

Colecciones

El servicio de colecciones es responsable de proveer la información que permite al conjunto de servicios interactuar dentro de la estructura de una biblioteca digital.

El servicio de colecciones provee la siguiente información:

- Como se encuentran organizadas las colecciones.
- La localización de los servicios de indexación dentro de la red.
- La meta-información de cada servidor de indexación.

2.4.3. Phronesis

En México se han hecho esfuerzos para la construcción de bibliotecas digitales, entre estos se encuentra Phronesis [34], desarrollado en el Instituto Tecnológico de Estudios Superiores de Monterrey.

Phronesis, palabra de origen griego que significa *Sabiduría*, es una herramienta para la creación de bibliotecas digitales distribuidas en Internet. Puede almacenar una gran cantidad de documentos digitales en diferentes formatos, así como realizar búsquedas en el texto completo de los documentos, en sus metadatos o en ambos al mismo tiempo.

Phronesis fue diseñado inicialmente para contener repositorios referentes al área de Ciencias de la Computación, sin embargo su uso ha sido extendido a otras áreas del conocimiento humano.

Una biblioteca Phronesis puede consistir de varios repositorios localizados en diferentes lugares del mundo, los usuarios pueden acceder a cualquier repositorio vía Web y desde ese repositorio tienen la capacidad de realizar búsquedas de expresiones en cualquier repositorio disponible (búsqueda distribuida). Estas búsquedas pueden ser booleanas o por relevancia.

Las búsquedas pueden ser realizadas de cuatro formas:

1. *Búsquedas booleanas en documentos* En estas búsquedas se muestran, en el mismo orden en el que se encuentran, aquellos documentos que cumplen con la expresión especificada.

2. *Búsquedas booleanas en metadatos* En estas búsquedas se muestran, en el mismo orden en el que se encuentran, aquellos documentos que sus metadatos cumplan con la expresión especificada.

3. *Búsquedas booleanas en documentos y metadatos* El resultado de esta búsqueda en la unión de *Búsquedas booleanas en documentos* y *Búsquedas booleanas en metadatos*.

4. *Búsquedas por relevancias en documentos* El resultado de esta búsqueda son aquellos documentos que contienen la expresión ordenados por la suma de coincidencias encontradas en los documentos.

Las búsquedas en Phronesis se encuentran auxiliadas con un *stemmer* en español, la cual es una herramienta que sirve para hacer búsquedas de palabras derivadas, como por ejemplo: beba, bebita, bebota.

Además Phronesis tiene opciones de indexamiento, por ejemplo puede evitar indexación de stopwords, que son palabras que se repiten en cada documento de la colección, y causan un índice demasiado grande y lento en su utilización.

Una vez encontrados los documentos por medio de búsquedas o navegando en los repositorios, es posible ver los metadatos del documento, guardar el documento original en el disco ó extraer el texto del mismo. Así mismo,

los usuarios de Phronesis disponen de una interfaz Web para el envío de documentos, los cuales viajan en su formato original.

Phronesis utiliza el sistema Managing Gigabytes (MG) [42] para la indexación y búsqueda en las colecciones digitales

Los metadatos, que es la información acerca de los documentos usados por Phronesis están basados en el estándar internacional Dublin Core definido por OCLC (On Line Computer Library Center).

Dentro de la administración de la biblioteca digital, existen tres tipos de usuarios en Phronesis:

- *Administradores* Son los encargados de asignar cuentas a usuarios, actualizar colecciones, eliminar documentos y de todo lo relacionado con la administración técnica del repositorio.
- *Usuarios Registrados* Son aquellos usuarios que pueden almacenar documentos dentro de la biblioteca digital.
- *Usuarios de búsquedas* Son todos aquellos usuarios que pueden realizar búsquedas dentro de la biblioteca digital.

Phronesis soporta estándares internacionales para comunicarse con otros servidores, entre estos estándares se encuentran:

- *Z39.50* Es un protocolo y estándar de comunicación del tipo cliente-servidor que define los procesos de búsquedas y resultados de bases de datos distribuidas. Este protocolo proporciona la capacidad de realizar operaciones de búsquedas en otras bibliotecas digitales independientes de la arquitectura de Phronesis, como la librería del Congreso, Laboratorios Bell, MIT y otras instituciones de prestigio internacional.
- *OAI (Open Archives Initiative)* OAI promueve la difusión de contenido científico entre múltiples entidades de comunicación. Permite realizar y responder a peticiones de bibliotecas digitales heterogéneas.

En resumen, el sistema Phronesis posee las siguientes características:

- Indexamiento y búsquedas en texto completo y/o en metadatos.

- Control de acceso de los usuarios que pueden contribuir material a la biblioteca.
- Soporte para búsquedas de documentos escritos en el idioma español e inglés.
- Interfaz de usuario basada en WWW en español e inglés.
- Búsquedas simultáneas en varias instancias de Phronesis.
- Soporte de búsquedas en documentos completos en varios formatos como lo son texto, PostScript, html, pdf y rtf.
- Soporte para almacenar cualquier tipo de documento y realizar búsquedas con base en sus metadatos.
- El software que constituye Phronesis es 100 % del dominio público
- Compresión eficiente de los documentos digitales.
- Stemmer en español.
- Comunicación a través de protocolos públicos (Z39.50, OAI)

2.4.4. UpLib

Uno de lo sistemas de bibliotecas digitales que se han desarrollado actualmente (2003), es UpLib [18], el cual consiste de una serie de repositorios de información indexados accesibles vía Web.

UpLib provee funciones de seguridad, organización, acceso y disponibilidad de documentos personales (como pueden ser cuentas de tarjetas de crédito, fotografías familiares, libros favoritos, cartas, diversos documentos en una gran variedad de formatos, páginas Web, etc.) desde Internet ocupando el protocolo Web, sin embargo, se tiene planeado como trabajo futuro integrarlo en dispositivos PDA.

UpLib realiza una clara distinción entre el tratamiento de los formatos texto e imagen, y se especializan en las imágenes argumentando que ellas son fácilmente transportables, sin perdidas de bits entre las diferentes plataformas.

Para el desarrollo de UpLib se consideraron las siguientes características:

- *Universalidad* Los documentos pueden tener cualquier tipo de origen y se mantienen en su formato original.
- *Disponibilidad* Los documentos tratados en UpLib, aun los mas grandes libros, pueden ser fácilmente encontrados, navegados y leídos con cualquier tipo de Navegador de Internet, y estos documentos se encuentran accesibles auque la librería personal no este corriendo.
- *Extensibilidad* Diversos tipos de funcionalidades pueden ser añadidas al manejo de nuevos tipos de búsquedas, conversiones, organizaciones o accesos.
- *Búsquedas* Los documentos pueden ser obtenidos usando combinaciones de búsqueda en texto completo y metadatos.
- *Escalabilidad* Cualquier cantidad de documentos pueden ingresar al sistema sin que este sufra alguna disminución en sus tiempos de respuesta.
- *Seguridad* Documentos tales como reportes médicos personales y registros financieros pueden ser almacenados seguramente.

El sistema UpLib está dividido en tres partes funcionales:

1. Captura.
2. Normalización de documentos y almacenamiento.
3. Búsqueda y uso de documentos.

Normalización y almacenamiento

UpLib hace referencia sobre lo inconveniente de los primeros sistemas de bibliotecas digitales, los cuales no permitían al usuario conocer donde se guardaban sus documentos, todo el mecanismo de almacenamiento era parecido a una caja negra para el usuario. UpLib consigue ir un paso adelante, permitiendo a los usuarios mantener sus archivos en una estructura de directorios creadas por ellos mismos, y ser accesible aunque el sistema Uplib se encuentre fuera de servicio.

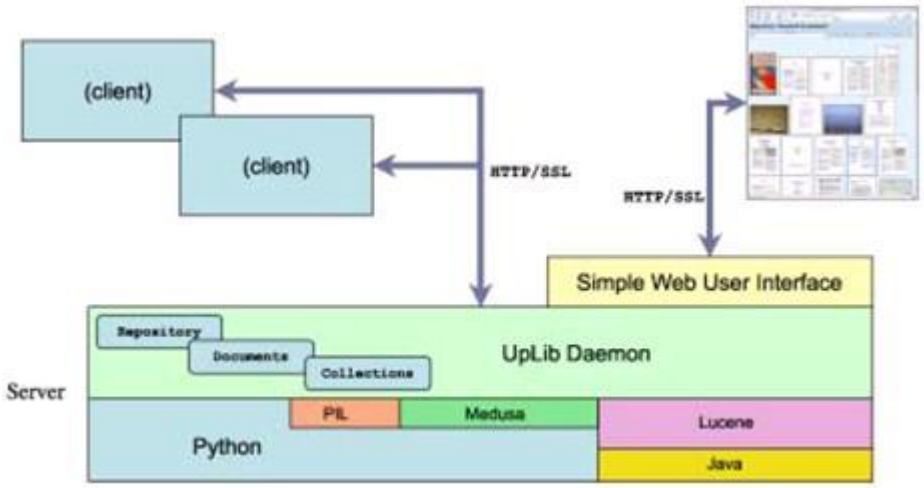

Figura 2.6: Arquitectura del sistema UpLib

UpLib (Figura 2.6) utiliza rippers (analizadores de texto) para obtener los metadatos de los documentos de forma automática. Los cuales corren secuencialmente hasta obtener el mejor grado posible de *fragmentación* del documento para facilitar las operaciones de búsqueda e indexamiento. Estos mismos se encargan de generar, como parte de los metadatos, la sumarización respectiva de cada documento.

Cada ripper puede operar de forma independiente o dependiente de los demás, teniendo una gran diversidad de funciones como: extracción de metadatos relevantes en investigación, extracción de urls de páginas web, extracciones de fechas de eventos, reconocimiento de rostros en fotografías familiares, clasificación de e-mails, etc. Como parte de la normalización particular cada usuario decide que rippers implementar sobre su información.

Búsqueda y uso de documentos

Una vez que los documentos han sido normalizados, pasan al repositorio de información. Los cuales podrán ser manejados vía Web a traves de conexiones seguras (SSL). La presentación de estos documentos almacenados en UpLib tiene como característica particular que muestra una imagen con el resumen del documento (thumbnail) extraído previamente en uno de los rippers. Estas presentaciones de los documentos comparten todas el mismo tamaño

y pueden arrastrarse en la pantalla. Otra de las formas de presentación de información de UpLib consiste en el uso de abstracts o resúmenes sobre una lista de los documentos encontrados.

UpLib realiza la búsqueda a través de pares de valores (nombre / valor) conocidos como términos, los cuales pueden ser independientes del texto completo del documento (metadatos). Y puede indexar cualquier cantidad de términos. Teniendo la capacidad de poder buscar solo en diferentes partes de la información almacenada. UpLib tiene funciones como búsquedas de frases, coincidencias de palabras completas, búsquedas por colocación de palabras, por rangos y combinaciones boleanas.

Una de las características importantes de las búsquedas en UpLib es que están pueden ser almacenadas como consultas persistentes y nombradas como colecciones. Cada vez que se abren están consultas almacenadas se vuelve a ejecutar la búsqueda y los documentos que coinciden son mostrados.

Arquitectura

UpLib ha sido construido basado en el modelo cliente/servidor. Con una interfaz Web HTML con conexiones seguras y un servidor back-end escrito en Phyton.

Los usuarios en UpLib tiene la capacidad de agregar rippers, o fragmentos de código que suman alguna funcionalidad de búsqueda sobre los datos, conocidos como acciones. Estas acciones son agregadas como módulos de Phyton, en el “actions-path”.

Las acciones operan sobre tres tipos de objetos: Repositorios, Documentos y Colecciones. Y los usuarios tienen la capacidad de crear subtipos de esos objetos y poder utilizarlos.

En general los documentos en UpLib son almacenados en el FileSystem, asi como los metadatos, los cuales se encuentran en archivos planos al lado de los documentos. Los metadatos acerca de los repositorios completos son almacenados en los directorios de mas altos nivel, estos en formatos XML.

Seguridad

Todas las comunicaciones en UpLib son realizadas de manera segura a través de la encriptación del protocolo SSL sobre HTTP. Y se usan directivas apropiadas sobre el caché de los clientes, para impedir que documentos se queden almacenados temporalmente sin conocimiento del usuario y queden expuestos a cualquier persona no autorizada.

Así mismo cada repositorio puede tener contraseñas asignadas de acceso individualmente.

2.5. Conclusión

En este capítulo se describieron los principales conceptos relacionados con las bibliotecas digitales, la importancia de sus componentes y su principal funcionamiento.

También revisamos algunos de los sistemas que han sido estandartes en el ámbito de las bibliotecas digitales, sin embargo, al intentarlos ocupar como la base del proyecto PDLib, para el manejo de documentos personales, se presentaron los siguientes inconvenientes:

- *Toda la información es almacenada en el sistema de archivos.* Almacenar toda la información en el sistema de archivos no permite la explotación de tecnologías como las bases de datos ni sus recientes características de búsqueda en texto completo.

  No permitir el uso de estas tecnologías limita la incorporación de mejoras sustanciales y el crecimiento continuo en la incorporación de formatos y características para el tratamiento de documentos personales.

- *No proporcionan seguridad entre información pública, compartida y privada.* A menudo las bibliotecas digitales públicas y personales, difieren en que para las primeras, los documentos son visibles a todos los usuarios de la misma, exponiendo su información a quien consulte los documentos, mientras que para las bibliotecas digitales personales, la seguridad en la información toma un papel relevante, ya que un documento confidencial nunca será publicado a todos los usuarios de

la biblioteca digital, sin embargo, si puede ser compartido a ciertos usuarios o grupos de usuarios, ya sea dentro de la propia biblioteca digital o fuera de esta.

- *No manejan metadatos definidos por el usuario.* En el caso de documentos personales, los metadatos representan una parte importante en la explotación de información, ya que describen el contenido de los documentos, sin embargo, algunas bibliotecas digitales no permiten el manejo de los documentos junto con sus metadatos, impidiendo un mejor aprovechamiento de la biblioteca digital.

- *Los cambios en los documentos no son reflejados inmediatamente.* Los documentos personales, se distinguen de otros documentos, particularmente porque su información cambia constantemente, estos cambios requieren ser integrados inmediatamente en la biblioteca digital, sin embargo, algunas bibliotecas digitales necesitan reindexar todos los documentos para integrar los cambios de un solo documento, por lo general este proceso es realizado en las noches, teniendo que esperar un día después la integración de los cambios en la biblioteca digital.

- *La arquitectura es pobremente escalable y adaptable a nuevos cambios.* Los formatos de los documentos personales varían constantemente de acuerdo a la tecnología, desde archivos de audio comprimidos, ejemplo mp3, hasta imágenes y videos; por esta razón la arquitectura de las bibliotecas requiere de escalabilidad inmediata tanto tecnológicamente como a nivel plataforma, y que no ofrezca resistencia a los cambios, permitiendo la incorporación de los nuevos formatos de los documentos a la biblioteca digital de manera rápida y efectiva.

- *No comparten información con otras bibliotecas digitales y no pueden navegar en bibliotecas digitales foráneas.* Debido al avance en las telecomunicaciones, la distribución de información entre bibliotecas digitales se ha convertido en una necesidad de alta prioridad, una biblioteca digital que no pueda navegar en bibliotecas digitales foráneas ó que no pueda compartir su información, se encuentra aislada y ofrece soluciones pobres a los usuarios que la consultan.

Estos inconvenientes serán resueltos con la propuesta de esta tesis, facilitando el trabajo colaborativo, con una arquitectura altamente escalable

y adaptable a los cambios, cuantificable, y un óptimo tratamiento en el manejo de documentos personales.

Capítulo 3

Manejo de información en Bibliotecas Digitales Personales

En este capítulo se describen diversos escenarios sobre el manejo de información en Bibliotecas Digitales Personales, la tecnología necesaria involucrada y un panorama general de los componentes del proyecto PDLib.

3.1. Escenario motivacional

3.1.1. Escenario 1

Miguel, originario de Oregon E.U., asiste a una conferencia en Nuevo León, México. Al terminar la conferencia Miguel solicita al expositor el material visto, para esto el expositor enciende su PDA y envía por mail la presentación a Miguel.

De regreso en Oregon, Miguel revisa su mail en su Desktop, encuentra la presentación enviada por el expositor y decide guardarla en su biblioteca digital personal, para esto arrastra el documento de su mail a un ícono del lado derecho de su reloj en su computadora.

Posteriormente Miguel decide asistir a otra conferencia en Tokio, Japón. Y durante el viaje va revisando sus documentos a través de su PDA, entre ellos la presentación enviada por el expositor.

Mientras asiste a la conferencia en Tokio se encuentra con su colega Jacqueline, y conversa sobre los documentos obtenidos en Nuevo León, México. Y decide compartirlos, para esto, Jacqueline accede desde su PDA a los documentos públicos de la biblioteca digital personal de Miguel.

Un tiempo después Jacqueline y Miguel son contactados por otro investigador residente en Europa, el cual se encuentra muy interesado al igual que ellos en el tema de la conferencia en Monterrey; este investigador realizó una búsqueda en diferentes repositorios OAI y se encontró con los metadatos del documento que tienen guardado en su biblioteca personal, de esta forma los contactó para solicitarles el documento y compartir ideas. Así los tres investigadores formaron un equipo de trabajo, al cual paulatinamente se van uniendo más investigadores de diferentes partes del mundo, el panorama completo de puede observar en la figura 3.1.

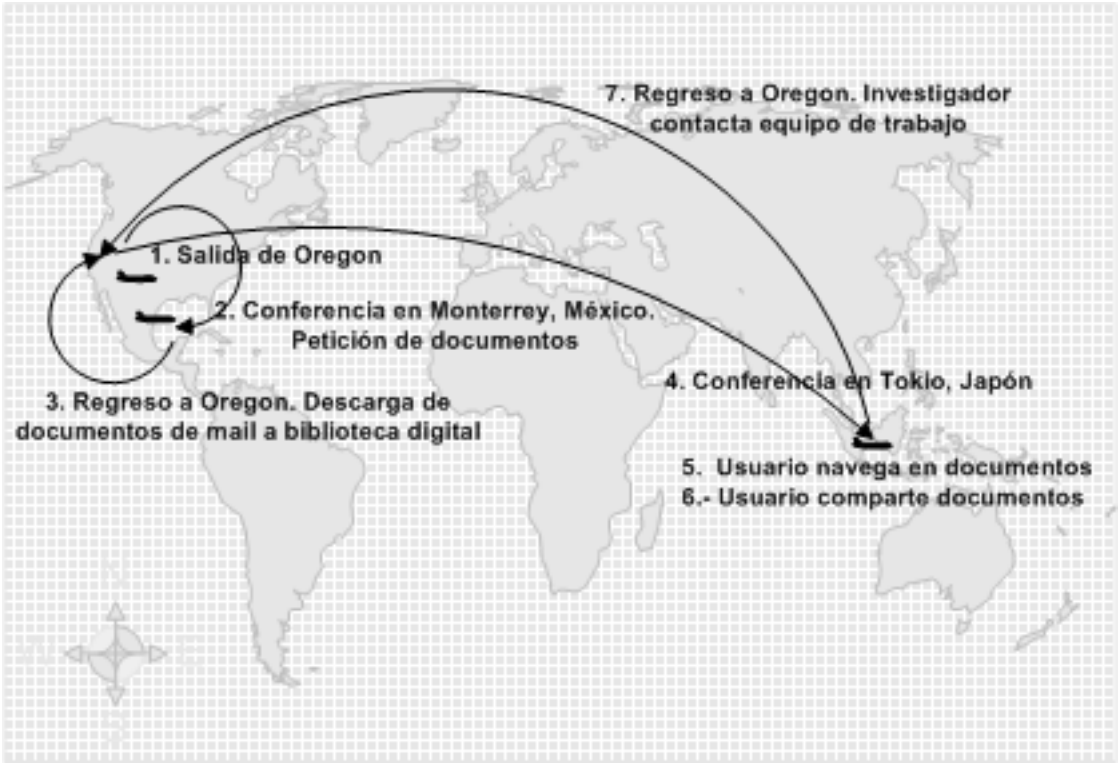

Figura 3.1: Escenario 1

3.1.2. Escenario 2

El director de una empresa en México, se encuentra atareado por la gran cantidad de documentos y facturas que recibe diariamente, tanto digitales como físicos. Por lo tanto decide escanearlos (en el caso de los documentos físicos) y guardarlos en su biblioteca digital personal.

Figura 3.2: Escenario 2

El director viaja a Estados Unidos, después a Francia, y recibe una llamada para confirmar la cantidad de productos de un lote que acaba de llegar, revisa desde la PDA su biblioteca digital personal y encuentra la factura de ese pedido y responde inmediatamente a la petición.

Posteriormente decide navegar desde su PDA entre las facturas recientes de su empresa y enviar algunas por correo electrónico a los miembros de la junta directiva.

De regreso en México, el director de la empresa enfrenta una auditoria, en

el cual le solicitan todo tipo de documentos de diversas fechas, simplemente realiza búsquedas a través de su biblioteca digital y encuentra todos los documentos solicitados. De esta forma mantiene una administración eficiente y puede responder inmediatamente a cualquier decisión. Las actividades se pueden observar en la figura 3.2.

3.1.3. Discusión de los escenarios

A continuación presentamos la tecnología necesaria para satisfacer los requerimientos de cada escenario.

Escenario 1

Para que Miguel reciba la presentación, se requiere que el expositor tenga su información almacenada en algún sitio manipulable desde su PDA, esto es, una biblioteca digital personal con funciones de envío de documentos a través de mail y de búsquedas de documentos en diferentes formatos.

Posteriormente Miguel recibe el documento en su computadora de escritorio, dicha computadora, requiere de un punto de acceso a su biblioteca personal del tipo *drag and drop*. Este punto de acceso estará al lado del reloj de la computadora y todos los documentos que reciba los guardará en la biblioteca digital personal del usuario, por lo que se requiere que esta se adaptable a nuevos puntos de acceso.

A su vez, para compartir documentos entre los usuarios de la biblioteca digital personal, requiere administración de permisos para decidir aquellos documentos que son públicos y privados.

También la biblioteca personal digital debe de poder comunicarse con otros sistemas completamente heterogéneos para compartir metadatos y documentos (interoperabilidad), y así tender líneas de comunicación entre personas interesadas en las mismas temáticas.

Los requerimientos mencionados en este escenario se pueden observar en la figura 3.3.

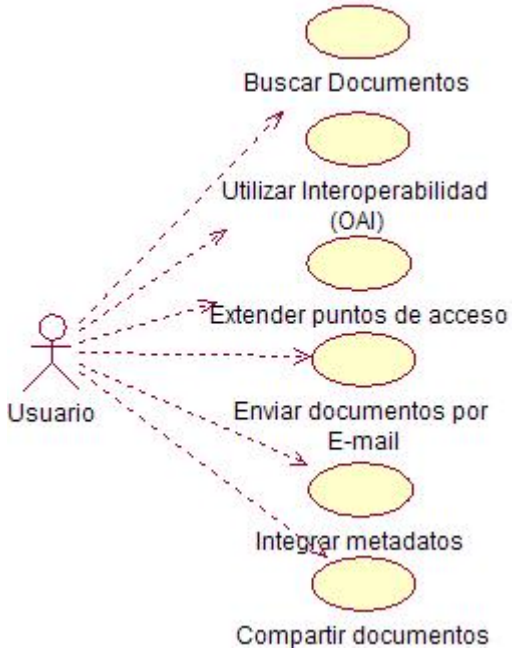

Figura 3.3: Requerimientos del caso 1

Escenario 2

La biblioteca digital personal ocupada por el director extrae la información textual de las facturas para indexarlas. De esta manera es posible realizar búsquedas en cualquier tipo de documento sin importar el formato del mismo.

Cuando el director se traslada a diferentes lugares, debido al retardo en las comunicaciones para enviar y recibir información, es necesario el menor tiempo posible de espera para mantener funcionales los dispositivos móviles, por lo tanto la información va siguiendo al usuario a donde este se dirija concentrándolo en el servidor de biblioteca digital más cercano.

Además de la funcionalidad de búsqueda es necesario que la biblioteca digital cuente con otras funcionalidades como envío de documentos por e-mail, los requerimientos se pueden observar en la figura 3.4.

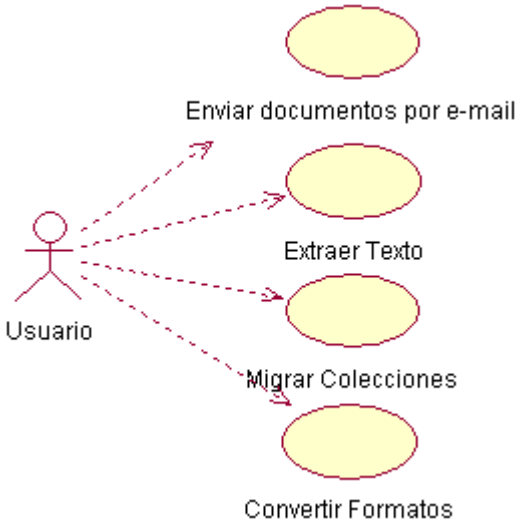

Figura 3.4: Requerimientos del caso 2

3.2. PDLib

Para satisfacer los requerimientos de los escenarios planteados anteriormente se necesita de una biblioteca digital personal con capacidades de comunicación con bibliotecas heterogéneas, indexamiento y búsquedas sobre documentos personales, soporte a la movilidad y adaptación al contexto, migración de colecciones, envío de documentos por mail y soporte offline.

El soporte de bibliotecas digitales en ambientes móviles, requiere de una infraestructura de base de datos, la cual es utilizada en el manejo de permisos y preferencias para los usuarios y para mantener la organización de colecciones y la jerarquía de la biblioteca digital en general. La base de datos mantendrá el orden de acceso dentro de PDLib, los envíos y transferencias de información realizados entre los dispositivos móviles y entre los mismos servidores PDLib. Así mismo permitirá definir los metadatos de las colecciones sin ningún tipo de restricción, dando la libertad al usuario de definir con sus propias palabras la información que describirá a las colecciones.

A su vez la base de datos estará operando junto con el indexador, encargado de realizar las búsquedas en las colecciones de datos y mantener un índice único. Este índice podrá ser enviado a otros servidores PDLib cuando

así se requiera, sin afectar la integridad de los datos.

Además de esto, el modelo de datos definido en esta tesis establecerá condiciones de comunicación entre los usuarios, controlando el acceso a la información mediante mecanismos de seguridad en colecciones y documentos, definiendo permisos y alcances, bases de datos, ámbito y esfuerzo computacional, garantizando la seguridad y confidencialidad de la información.

Todas estas características se encuentran cubiertas por el proyecto PDLib, y el soporte al manejo de información que realiza es el propósito de esta tesis.

A continuación describimos de manera general los componentes del proyecto PDLib, y en el siguiente capítulo profundizamos en la extracción y publicación de información.

3.2.1. Panorama General

PDLib se encuentra conformado por tres capas[21](figura 3.5):

- *Capa Cliente.* Esta capa incluye una gran variedad de dispositivos con los cuales los usuarios pueden interactuar con PDLib.
- *Capa Servidor.* Es capa contiene la infraestructura que provee servicio a los clientes. DataServer, Mobile Connection Middlware (MCM) y Web Front-end.
- *Capa de Interoperabilidad.* Incluye comunicación entre servidores PDLib y con otros servidores a través de OAI-PMH.

Los dos primeros módulos son transparentes a los usuarios, su función es proporcionar la funcionalidad requerida en los módulos de nivel superior a través de peticiones en formato XML-RPC. Pueden ubicarse en diferentes servidores, sin embargo, es requerido un puerto abierto que permita la interacción con los demás componentes.

Estos módulos manejan la información de las bibliotecas digitales de los usuarios proporcionando servicios como indexamiento, búsquedas, caché, visualización de contenidos, etc.

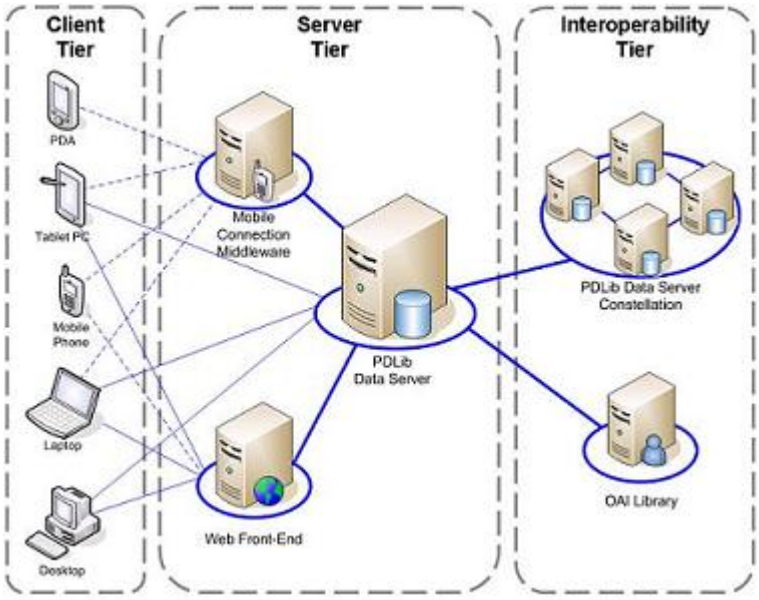

Figura 3.5: Sistema PDLib

Los dispositivos de la capa cliente se comunican con la capa servidor para acceder a los servicios de la biblioteca digital personal. El tipo de acceso de los dispositivos varía de acuerdo a las capacidades de los mismos. La arquitectura de PDLib refleja los siguientes tipos de accesos:

1. *Middleware Access.* Soporta los dispositivos móviles, especialmente aquellos que tienen recursos computacionales limitados (ejemplos PDAs).

2. *Web Access.* Provee acceso HTTP a cualquier dispositivo que incluya un navegador Web .

3. *Direct Access.* Ciertas aplicaciones con particulares requerimientos pueden acceder directamente al DataServer.

A continuación describimos los módulos principales de la arquitectura PDLib.

3.2.2. Clientes

A través de los clientes los usuarios acceden a los servicios de las bibliotecas digitales. De acuerdo al tipo de acceso los clientes se clasifican en[10]:

- *Clientes Móviles (mobile thick clients).* Los clientes móviles tienen que lidiar con las limitaciones del ambiente móvil. Por lo tanto usan los servicios de un middleware para realizar sus actividades.
- *Clientes Web (fixed and mobile thin clients).* Esta categoría incluye a todos aquellos dispositivos que cuentan con un navegador Web capaz de desplegar páginas HTML ó WML.
- *Aplicaciones Clientes.* Las aplicaciones clientes son todas aquellas que se ejecutan en servidores, computadoras de escritorio ó laptops prácticamente sin limitaciones de recursos, ejemplo el MCM (Middleware Conexion Mobile), que será visto a detalle mas adelante.

Para desempeñar sus actividades los clientes móviles realizan las siguientes funciones:

- *Mecanismo de almacenamiento local.* Almacenan los documentos en el dispositivo móvil para revisiones offline.
- *Mecanismo de Adaptación a la Conexión.* Este mecanismo proporciona un tiempo de reacción constante a pesar de la variabilidad de la conexión. Logrando ser calculado por el tamaño de la ventana de transferencia y prediciendo el estado de la red.
- *Soporte para la interacción del usuario.* Proporciona una interfaz gráfica que permite a los usuarios manejar el contenido de su biblioteca digital personal.

A continuación describiremos dos de los clientes de PDLib: Cliente Móvil y Cliente Web.

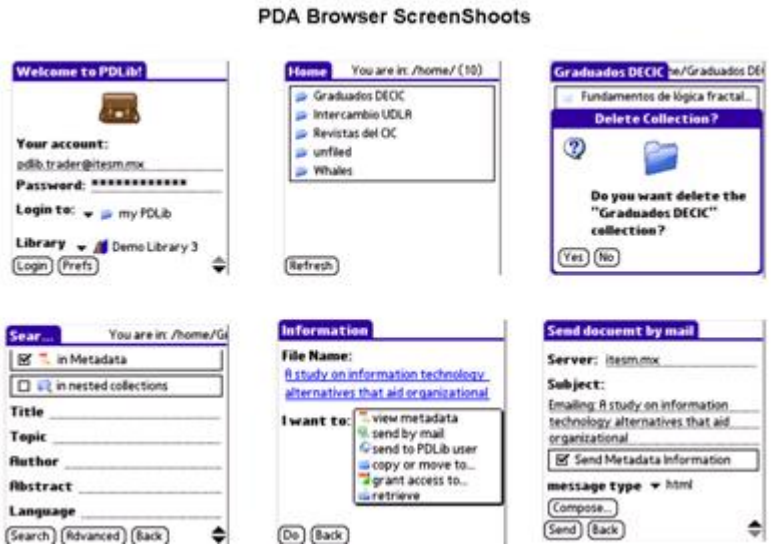

Figura 3.6: Cliente móvil.

Cliente Móvil

El cliente móvil es el módulo más cercano al usuario móvil, esta conformado de pantallas y menús que permiten la navegación de la biblioteca digital a través de PDAs.

Posee características importantes para los dispositivos móviles que poseen una cantidad de memoria limitada, entre ellas lo reducido de su tamaño y su portabilidad, permitiendo su funcionalidad en una gran variedad de dispositivos.

Para los usuarios finales, el cliente móvil representa una interfaz sencilla e intuitiva, fácil de manipular y con características como visualización de información en espacios reducidos (figura 3.6).

Las características destacables con que cuenta el cliente móvil son:

- *Facilidad de uso.* Debido a que su operación es desde un dispositivo móvil, la interfaz se ha simplificado para acceder fácilmente a la funcionalidad de la biblioteca digital.
- *Arquitectura enfocada a objetos.* Con la arquitectura orientada a objetos del cliente móvil, se pueden mejorar las funcionalidad que ofrece

sin alterar la interfaz de los objetos, generando un entorno escalable.

- *Multiplataforma.* Con su fundamento en Java (J2ME)[24] permite al cliente ejecutarse en los diversos sistemas operativos de los dispositivos móviles.
- *Administración de colecciones.* Desde el cliente móvil se pueden ejecutar operaciones sobre las colecciones, como por ejemplo altas, bajas.
- *Administración de documentos.* Los documentos también son administrados desde el dispositivo móvil, se pueden borrar, dar de alta ó revisar su contenido.
- *Envío de documentos por E-mail.* Como parte de las prestaciones de la biblioteca digital, el cliente móvil permite enviar por e-mail los documentos a otros usuarios.
- *Búsquedas.* También se pueden realizar búsquedas, y navegar sobre los documentos devueltos.
- *Navegación en repositorios locales y foráneos.* De igual forma es posible navegar tanto en los repositorios locales como foráneos a través de OAI.

Cliente Web

El Cliente Web es el punto de acceso Web a PDLib, esta diseñado para aquellos dispositivos que poseen navegadores de Internet, y puede entregar contenido WML ó HTML según el tipo de dispositivo que realiza la petición. El Cliente Web ha sido desarrollado utilizando la tecnología Apache Tomcat[30].

La funcionalidad del cliente móvil es mantenida como base en el cliente Web, sin embargo, dado que esta creado en un ambiente Web (figura 3.7), la facilidad de uso permite realizar operaciones eficientemente y más rápidas que en el cliente móvil.

El cliente Web mantiene la base de la movilidad de PDLib, ya que los usuarios pueden ingresar desde cualquier sitio a sus archivos.

Entre las características destacables del cliente Web se encuentran:

Figura 3.7: Cliente Web

- Navegación de bibliotecas locales y foráneas.
- Búsquedas por documentos y metadatos (figura 3.8).
- Administración de documentos.
- Administración de colecciones.
- Visualización de metadatos y de contenido.
- Envío de E-mail con documentos.

3.2.3. MCM (Mobile Conexion MiddleWare)

Debido a que el servidor de datos no ha sido diseñado para trabajar con dispositivos móviles directamente, el MCM (Mobile Conexion MiddleWare) es el módulo intermediario entre el servidor de datos y los clientes móviles ligeros (PDAs, celulares)[10].

Los dispositivos móviles, por sus características como ancho de banda limitado, poca duración de la pila; requieren de una carga de trabajo ligera y una interfaz adecuada para que los dispositivos móviles puedan interactuar con el servidor de datos.

Figura 3.8: Cliente Web. Navegación en documentos

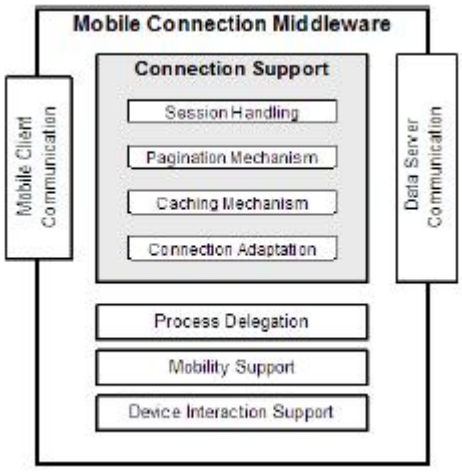

Figura 3.9: MCM Arquitectura

La arquitectura del MCM esta conformada por los siguiente componentes [10], como se puede observar en la figura 3.9:

- *Soporte a sesiones.* Los dispositivos en movimiento, pueden sufrir desconexiones repentinas por cambio de redes o celdas; estas conexiones interrumpen el flujo de la comunicación que se este realizando en ese momento. Sin un soporte de sesiones, cuando la conexión es reiniciada todo el trabajo previamente realizado queda anulado y el usuario tiene que empezar el proceso nuevamente, el módulo de soporte a sesiones garantiza que no exista pérdida de información y que los recursos del dispositivo se utilicen efectivamente.

- *Caché.* En ocasiones los dispositivos móviles no pueden procesar completamente los resultados devueltos por el DataServer, por lo tanto estos se procesan paulatinamente, sin embargo, el MCM no vuelve a realizar la misma consulta, en lugar de esto, guarda los resultados temporalmente en un espacio de memoria denominado Caché, el cual es reutilizado hasta que se completa la operación inicial.

- *Paginación.* Las peticiones realizadas por los usuarios pueden involucrar resultados que los dispositivos no pueden procesar completamente, por ejemplo un documento conformado por 1000 páginas; en estos casos se recurre a la paginación realizada dentro del MCM, la cual fragmenta el documento y devuelve los resultados de acuerdo a las solicitudes que recibe.

- *Soporte a la conexión.* Es requerido por los clientes móviles para adaptar la variabilidad del ancho de banda y enfrentar las desconexiones frecuentes.

- *Delegación de procesos.* Ejecuta funciones que podrían demandar una excesiva cantidad de recursos computacionales en los dispositivos móviles.

- *Soporte a la movilidad.* Ciertas operaciones como *prefetching* pueden llevarse a cabo para mejorar el desempeño en la navegación de documentos, así como almacenarlos en caché más cerca del usuario.

- *Soporte a la interacción del dispositivo.* Realiza adaptación de contenido de acuerdo a las características del dispositivo.

El MCM se encuentra en constante comunicación con el servidor de datos, cuya funcionalidad describimos en la siguiente sección.

3.2.4. DataServer

La parte central de PDLib es el servidor de datos (DataServer), en el se realiza el almacenamiento y tratamiento de los datos. El DataServer provee la siguiente funcionalidad:

- *Biblioteca digital personal.* El DataServer realiza operaciones de creación, navegación, búsquedas, actualización y borrado sobre los objetos almacenados en la biblioteca digital personal (Colecciones, documentos y metadatos). También provee servicios como realizar copias de documentos, envío de documentos por mail, conversión de formatos y otros.
- *Almacenamiento.* El DataServer provee la infraestructura necesaria para la conservación de los documentos de los usuarios. Dichos documentos se encuentran físicamente almacenados en la base de datos de PDLib, y con un herramienta IR (Information Retrieval) provee rápidas referencias a los mismos en las búsquedas. El manejo de esta información constituye la parte fundamental de esta tesis y será ampliamente descrito en el siguiente capítulo.
- *Comunicaciones.* Provee comunicación con otros DataServers, para facilitar el manejo global de los datos móviles [22], y favorecer a la migración de colecciones, enviando los documentos de los usuarios al servidor mas cercano de éste.
- *Soporte OAI-PMH.* A través de OAI-PMH, PDLib muestra los metadatos de los documentos almacenados y permite navegación en otros servidores OAI-PMH.

La arquitectura del DataServer permite adaptaciones y modificaciones en sus diversos componentes, sin afectar la funcionalidad de otros módulos, se divide en tres grandes grupos: manejo de datos internos (ejemplo: manejo de usuarios, colecciones, etc.), extensión a servicios internos de PDLib (ejemplo: XML-RPC), extensión a servicios externos (ejemplo: OAI).

Manejo de datos Internos

El manejo de datos internos se encuentra dividido en diferentes funcionalidades:

Usuarios. Los usuarios son los principales actores de PDLib, se encargan de proporcionar las órdenes para el manejo de la información. Suben y bajan archivos desde cualquier dispositivo con acceso a PDLib, comparten su información con otros usuarios, visualizan documentos, navegan en librerías foráneas. El DataServer proporciona estas funcionalidades requeridas para los usuarios.

Colecciones. Las colecciones mantienen la separación lógica de los diversos documentos que se encuentran en las bibliotecas digitales. En PDLib mediante las colecciones es posible definir permisos de accesos y documentos relacionados dentro de una biblioteca digital, así como la asignación de metadatos los cuales serán heredados por los documentos contenidos dentro de esta.

Documentos. Los documentos representan la información mínima almacenada en las bibliotecas digitales, en PDLib los documentos son indexados, tanto en contenido como en metadatos, se realizan operaciones sobre ellos como extracciones de texto, compresión, búsquedas, envío y recepción. Los documentos pueden estar en diferentes formatos, entre los formatos soportados por PDLib se encuentran pdfs, docs, xsl, richtext, ascii, PostScript.

Indexamiento y búsquedas. El indexamiento consiste en la formación de directorios de palabras y frases con punteros en el documento de origen, esta técnica permite optimizar el tiempo de búsqueda en las bibliotecas digitales. El indexamiento es realizado sobre el contenido de los documentos y sobre los metadatos de los mismos, actualmente existen diversos motores de indexamiento, en PDLib, el indexamiento y la búsqueda son realizados a través de Yakarta Lucene [31]. La búsqueda dentro de PDLib es obtenida con mecanismos de relevancias de palabras en los documentos.

Extensión a servicios internos de PDLib

Para proporcionar servicios a los demás módulos, el DataServer contiene en esta capa la funcionalidad requerida para el transporte de los datos, de

esta manera cada objeto de datos dentro de PDLib es serializado, comprimido y transportado a clientes que lo soliciten.

El protocolo de comunicación utilizado para la prestación de los servicios internos de PDLib es XML-RPC, debido a que este protocolo permite manipular a nivel de tramas la prestación del servicio, eliminando el transporte de información no necesaria como sucede con otros protocolos.

Cada módulo tiene definido un puerto específico para permitir la comunicación entre las diferentes entidades, dicho puerto es colocado a través de los archivos de configuración de cada módulo.

Extensión a servicios externos de PDLib

Una parte importante del DataServer lo constituyen el soporte a los servicios externos de PDLib, esta funcionalidad provee de las herramientas necesarias para la comunicación con otros servidores, así también para la navegación en bibliotecas foráneas bajo protocolos estándares como AOI.

De esta forma es posible compartir datos entre diferentes instituciones completamente independientes, y entre bibliotecas digitales con arquitecturas heterogéneas. Se desciribirá ampliamente la implementación de OAI en el siguiente capítulo, así como las prestaciones que brinda al DataServer.

3.2.5. Modelo de datos.

El modelo de datos es la base del proyecto PDLib, en él se encuentran plasmados los objetos de datos y las reglas existentes entre ellos (como se vió en el capítulo 2), actualmente este modelo de datos se encuentra implementado en MySQL [37], sin embargo es adaptable a cualquier gestor de base de datos con soporte a búsquedas de texto completo.

El modelo conceptual simplificado de datos entidad-relación [21] es mostrado en la figura 3.10, para conformar la base de la biblioteca digital personal se establece que la entidad *library* puede contener a una o más entidades *collection*, las cuales tienen un conjunto de metadatos asociado (*Metadata Set*), y estas a su vez contienen *documents* los cuales están asociados a *Document Metadata*, donde están definidos los conjuntos de metadatos de los documentos.

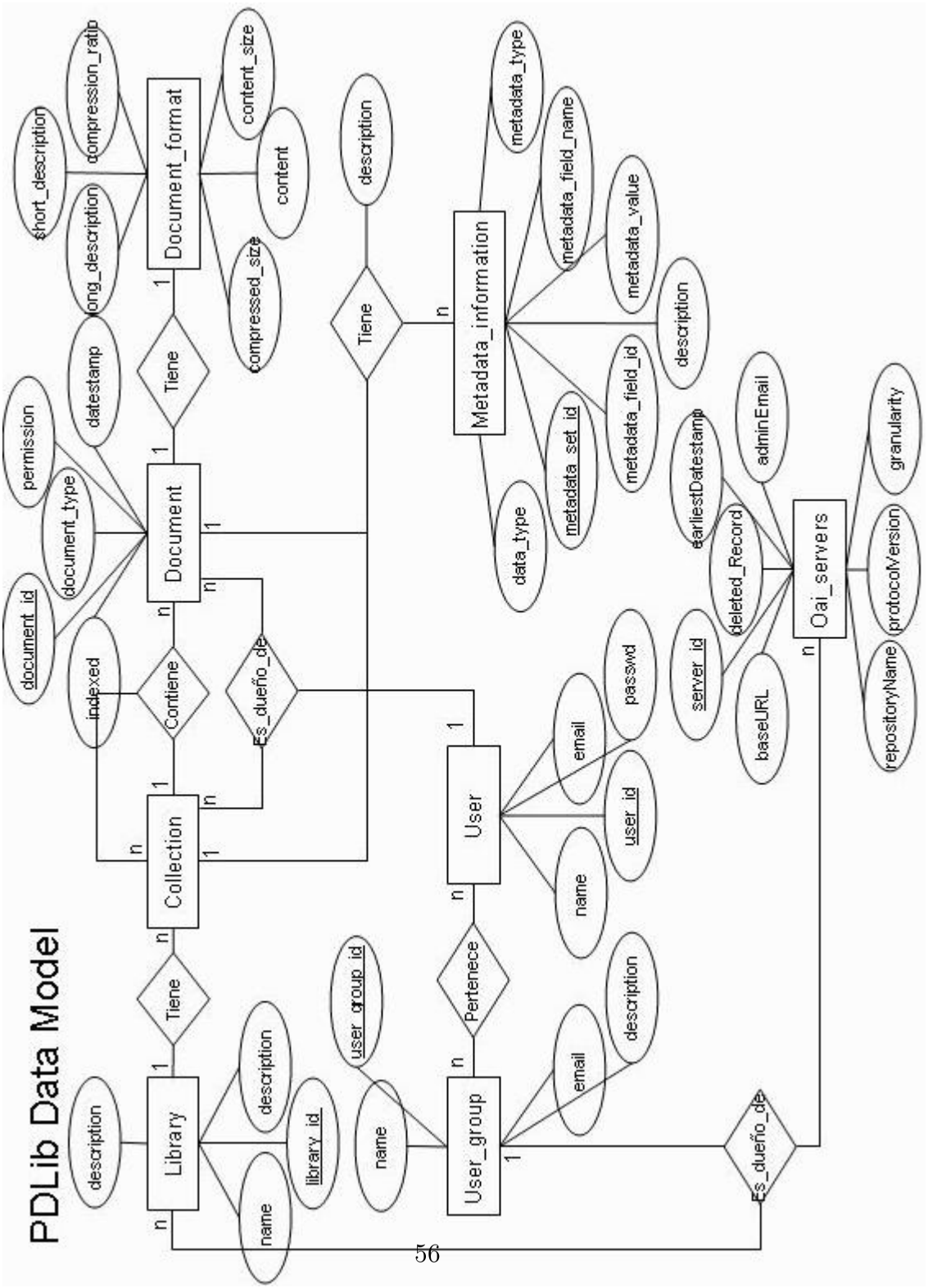

Figura 3.10: Modelo de datos simplificado

La seguridad de los documentos personales se encuentra basada en la entidad *Permission*, la cuál tiene asociados los permisos sobre las colecciones y documentos, los cuales pueden ser:

1. *Personal.* El acceso es restringido exclusivamente al dueño de la biblioteca.
2. *Compartido.* Otros usuarios tienen permitido crear documentos y colecciones.
3. *Compartido Público.* Otros usuarios tienen permitido navegar en las colecciones y hacer búsquedas sobre los documentos.
4. *Mantenimiento.* Otros usuarios tienen permitido actualizar los documentos y sus metadatos asociados.
5. *Público.* Otros usuarios tienen completo acceso sin restricciones a las colecciones, los documentos y sus metadatos asociados.

Cada usuario (*User*) pertenece a un grupo (*User_group*) y cada grupo tiene asociado una serie de servidores OAI (*OAI_Servers).* A su vez, los usuarios son dueños de colecciones y documentos.

Este modelo de datos (figura 3.10) mantiene una simplicidad en su diseño, esta simplicidad permite que la biblioteca digital sea adaptable a nuevos cambios o requerimientos, y que pueda ser escalable para ofrecer nuevas prestaciones.

A través de este modelo es posible cubrir los requerimientos de la biblioteca digital para el manejo de documentos personales.

3.3. Conclusión

En este capítulo se describió una visión global de PDLib, los escenarios de pruebas describieron los alcances y las limitaciones de la biblioteca digital personal, así como los requerimientos técnicos asociados al tratamiento de documentos personales.

También se revisaron de manera general los principales componentes de PDLib, los clientes móviles y los clientes Web conforman las principales

herramientas de interacción con el usuario, también describimos el modulo de adaptación a la conectividad para clientes móviles, y profundizamos en el tratamiento de los datos mantenidos por el DataServer, quien es el principal componente de PDLib para el tratamiento de los documentos personales. Revisamos el manejo interno de la información que se genera entre los componentes de PDLib y la comunicación XML-RPC establecida entre ellos, y el tratamiento externo de la información a través del protocolo OAI, permitiendo al DataServer compartir documentos entre bibliotecas digitales con arquitecturas heterogéneas, así como navegar en repositorios foráneos de información.

En el siguiente capítulo profundizaremos en el tratamiento interno de la información proporcionada por el DataServer a través de un módulo altamente escalable y adaptable a cambios llamado DSAPI, así como el manejo externo de la información mediante el protocolo OAI.

Capítulo 4

Data Storage API (DSAPI) e Interoperabilidad (OAI)

El manejo de información dentro de una biblioteca digital constituye la parte medular de su funcionamiento, este manejo de información es realizado tanto internamente como externamente. Internamente realiza operaciones de borrado, alta y actualizaciones de información, mientras que externamente comparte información con otras bibliotecas digitales. El manejo de datos internos en PDLib es desarrollado por el DataServer.

El DataServer es un módulo muy complejo, en su interior se realiza la administración de usuarios, el almacenamiento y gestión de las búsquedas, así como muchas otras operaciones vistas en el capítulo anterior, respondiendo a las peticiones del MCM, así como las de los clientes Web y móviles.

La gestión de búsquedas representa una carga de trabajo importante para el DataServer y en general para cualquier biblioteca digital, estas herramientas de búsquedas a menudo trabajan como cajas negras las cuales no tienen sus componentes claramente identificados, las actualizaciones sobre estos componentes son escasas o nulas, y no ofrecen métricas para cuantificar la utilización de recursos dentro del motor de búsqueda [41]. Debido a esto en PDLib se diseñó un módulo escalable y adaptable a cambios y modificaciones de manera rápida y consistente, en el cuál es posible identificar claramente cada una de sus partes y realizar mediciones sobre estos, este módulo dentro del DataServer es llamado DSAPI (Data Storage Application Programming

Interface).

La estrategia para el manejo de información utilizada por el DSAPI es el acoplamiento de una base de datos con una herramienta de indexación para manejar el almacenamiento, indexamiento, actualizaciones y búsquedas sobre los documentos de los clientes PDLib.

La herramienta de indexación mantiene una estructura de índice de la información textual de los documentos, mientras que en la base de datos son almacenados los metadatos de los documentos (información sobre los documentos) y los mismos documentos, una consulta puede llevar a la utilización de ambas herramientas; de esta forma las partes de las consultas que tienen que ver con búsquedas en texto completo, son evaluadas por la herramienta de indexación, mientras que otras consultas que tienen que ver con los metadatos de los documentos son evaluadas por la base de datos, siendo coordinado el trabajo de ambas herramientas por el DSAPI para devolver un único resultado.

Por otra parte, el tratamiento externo de información es controlado a través del módulo OAI del DataServer, el cual permite la obtención de los metadatos de colecciones y documentos foráneos, así como la publicación de la información almacenada en él.

Abordaremos primeramente el tema del manejo interno de la información y posteriormente el manejo externo de ésta información.

4.1. Arquitectura del DSAPI

El DSAPI es el módulo encargado del manejo de datos internos en la biblioteca digital, se encuentra conformado por dos partes: DBAPI(DataBase API), maneja la interacción con las bases de datos e IRAPI(IRTool API), soporta la interacción con la herramienta de indexación.

En la figura 4.1 se puede obervar que estos módulos funcionan independientemente uno del otro, manteniendo cada uno su propio espacio de trabajo y la organización de la información que manejan; y el DSAPI se encarga de la coordinación de estos módulos para responder al flujo del manejo de información dentro del DataServer.

A continuación describimos a detalle estos módulos.

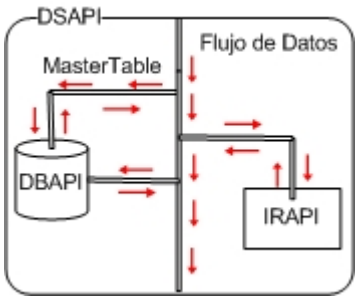

Figura 4.1: Arquitectura del DSAPI

4.1.1. DBAPI (DataBase Application Programming Interface)

El DBAPI es el componente software encargado de la interacción con las bases de datos, su diseño se encuentra basado en el L-DBMS[21] (Sistema manejador de base de datos local). El sistema L-DBMS es un componente atómico de una arquitectura mayor de un sistema de bases de datos distribuidas(D-DBS), y se puede visualizar como la implementación de un API de acceso a datos de una base de datos local, como se puede observar en la figura 4.2.

Figura 4.2: L-DBMS como una API de acceso a una base de datos

En la figura 4.2 un usuario (**Usuario**) accede de forma remota a un conjunto de funciones (**API**) implementadas por el **L-DBMS**. El L-DBMS implementa esas funciones de acuerdo a la interfaz de la base de datos local. Donde la interfaz de la base de datos local esta constituida por su soporte de SQL[7][21].

De acuerdo al diseño basado en el L-DBMS, cuando el DBAPI recibe una petición, traduce esta petición en términos que la base de datos (predeterminada por el DataServer en ese momento) pueda entender de acuerdo a su interfaz SQL, la consulta es procesada por la base de datos y el resultado es devuelto al DBAPI quien reestructura la respuesta adecuandola

a su interfaz. Ahora el resultado es envíado al DSAPI quien recibe la respuesta y puede realizar alguna otra operación con estos resultados (si la consulta involucra al IRAPI) ó devolverla al solicitante.

El DBAPI sigue un diseño orientado a objetos a través de una serie de clases descritas a continuación.

Diseño del DBAPI

El DBAPI se encuentra formado por un conjunto de clases[1] encargadas de la interacción con la base de datos. Estas clases se dividen en dos grupos: Las que realizan operaciones sobre el esquema de la base de datos y las que realizan operaciones sobre los datos.

- ***Operaciones sobre el esquema.*** Este grupo de clases modifican ó alteran la estructura de las tablas de una base de datos. Entre estas clases tenemos:

  AlterTable. La clase AlterTable es utilizada para actualizar las definiciones de las tablas de la base de datos.

  CreateTable. La clase CreateTable es utilizada para crear nuevas tablas en la base de datos.

  DBColumn. La clase DBColumn es utilizada para representar las columnas de las tablas de la base de datos, su principal función es obtener y modificar los atributos de las columnas de las tablas.

  DBMetaData. La clase DBMetaData está encargada de proporcionar información de la base de datos.

  DBTable. La clase DBTable es una de las clases fundamentales del DBAPI, representa a una tabla de la base de datos, proporcionando operaciones para leer ó escribir los atributos de las tablas.

  DropTable. La clase DropTable es una de las mas sencillas dentro del DBAPI, encargada de borrar las tablas de la base de datos.

  DBConnection. Esta clase implementa un mecanismo responsable de crear los objetos a partir de sus clases de implementación y proporcionar acceso a los mismos.

[1]Para mayor información de estas clases y ejemplos véase apéndice C: Código Fuente DSAPI

SQLSchemaOperation. Esta es una clase abstracta y constituye la interfaz de las funciones AlterTable, CreateTable y DropTable vistas anteriormente.

- ***Operaciones sobre los datos.*** Este grupo de clases modifican, insertan ó borran los datos de los usuarios. Son a menudo las clases que soportan mayor carga de trabajo dentro del DBAPI, debido a su continúo uso.

  A continuación describimos las clases que soportan las operaciones sobre los datos:

  Insert. Esta clase es encargada de realizar las inserciones en las tablas de la base de datos. Mantiene internamente dos vectores, uno de nombres de columnas y otro de valores.

  Delete. Esta es encargada de eliminar información de las tablas. Mantiene internamente 3 vectores: whereColumns, whereOperators y whereValues que contienen las columnas involucradas en las condiciones para el borrado de información, los operadores de estas condiciones y los valores utilizados.

  Update. La clase Update es utilizada para la actualización de información. Mantiene 5 vectores para el manejo de información: columns, values, whereColumns, whereOperators y whereValues, donde se guardan los nombres de las columnas, los nuevos valores de las columnas, las columnas involucradas en las condiciones para la actualización de información, los operadores de estas condiciones y los valores utilizados.

  Select. La clase Select esta encargada de la formación de consultas a la base de datos.

  QueryResult. Esta clase es responsable del almacenamiento de los resultados de las consultas realizadas a la base de datos, es el objeto devuelto tras realizar una consulta con la clase Select.

  SQLOperation. Esta clase representa un interfaz para las clases de Insert, Delete, Select y Update. El tipo de operación esta determinado dentro de la clase por una varible que puede tomar los siguiente valores (cuadro 4.1):

ID	Operación
0	SELECT
1	INSERT
2	DELETE
3	UPDATE

Cuadro 4.1: SQLOperation. Tipos de Operaciones

Escenarios

Para tener una visión global de como las clases del DBAPI interactúan entre sí, exponemos a continuación una serie de escenarios.

Escenario 1: Agregar documentos y sus metadatos a la biblioteca digital a través del DBAPI. Grace requiere indexar en su biblioteca digital personal, una de las calificaciones digitalizadas del semestre actual, para su tratamiento como un documento personal.

La información que guardará asociada al documento (esto es metadatos) consiste de su calificación, el número de aciertos y errores obtenidos, esto, debido a que Grace quiere llevar un control estadístico de su avance académico. Sus notas consisten de una calificación de 9, 8 aciertos y 2 errores.

A continuación describimos el proceso efectuado en la biblioteca digital para guardar su documento digitalizado (Proceso Insert):

Proceso Insert
1. Insert insert = dbc.getInsert("Grace_Academico");
2. insert.value("C", 9);
3. insert.value("A", 8);
4. insert.value("E", 2);
5. insert.value("D", "Documento Digitalizado");
6. insert.execute();

Este proceso muestra la inserción de un documento en el espacio de trabajo de la biblioteca digital llamda *Grace_Academico*, manejado exclusivamente por el DBAPI en este escenario. Los valores numéricos 9, 8 y 2 son insertados en los campos C (Calificación), A (Aciertos) y E (Errores). Y el documento

digitalizado en D (Documento).

Los campos numéricos y el documento digital son almacenados en la base de datos, debido a la exclusividad de este escenario sobre el DBAPI.

En la figura 4.3 se puede ver el diagrama de secuencia correspondiente a al proceso de inserción de este documento. Observamos que originalmente tenemos un objeto DBC de la clase DBConnection que funciona como una fábrica de objetos de las operaciones sobre la base de datos.

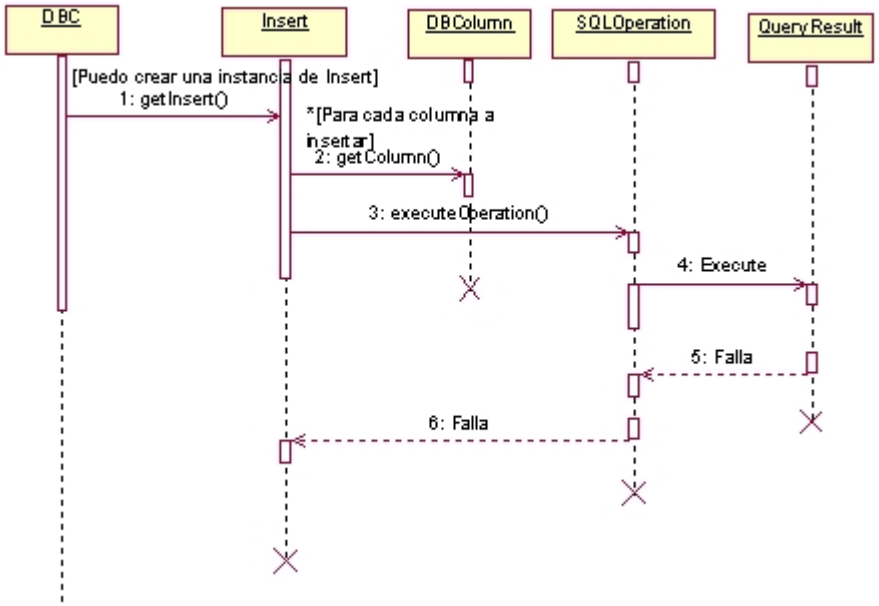

Figura 4.3: Diagrama de secuencia de inserción en el DBAPI

En el paso numero 1 de la figura observamos la llamada a la función getInsert() la cual crea una instancia de la clase Insert basada en el espacio de trabajo *Grace_Academico*(como se puede ver en la línea 1 del código fuente Insert). En el paso número 2 por cada valor que se insertará se crea un objeto DBColumn el cual mantiene el valor asociado con el nombre del campo respectivo. En el paso número 3 se manda a ejecutar la operación de inserción la cual genera una sentencia SQL, esta cadena es enviada al objeto SQLOperation encargado de ejecutar la sentencia. Y por último se guarda el resultado de la operación en la clase QueryResult.

Escenario 2: Recuperación de documentos. Un tiempo después cuando Grace quiere recuperar su documento, genera la siguiente consulta en el DBAPI.

Codigo fuente Select

```
1. Select select = dbc.getSelect("Grace_Academico");
2. select.column("C");
3. select.column("A");
4. select.column("E");
5. select.column("D");
6. select.where("C", "=", "9");
7. select.where("A", "=", "8");
```

El proceso *Select* abre el espacio de trabajo *Grace_Academico* y recupera sus calificaciones (C), aciertos (A), errores (E) y el documento (D), de aquellos donde tuvo una calificación de 9 y 8 aciertos.

En la figura 4.4 se puede ver el diagrama de secuencia correspondiente a la selección sobre esa tabla.

Donde en el paso 1 existe un objeto de la clase DBConnection al cual se le ejecuta un método getSelect() para obtener el espacio de trabajo *Grace_Academico*. En el paso 2 por cada columna a obtener y por cada condición se forma un objeto DBColumn, en el paso 3 se manda a ejecutar la operación enviando una sentencia SQL al objeto SQLOperation quien a su vez manda a ejecutar directamente en la base de datos la sentencia, guardando el resultado en un objeto de la clase QueryResult.

El DBAPI es una interfaz de funciones sobre las bases de datos, la comunicación existente entre el DBAPI y las bases de datos es a través del estándar ANSI SQL, de esta forma cualquier base de datos que sea capaz de interpretar comandos SQL y responder a estos, puede tener al DBAPI como una interfaz de acceso.

Utilizando al DBAPI para el acceso a bases de datos se consiguen sistemas capaces de utilizar diversas bases de datos sin sufrir variaciones en su interfaz,

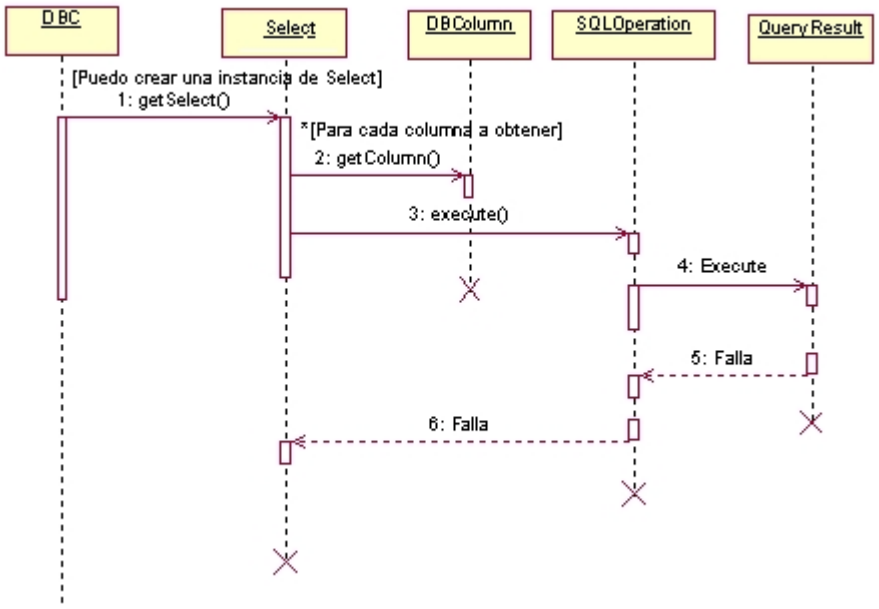

Figura 4.4: Diagrama de secuencia select en el DBAPI

lo que conlleva a la reutilización de código fuente (no es necesario crear nuevas interfaces) y a sistemas escalables (es posible cambiar de base de datos por ejemplo cuando aumente la demanda del manejo de información).

Otra de las herramientas utilizadas en el diseño del DSAPI es el IRAPI, encargado de la interacción con las herramientas de indexación. Su funcionalidad es muy parecida al DBAPI pero trasladando su manejo a las herramientas de indexación. El IRAPI es el siguiente tema.

4.1.2. IRAPI (Information Retrieval Application Programming Interface)

El IRAPI es un componente software encargado de la interacción con las herramientas de indexación (IRTools), esta diseñado para trabajar con diferentes herramientas de indexación; sin embargo como las herramientas de indexación no tienen un lenguaje común para el manejo de información (como es el caso del SQL en el DBAPI), el soporte a éstas no es transparente. Para

enfrentar esto, el IRAPI posee una serie de funciones comunes predefinidas de todas las herramientas de indexación, estableciendo puntos específicos de adaptación para otras IRTools.

Estas funciones comunes de las herramientas de indexación son explicadas a continuación, en el diseño del IRAPI.

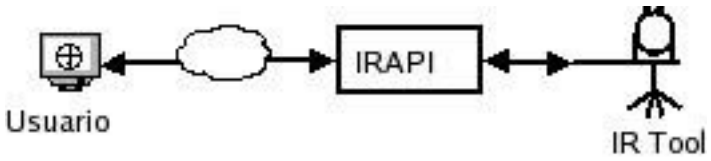

Figura 4.5: IRAPI como una API de acceso a herramientas de indexación

Diseño del IRAPI

Un usuario puede interactuar directamente con el IRAPI para acceder a los servicios de la herramienta de indexación, tal como se muestra en la figura 4.5. Entre estos servicios los usuarios pueden agregar o borrar documentos de los índices, crear y borrar índices y realizar búsquedas sobre ellos, como se puede observar en la figura 4.6.

Debido a que el IRAPI trabaja como una interfaz para las herramientas de indexación, se puede cambiar la IRTool de la figura 4.5 por otra, sin modificar las peticiones y respuestas del usuario de la misma figura.

El IRAPI es un paquete consistente en una sola clase, como se muestra en la figura C.17. Esta clase posee diversas funciones implementadas comunes a todos los indexadores, de tal forma que es posible cambiar a una IRTool por otra, reemplazando específicamente la interfaz de acceso a la IRTool en cada una de las funciones del IRAPI.

La funciones[2] que forman al IRAPI son las siguientes:

- ***newindex.*** Esta función tiene como objetivo la creación de un nuevo índice en la IRTool. Recibe como parámetro una cadena de texto representando el nombre del nuevo índice.

[2]Una guía mas detallada con ejemplos se puede encontrar en el Anexo C: Código Fuente DSAPI

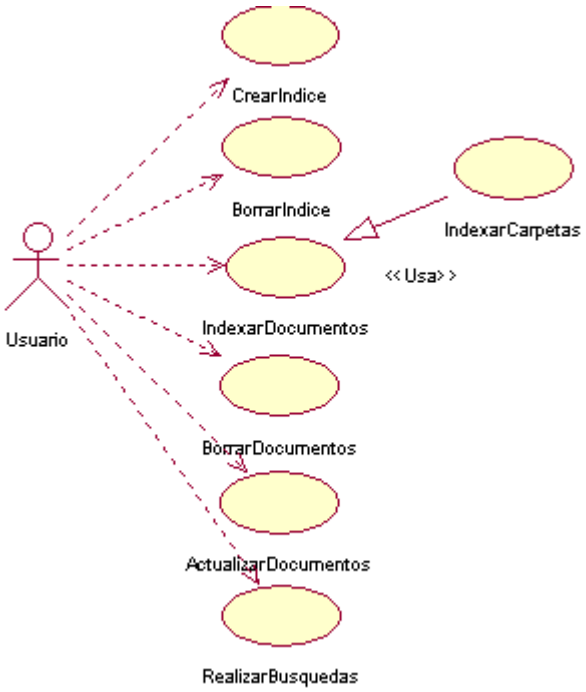

Figura 4.6: IRAPI casos de uso

Este índice es creado en el directorio donde se encuentra el IRAPI, en el caso de que el índice ya exista devuelve un mensaje de error.

- ***index.*** Esta función se encarga del indexamiento de directorios o archivos de manera recursiva. Este indexamiento es realizado exclusivamente sobre información de tipo textual. Siendo el índice creado en el FileSystem.

- ***deleteDocument.*** Esta función se encarga de eliminar documentos del índice.

  Recibe como parámetro una cadena indicando la dirección del índice y una cadena de texto conteniendo las palabras claves a eliminar.

  Eliminará todos aquellos documentos que coincidan con las palabras claves, por lo general, estas palabras claves se refieren al identificador del documento.

- ***search.*** La función Search recibe como parámetro la ubicación del índice sobre el cual se realizará la operación de búsqueda, y un par de vectores conteniendo los metadatos del documento con un valor asociado.

  Los valores de los vectores son tratados como búsquedas lógicas de tipo AND, de manera que un documento deberá de coincidir con todos los criterios de búsqueda.

En el paquete IRAPI recae la interacción con las herramientas de indexación, y como se puede observar, la sencillez en su diseño constituye una parte fundamental de su funcionamiento.

A continuación describiremos el escenario de indexación por parte del IRAPI.

Escenarios

Escenario 3. Indexación. Luis es un coleccionista de libros digitales, es dueño de una gran cantidad de estos, sin embargo, le es difícil buscar alguna reseña en algún libro digital, debido precisamente a la gran cantidad de material que posee. Para solucionar esto recurre al IRAPI.

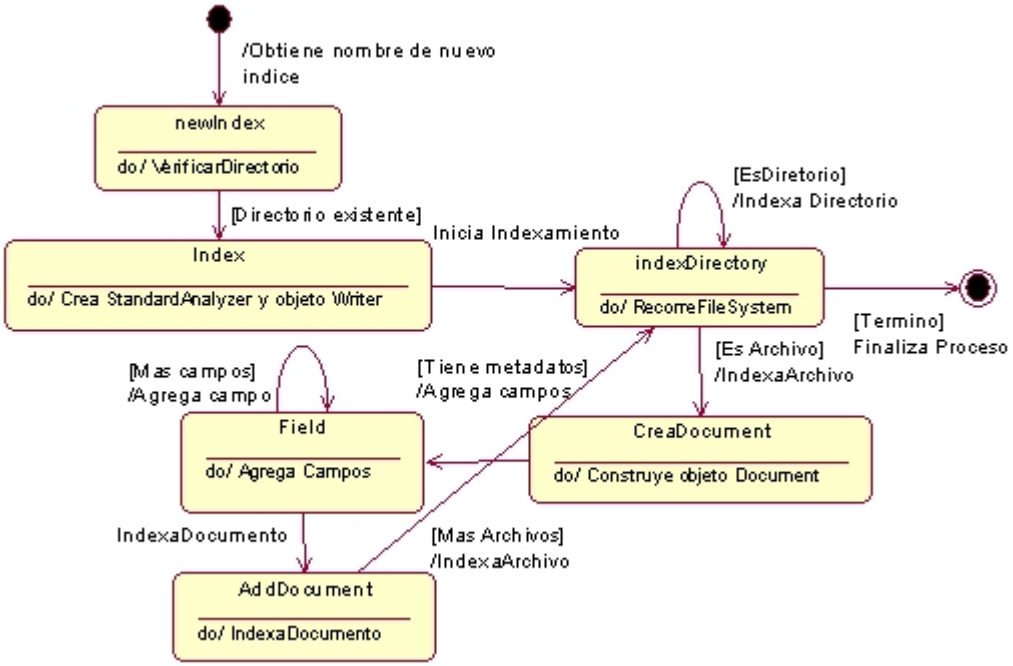

Figura 4.7: IRAPI. Diagrama de estados para el indexamiento.

Para la creación de un índice (figura 4.7) se requiere conocer el nombre y la ubicación donde estará el nuevo índice, y de los libros digitales de Luis.

Despues de verificar que el directorio de archivos a indexar se encuentre existente en el sistema de archivos, crea los objetos responsables de la indexación e inicia el indexamiento, como se observa en la figura 4.7.

Si es un directorio se itera recursivamente hasta obtener los documentos en caso de que estos existieran, y sea agregan propiedades a los documentos conocidas como campos o metadatos, entre estos campos se encuentran el path, la fecha de indexamiento etc.

Una característica destacable del IRAPI es que siempre mantiene una copia idéntica de todos los campos de información pasados en la indexación en un solo campo llamado **all**. Esto con el fin de simplificar las búsquedas en texto completo, ya que evita la identificación de cada campo llamando simplemente a uno en las búsquedas de texto completo:**all**.

Una vez que se han recorrido todos los documentos se finaliza el proceso de indexamiento. Y se genera como resultado un índice en el sistema de archivos de los documentos indicados, el cuál puede ser utilizada en la gestión de búsquedas por Luis y encontrar rápidamente cualquier reseña de un libro.

Entre las partes importantes a destacar del IRAPI consiste en la definición de funciones que permiten el indexamiento incremental[3] sobre un documento o conjunto de documentos (Esta característica solo esta disponible en aquellos indexadores con soporte al indexamiento incremental).

A continuación veremos el funcionamiento del DSAPI, la interacción de los componentes DBAPI e IRAPI y el flujo de trabajo necesario para desarrollar sus funciones.

[3]Consiste en la actualización del índice documento por documento sin necesidad de reindexar todos los documentos nuevamente, los doucumentos actualizados estan disponibles inmediatamente una vez acabado el proceso de indexamiento incremental.

4.2. Funcionamiento del DSAPI

La estrategia del DSAPI consiste en el acoplamiento y coordinación de una base de datos con una herramienta de indexación para gestionar el manejo de información dentro de la biblioteca digital.

El DBAPI contiene la funcionalidad para el acceso a la base de datos, mientras que el IRAPI contiene la funcionalidad para realizar las operaciones del indexador, como se vio en la sección anterior.

Las porciones de las consultas que tienen que ver con búsquedas en texto completo son evaluadas por la herramienta de indexación, mientras que las consultas que tienen que ver con campos de la base de datos son evaluadas por el DBAPI, la coordinación de ambos módulos es llevada a cabo por el DSAPI, quien entrega un único resultado al solicitante.

Para la coordinación de ambos componentes (DBAPI e IRAPI) el DSAPI se apoya primeramente en una tabla llamada Master, como se puede ver en la figura 4.8.

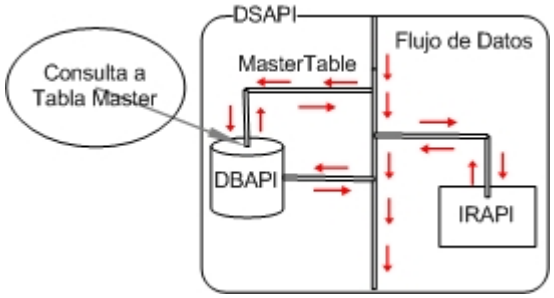

Figura 4.8: DSAPI. Consulta a tabla Master.

4.2.1. Tabla Master

Esta tabla consiste en el punto de partida de las operaciones del DSAPI, apoyándose en una tabla de la base de datos llamada *master*, la cual contiene descritas las estructuras internas para el manejo de información de ambos módulos. En la tabla 4.2 se observa la estructura de la tabla master la cual consta de 5 campos que describen al nombre de la tabla, el nombre del campo,

su tipo de dato y si es un campo primario ó de tipo índice[4].

Field	Type	Description
tablename	varchar	Nombre de la tabla
fieldname	varchar	Nombre del campo
dtype	varchar	Tipo de dato (Integer, Double, Text, Varchar, Boolean)
isprimary	varchar	Señala dato de tipo primario
isindex	varchar	Señala dato en el indexador

Cuadro 4.2: Tabla Master

Un ejemplo del contenido de la tabla master se puede observar en la figura 4.9. Donde se observa la estructura del espacio de trabajo T1, el cuál contiene 4 campos; los tres primeros numéricos y el último de tipo textual. Donde los dos primeros (C1, C2) son llaves primarias en la base de datos y en el indexador, el tercero (C3) solamente se encuentra guardado en la base de datos, mientras que en el último (campo C4) su contenido se encuentra indexado en la IRTool.

La tabla *master* ayuda a identificar aquellos atributos que se encuentran en la base de datos, en el indexador ó en ambos, brindando un mapa para el DSAPI. Debido a su constante utilización la información de la tabla *master* es leída una sola vez de la base de datos (por la clase IRMasterTable) y utilizada durante todo el tiempo de vida de la aplicación.

tablename	fieldname	dtype	isprimary	isindex
T1	C1	INTEGER	Yes	Yes
T1	C2	INTEGER	Yes	Yes
T1	C3	INTEGER	No	No
T1	C4	VARCHAR(255)	No	Yes

Figura 4.9: Contenido de la tabla master

Si el DSAPI, a través de la tabla master, determina que la operación de información involucra al IRAPI, inicia la ejecución del módulo, con lo cual,

[4]Un campo primario se encuentra contenido tanto en la base de datos como en la IRTool, mientras que un campo de tipo índice mantiene su índice en el indexador y su contenido en la base de datos

el control de la ejecución del DSAPI queda en espera de la respuesta del IRAPI.

4.2.2. IRAPI

En la figura 4.10 se observa que el control de la ejecución del DSAPI es transferido al IRAPI. Esto sucede en aquellos casos en los que la tabla master informa que el campo no es llave primaria y se encuentra indexado. En el ejemplo de la tabla master de la figura 4.9 el campo C4 del espacio de trabajo T1 su valor del campo isprimary es No, y el valor del campo isindex es Yes, lo cual indica que este campo es controlado por el IRAPI.

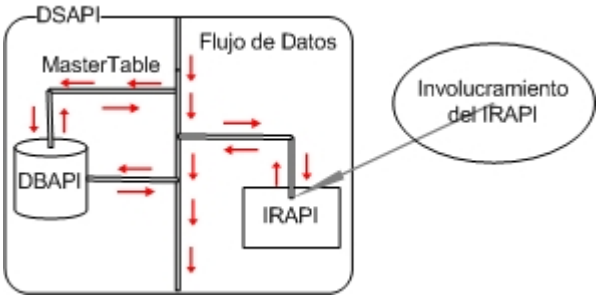

Figura 4.10: Involucramiento del IRAPI.

El camino hacia el IRAPI es opcional, determinado por la tabla master, sin embargo, si entra en acción, la ejecución del DSAPI permanece en espera del retorno de las llaves primarias de los documentos que coincidieron con los criterios de la operación de información del IRAPI.

Estas llaves primarias se utilizarán para que el DSAPI forme el criterio de la operación sobre el DBAPI, el cual se comenta a continuación.

4.2.3. DBAPI

Cuando el control de la operación es transferido al DBAPI, el DSAPI queda en espera del retorno de los documentos, como se puede ver en la figura 4.11

El DBAPI tiene un doble trabajo en este punto:

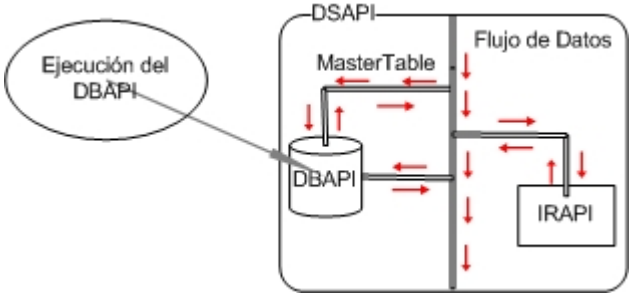

Figura 4.11: Involucramiento del IRAPI.

- Realizar las consultas necesarias a la base de datos que involucren al conjunto de resultados devueltos por el IRAPI.
- Regresar los documentos almacenados en la base de datos.

Las consultas que realiza el DBAPI sobre la base de datos pueden ser una o varias, dependiendo del tipo de operación, estas consultas son determinadas por el DSAPI. Al final de su trabajo el DBAPI retorna los documentos que coincidieron con los criterios de operación del usuario, transfiriendo este conjunto de documentos resultantes al DSAPI.

A manera de resumen del funcionamiento del DSAPI , vemos la estrategia global general ilustrada en la figura 4.12.

En la figura 4.12 se puede observar que el DSAPI inicialmente evalúa la consulta a través de la tabla Master y si encuentra que alguna operación o consulta involucra al IRAPI, ejecuta la parte correspondiente al IRAPI, el resultado devuelto por el IRAPI es ahora utilizado para formar las consultas necesarias en la base de datos (DBAPI) con la información filtrada por el IRAPI, mientras tanto el DSAPI se encuentra esperando a que termine la ejecución del DBAPI, una vez finalizado el resultado devuelto por el DBAPI es enviado al usuario.

Estos dos componentes de software (DBAPI e IRAPI) constituyen el DSAPI, y forman una arquitectura con las siguientes características:

- ***Adaptable a los cambios.*** Es posible reemplazar alguno de los

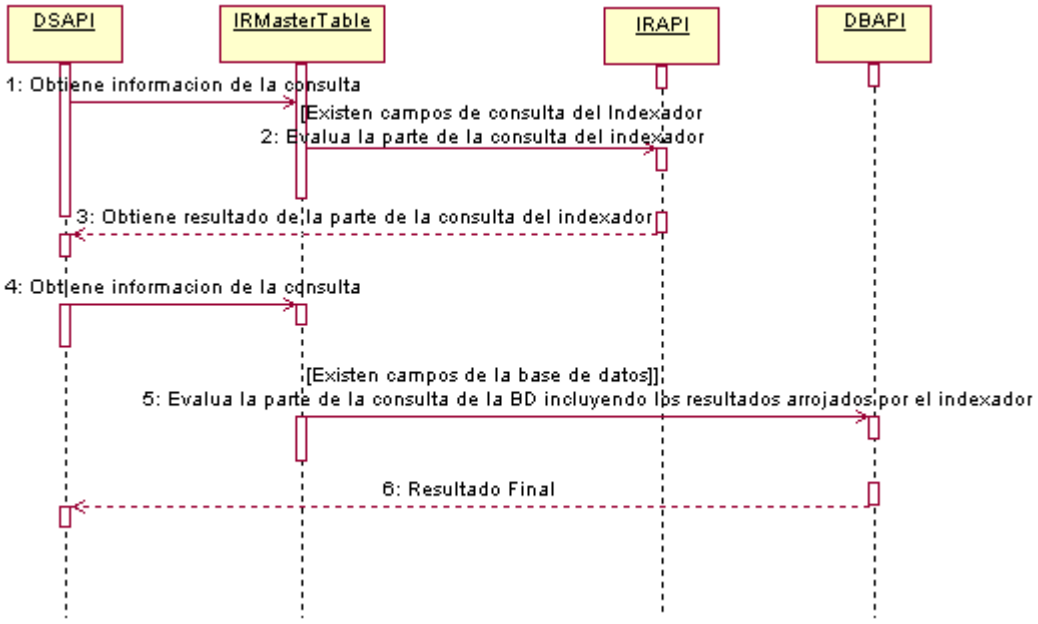

Figura 4.12: Estrategia General del DSAPI

componentes (según sea el tipo de propósito al que sea sometido) sin que esto afecte el funcionamiento global ni la interfaz pública del sistema.

- ***Escalable.*** Es capaz de soportar mayores demandas en el manejo de información, a través de una actualización de los componentes DBAPI e IRAPI.
- ***Transparente.*** Es posible determinar e identificar el trabajo que desarrollan cada una de sus partes.
- ***Medible.*** Se puede cuantificar el costo de las operaciones del sistema, ofreciendo información sobre los procesos que intervienen en el DSAPI.

A continuación se plantean una serie de ejemplos con algunas de las operaciones que realiza el DSAPI.

4.2.4. Ejemplos

Los ejemplos mostrados a continuación describen el comportamiento del DSAPI en las operaciones de manipulación de información a los que es sometido. Iniciaremos con la descripción de la operación básica del DSAPI, CREATE TABLE, quién es la encargada de crear el espacio de trabajo en los indexadores y en la base de datos, para posteriormente mostrar las inserciones (INSERT), búsquedas (SELECT), actualizaciones (UPDATE) y borrado de información (DELETE) entre otras funcionalidades del DSAPI.

1. *CREATE TABLE.* Esta operación se usa en el DataServer para crear el esquema (espacio de trabajo) tanto de la base de datos como del indexador. DSAPI reconoce la sentencia y después de crear el espacio de trabajo en la base de datos, crea un índice vacío en la IRTool.

   Un ejemplo de *CREATE TABLE* utilizando el DSAPI es el ejemplo referido a continuación, que forma el espacio de trabajo mostrado en la figura 4.13:

   Ejemplo CREATE TABLE
   CreateTable ct = dbc.getCreateTable(“T1”);
   ct.addColumnAsKey(“C1”, “INTEGER”);

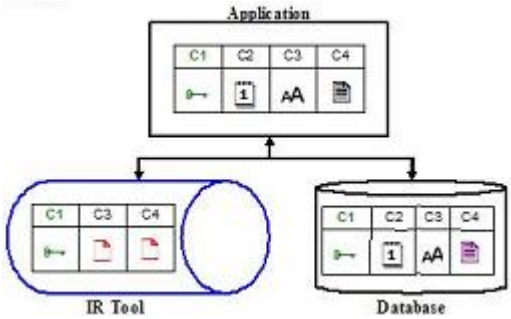

Figura 4.13: DSAPI CREATETABLE

ct.addColumn("C2", "DATE");
ct.addIRColumn("C3", "VARCHAR(255)");
ct.addIRColumn("C4", "TEXT");
ct.execute();

Donde el espacio de trabajo *T1* es creado en la base de datos, y en el indexador es creado un índice, posteriormente se agregan las llaves primarias con la función *addColumnAsKey*, estas son creadas tanto en la base de datos como en el indexador y nos permite identificar y relacionar los elementos de ambos módulos. Los campos que se guardan únicamente en la base de datos están identificados por la función *addColumn*; para aquellos campos en los cuales su contenido se guarda en la base de datos y el índice de este contenido es administrado por el indexador se identifican por la función *addIRColumn*. En la figura (figura 4.13) podemos identificar que el campo *C1* se encuentra tanto en la base como en el indexador debido a su condición de llave primaria (en la IRTool su contenido no se encuentra indexado, solo sirve para identificar al documento), el campo *C2* se encuentra exclusivamente en la base de datos, el contenido de los campos *C3,C4* se encuentra en la base de datos y sus índices son manejados por el indexador, de esta forma cualquier consulta referente a estos campos *C3,C4* causará la activación del IRAPI.

2. *INSERT.* Para realizar inserciones el DSAPI se ayuda de la tabla *master* y de acuerdo a su contenido insertará en la base de datos y/o

creará índices. Un ejemplo de la inserción basado en la figura 4.13 es el siguiente:

Ejemplo INSERT
Insert insert = dbc.getInsert("T1");
insert.value("C1", 1);
insert.value("C2", 2);
insert.value("C3", "Metadatos del documento");
insert.value("C4", "Contenido del documento");

En este ejemplo (INSERT) se insertará en el espacio de trabajo *T1* el valor 1 de *C1* como llave primaria, tanto en la base de datos como en el índice. El valor 2 de *C2* solamente se inserta en la base de datos, mientras los valores "Metadatos del documento" de *C3* y "Contenido del documento" de *C4* son insertados en la base de datos e indexados a través del IRAPI.

3. *SELECT.* En la selección de información se pueden dar los siguientes casos:

   - *DBAPI.* Solo se requiere la asistencia de la base de datos en aquellas consultas donde se busque a través de llaves primarias o de campos que no están indexados.
   - *IRAPI/DBAPI.* Se requiere de la asistencia de ambos componentes cuando los criterios de selección coinciden en campos que se encuentran indexados.
   - *IRAPI.* Este es una caso especial, donde la configuración del módulo depende exclusivamente del IRAPI y no esta presente el DBAPI, todas las consultas son administradas por el IRAPI y no se cuenta con una tabla master.

   En el siguiente ejemplo participan IRAPI/DBAPI y se encuentra basado en la figura 4.13:

   Ejemplo SELECT

Select select = dbc.getSelect("T1");
select.column("C1");
select.column("C4");
select.where("C4", "=", "contenido");

En este ejemplo (SELECT) se piden las columnas *C1, C4* de aquellos documentos en cuyo campo C4 se encuentre la palabra *"contenido"*. Este campo como vemos en la figura 4.13 se encuentra administrado por el indexador, recurriendo al IRAPI para la obtención de los resultados.

4. *JOIN.* El Join consiste de la ejecución de varios SELECTs en los cuales se cumplen las condiciones vistas en los mismos.

   Un ejemplo del código de JOIN es el siguiente:

   Ejemplo JOIN
   select = dbc.getSelect("T1");
   select.from("T2");
   select.column("T1.C1");
   select.column("T2.F4");
   select.join("T1.C1", "T2.F1");
   select.join("T1.C2", "T2.F2");

   En este ejemplo (JOIN) se realiza un Join entre las tablas *T1, T2* donde los campos *T1.C1, T1.C2* coincidan con *T2.F1, T2.F2* respectivamente.

5. *DELETE.* El DELETE consiste primero en la ejecución de una selección, se obtienen las llaves primarias de los documentos y posteriormente se borra la información tanto en el DBAPI como en el IRAPI (si existe información referenciada por la tabla master), como el caso del espacio de trabajo T1 de la figura 4.13 cuyo ejemplo de describe a continuación.

   Un ejemplo del código del DELETE es el siguiente:

   Ejemplo DELETE
   Delete delete = dbc.getDelete("T1");

delete.where("C1", "=", 9);
delete.whereMatch("C4", "+" + "¨" + "prueba" + "¨", Delete.IN_BOOLEAN_MODE);
delete.execute();

En este ejemplo (DELETE) se eliminan los documentos tanto del IRAPI como del DBAPI cuyo campo primario C1 sea 9 y el campo C4 contenga la palabra "prueba".

6. *UPDATE.* El UPDATE al igual que el DELETE realiza primeramente una selección y posteriormente actualiza tanto en el DBAPI, como en el IRAPI. Si el campo a actualizar es exclusivo de la base de datos, el IRAPI no realiza ninguna acción de actualización.

Ejemplo UPDATE
Update update = dbc.getUpdate("T1");
update.value("C3", 3);
update.value("C4", "nuevo contenido");
update.where("C1", "=", 1);

En este ejemplo (UPDATE) se actualizan dentro del espacio de trabajo T1 los documentos cuya llave primaria contenga 1, y tomando como referencia a la figura 4.13 esta actualización afecta tanto al DBAPI como al IRAPI.

Una vez descritos los ejemplos del funcionamiento del DSAPI, aplicaremos estos ejemplos en una evaluación en la que veremos la cuantificación y evaluación de sus características.

A continuación mostramos la descripción del ambiente de pruebas donde se utilizó al DSAPI.

4.3. Pruebas del DSAPI

Las pruebas desarrolladas sobre el DSAPI permiten la verificación de los márgenes de operación de ésta herramienta y su comparación con otras.

Además de esto, permiten la valorización de la herramienta DSAPI para expresar claramente sus limitaciones y alcances en el indexamiento de información, respuesta a las consultas y tamaño de índices.

A continuación mostramos un breve descripción de la evaluación al que es sometido el DSAPI.

4.3.1. Descripción de la prueba

Estas pruebas fueron realizadas en orden incremental, primero con un único archivo de texto y posteriormente fueron aumentando hasta llegar a un total de 1047 archivos, ocupando 2 GB de espacio en disco duro. Además, se desarrolló un scanner de páginas Web para representar lo más fielmente posible la gran variedad de documentos que serán manejados por PDLib. Este scanner toma como alimentación un diccionario de palabras en español de OpenOffice y realiza una búsqueda de cada palabra en el buscador Web Google [13], itera en toda la lista y va extrayendo el texto limpio de marcas HTML sobre cada referencia encontrada. Esta información se almacena en el filesystem en forma de archivos con formato textual. Inicialmente las pruebas comenzaron con dos archivos de 99 bytes cada uno, posteriormente se utilizó la versión española de *Don Quijote* la cual consiste en 5 archivos con un total de 2.1 MB, posteriormente se aumentó la complejidad con la coleción de documentos de la *Biblia* la cual consta de 8 elementos y tiene un total de 3.9 MB y por ultimo se ocupó la información de la *Text REtrieval Conference Extendido* [39] junto con los archivos del scanner de Internet los cuales consistían en total de 1047 elementos, contabilizando 1.9 GB.

A continuación describimos los recursos sobre los cuales fueron ejecutadas estas pruebas.

Sistema Operativo

Esta prueba fue realizada bajo el sistema operativo Linux con las siguientes características:

- Distribución Fedora Core 3.
- kernel 2.6.9-1.667
- Arquitectura i686.

Hardware

El Hardware utilizado cuenta con las siguientes características:

- 1 Procesador Xeon 3.02 Ghz.
- 1 GB RAM.
- 176 GB de espacio en Disco Duro.

Una descripción detalla de estas pruebas puede encontrarse en el apéndice *Implementación del DSAPI.*

A continuación mostramos los resultados de la evaluación del rendimiento de esas pruebas en cada una de las diversas muestras de información.

4.4. Evaluación del rendimiento del DSAPI

Las pruebas anteriores fueron aplicadas sobre los indexadores MG, Lucene, MySQL (con funcionalidad de búsqueda en texto completo) y sobre el DSAPI (Que a su vez utiliza a MySQL y Lucene).

El resultado de estas pruebas permite conocer el comportamiento constante del DSAPI en cargas de trabajo (al lado de indexadores en bruto), ya que su comparación con indexadores nativos sería injusta e inequitativa, porque el DSAPI utiliza a su vez dos de los participantes mostrados en los resultados de estas pruebas (MySQL y Lucene, como parte del DBAPI e IRAPI respectivamente), sin embargo, proporciona los limites de operación

del DSAPI, además de esto, en el siguiente capítulo se mostrará como parte del trabajo futuro la optimización sobre los resultados del DSAPI.

Los resultados obtenidos fueron divididos en 3 áreas de trabajo:

1. ***Tiempo de Indexación*** El tiempo de indexación consiste en el tiempo que destina la IRTool para formar un índice a partir de una muestra de datos.

   En la tabla 4.3 podemos observar que el indexador más rápido en el indexado de información en la muestra mayor (TREC EXTENDIDO) fue MG con 817.75 segundos.

   También podemos ver que el DSAPI mantiene un factor de retardo del 139 % con respecto al indexador con menor tiempo de indexado, con 1137 segundos.

   En la figura 4.14 podemos observar visualmente el resultado de los indexadores en la mayor muestra.

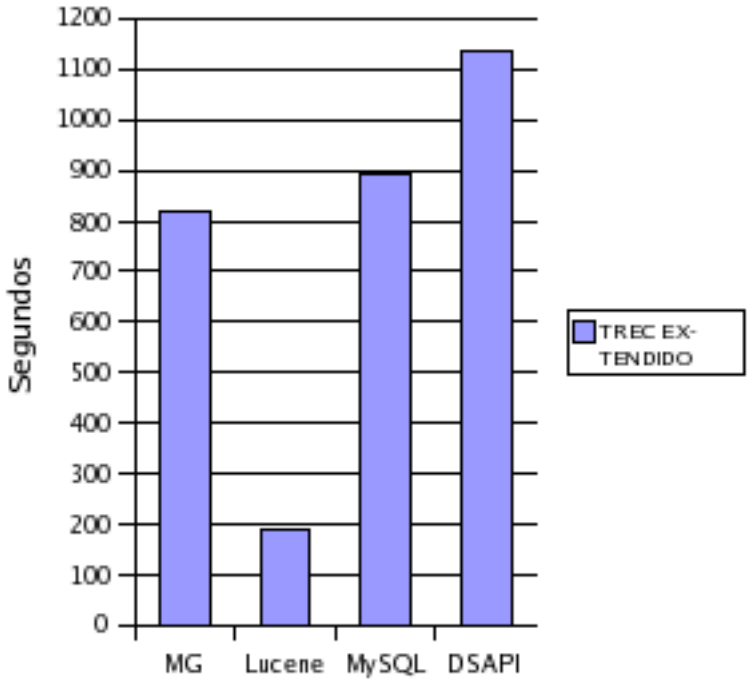

Figura 4.14: DSAPI Resultados: Tiempo de Indexación en segundos

Indexador	Mínima	DON QUIJOTE	BIBLIA	TREC EXTENDIDO
MG	0.20s	1.35s	2.07s	817.75s
Lucene	0.29s	0.60s	0.91s	189.03s
MySQL	0.44s	0.86s	1.19s	890.00s
DSAPI	0.70s	2.21s	3.23s	1137.00s

Cuadro 4.3: DSAPI Resultados Tiempos de Indexación

2. ***Tamaño de índice*** El resultado del tamaño del índice responde a la pregunta: Que tan eficiente es el DSAPI administrando los recursos de espacio en disco duro?.

   En la tabla 4.4 podemos observar que el indexador que ocupa menos espacio en disco sobre la muestra mayor de información es Lucene con 31.4MB, y que el DSAPI se mantiene arriba con una factor del 199 % con 62.6MB.

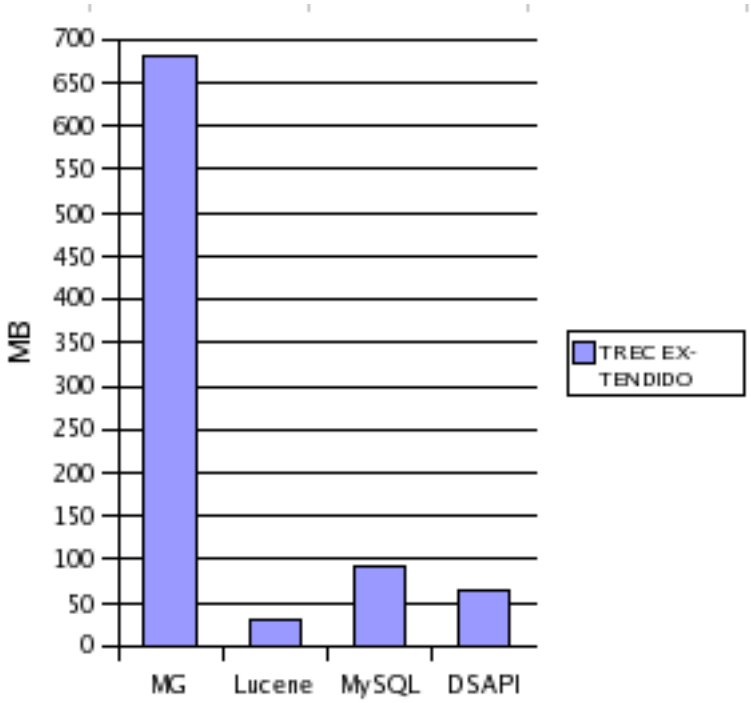

Figura 4.15: DSAPI Resultados: Tamaño índice

Indexador	**Mínima**	**DON QUIJOTE**	**BIBLIA**	**TREC EXTENDIDO**
MG	11.20KB	1.70MB	2.30MB	681.10MB
Lucene	520.00 bytes	46.70KB	108.00KB	31.40MB
MySQL	16.90KB	154.90KB	407.20KB	91.90MB
DSAPI	4.70KB	97.00KB	219.30KB	62.60MB

Cuadro 4.4: Resultados Tamaño índice.

Estos resultados conllevan a la siguiente pregunta: Cómo es esto posible, si el DSAPI esta construído con Lucene, debería ser lo mismo?

No es lo mismo, ya que Lucene muestra los datos en crudo de indexamiento, simplemente indexa la información y responde a las búsquedas. Mientras que el DSAPI , como se mencionó en la sección IRAPI, es un módulo más complejo que sigue ciertas reglas, entre ellas mantener siempre aparte de los campos de información, un solo campo llamado all, el cuál es utilizado directamente en las búsquedas en texto completo y mantiene una concatenación de todos los campos de información, esto evita desde la codificación principal investigar los tipos de campos que tiene en ese momento el IRAPI y realizar a través de un ciclo una búsqueda sobre ellos.

Por esta razón, agregando el campo *all* en el DSAPI tenemos que el resultado del tamaño de los archivos de indexación es aproximadamente el doble, en este caso 199 %. Esta opción puede ser deshabilitada como parte de una optimización del DSAPI, como se verá en el siguiente capítulo.

En la figura 4.15 podemos observar el tamaño del índice creado en la mayor muestra por los indexadores.

3. ***Tiempo de búsquedas*** El tiempo en las búsquedas consiste en el tiempo de respuesta a una consulta. La tabla 4.5 no solo muestra el tiempo de búsqueda, sino el tiempo de obtención del documento.

   Como podemos ver en la tabla 4.5 mencionada, el indexador con el menor tiempo de respuesta en las búsquedas sobre la muestra mayor es MG con 0.226 segundos.

   Con respeto a este factor el DSAPI se mantiene aproximadamente 332 % arriba de este resultado. Esto debido principalmente a la utilización de dos herramientas.

En la figura 4.16 podemos observar el tiempo de respuesta en la mayor muestra de los indexadores.

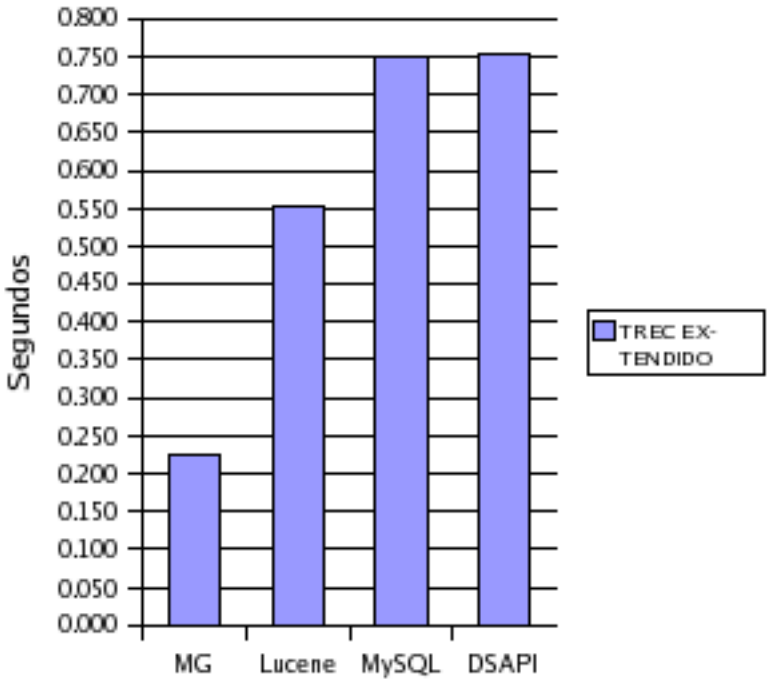

Figura 4.16: DSAPI Resultados: Tiempos de respuesta

Indexador	**Mínima**	**DON QUIJOTE**	**BIBLIA**	**TREC EXTENDIDO**
MG	0m0.142s	0m0.119s	0m0.127s	0m0.226s
Lucene	0m0.383s	0m0.383s	0m0.382s	0m0.553s
MySQL	0m0.436s	0m0.435s	0m0.435s	0m0.751s
DSAPI	0m0.544s	0m0.544s	0m0.544s	0m0.752s

Cuadro 4.5: Resultados Tiempos de respuestas.

Los resultados de estas pruebas ayudan a visualizar los márgenes de operación del DSAPI, y establecen claros límites en cada una de las áreas a optimizar, entre estas, la más importante que destaca es el tiempo de respuesta en las búsquedas con un margen del 300 % con respecto al valor óptimo.

Entre los puntos destacables de las evaluaciones realizadas sobre el DSAPI, establecen que el comportamiento del DSAPI es constante a medida que la información crece, esto a pesar de toda la funcionalidad interna y de los diversos componentes involucrados.

Como conclusión de estas pruebas podemos decir que el DSAPI en relación a los indexadores nativos o crudos[5], es un módulo con áreas de oportunidades en cuestiones de optimización y rendimiento. La diferencia principal del DSAPI con respecto a los demás indexadores, radica en su arquitectura abierta, escalable y adaptable a los cambios y necesidades. Permitiendo la exploración de nuevas soluciones y tiempos, cambiando las herramientas administradas por el DBAPI ó el IRAPI (MYSQL y Lucene en el caso de esta evaluación), y estableciendo marcas en las diversas combinaciones de ellas ó permitiendo la especialización del DBAPI en determinadas tareas.

Finalmente podemos decir que el manejo de información interna administrada por el DSAPI se encuentra costruida bajo una arquitectura personalizable, cuyo desempeño y operación pueden ser moldeados particularmente dependiendo del tipo de datos y requerimientos a los cuáles es sometido, y del tipo de herramientas administradas por el módulo.

Como reto se propone la optimización del DSAPI para que se aproxime en todos los casos al óptimo con valores menores al 100 % de los resultados óptimos obtenidos en estas evaluaciones.

Aquí terminamos el manejo de información interna realizado por el DSAPI, y a continuación describiremos el manejo externo de la información que realiza el DataServer a través del protocolo OAI (The Open Archives Initiative).

4.5. Interoperabilidad

El manejo de información externa (interoperabilidad) es una condición necesaria en los sistemas de biblioteca digitales como se observó en el capítulo 2, con el objetivo de realizar búsquedas distribuidas de información entre sistemas heterogéneos.

[5]Son aquellos indexadores que no tienen ninguna otra funcionalidad integrada, viene tal cual lo proporciona el fabricante.

Como parte del DataServer presentamos el protocolo OAI-PMH, el cual permite a PDLib navegar en otras bibliotecas OAI ó funcionar como servidor de datos para que el resto del mundo navegue en los metadatos de los documentos almacenados en PDLib.

A continuación describimos el protocolo OAI.

4.5.1. OAI-PMH

El servidor de datos de PDLib soporta el protocolo OAI (The Open Archives Initiative) [17] el cual desarrolla y promueve normas que facilitan la diseminación eficiente de contenido, en un esfuerzo por permitir la comunicación entre investigadores y universidades.

El término *archivo* denota cualquier documento con información útil acompañado de los metadatos que definen a ese archivo.

OAI se construye básicamente de tres entidades:

- *Data provider* El proveedor de datos (Data provider) mantiene uno o más repositorios (Servidores Web) que exponen los metadatos de los documentos contenidos en él a través de OAI.
- *Service provider* Los proveedores de servicios (Service provider) realizan peticiones de OAI a los proveedores de datos y construyen servicios sobre estos.
- *Client* El cliente es el realizador de las peticiones OAI.

Cuando los clientes realizan peticiones, posiblemente deseen ver los documentos de los metadatos asociados con su interés, OAI no proporciona directamente alguna funcionalidad para obtener el contenido de un documento, sin embargo el *Data provider* puede ocupar alguno de los metadatos para indicarle al cliente la ubicación de ese documento.

El protocolo OAI está formado por seis verbos o funcionalidades, las cuales se describen a continuación.

4.5.2. Verbos OAI

El manejo de información externa OAI se encuentra formado por seis verbos:

1. *GetRecord.* Este verbo devuelve los metadatos de un determinado documento. Requiere dos parámetros: el identificador único del documento y el tipo formato de metadatos solicitado (por ejemplo Dublin Core [8]).

2. *Identify.* Este verbo es usado para solicitar la información de un repositorio. Entre la información que se obtiene es:

   - repositoryName. Nombre legible del repositorio.
   - baseURL. La URL base del repositorio.
   - protocolVersion. La versión del protocolo OAI soportada.
   - earliestDatestamp. La fecha del primer documento insertado en el repositorio.
   - deletedRecord. Soporte a borrado de registros, tiene los valores *no* si el repositorio no mantiene información acerca de los registros borrados; *persistent* si el repositorio mantiene esta información si limite; *transient* si el repositorio no garantiza que esta información sea mantenida.
   - granularity. El formato de fechas soportado por el repositorio, entre los posibles valores están YYYY-MM-DD y YYYY-MM-DDThh:mm:ssZ.

3. *ListIdentifiers.* Este verbo es una forma abreviada de ListRecords debido a que solamente devuelve los encabezados de los documentos, esto es el identificador único, la fecha de creación o modificación y la colección en la cual se encuentra.

4. *ListMetadataFormats.* Este verbo informa el tipo de formatos de metadatos soportados por el repositorio (ejemplo Dublin Core)

5. *ListRecords.* La lista completa de todos los metadatos de los documentos es devuelta por este verbo. Algunas veces ocupa un *resumptionToken* para fragmentar la lista cuando esta demasiado grande.

6. *ListSets.* Informa que el repositorio no soporta colecciones ó en caso contrario muestra las colecciones que forman al repositorio.

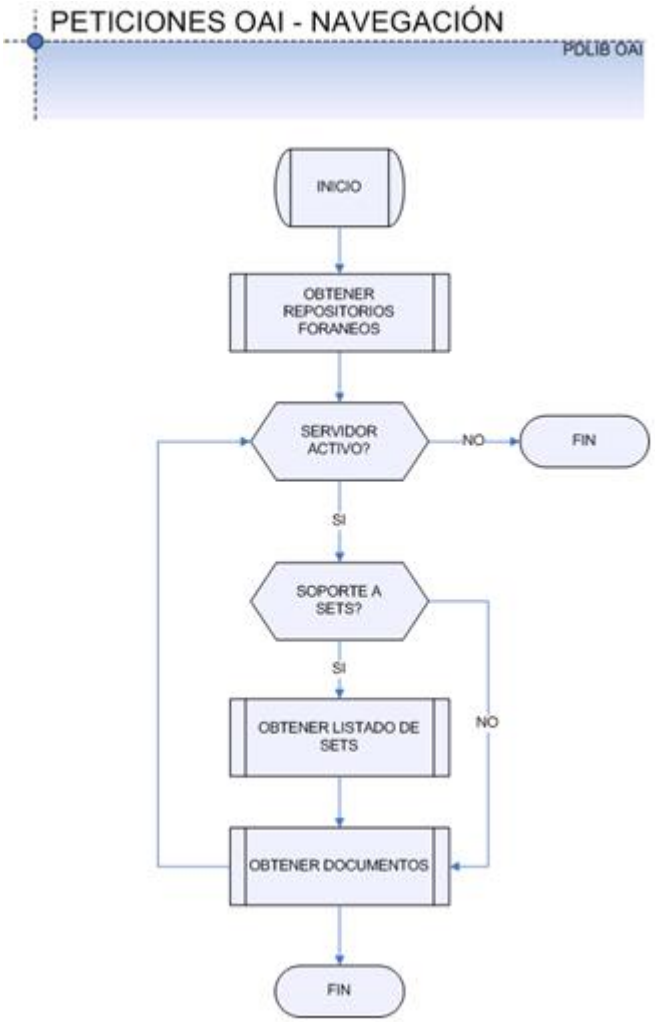

Figura 4.17: Proceso de navegación básico OAI

A continuación describimos la utilización de estos verbos en el proceso de navegación OAI.

4.5.3. Navegación OAI

El servidor de datos de PDLib es un *Data provider* y a la vez un *Service provider* proporcionando a sus clientes soporte para navegación, administración y búsquedas en repositorio foráneos bajo OAI.

El proceso de navegación esta representado en la figura 4.17.

Entre las funciones involucradas en el proceso de navegación tenemos:

- *Obtener repositorios foráneos.* Esta función retorna una lista con los identificadores y los nombres de los repositorios foráneos almacenados en PDLib.
- *Soporte a sets.* Esta funcionalidad puede tomar diferentes caminos de acuerdo al soporte de sets del repositorio foráneo.
  - Si el repositorio no soporta Sets la función devuelve todo el conjunto de documentos encontrados en el nodo raíz.
  - Si el repositorio soporta Sets entonces devuelve la lista de *colecciones* que muestran la estructura del repositorio o la lista de documentos de una determinada colección.
- *Obtener documentos.* Esta funcionalidad devuelve todos los metadatos de un determinado documento.

La búsqueda de documentos puede realizarse a través de dos modos:

- *Básico.* La frase de búsqueda es rastreada en todos los metadatos del documento. Si algún metadato contiene esta palabra de búsqueda, ese documento es devuelto en la búsqueda.
- *Avanzado.* En el modo avanzado se indican los metadatos y las frases que serán buscados en esos metadatos.

En la Administración de OAI en el DataServer se disponen opciones para altas y bajas de repositorios foráneos (figura 4.18):

- *Alta de repositorios foráneos.* Los clientes envían la dirección del repositorio que tiene soporte OAI y el servidor de datos de PDLib se encarga de obtener la identificación de ese repositorio y agregar sus datos al sistema ó en caso de que exista un error informar al usuario.

Figura 4.18: OAI Administración

- *Baja de repositorio foráneos.* Para realizar las bajas de repositorios foráneos se indica al DataServer el identificador numérico del servidor OAI y se procede con la baja del sistema.

Todo este manejo de información externa en el DataServer permite la comunicación entre bibliotecas digitales con arquitecturas heterógenas, estableciendo redes de comunicación en la búsqueda de información.

Permitiendo al DataServer navegar en repositorios foráneos de información y su vez servir como repositorio de información navegable por usuarios foráneos a través del protocolo OAI.

La implementación de protocolos de comunicación en las bibliotecas digitales actuales forma un mundo de información globalizado, facilitando el manejo externo de la información.

4.6. Conclusión

La conclusión de este capítulo forma parte de la conclusión de esta tesis, y se encuentra plasmada en el siguiente capítulo, donde concluiremos sobre el manejo de información interna y externa, el trabajo futuro pendiente de realizarse, y veremos las características destacables del por qué elegir las técnicas mencionadas en este tesis para el tratamiento de información externa e interna de documentos personales en bibliotecas digitales.

Capítulo 5

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones de éste trabajo sobre el manejo interno y externo de información en documentos personales para bibliotecas digitales y el trabajo propuesto para futuras investigaciones.

5.1. Conclusiones

Como vimos en los capítulos anteriores, el manejo de información en las bibliotecas digitales personales representa un reto muy importante en nuestros tiempos, debido a la gran cantidad de información digital que se maneja.

Este manejo de información puede ser clasificado en interno y externo. El manejo interno sucede con el tratamiento que se le da a los documentos personales dentro de la biblioteca digital, mientras que el manejo externo de la información consiste en la forma de compartir información entre bibliotecas digitales con arquitecturas heterogéneas.

A continuación veremos la conclusión realizada en esta tesis sobre el tratamiento de los documentos personales como parte del manejo interno de la información y posteriormente describiremos las conclusiones sobre el manejo externo de la información.

5.1.1. Manejo interno de la información

Para manejar grandes volúmenes de información se requieren de módulos de software transparentes, escalables y adaptables a los cambios. En esta tesis se propuso el componente DSAPI para el manejo de información interna de una biblioteca digital personal (PDLib), el cual integra una base de datos con una herramienta de indexación. El DSAPI forma una interfaz de alto nivel basada en objetos, y los usuarios pueden acceder a ésta sin sufrir modificaciones por cambios en la arquitectura o expansión de información generando un módulo de software escalable y adaptable a cambios. Esta integración le permite distribuir la carga de trabajo entre los componentes proporcionando una arquitectura basada en componentes especializados en diversas áreas.

Además de esto, los documentos personales se distinguen de otros tipos de documentos, porque varían constantemente en el transcurso del tiempo, tanto en la forma como en su contenido. Por esta razón la arquitectura propuesta para el manejo interno de la información es escalable, ya que permite fácilmente la integración de nuevas tecnologías; adaptable, debido a que permite la integración en su arquitectura de cambios sustanciales para la extracción de información en diversos formatos de documentos personales; medible, ya que sus componentes se encuentran claramente identificados; y abierto, ya que su transparencia permite especializarlo en diversas tareas dependiendo del tipo de herramientas administradas por el DSAPI y el IRAPI.

La seguridad es también un punto importante en el manejo de información, por esta razón, el DSAPI cuenta con un modelo de datos, el cual integra un tipo de seguridad particular por cada tipo de documento y, una seguridad global en la definición de grupos de trabajo.

En esta tesis también definimos mediante una serie de evaluaciones, los alcances y límites de la arquitectura propuesta con el DSAPI. Dichas evaluaciones proporcionaron la pauta a seguir para la experimentación y optimización del DSAPI.

A continuación describiremos las conclusiones del manejo externo de información.

5.1.2. Manejo externo de la información

Como parte fundamental del manejo de información externa, se implementó un protocolo de difusión conocido como OAI el cual permite establecer una comunicación con el resto de las biblioteca digitales, a pesar de tener arquitecturas heterogéneas, permitiendo navegar en documentos de otras bibliotecas digitales, así como permitir que otros usuarios naveguen en los metadatos de los documentos de PDLib lo cual ayuda a diseminar el conocimiento de manera transparente creando un punto de encuentro para investigadores y personas interesadas en temáticas comunes.

Para finalizar, contestamos los puntos observados sobre la arquitectura de las bibliotecas digitales formuladas en la conclusión del capítulo 2, las cuales son respondidas con el presente trabajo de tesis:

- *Toda la información es almacenada en el sistema de archivos.* El DSAPI (Data Storage API) permite la explotación de nuevas tecnologías, a través de las herramientas DBAPI (manejo de bases de datos) e IRAPI (manejo de herramientas de indexación), entre las tecnologías soportadas por el DBAPI, se encuentran bases de datos capaces de almacenar documentos en forma de datos, con lo cual se tiene una mejor administración y uso de estos documentos.
- *No proporcionan seguridad entre información pública, compartida y privada.* Los documentos personales, a diferencia de los documentos públicos, requieren de altos niveles de seguridad para que no puedan ser accesados por personas no autorizadas. El DSAPI cuenta con un modelo de datos que cubre con los requerimientos de la seguridad en la información personal y la disponibilidad de la información compartida o pública.
- *No manejan metadatos definidos por el usuario.* Debido a que los documentos personales a menudo necesitan de descripciones sobre la información que contienen, el DSAPI permite la integración de metadatos en los documentos que maneja, estos metadatos son definibles para cada espacio de trabajo configurado por los usuarios.

- *Los cambios en los documentos no son reflejados inmediatamente.* El DSAPI como regla de su funcionamiento, requiere de indexadores incrementales, esto es, de indexadores que no requieran reconstruir toda la biblioteca digital para agregar un documento. Por esta razón, éste requerimiento es cubierto por la arquitectura del DSAPI.

- *La arquitectura es pobremente escalable y adaptable a nuevos cambios.* El DSAPI presenta una arquitectura fácilmente escalable para soportar cambios en las demandas en los requerimientos de la información, así como adaptación a los cambios tanto de formatos como de herramientas de bases de datos o de indexación, lo cuál lo mantiene constantemente actualizado a las nuevas tecnologías.

- *No comparten información con otras bibliotecas digitales y no pueden navegar en bibliotecas digitales foráneas.*

  Gracias a la implementación del protocolo OAI, el DataServer permite comunicarse con otras bibliotecas digitales que implementen el protocolo OAI como parte de sus servicios externos de comunicación. Y al mismo tiempo, muestra aquellos documentos públicos al nivel mundial de usuarios OAI.

Como podemos observar, el presente trabajo de tesis cumple con todos los requerimientos planteados inicialmente como base de las bibliotecas digitales modernas y se presenta como una arquitectura elegible en el manejo de documentos personales.

A continuación describimos el trabajo futuro para el manejo de información concerniente a esta tesis.

5.2. Trabajo Futuro

Como trabajo futuro se destacan los siguientes puntos:

5.2.1. Cambios en el manejo de información interna.

Como parte del trabajo futuro de esta tesis se proponen los siguientes cambios:

Unificación de las consultas del DBAPI y el IRAPI.

Si las consultas del DBAPI y el IRAPI se unificaran en procesos multihilo como se puede ver en la figura 5.1 se eliminaría el tiempo de espera del IRAPI, con lo cual se conseguiría un desempeño muy cercano al óptimo.

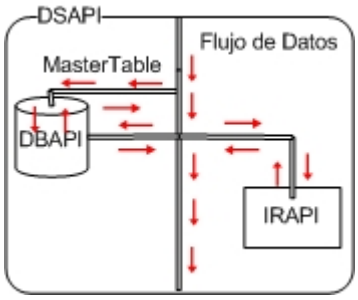

Figura 5.1: Unificación de las consultas del DBAPI e IRAPI.

Esta unificación es posible si se utilizan procesos multihilos, ya que mientras se ejecutan las consultas en el IRAPI, se están ejecutando las consultas en el DBAPI, y al final el DSAPI se encargará de hacer una combinación de los dos resultados (DBAPI e IRAPI) para devolver al usuario un solo conjunto de metadatos.

Es importante resaltar que la optimización basada en esta arquitectura es viable en aquellos casos donde la herramienta de indexación mantiene un continuo uso. Ya que el ahorro en el tiempo computacional ocurre en la llamada del IRAPI.

Esta arquitectura también exige un mayor número de recursos computacionales, porque en entornos con bajo rendimiento, podría aumentar el tiempo de uso del DSAPI, debido a factores externos como la paginación de la memoria virtual.

Eliminación de la tabla master.

Con la eliminación de la tabla master, como se puede ver en la figura 5.2, se podría ahorrar el tiempo inicial destinado a conocer los espacios de trabajo y sus implicaciones en ambos módulos.

Se propone que las consultas realizadas se envíen a cada módulo, y sea

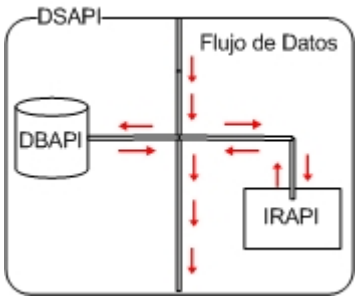

Figura 5.2: Eliminación de la tabla master.

cada módulo, capaz de decidir que campos de información le pertenecen o no.

Si a esta arquitectura le agregamos el procesamiento multihilo de la arquitectura anterior, conseguiriamos un DSAPI optimizado en velocidad en algunos casos.

Si embargo, es importante mencionar, que ahora la complejidad de la información podría ser un factor desgastante para el DSAPI. Esto debido, a que con el apoyo de la tabla master, el DSAPI es capaz de discriminar ó filtrar consultas. Mientras con ésta arquitectura propuesta como parte del trabajo futuro, los módulos encargados de realizar el manejo de información asumen que son los únicos, cada uno en su propio espacio de trabajo, y pueden regresar un gran número de resultados, que podrían haberse filtrado con la respuesta del otro módulo. Además de esto, el mezclado de los resultados de ambos módulos, ocasionaría en el DSAPI un esfuerzo computacional mayor.

Unificación de los módulos IRAPI/DBAPI.

La optimización del DSAPI podría obtenerse de una arquitectura debilmente acoplada (Base de datos e indexador separados) a una arquitectura fuertemente acoplada (Base de datos e indexador juntos), como se muestra en la figura 5.3. Con este desarrollo se reducirá a una sola consulta el tiempo de espera del DSAPI.

En esta arquitectura el control de la información recae fuertemente sobre el DBAPI, encargada de referenciar al IRAPI en los casos en los que lo necesite.

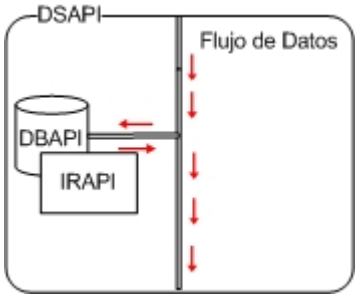

Figura 5.3: Unificación de los componentes.

Debido a su arquitectura fuertemente acoplada el índice administrado por el IRAPI, podría estar contenido en el DBAPI.

Exploración de herramientas DBAPI.

Otra forma de obtener mejores resultados que los mostrados en el capítulo anterior, consiste en la exploración de diversas bases de datos y el desarrollo de la interfaz en el DBAPI.

Para esto, se requieren pruebas exhaustivas del DSAPI sobre diversas bases de datos, de tal forma que se pueda formar un catálogo de las bases de datos soportadas asociadas con su rendimiento.

Exploración de herramientas IRAPI.

Debido a las constantes aportaciones recientes del mundo open source, es posible encontrar una gran variedad de indexadores y hacer pruebas sobre ellos.

A continuación vemos algunas de las mejoras propuestas para el manejo externo de información.

5.2.2. Cambios en el manejo de información externa.

El manejo de información externa, como parte de las comunicaciones con otras bibliotecas digitales, también es susceptible de mejoras, las cuales se listan a continuación.

Implementación de protocolos de comunicación.

En esta tesis se ha hablado de OAI-PMH como protocolo de comunicación entre bibliotecas digitales heterogéneas, sin embargo, aún falta por incluir más protocolos que garanticen la comunicación de PDLib con otras entidades, como por ejemplo Z39.50

Un mayor número de protocolos implementados ampliará las posibilidades de comunicación de la biblioteca digital, con otras bibliotecas digitales a nivel mundial.

Caché OAI.

Para un desempeño mejor de la interfaz OAI, se requiere construir una caché de documentos y servidores, de forma que las consultas a los servidores sean ejecutadas una sola vez, y todo el manejo de la información sea local.

Esto debido a que el coste de las comunicación es a menudo lenta y con posibilidades de fallos. Con la construcción de un caché OAI y su constante actualización, permitimos a la biblioteca digital la navegación inmediata en repositorios foráneos de información.

En este punto concluimos la presente tesis, esperando que los razonamientos postulados aquí, sirvan de base para la construcción de un mundo mejor.

Como parte final se incluyen las guías de instalación del DSAPI, los indexadores MG, Lucene y todo el código fuente discutido en esta tesis.

Bibliografía

[1] Nabil R. Adam and Shamim Naqvi. Universal access in digital libraries. *ACM Computing Surveys (CSUR)*, 28, December 1996.

[2] N.R. Adam, R. Holowczak, M. Halem, N. Lal, and Y Yesha. Digital library task force. *Computer*, 29:89–91, August 1996.

[3] D. S. Batory and C.C. Gotlieb. A unifying model of physical databases. *ACM Transactions on Database Systems (TODS)*, 7:509–539, December 1982.

[4] Shannon Bradshaw, Andrei Scheinkman, and Kristian Hammond. Guiding people to information: Providing an interface to a digital library using reference as a basis for indexing. *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 37–43, 2000.

[5] Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 398–409, 1995.

[6] Peter Pin-Shan Chen. The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1:9–36, 1976.

[7] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 1970.

[8] Dublin Core. Dublin core metadata initiative. *Available at: http://dublincore.org/, last visited Abril 2005*, 2005.

[9] Edward A. Fox, Robert M. Akscyn, Richard K. Furuta, and John J. Leggett. Digital libraries. *Communications of the ACM*, 38:22–28, 1995.

[10] R. Garcia-Sanchez. Connection adaptation techniques for mobile clients that access digital library services. *Master thesis, ITESM, Campus Monterrey,*, 2004.

[11] Managing Gigabytes. Download mg. *Available at: http://www.cs.mu.oz.au/mg/, last visited Abril 2005*, 2005.

[12] Managing Gigabytes. Gnu public licence. *Available at: http://mds.rmit.edu.au/mg/intro/copying.html, last visited Abril 2005*, 2005.

[13] Inc. Google. Google. *Available at: http://www.google.com/ last visited February 2005*, 2005.

[14] Erik Hatcher. Lucene intro. *java.net*, 2004.

[15] Erik Hatcher and Otis Gospodnetic. *Lucene in Action*. Manning Publications, 2004.

[16] HtDig.org. Htdig. *Available at: http://www.htdig.org, last visited February 2005*.

[17] The Open Archives Initiative. The open archives initiative. *Available at: http://www.openarchives.org/ last visited February 2005*, 2005.

[18] William C. Janssen and Kris Popat. Uplib: a universal personal digital library system. *Proceedings of the 2003 ACM symposium on Document engineering*, pages 234–242, 2003.

[19] JSE. Java search engine. *Available at: http://www.me.lv/jse/ last visited May 2005*, 2005.

[20] Nicholas Lester, Justin Zobel, and Hugh E. Williams. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. *Proceedings of the 27th conference on Australasian computer science*, 26:15–23, 2004.

[21] Francisco Álvarez Cavazos. Distribución de datos para bases de datos distribuidas, una arquitectura basada en componentes de software. Master's thesis, Instituto Tecnológico de Estudios Superiores de Monterrey, 2003.

[22] S. K. Madria and S. S. Bhowmick. Mobile data management. *IEEE Potential Magazine*, pages 11–15, October/November 2001.

[23] William C. McGee. On user criteria for data model evaluation. *ACM Transactions on Database Systems (TODS)*, 1:370–387, 1976.

[24] Sun Microsystems. The java platform for consumer and embedded devices. *Available at: http://java.sun.com/j2me/docs/ last visited December 2004*, 2002.

[25] Sun Microsystems. The java platform. *Available at: http://java.sun.com/ last visited May 2005*, 2005.

[26] The University of Calgary. The university of calgary, canada. *Available at: http://www.cpsc.ucalgary.ca/, last visited Abril 2005*, 2005.

[27] The University of Canterbury. The university of canterbury, new zealand. *Available at: http://www.cosc.canterbury.ac.nz/, last visited Abril 2005*, 2005.

[28] The University of Melbourne. The university of melbourne, australia. *Available at: http://www.cs.mu.oz.au/, last visited Abril 2005*, 2005.

[29] The University of Waikato. The university of waikato, new zealand. *Available at: http://www.cs.waikato.ac.nz/, last visited Abril 2005*, 2005.

[30] The Apache Jakarta Project. Apache jakarta tomcat. *Available at: http://jakarta.apache.org/tomcat/ last visited May 2005.*

[31] The Apache Jakarta Project. Jakarta lucene. *Available at: http://jakarta.apache.org/lucene/docs/s last visited February 2005*, 2005.

[32] RMIT. Rmit, australia. *Available at: http://www.rmit.edu.au/csit, last visited Abril 2005*, 2005.

[33] David A. Garza Salazar, Lorena G. Gomez Martinez, Juan C. Lavariega Jarquin, Juan A.Ñolazco Flores, and Martha Sordia Salinas. Pdlib: The personal digital library project. Technical report, ITESM Monterrey, 2004.

[34] David A. Garza Salazar, Yolanda Martínez Treviño, and Martha Sordia Salinas. Proyecto phronesis. *Available at: http://copernico.mty.itesm.mx/phronesis/Proyectos/, last visited February 2005*, 2005.

[35] Ben Shneiderman, David Feldman, Anne Rose, and Xavier Ferré Grau. Visualizing digital library search results with categorical and hierarchical axes. *Proceedings of the fifth ACM conference on Digital libraries*, pages 57–66, 2000.

[36] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. Data models. *ACM Computing Surveys (CSUR)*, 28:105–108, 1996.

[37] MYSQL Team. Mysql. *Available at: http://www.mysql.org last visited May 2005*, 2005.

[38] Anthony Tomasic, Héctor García-Molina, and Kurt Shoens. Incremental updates of inverted lists for text document retrieval. *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 289–300, 1994.

[39] TREC. Text retrieval conference. *Available at: http://trec.nist.gov/, last visited Abril 2005*, 2005.

[40] Ullman, Jeffrey, and Widom Jennifer. *Introducción a los Sistemas de Bases de Datos*. Prentice Hall, 1999.

[41] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. Managing gigabytes, compressing and indexing documents and images. pages 367–368, 1994.

[42] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes, Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 1994.

[43] Tak W. Yan and Héctor Sellen, García-Molina. Information finding in a digital library: the stanford perspective. *ACM SIGMOD Record*, 24:62–70, September 1995.

Apéndice A

MG. Guía de uso

MG (Managing Gigabytes) es una herramienta open-source (GNU public licence [12]) para indexamiento y búsqueda de información para texto, imágenes e imágenes textuales.

La versión de MG utilizada en esta tesis se encuentra disponible en la página oficial de MG [11].

MG fue desarrollado inicialmente por Ian Witen (Profesor de Ciencias Computacionales en la Universidad de Waikato, Nueva Zelanda), Tim C. Bell (Universidad de Canterbury), Alistair Moffat (Universidad de Melbourne), Justin Zobel (RMIT) y otros colaboradores.

Entre las instituciones que han cooperado para el desarrollo de MG se encuentran:

- The Australian Research Council.
- Collaborative Information Technology Research Institute, Melbourne.
- The University of Calgary, Canada. [26]
- The University of Canterbury, New Zealand. [27]
- The University of Melbourne, Australia.[28]
- RMIT, Australia. [32]
- The University of Waikato, New Zealand. [29]

El sistema MG es una colección de pequeños programas los cuales en su totalidad constituyen el sistema de búsqueda e indexamiento de información. Este sistema permite crear colecciones de documentos y hacer búsquedas de texto completo sobre estas colecciones.

El uso mas común de MG ha sido búsquedas sobre archivos de mail en UNIX. Sin embargo, su uso puede ser extendido prácticamente a cualquier aplicación imaginable, como la presentada en esta tesis.

A.1. Instalación.

Una vez que se baja el archivo de instalación de MG (la versión ocupada en esta tesis es mg-1.2.1.tar.gz) se descomprime en un directorio. Es importante recordar que esta versión de MG solo funcionará en un entorno Linux o Unix, sin embargo existen otras versiones que pueden ser instaladas en otros sistemas operativos.

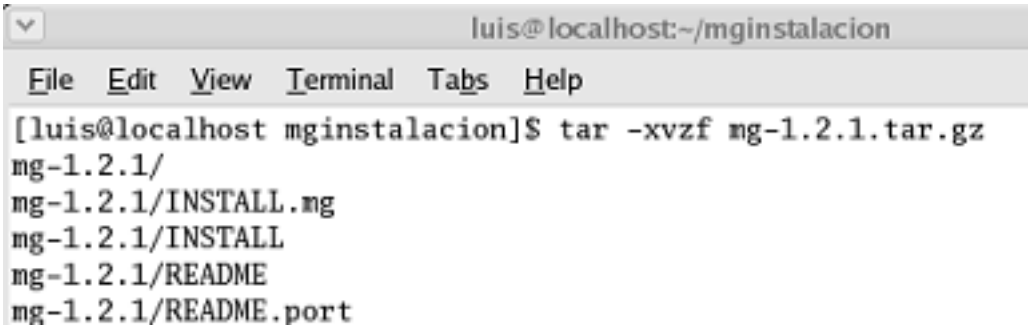

Figura A.1: Descompresión de MG

En una ventana de consola *bash* sobre el directorio donde se encuentra el archivo comprimido de MG se tiene que ejecutar el comando *tar -xvzf mg-1.2.1.tar.gz* como se muestra en la figura A.1 para descomprimir el archivo.

Esto creará una carpeta con el mismo nombre del archivo comprimido en el directorio actual. Posteriormente se ingresa a la carpeta creada *cd mg-1.2.1* y se teclea en el siguiente orden los comandos:

1. *./configure*

2. *make*

3. *make install*

Los dos primeros comandos previos comprobarán que el sistema pueda compilar e instalar MG en el sistema operativo Linux. El tercer comando realiza la instalación de MG.

Una vez instalado se debe crear una carpeta que albergará los índices de MG. para esto crea una carpeta llamada *mgdata* en */home/mgdata*, y agrega la variable de entorno MGDATA con la dirección donde estarán los índices, para esto realiza lo siguiente.

1. *cd /home*

2. *mkdir mgdata*

3. *export MGDATA=/home/mgdata/*

A.2. Uso.

Posteriormente es necesario crear en la carpeta donde se ejecutaran los comandos de MG un archivo llamado .mg_getrc Este archivo deberá de contener información referente a las colecciones como su nombre y la dirección en que se encuentran sus documentos, ejemplo A.2.

De esta forma MG conoce que existe una colección llamada tesis.

Posteriormente para indexar la colección llamada tesis, se utiliza el comando *mgbuild tesis*, con lo cual se creará un índice de todos los documentos .txt contenidos en la colección tesis que se encuentran en la ruta /home/luis/testdata.

Aparte de *DIR* existen otras formas de indexar *(PARA, MAIL, PLAIN)* de los cuales se puede observar su funcionamiento en el archivo *mg_get*.

El índice es guardado donde apunta la variable de entorno MGDATA definida anteriormente. La salida del programa *mgbuild* se puede observar a continuación A.3.

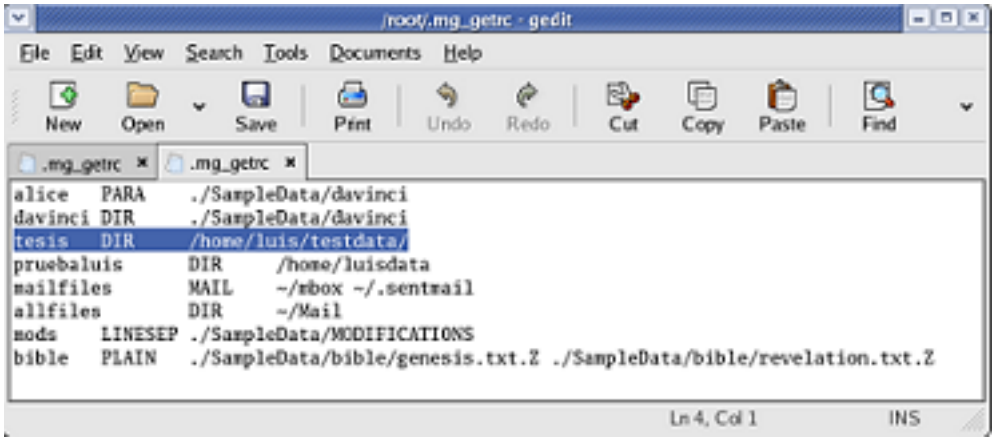

Figura A.2: Archivo .mg_getrc

```
[root@localhost luis]# time mgbuild tesis
---------------------------------
localhost.localdomain, Tue Mar  1 19:18:36 EST 2005
tesis --> tesis/tesis
---------------------------------
/usr/local/bin/mg_get tesis -init
/root/.mg_getrc
Probando para Luis...
---------------------------------
/usr/local/bin/mg_get tesis -text | mg_passes -f tesis/tesis -2
ivf.pass1 : Mem reqd for 1 chunk:        7 bytes
ivf.pass1 : Chunks written      : 1
ivf.pass1 : Peak memory usage   :        1.2 Mb
ivf.pass1 : Stems size          :        0.2 kB  107.37%
ivf.pass1 : Lev 1 inverted file :        0.0 kB    5.07%
ivf.pass1 : Lev 2 inverted file :        0.0 kB    7.37%
ivf.pass1 : Lev 3 inverted file :        0.0 kB   13.36%
ivf.pass1 : Record addresses    :        0.0 kB    0.00%
```

Figura A.3: Salida de mgbuild

Para realizar las búsquedas en MG se utiliza el comando mgquery [nombre_colección]. El cual proporciona por default una consola donde se pueden ir escribiendo las búsquedas.

Apéndice B

Jakarta Lucene. Guía de uso

Jakarta Lucene es un motor de búsqueda escrito en Java. Fue creado por Doug Cutting en 1997 y liberado como software libre en 2000. Permite ser empleado en cualquier aplicación que requiera búsquedas en texto, desde búsqueda en páginas Web hasta gestores de correo. La última versión (1.4) incluye una clase para hacer caché en memoria de los resultados de algunos términos de búsqueda.

Lucene se encuentra disponible en dos tipos de distribuciones:

- Código fuente de clases (lucene-1.4-final.tar.gz)
- Clases empaquetadas en un único archivo .jar (lucene-1.4-final.jar)

Ambas distribuciones constituyen la API de Lucene, es decir, no es un programa ejecutable, sino un conjunto de funciones las cuales deben ser invocadas a través de la construcción de un software que las utilice.

En esta guía utilizaremos la segunda distribución de Lucene (lucene-1.4-final.jar) la cual se encuentra en su sitio oficial [31], la versión de Java 2 Platform SE v1.4.2. y nos basaremos en el artículo *Lucene Intro* publicado en java.net por Erik Hatcher [14].

B.1. Indexamiento

Para utilizar las funciones de indexamiento de Lucene necesitamos crear una clase que haga uso de esta funcionalidad. Llamaremos a esta clase *Indexer* y la utilizaremos para indexar todos los archivos textuales de un determinado directorio.

Esta clase contendrá un único método público *index()* que toma dos argumentos. El primer argumento *indexDir* es un objeto de tipo *File* que corresponde al directorio donde el índice será creado. El segundo argumento es otro objeto de tipo *File* que corresponde al directorio que será indexado.

```
public static void index(File indexDir, File dataDir) throws IOException {
if (!dataDir.exists() || !dataDir.isDirectory()) {
throw new IOException(dataDir + "does not exist or is not a directory");
}
IndexWriter writer = new IndexWriter(indexDir, new StandardAnalyzer(),
true);
indexDirectory(writer, dataDir);
writer.close();
}
```

Esta función crea un objeto de tipo *IndexWriter* que será ocupado para construir el índice, además manda a llamar *indexDirectory* que mostramos a continuación:

```
private static void indexDirectory(IndexWriter writer, File dir) throws
IOException {
File[] files = dir.listFiles();
for (int i=0; i < files.length; i++) {
File f = files[i];
if (f.isDirectory()) {
indexDirectory(writer, f);
} else if (f.getName().endsWith(".txt")) {
indexFile(writer, f);
} } }
```

El método *indexDirectory* se llama recursivamente en el caso de que existan subdirectorios, en el caso de archivos utilizamos la función *indexFile*,

la cual se define a continuación:
private static void indexFile(IndexWriter writer, File f) throws IOException
{
System.out.println("Indexing" + f.getName());
Document doc = new Document();
doc.add(Field.Text("contents", new FileReader(f)));
doc.add(Field.Keyword("filename", f.getCanonicalPath()));
writer.addDocument(doc);
}

Con esto hemos completado el proceso de indexamiento. El cual indexará de un determinado directorio todos los archivos con extensión *.txt*.

En general el proceso de indexamiento realiza 4 pasos:

1. Crea una instancia de la clase IndexWriter
2. Localiza los archivos con extensión .txt
3. Para cada archivo encontrado crea un *Document* con sus respectivos campos *Fields*.
4. Agrega los documentos a la instancia de la clase IndexWriter.

Los documentos son las unidades primarias de información para Lucene. Cada Documento *(Document)* consiste de una serie de campos *(Fields)*. Cada campo tiene un nombre y un contenido.

Los campos pueden tener tres diferentes atributos:

- *Stored.* El texto original del documento se almacena junto con el índice.
- *Indexed.* Indexa el contenido del campo.
- *Tokenized.* El texto agregado es descompuesto por el analizador

Para agregar contenido a la clase *Field* se dispone de métodos estáticos los cuales combinan los atributos mencionados, como por ejemplo:

- *Field.Keyword.* +Indexed +Stored -Tokenized.

- *Field.Text.* +Indexed +Tokenized.
- *Field.UnIndexed.* +Stored -Indexed -Tokenized.
- *Field.UnStored.* +Indexed +Tokenized -Stored

La clase *Indexer* completa se describe a continuación:

```
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;

import java.io.File;
import java.io.IOException;
import java.io.FileReader;

public class Indexer {
 public static void index(File indexDir, File dataDir) throws IOException {
if (!dataDir.exists() || !dataDir.isDirectory()) {
throw new IOException(dataDir + "does not exist or is not a directory");
}
IndexWriter writer = new IndexWriter(indexDir, new StandardAnalyzer(),
true);
indexDirectory(writer, dataDir);
writer.close();
}
private static void indexDirectory(IndexWriter writer, File dir) throws
IOException {
File[] files = dir.listFiles();
for (int i=0; i < files.length; i++) {
File f = files[i];
if (f.isDirectory()) {
indexDirectory(writer, f);
} else if (f.getName().endsWith(".txt")) {
indexFile(writer, f);
} } }
private static void indexFile(IndexWriter writer, File f) throws IOException
```

```
{
System.out.println("Indexing" + f.getName());
Document doc = new Document();
doc.add(Field.Text("contents", new FileReader(f)));
doc.add(Field.Keyword("filename", f.getCanonicalPath()));
writer.addDocument(doc);
}

 public static void main(String[] args) throws Exception {
if (args.length != 2) {
throw new Exception("Usage: " + Indexer.class.getName() + " <index dir>
<data dir>");
}
File indexDir = new File(args[0]);
File dataDir = new File(args[1]);
index(indexDir, dataDir);
} }
```

B.2. Búsquedas

Una vez que se han indexado los documentos, se pueden hacer búsquedas sobre ellos. Para esto Lucene ocupa la clase *IndexSearcher*.

Definiremos la clase *Searcher* con el objetivo de realizar una búsqueda ocupando la clase *IndexSearcher*.

```
import org.apache.lucene.document.Document;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.Hits;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.store.Directory;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import java.io.File;
```

```
public class Searcher {
public static void main(String[] args) throws Exception {
if (args.length != 2) {
throw new Exception("Usage: " + Searcher.class.getName() + " <index
dir> <query>");
}
 File indexDir = new File(args[0]);
String q = args[1];
if (!indexDir.exists() || !indexDir.isDirectory()) {
throw new Exception(indexDir + " is does not exist or is not a directory.");
}   search(indexDir, q);
} }
```

```
 public static void search(File indexDir, String q) throws Exception{
Directory fsDir = FSDirectory.getDirectory(indexDir, false);
IndexSearcher is = new IndexSearcher(fsDir);  Query query =
QueryParser.parse(q, "contents", new StandardAnalyzer());
Hits hits = is.search(query);
System.out.println("Found " + hits.length() + " document(s) that matched
query " + q + ":");
for (int i = 0; i < hits.length(); i++) {
Document doc = hits.doc(i);
System.out.println(doc.get("filename"));
} }
```

Ahora con el programa *Searcher* desde la línea de comandos se puede invocar escribiendo en primer lugar la dirección donde se encuentra el índice y posteriormente la palabra a buscar.

Tanto para indexar como para buscar necesitas agregar al path de Java el *.jar* de Lucene.

Este es un pequeño ejemplo de la funcionalidad de Lucene, si se desea conocer mas ampliamente se puede consultar el libro Lucene in Action [15].

Apéndice C

Código Fuente DSAPI

El siguiente source code forma parte del DSAPI mostrado en esta tesis, el cual esta dividido en clases para su mejor referencia.

El source code DBAPI se encuentra basado en el L-DBMS[21] (Sistema manejador de base de datos local)

C.1. AlterTable.

La clase AlterTable es utilizada para actualizar las definiciones de las tablas de la base de datos, entre las funciones principales de esta clase destacan (ver figura C.1):

- addColumn. Esta operación agrega una columna a una tabla de una base de datos específica.
- addColumnAsKey. Esta operación agrega una columna que será la llave primaria de la tabla de una base de datos específica.
- alterColumn. Esta operación modifica los atributos de una columna de la tabla de una base de datos específica.
- alterColumnAsKey. Esta operación modifica los atributos de una columna primaria de la tabla de una base de datos específica.

- dropColumn . Esta operación borra una columna de la tabla de una base de datos específica.
- dropPrimaryKey . Esta operación borra una columna primaria de la tabla de una base de datos específica.
- createTable . Esta operación crea una tabla de una base de datos específica.

Un ejemplo del funcionamiento de esta clase dentro del DSAPI es el siguiente:

Codigo fuente AlterTable
1. AlterTable alterTable = dbc.getAlterTable("master");
2. alterTable.addColumn("name", "TEXT"); // Add column
3. boolean flag = alterTable.execute();

El código fuente AlterTable tiene como propósito agregar una columna (name) a una tabla de la base de datos (master). El dbc es un objecto de la clase DBConnection, el cual será explicado mas adelante. En la línea 1 se proporciona el nombre de la tabla a modificar y en la línea 2 se especifica el nombre de la columna a agregar así como su tipo de datos. En la línea 3 se ejecuta la operación y se guarda su resultado, el cual es un valor booleano verdadero si la operación tuvo éxito ó falso en caso contrario.

C.2. CreateTable

La clase CreateTable es utilizada para crear nuevas tablas en la base de datos, entre las funciones principales de esta clase destacan(ver figura C.2):

getTableObject. Esta operación carga en memoria una tabla nueva, y se prepara para agregarle atributos, retorna un valor DBTable que corresponde con el nombre de la tabla porporcionado a la función.

addColumnAsKey. Esta operación agrega una columna que será la llave primaria de la tabla recientemente definida.

addColumn. Esta operación agrega una columna a una tabla nueva.

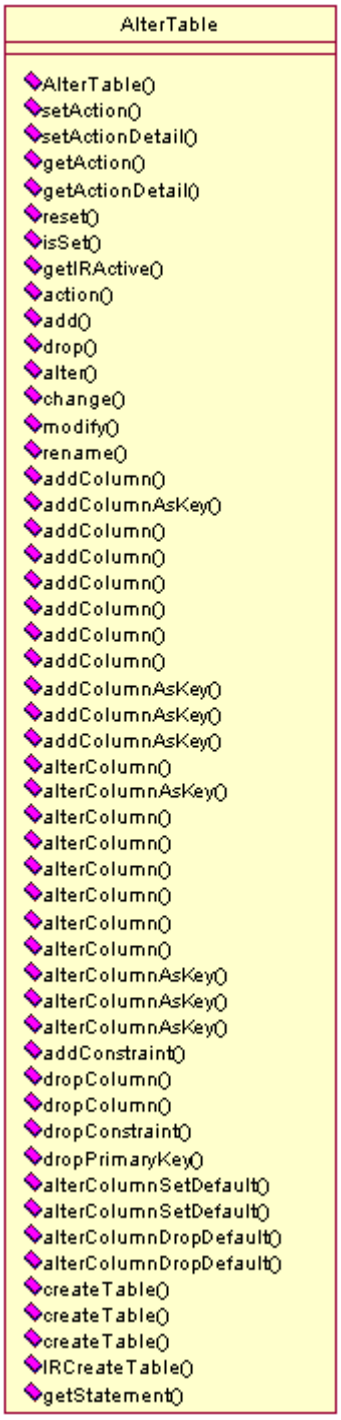

Figura C.1: Clase AlterTable

getStatement. Esta operación retorna una cadena SQL correspondiente a la operación de CREATE TABLE.

Un ejemplo de la clase CreateTable en funcionamiento es el siguiente:

Codigo fuente CreateTable

```
1. CreateTable ct = dbc.getCreateTable("T1");
2. ct.addColumnAsKey("C1","INTEGER");
3. ct.addColumnAsKey("C2","INTEGER");
4. ct.addColumn("C3","INTEGER");
5. ct.addColumn("C4","TEXT");
6. ct.execute();
```

El código fuente CreateTable en la línea 1 se realiza una asignación del nombre de la nueva tabla en la base de datos, en las líneas 2 y 3 se agregan las columnas (C1 y C2) con sus tipos de datos INTEGER, estas columnas serán las llaves primarias de la tabla; y en las línea 4 y 5 se agregan el resto de las columnas de datos, finalmente en la línea 6 se ejecuta la operación y se crea la tabla T1 en la base de datos con el esquema mostrado en el cuadro C.1. Donde C1 y C2 son tipos enteros y llaves primarias de la tabla, C3 y C4 son campos de la tabla de tipo entero y texto respectivamente.

Field	Type	Description
C1	Entero	Llave primaria de la tabla
C2	Entero	Llave primaria de la tabla
C3	Entero	Campo de la tabla
C4	Texto	Campo de la tabla

Cuadro C.1: CreateTable T1

C.3. DBColumn

La clase DBColumn es utilizada para representar las columnas de las tablas de la base de datos, su principal función es obtener y modificar los atributos de las columnas de las tablas, entre sus funciones principales destacan (ver figura C.3):

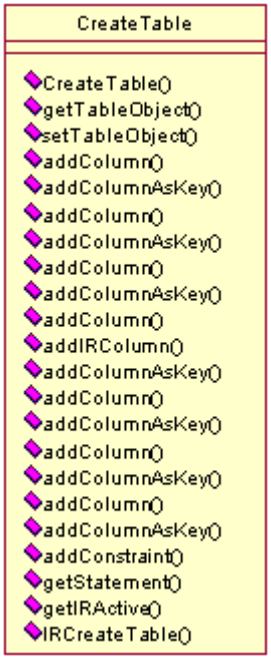

Figura C.2: Clase CreateTable

getName. Esta operación devuelve el nombre de la columna de una tabla.

getIsPrimary. Esta operación devuelve verdadero en caso de que la columna sea la llave primaria de la tabla, falso en caso contrario.

setDatatype. Esta operación declara el tipo de dato de una columna.

setIsPrimary. Esta operación declara a la columna como llave primaria de una tabla.

setName. Esta operación asigna un nombre a una columna de una tabla.

A continuación mostramos un ejemplo de código fuente del constructor de la clase DBColumn y su implementación.

Codigo fuente DBColumn Constructor

```
1. public DBColumn(
2. String columnName,
3. String datatype ) {
4.
5. this.name = columnName;
6. this.datatype = datatype;
7. this.datasize = 0;
8. this.decimalDigits = 0;
9. this.isNull = true;
10. this.isPrimary = false; }
```

Codigo fuente DBColumn Implementación

```
11 CreateTable ct = dbc.getCreateTable("T1");
12. ct.addColumn("C3","INTEGER");
```

En el código fuente DBColumn Constructor podemos observar sobre las líneas 2 y 3 la declaración del constructor de la clase DBColumn utilizando dos cadenas de texto representando al nombre y al tipo de datos respectivamente. Los tipos de datos dependen de la base de datos que se este usando en ese momento, ya que el string que representa al tipo de datos es pasado integramente al manejador SQL de la base de datos y esta generará un error si no lo puede interpretar o realizará la operación en caso contrario.

Por default se asignan los valores de tamaño de datos a un valor de cero (línea 7), sin dígitos decimales (línea 8), permite la aceptación de nulos (línea 9) y no es una columna primaria por default (línea 10); estos valores pueden ser modificados posteriormente utilizando las funciones de la clase.

El código fuente DBColumn Implementación muestra en la línea 12 la creación de una instancia de la clase DBColum llamada C3 y de tipo de datos entero.

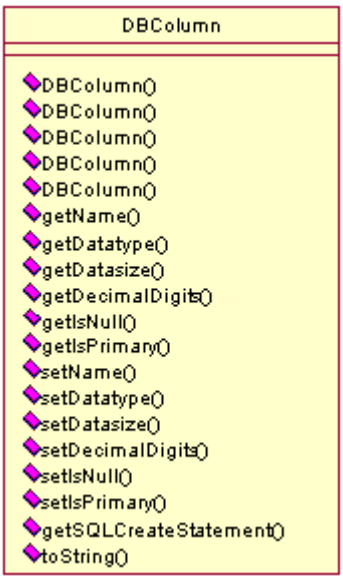

Figura C.3: Clase DBColumn

C.4. DBConnection

Esta clase implementa un mecanismo responsable de crear los objetos a partir de sus clases de implementación y proporcionar acceso a los mismos.

Implementa las siguientes funciones (ver figura C.4).

connected. Esta operación responde con un valor verdadero si se encuentra actualmente conectado o falso en caso contrario.

getAlterTable. Esta operación devuelve un objeto AlterTable dado un objeto de la clase DBTable.

getConnection. Este método retorna un objeto de la clase Connection, el cual repesenta a la conexión actual.

getCreateTable. Esta operación retorna un objeto CreateTable dada una cadena representando el nombre de una tabla de la base de datos.

getDelete. Esta operación retorna un objeto de la clase Delete dado el nombre de una tabla.

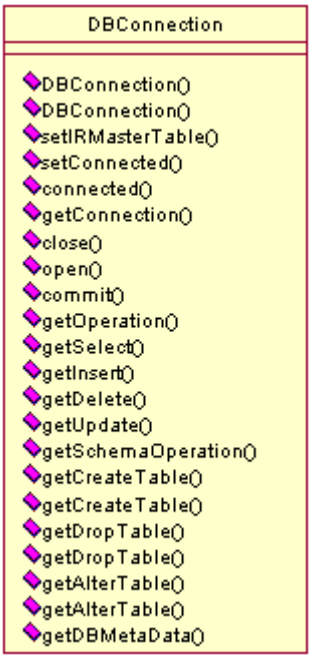

Figura C.4: Clase DBConnection

C.5. DBMetaData

La clase DBMetaData está encargada de proporcionar información de la base de datos, entre sus funciones destacan(ver figura C.5):

getTableCount. Esta operación devuelve el número de tablas que contiene la base de datos.

exists. Esta función comprueba que exista una tabla en la base de datos.

getTableIndex. Esta función devuelve el índice de la tabla en la base de datos.

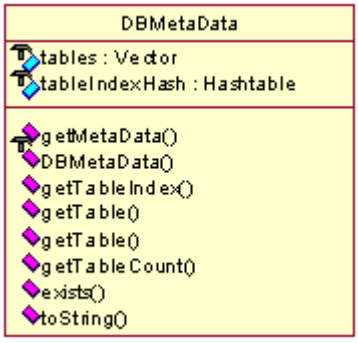

Figura C.5: Clase DBMetaData

C.6. DBTable

La clase DBTable es una de las clases fundamentales del DBAPI, representa a una tabla de la base de datos, proporcionando operaciones para leer ó escribir los atributos de las tablas. A continuación mostramos sus principales operaciones (ver figura C.6).

addColumn. Esta operación agrega una columna a la tabla.

addColumnAsKey. Esta operación agrega una columna que será la llave primaria de la tabla.

getColumns. Esta operación devuelve un array de metadatos conteniendo todas las columnas de la tabla.

getColumnName. Esta operación devuelve el nombre de la columna dado su índice dentro de la tabla.

getColumnSize. Esta operación devuelve el tamaño(size) de una columna.

getPrimaryKeyCount. Esta función retorna el número de columnas que son llaves primarias de la tabla.

getSQLCreateStatement. Esta función regresa la sentencia SQL que

afecta a la tabla.

A continuación se muestra un ejemplo de clase DBTable:

Codigo fuente DBTable

```
1. DBTable tableObject;
2. tableObject = new DBTable("T2");
3. tableObject.addColumnAsKey("C2","INTEGER");
4. tableObject.addColumn("C3","INTEGER");
5. tableObject.getSQLCreateStatement();
```

En el código fuente DBTable se observa la creación de una tabla llamada T2 (línea 2), en la línea 3 y 4 se crean dos nuevas columnas y finalmente en la línea 5 se obtiene el SQL resultante de las operaciones anteriores. Este SQL puede ser ejecutado sobre la base de datos para la creación de la nueva tabla.

C.7. Delete

Esta es encargada de eliminar información de las tablas. Mantiene internamente 3 vectores: whereColumns, whereOperators y whereValues que contienen las columnas involucradas en las condiciones para el borrado de información, los operadores de estas condiciones y los valores utilizados.

Entre sus funciones principales destacan (ver figura C.7):

getStatement. Este método devuelve una sentencia SQL correspondiente al borrado de información.

where. Este método recibe tres parámetros (una cadena representando la tabla que se va a modificar, el operador de la condición y el valor utilizado durante la condición) que indican la condición necesaria para el borrado de información.

whereMatch. Este método se utiliza para el borrado de información basando en las búsquedas de texto completo.

executeOperation. Este método se encarga de realizar la inserción dentro de la tabla generando el SQL de inserción y ejecutándolo en la base de datos.

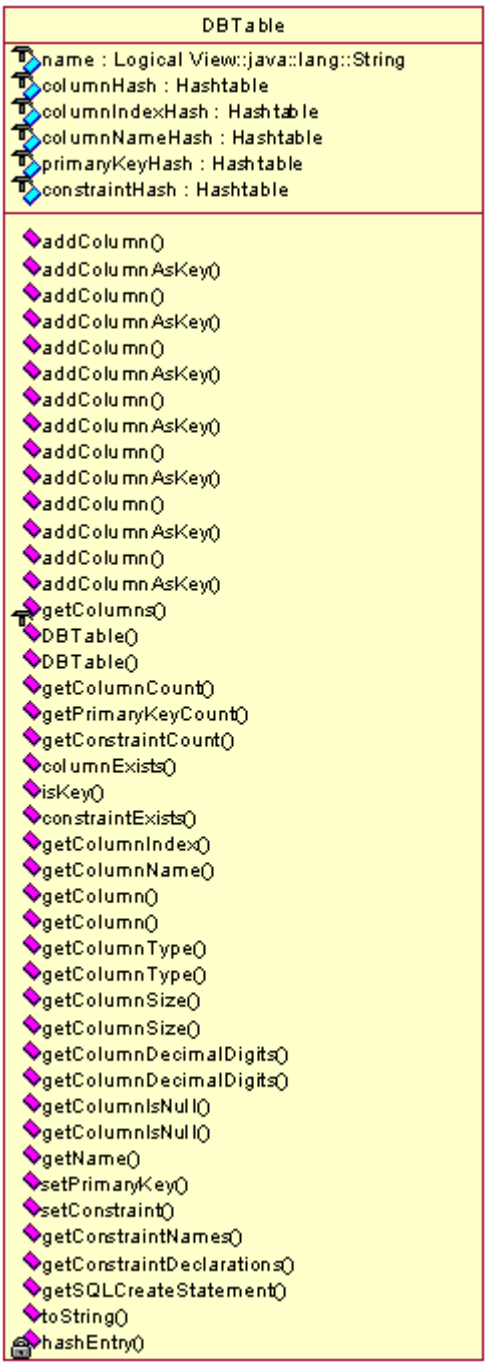

Figura C.6: Clase DBTable

Un ejemplo del código Delete es el siguiente:

Codigo fuente Delete
1. Delete delete = dbc.getDelete("T1");
2. delete.where("C1", "=", 9);
3. delete.execute();

El código fuente Delete borra el ó los datos cuya columna C1 sea igual a 9. En la línea 1 se crea una instancia de la clase Delete que se basa en la tabla T1, en la línea 2 se forma la condición incluyendo como parámetros el nombre de la columna, el operador utilizado y el valor delimitante del alcanze de Delete. Y en la línea 3 del código fuente se genera la ejecución del borrado de información.

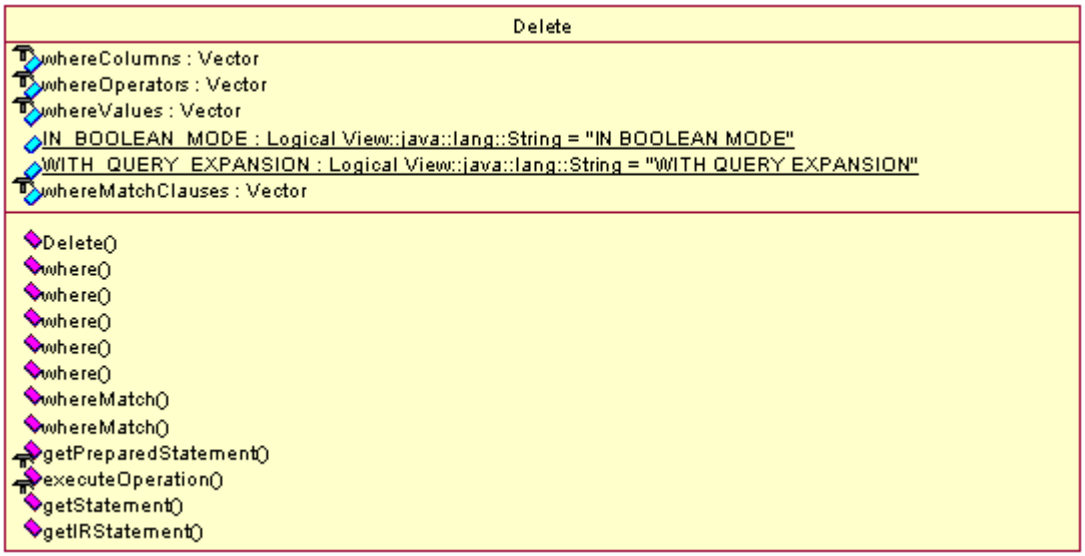

Figura C.7: Clase Delete

C.8. DropTable

La clase DropTable es una de las mas sencillas dentro del DBAPI, encargada de borrar las tablas de la base de datos. A continuación mostramos

sus operaciones (ver figura C.8):

DropTable (Constructor). El constructor de la clase DropTable recibe dos parámetros, la conexión y el nombre de la tabla a borrar respectivamente.

getStatement. Esta operación genera una sentencia SQL de tipo DROP Table.

A continuación se muestra un ejemplo de clase DropTable:

Codigo fuente DropTable

1. DropTable droptable;
2. droptable = new DropTable(dbc,"T2");
3. tableObject.getStatement();

El código fuente DropTable genera como resultado una cadena SQL que puede ser ejecutada sobre la base de datos para eliminar una tabla, en la línea 1 se declara un objeto de la clase DropTable, en la línea 2 se establece el nombre de la tabla a eliminar y en la línea 3 se genera la sentencia SQL para el borrado de la tabla.

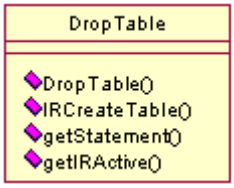

Figura C.8: Clase DropTable

C.9. Insert

Esta clase es encargada de realizar las inserciones en las tablas de la base de datos. Mantiene internamente dos vectores, uno de nombres de columnas y otro de valores.

Las operaciones implementadas por la clase Insert son las siguientes (ver figura C.9):

value. Este método se encarga de asignar un valor a una columna determinada, sus parámetros son el nombre de la columna de la tabla y el valor asignado.

executeOperation. Este método se encarga de realizar la inserción dentro de la tabla generando el SQL de inserción y ejecutándolo en la base de datos.

getStatement. Este operación retorna el SQL de inserción.

Un ejemplo de un código de inserción es el siguiente:

Codigo fuente Insert

```
1. Insert insert = dbc.getInsert("T1");
2. insert.value("C1", 1);
3. insert.value("C2", 2);
4. insert.value("C3", 3);
5. insert.value("C4", "cadena");
6. insert.execute();
```

El código fuente Insert muestra la inserción en la tabla T1 de los valores 1, 2, 3 y una cadena de texto en los campos C1, C2, C3 y C4 respectivamente.

En la línea 1 se crea un objeto de inserción que referencía a la tabla T1, en las líneas 2, 3 y 4 se insertan valores enteros, en la línea 5 se inserta una cadena de texto y en la línea 6 se ejecuta la operación de inserción.

Si alguna de las columnas no concuerda con el tipo establecido en la tabla de la base de datos, la inserción generará un error capturable por el usuario.

C.10. IRDBTable

La clase IRDBTable (ver figura C.11) es una de las herramientas más importantes para el manejo de información del DSAPI. Se encarga de relacionar a nivel de código fuente la herramienta de indexación (IRAPI) con el DBAPI.

Se puede considerar a la clase IRDBTable como el punto común de ejecución ó de encuentro que tienen los componentes DBAPI e IRAPI.

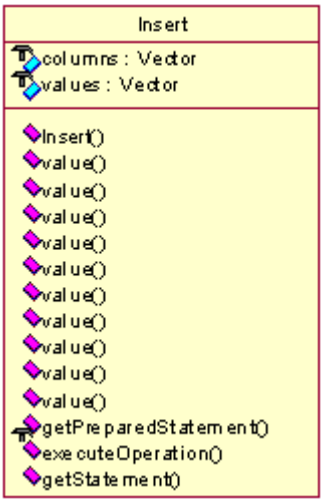

Figura C.9: Clase Insert

Entre las funciones principales de la clase tenemos:

- ***addIRColumn.*** Esta función se encarga de indexar el contenido de un campo almacenado en la base de datos.
- ***IRDBTable.*** Se encarga de crear una nueva tabla de la clase IRDBTable. que incorpora una tabla DBAPI con soporte IRAPI.
- ***getIRKeyCount.*** Esta función devuelve la cantidad de llaves primarias que contiene la herramienta de indexación.
- ***getIRActive.*** Esta función devuelve un valor de falso si no se esta utilizando al IRAPI, es decir si solamente se esta trabajando con el DBAPI; verdadero en caso contrario.
- ***createIndex.*** Esta función se encarga de efectuar la operación de nuevo índice.
- ***IRMasterCreateTable.*** Esta función se encarga de guardar los datos de la estructura de la tabla en la tabla master.

  Primero inserta los datos de la base, posteriormente lo datos de la herramienta de indexación y por último actualiza la propiedad de las llaves primarias en los campos que así o requieran.

- ***executeIRInsert.*** Esta función se encarga de efectuar la operación Inserción en una tabla distribuyendo los datos que van a la IRTool o al DBAPI.

- ***executeIRDelete.*** Esta función se encarga de efectuar la operación Delete en una tabla. Borra los datos tanto de la tabla como del indexador.

C.11. IRMasterTable

La clase IRMasterTable (ver figura C.10) se encarga de mantener en memoria y siempre disponible el contenido de la tabla master de la base de datos. Esta clase contiene funciones fundamentales en el manejo de información del DataServer, ya que proporciona la información necesaria para conocer los campos que se encuentran en la base de datos y los campos que se encuentran en el indexador.

Entre sus funciones principales se encuentran:

- ***DBsize.*** Esta función recibe como parámetro el nombre de una tabla y devuelve el número de columnas de la tabla que se encuentran en la base de datos.

- ***isDB.*** Esta función recibe como parámetros dos cadenas de texto representando al nombre de una tabla y de una columna respectivamente. La función devuelve un valor verdadero si el campo pertenece a la tabla y su contenido se encuentra en la base de datos, y falso en caso contrario.

- ***isIR.*** Esta función recibe como parámetros dos cadenas de texto representando al nombre de una tabla y de una columna respectivamente. La función devuelve un valor verdadero si el campo pertenece a la tabla y su índice es administrado por la herramienta de indexación, y falso en caso contrario.

- ***isPrimary.*** Esta función recibe como parámetros dos cadenas de texto representando al nombre de una tabla y de una columna respectivamente. Devuelve un valor verdader si la columna pertenece a

la tabla y es una llave primaria (por lo tanto se encuentra tanto en la base de datos como en el indexador).

- ***getPrimaries.*** Esta función recibe como parámetro el nombre de una tabla y devuelve un vector conteniendo las llaves primarias de la tabla.
- ***getType.*** Esta función recibe como parámetros dos cadenas de texto representando al nombre de una tabla y de una columna respectivamente. Devuelve el tipo de dato de la columna.
- ***loadIRMasterTable.*** Esta función carga en memoria el contenido de la tabla master.

  Un ejemplo del código fuente IRMasterTable es el siguiente:

Codigo fuente IRMasterTable

1. IRMasterTable irmastertable = new IRMasterTable();
2. if (irmastertable.isDB("T1","C1")) {
3. ....

El código fuente IRMasterTable muestra en la línea 2 la utilización de la función isDB para conocer si el campo C1 prtenece a la tabla T1 y su valor se encuentra en la base de datos.

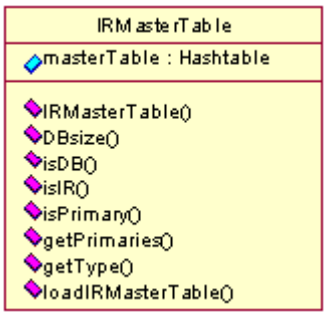

Figura C.10: Clase IRMasterTable

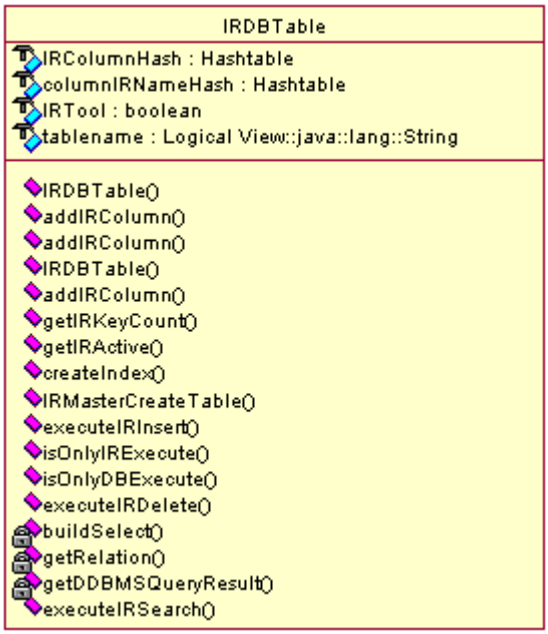

Figura C.11: Clase IRDBTable

C.12. QueryResult

Esta clase es responsable del almacenamiento de los resultados de las consultas realizadas a la base de datos, es el objeto devuelto tras realizar una consulta con la clase Select.

Entre las operaciones destacables tenemos (ver figura C.12):

getColumnNames. Este método regresa un vector conteniendo los nombres de las columnas involucradas en el resultado.

getColumnTypes. Este método regresa los tipos de las columnas involucradas en el resultado.

getQuery. Este método regresa una sentencia SQL correspondiente a la selección creada.

getString. Este método regresa el valor relacionado con el nombre de la columna pasado como parámetro.

hasNext. Este es un método iterativo, regresa verdadero si aún hay

mas elementos en el arreglo de resultados ó falso en caso contrario.

isEmpty. Este método regresa el valor verdadero su la consulta no arrojo ningun resultado, falso en caso contrario.

isFirst. Este método regresa el valor verdadero si el cursor actual apunta a la primera tupla devuelta por la consulta.

isLast. Este método regresa el valor verdadero si el cursor actual apunta a la última tupla devuelta por la consulta.

Un ejemplo del código fuente QueryResult es el siguiente:

Codigo fuente QueryResult

```
1. Select select = dbc.getSelect("T1");
2. select.column("C1");
3. select.column("C4");
4. select.where("C3", "=", "9");
5. select.where("C4", "=", "prueba");

6. QueryResult qrs = select.execute();
7. System.out.println("Documents found: "+ qrs.size());
8. while (qrs.next()) {
9. String valor = qrs.getString(Ç4");
10. System.out.println("S1 El valor:"+ valor);
11. }
```

La primera parte del código QueryResult pertenece a la operación Select explicada en el apartado anterior, porlo que solo nos limitaremos a explicar la segunda parte a partir de la línea 6. En la línea 6 se ejecuta la operación Select y el resultado de la consulta es guardado en un objeto de tipo QueryResult llamado qrs, en la línea 7 mandamos a pantalla la cantidad de tupas retornadas por el query, y a partir de la línea 8 a la 11 generamos un ciclo para recorrer cada una de las tuplas e imprimir, el ciclo se forma preguntando inicialmente si exiten más tuplas a partir del cursor actual, si esto es verdadero imprimimos el valor de la columna C4. Cuando la operación qrs.next() en la línea 8 retorne un valor falso, significa que se ha llegado al final del conjunto de tuplas y ya no entra al ciclo.

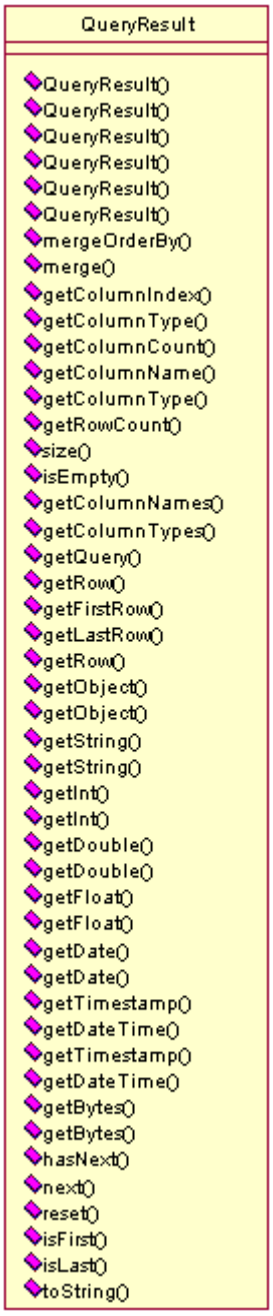

Figura C.12: Clase QueryResult

C.13. Select

La clase Select esta encargada de la formación de consultas a la base de datos. Entre sus funciones destacables tenemos (ver figura C.13).

getColumns. Este método devuelve un vector conteniendo a las columnas que regresara el query.

getColumnSortOrder. Este método devuelve un vector conteniendo a las columnas incluidas en el orden de resultados, correspondiente al ORDER BY de SQL.

getStatement. Este método regresa una sentencia SQL correspondiente a la selección creada.

getTables. Este método devuelve un vector conteniendo a las tablas involucradas en la consulta.

selectWhereClause. Este método devuelve un valor verdader en caso de que ya este formada una cláusula where, en caso contrario devuelve falso.

where. Este método recibe tres parámetros (una cadena representando la columna de la tabla incluida en la condición, el operador de la condición y el valor utilizado durante la condición) que indican la condición necesaria para la selección de información.

Un ejemplo del código fuente Select es el siguiente:

Codigo fuente Select

```
1. Select select = dbc.getSelect("T1");
2. select.column("C1");
3. select.column("C4");
4. select.where("C3", "=", "9");
5. select.where("C4", "=", "prueba");
```

El código fuente Select tiene como propósito seleccionar de la tabla T1 las columnas C1 y C4 en aquellas filas donde la columna C3 es igual a 9 y la columna C4 es igual a la cadena "prueba".

En la línea 1 podemos observar que la información será buscada en la tabla T1, en la línea 2 y 3 se selecciona la información contenida en las columnas

C1 y C4, y en las líneas 4 y 5 se forma la clausuala where indicando que la columna C3 debe ser igual a el valor 9 y la columna C4 debe ser igual a la cadena “prueba”.

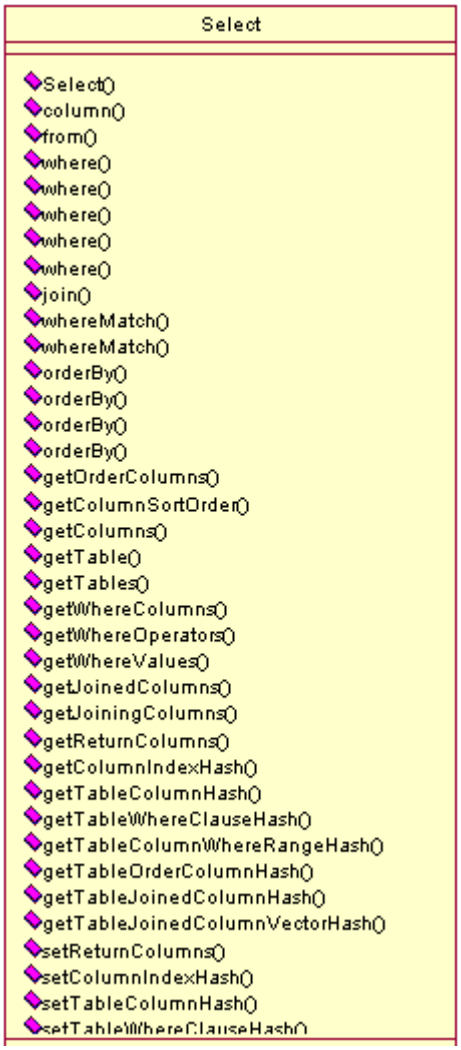

Figura C.13: Clase Select

C.14. SQLOperation

Esta clase representa un interfaz para las clases de Insert, Delete, Select y Update.

En el cuadro 4.1 se muestran los identificadores para cada una de las clases que implementan SQLOperation.

Entra las operaciones de la clase SQLOperation tenemos:

execute. Este método genera la ejecución SQL de las clases que implementan la interfaz.

getPreparedStatement. Función virtual definida en los hijos de la clase que genera el SQL de la operación existente en ese momento.

setOperation. Declara el tipo de operación a realizar basándose en el cuadro 4.1

setTable. Este método declara la tabla sobre la cual se efectuarán las operaciones.

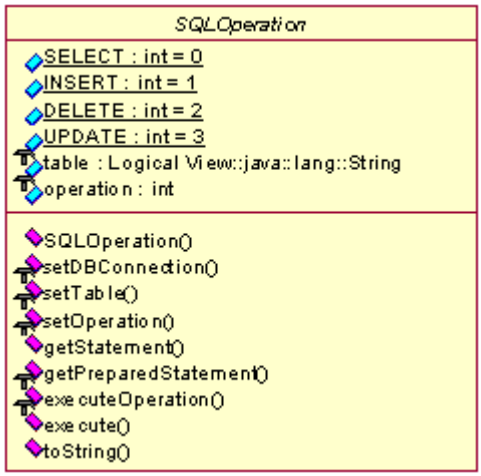

Figura C.14: Clase SQLOperation

C.15. SQLSchemaOperation

Esta es una clase abstracta y constituye la interfaz de las funciones AlterTable, CreateTable y DropTable vistas anteriormente, define las siguientes funciones (ver figura C.15).

setDBConnection. Este método asigna un objeto DBConnection a la operación actual.

setTable. Esta operación asigna una cadena conteniendo el nombre de la tabla sobre la cual desea realizarse alguna operación.

setOperation. Este método establece el tipo de operación que desea realizarse. Pueden ser tres tipos de operaciones asignadas a un ID como se muestra en la siguiente tabla C.2. Donde para asignar una operación de CREATE TABLE, DROP TABLE y ALTER TABLE se pasa un parámetro de 0, 1 y 2 respectivamente.

ID	Operación
0	CREATE TABLE
1	DROP TABLE
2	ALTER TABLE

Cuadro C.2: SQLSchemaOperation. Tipos de Operaciones

executeOperation. Esta operación realiza la ejecución de la operación establecida.

toString. Esta operación devuelve la sentencia SQL de la operación seleccionada.

C.16. Update

La clase Update es utilizada para la actualización de información. Mantiene 5 vectores para el manejo de información: columns, values, whereColumns, whereOperators y whereValues, donde se guardan los nombres de las columnas, los nuevos valores de las columnas, las columnas involucradas en las condiciones para la actualización de información, los operadores de estas condiciones y los valores utilizados.

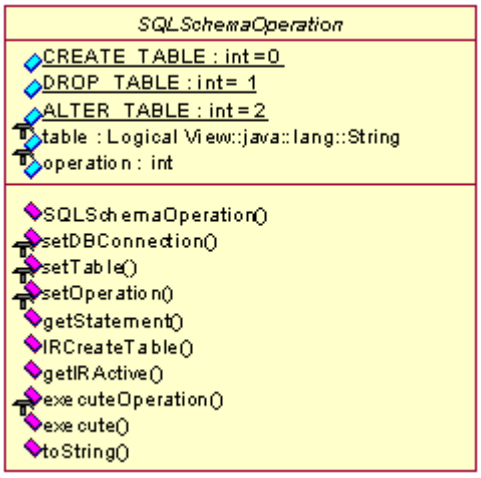

Figura C.15: Clase SQLSchemaOperation

Entre las funciones principales de la clase Update tenemos (ver figura C.16):

value. Este método se encarga de asignar un valor a una columna determinada, sus parámetros son el nombre de la columna de la tabla y el valor asignado.

where. Este método recibe tres parámetros (una cadena representando la columna de la tabla incluida en la condición, el operador de la condición y el valor utilizado durante la condición) que indican la condición necesaria para la actualización de información.

whereMatch. Este método se utiliza para la actualización de información basando en las búsquedas de texto completo.

executeOperation. Este método se encarga de realizar la actualización de información dentro de la tabla generando el SQL de inserción y ejecutándolo en la base de datos.

getStatement. Este método devuelve una sentencia SQL correspodiente a la actualización de información.

Un ejemplo del código de actualización es el siguiente:

Codigo fuente Update

1. Update update = dbc.getUpdate("T1");

2. update.value("C1", 11);
3. update.where("C1", "=", 9);
4. update.execute();

El código fuente Update cambia el valor de la columna C1 de 9 a 11 en la tabla T1. En la línea 1 se obtiene la referencia a la tabla T1, en la línea 2 se asigna el nuevo valor, en la línea 3 se coloca la condición de actualización y en la línea 4 se realiza la ejecución de la operación.

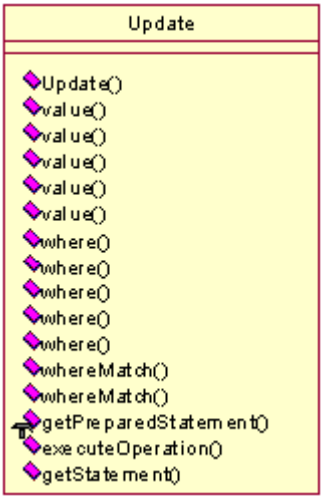

Figura C.16: Clase Update

C.17. IRIndex

En la figura C.17 se representa el diagrama de clases IRIndex. Entre las principales funciones destacan:

- ***newindex.*** Esta función tiene como objetivo la creación de un nuevo índice en la IRTool. Recibe como parámetro una cadena de texto representando el nombre del nuevo índice.

  Un ejemplo del código fuente newindex es el siguiente:

Codigo fuente newindex
1. IRIndex IRTool = new IRIndex();
2. IRTool.newindex("T1");

El código newindex crea un nuevo índice llamado T1 en la línea 2, mientras que en la línea 1 instancía un objeto de tipo IRIndex llamado IRTool, que contiene la funcionalidad del IRAPI.

- ***index.*** Esta función se encarga del indexamiento de directorios o archivos de manera recursiva.

  Un ejemplo del código fuente index es el siguiente:

Codigo fuente index
1. IRIndex IRTool = new IRIndex();
2. IRTool.index("C:/T1","C:/Mis Libros");

En la línea 1 código fuente index se crea un objeto de la clase IRIndex, mientras que en la línea 2 se ordena indexar aquellos documentos que se encuentren en C:/Mis Libros y que sean guardados en el índice C:/T1.

- ***deleteDocument.*** Esta función se encarga de eliminar documentos del índice.

  Un ejemplo del código fuente deleteDocument es el siguiente:

Codigo fuente deleteDocument
1. IRIndex IRTool = new IRIndex();
2. IRTool.deleteDocument("C:/T1",new Vector("ID"),new Vector("123456"));

El código fuente deleteDocument eliminará a todos aquellos documentos que su valor clave ID coincida con la cadena 123456.

- ***search.*** La función Search recibe como parámetro la ubicación del índice sobre el cual se realizará la operación de búsqueda, y un par de vectores conteniendo los metadatos del documento con un valor asociado.

Un ejemplo del código fuente search es el siguiente:

Codigo fuente search
1. IRIndex IRTool = new IRIndex();
2. IRTool.deleteDocument("C:/T1",new Vector("autor"),new Vector("Hurtado"));

En el código fuente search se buscan los documentos cuyo autor contenga la palabra Hurtado.

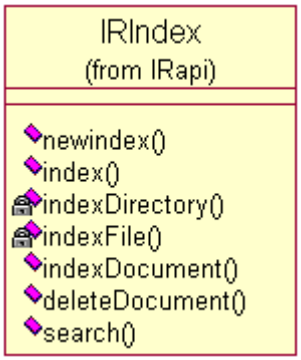

Figura C.17: IRAPI

```
package IRapi;

import org.apache.lucene.index.*;
import org.apache.lucene.search.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.store.*;
import org.apache.lucene.analysis.*;
```

```
import java.util.Vector;

import java.io.File;
import java.io.IOException;
import java.io.FileReader;

public class IRIndex {
// Variables
public static void newindex(String sindexDir) throws IOException {
File indexDir = new File(sindexDir);
File dataDir = new File(sindexDir);
index(indexDir, dataDir);
}

public static void index(File indexDir, File dataDir) throws IOException {

IndexWriter writer = new IndexWriter(indexDir, new StandardAnalyzer(),
true);
indexDirectory(writer, dataDir);
writer.close();
}

private static void indexDirectory(IndexWriter writer, File dir)
throws IOException {
File[] files = dir.listFiles();

for (int i = 0; i < files.length; i++) {
File f = files[i];
if (f.isDirectory()) {
indexDirectory(writer, f); // recurse
} else if (f.getName().endsWith(".txt")) {
indexFile(writer, f);
}
}
}

private static void indexFile(IndexWriter writer, File f)
throws IOException {
```

```
System.out.println("Indexing " + f.getName());

Document doc = new Document();
// doc.add(Field.UnStored("contents", new FileReader(f)));
// doc.add(Field.Keyword("filename", f.getCanonicalPath()));
writer.addDocument(doc);
}

public static void indexDocument(String indexDir, Vector keywords,
Vector kvalues, Vector contents, Vector cvalues) throws IOException {
IndexWriter writer = new IndexWriter(indexDir, new StandardAnalyzer(),
false);

Document doc = new Document();
StringBuffer allContent = new StringBuffer();

// Add Keywords
for (int i = 0; i < keywords.size(); i++) {
doc.add(Field.UnIndexed(keywords.get(i).toString(), kvalues.get(i)
.toString()));
allContent.append(keywords.get(i).toString() + ":"
+ kvalues.get(i).toString());
allContent.append(" ");
}

// Add contents
for (int i = 0; i < contents.size(); i++) {
doc.add(Field.UnStored(contents.get(i).toString(), cvalues.get(i)
.toString()));
allContent.append(contents.get(i).toString() + ":"
+ cvalues.get(i).toString());
allContent.append(" ");
}

// Add All
doc.add(Field.UnStored("contents", allContent.toString()));

writer.addDocument(doc);
```

```
writer.close();

}

public static void deleteDocument(String indexDir, Vector keywords,
Vector kvalues) throws IOException {
IndexReader reader = null;
reader = IndexReader.open(indexDir);
reader.close();
}

public static Hits search(String indexDir, Vector columns, Vector values) {
Hits hits = null;
try {
String stringSearch = "";
IndexReader reader = null;
reader = IndexReader.open(indexDir);
Analyzer analyzer = new StandardAnalyzer();
BooleanQuery query = new BooleanQuery();
Directory fsDir = FSDirectory.getDirectory(indexDir, false);
IndexSearcher searcher = new IndexSearcher(fsDir);

// Create the boolean query
for (int i = 0; i < columns.size(); i++) {

// Check the blank spaces
String[] bspc = values.get(i).toString().split(" ");
for (int j = 0; j < bspc.length; j++) {
TermQuery term = new TermQuery(new Term(columns.get(i)
.toString(), (String) bspc[j]));
query.add(term, true, false);
}
}

hits = searcher.search(query);
// System.out.println("Los documentos encontrados son:");
/*
 * for (int i = 0; i < hits.length(); i++) { Document doc =
```

```
 * hits.doc(i); System.out.println(doc.get("contents")); }
 */
reader.close();
} catch (Exception e) {
System.out.println(e.getMessage());
e.printStackTrace();
}
return hits;}}
```

Apéndice D

Implementación del DSAPI

Para el desarrollo de estas pruebas se requiere tener instalada y configurada la versión más reciente de la base de datos MYSQL[37]. El J2SDK versión 1.4[25] con el path de ejecución apropiadamente configurado, y se debe de contar con la herramienta de indexación a utilizar (en este caso Lucene [31]) con su path de ejecución configurado.

En la base de datos se debe contar con la tabla master creada, con los atributos que se indican en el cuadro 4.2.

Además de esto se debe de contar con el material textual a indexar.

Dando por hecho lo anterior, se debe de crear una sencillo programa que utilice los servicios del DSAPI. a continuación se expone el código fuente de un software de prueba documentado:

código CLIENTEDSAPI

```
//se declaran los paquetes que se van a utilizar
import java.io.DataOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.*;
import java.sql.*;
```

```
import java.util.*;

public class DSLcreatetable {

public static void main(String[] args) {

//Se estable la conexión conla base de datos
String databaseDriver = new String("com.mysql.jdbc.Driver");
String databaseName = new String("jdbc:mysql://localhost:3306/dslucene");
String databaseUser = new String("root");
String databasePassword = new String("");
DBConnection dbc;

//Se crea el objeto DBConnection
dbc = new DBConnection(databaseDriver, databaseName, databaseUser,
databasePassword);

//Si se quiere insertar datos, colocar esta
//variable a true, falso en caso contrario
boolean create = true;

if (create) { // Start 1st time

//Se crea la tabla T1
//con la columna C4 indexada por la IRTool
CreateTable ct = dbc.getCreateTable("T1");
ct.addColumnAsKey("C1", "INTEGER");
ct.addColumnAsKey("C2", "INTEGER");
ct.addColumn("C3", "INTEGER");
ct.addIRColumn("C4", "TEXT");
ct.execute();

// Memory copy of the master tables
dbc.setIRMasterTable();
// Probamos la primera cadena de prueba
Random generator = new Random();
String path = args[0];
```

```
try {
// call the function
//Funcion que lee archivos del FileSystem
//Y los inserta en el DSAPI
indexDirectory(generator, dbc, new File(path));
} catch (Exception e) {
e.printStackTrace();
}// end catch

//Insertamos un registro en la tabla T1
Insert insert = dbc.getInsert("T1");
insert.value("C1", 1);
insert.value("C2", 2);
insert.value("C3", 3); insert.value("C4",
"Primera cadena de Prueba");
QueryResult qr = insert.execute();

//Comprobamos la insercion
  if ( !(qr.next() && (qr.getInt("ROW_COUNT") > 0))) {
  System.out.println("Error en insercion..."); }

} // if (create)
else {
// Memory copy of the master tables
dbc.setIRMasterTable();

}

// Buscamos con el indexador.
 Select select = dbc.getSelect("T1");
 select.column("C1");
 select.column("C4");
 select.where("C3", "=", "9");
 select.where("C4", "=", "prueba");

//Muestra a pantalla el registro encontrado
 QueryResult qrs = select.execute();
 System.out.println("Documents found: " + qrs.size());
```

```
 while (qrs.next()) {
 String valor = qrs.getString("C4");
 System.out.println("S1 El valor:" + valor);
 }

// Borrando el documento donde la columna C4
//contenga la subcadena prueba
Delete delete = dbc.getDelete("T1"); delete.where("C1", "=", 9);
delete.whereMatch("C4", "+" + "\"" + "prueba" + "\"",
Delete.IN_BOOLEAN_MODE); delete.execute();

}

//Función que lee del FileSysem los documentos
//referenciados por la variable path y los guarda
//con el DSAPI
private static void indexDirectory(Random generator, DBConnection dbc,
File dir) throws IOException {
String path = "/home/luis/alldata/quijote/";
String currentDocument = "2donq10.txt";
String info = new String();

//Lee recursivamente dentro de directorios los archivos
//con extension .txt
File[] files = dir.listFiles();
for (int i = 0; i < files.length; i++) {
File f = files[i];
if (f.isDirectory()) {
indexDirectory(generator, dbc, f); // recurse
} else if (f.getName().endsWith(".txt")) {
try {

// File inputFile = new File(f.getPath(),f.getName());
File inputFile = new File(f.getPath());
FileInputStream input = new FileInputStream(inputFile);

byte[] textData = new byte[(int) inputFile.length()];
input.read(textData, 0, (int) inputFile.length());
```

```
input.close();
// System.out.println("Indexando documento...." +
// f.getPath());

//Inserta nuevo documento, con las variables
//enteras generadas al azar.
try {
info = new String(textData);
info = info.replaceAll("'", "");
Insert insert = dbc.getInsert("T1");
insert.value("C1", generator.nextInt());
insert.value("C2", generator.nextInt());
insert.value("C3", generator.nextInt());
insert.value("C4", info);
QueryResult qr = insert.execute();
if (!(qr.next() && (qr.getInt("ROW_COUNT") > 0))) {
System.out.println("Error en insercion...");
}
}catch(Exception e){

} catch (Exception e) {
System.out.print(e);
System.out.println("No existing table to delete");
}// end catch

} catch (Exception e) {
e.printStackTrace();
}// end catch
// indexFile(writer, f);
}}}}
```

El código fuente CLIENTEDSAPI muestra la funcionalidad del DSAPI, en este código se tienen inserciones, búsquedas y borrado de información.

Es importante mencionar que el código CLIENTEDSAPI es un código beta de prueba, en programas completos el código aparecerá más limpio. Con este software y ligeras modificaciones se realizaron las pruebas sobre el DSAPI.