INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN TECNOLOGÍAS DE
INFORMACIÓN Y ELECTRÓNICA

T E S I S

MAESTRÍA EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

Un enfoque de adaptación de contenido basado en la extracción automática de keyphrases para bibliotecas digitales en ambientes móviles

por

Miguel Angel Escoffié Puerto

Monterrey, N.L., Mayo de 2006

Un enfoque de adaptación de contenido basado en la extracción automática de keyphrases para bibliotecas digitales en ambientes móviles

por

Miguel Angel Escoffié Puerto

T e s i s

Presentada al Programa de Graduados en Tecnologías de Información y Electrónica

del

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

como requisito parcial para obtener el grado académico de

Maestro en Ciencias

en

Tecnología Informática

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L., Mayo de 2006

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

División de Graduados en Tecnologías de Información y Electrónica
Programa de Graduados en Tecnologías de Información y Electrónica

Los miembros del comité de tesis recomendamos que la presente tesis del Ing. Miguel Angel Escoffié Puerto sea aceptada como requisito parcial para obtener el grado de **Maestro en Ciencias en Tecnología Informática**.

Comité de Tesis

Dr. David A. Garza Salazar
Asesor principal

Dr. Juan Carlos Lavariega
Sinodal

Dr. José Luis Aguirre
Sinodal

Dr. David A. Garza Salazar
Director de los Programas de Graduados en Tecnologías de Información y Electrónica

Mayo de 2006

Resumen

Proveer los servicios de una biblioteca personal digital a usuarios de ambientes móviles involucra modificar la naturaleza de los servicios de esta para adecuarlos a las restricciones inherentes en el cómputo móvil. En este trabajo de tesis se presentan técnicas de adaptación de contenido mediante la extracción automática de *keyphrases* para bibliotecas personales digitales en ambientes móviles. Esta adaptación de contenido se hace obteniendo automáticamente información de los documentos digitales con la finalidad de presentar dicha información como sustituto del contenido del documento, de manera de que al ser presentada ésta información y el título del documento a un usuario, le permitan determinar la relevancia del mismo con respecto a una búsqueda realizada.

La extracción automática de *keyphrases* se realiza utilizando un sistema denominado KEA, al cual se le hicieron modificaciones con la finalidad de paralelizar el algoritmo mediante el uso de *threads* y proporcionarle mecanismos para interactuar con documentos almacenados en una base de datos relacional (e.j: MySQL). Las *keyphrases* extraídas automáticamente son usadas por un prototipo de cliente móvil desarrollado para dispositivos con el sistema operativo *Pocket PC* de Microsoft. Para la implementación de este prototipo se diseñó he implementó un *framework* tipo MVC (*Model-View-Controller*). El prototipo proporciona las funcionalidades básicas de un cliente móvil de acceso a bibliotecas personales digitales (e.j: autenticación, navegación de colecciones y búsqueda de documentos) y propone dos formas de utilización de las *keyphrases* en los resultados de una búsqueda de documentos. En una de las formas se muestran los títulos y las *keyphrases* de los documentos; en la segunda forma se muestra el título de cada documento y, al seleccionar alguno de la lista, se muestran sus *keyphrases* y los metadatos asociados al mismo.

Se realizaron experimentos para evaluar el algoritmo de extracción automática de *keyphrases* de KEA en paralelo, comparado con la versión original en secuencial del mismo. Las métricas evaluadas con estos experimentos son: el rendimiento en términos de tiempo de ejecución y el rendimiento en términos de precisión. La precisión se evaluó de forma exacta (comparación entre las palabras clave asignadas previamente por el autor contra las obtenidas automáticamente por el algoritmo) y de manera subjetiva (bajo criterio humano) . Estos experimentos fueron realizados con diferentes configuraciones de la versión en paralelo (variando el número de *threads* a usar) con la finalidad de determinar de manera empírica el número de *threads* adecuado para nuestros objetivos. Los resultados obtenidos indican que el número adecuado de *threads* a usar es de 2, ya que con este se presenta una mejora considerable en el rendimiento en términos de tiempo de ejecución, y la precisión exacta y subjetiva tienen la menor pérdida (en promedio 13.72 % y 0.43 % respectivamente) comparada con la obtenida usando más *threads*.

Dedicatoria

A mi madre,
que con su ejemplo y dedicación,
me mostró que el único camino
para cumplir nuestras metas es
el que nosotros mismos labramos.

A mi tío Jesús,
que me enseñó con su ejemplo
que lo más preciado que un hombre puede tener
es su familia,
que en Paz descanses.

A mi hermana,
por su apoyo y cariño,
aunque hayas sacado 7 de promedio en la prepa.

A Olga,
por tu apoyo, comprensión y amor incondicional
que me has demostrado todo este tiempo,
gracias por aguantarme.

Agradecimientos

A Dios,
porque a pesar de todo,
me demostraste que sigues conmigo,
mandándome esa luz que me hacía falta
para retomar mi camino.

Al Dr. David Garza,
por todo el tiempo dedicado y los consejos
que me ayudaron a llevar a buen término este trabajo de tesis,
como persona es un ejemplo a seguir.

A mis amigos y compañeros de trabajo,
Hugo Ortega, por compartir sus excelentes conocimientos en programación.
Héctor Ceballos, por sus contribuciones y por permitirme
dedicar el tiempo necesario a este documento,
Francisco Álvarez y el grupo de desarrollo de PDLib,
por sus valiosas contribuciones.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Una Biblioteca Digital proporciona acceso en línea a un gran número de recursos digitalizados (texto, imágenes, video, sonido, etc) de una manera integrada. Actualmente existen bibliotecas digitales de propósito general [56, 28, 58, 59] que proveen interfaces uniformes para proporcionar acceso a la información de manera transparente. Algunos de los servicios básicos que ofrecen las bibliotecas digitales son [2]:

- Creación de documentos
- Clasificación e indexamiento
- Búsqueda y recuperación
- Distribución
- Administración y control de acceso
- Personalización

Los usuarios de una biblioteca digital que acceden a dichos servicios no solo lo hacen a través de redes alámbricas fijas (LAN, WAN), sino que también la acceden a través de computadoras móviles en redes inalámbricas. Proveer servicios de acceso inalámbrico a las bibliotecas digitales es deseable debido a los diferentes usos que se le pueden dar (e.j: como una biblioteca digital personal).

El cómputo de ambientes móviles (Tablet PC, Pocket PC, Celulares) se caracteriza por tener más limitantes que el cómputo de alto rendimiento (computadoras portátiles y de escritorio). Estas limitantes son [43]:

- Riesgos de seguridad
- Limitaciones de poder de procesamiento, memoria y tamaño de pantalla
- Conectividad limitada

- Energía limitada

Proveer los servicios de una biblioteca digital a usuarios móviles involucra modificar la naturaleza de los servicios para adecuarlos a las restricciones inherentes en el cómputo móvil [7]. Las limitaciones de poder de procesamiento y memoria se han ido reduciendo debido a las mejoras constantes de dichas características, sin embargo, el tamaño de pantalla sigue siendo uno de los principales problemas a enfrentar, ya que los dispositivos se siguen haciendo cada vez más portables y por consiguiente, pequeños. Estas restricciones de pantalla representan un reto importante debido a que la información a presentar en estos dispositivos necesita ser *estructurada* y *convertida* de manera que pueda ser mostrada en el dispositivo[39].

El problema de *estructurar*, *convertir* y *visualizar* el contenido de paginas web o de bibliotecas digitales ha sido ampliamente investigado en diferentes artículos [10, 22, 12, 32, 62, 44]. La sumarización de contenido es una de las principales formas de atacar el problema del tamaño de pantalla en los dispositivos móviles. Una técnica muy usada para la sumarización de contenido es la extracción automática de *keyphrases*, la cual se describe y evalúa en diferentes trabajos [17, 60, 47, 55, 25, 24, 26]. Este trabajo de tesis está relacionado con el proyecto PDLib [4]. PDLib es una biblioteca personal digital que proporciona acceso universal a los repositorios de documentos de propósito general (bibliotecas personales digitales). El acceso es *universal* debido a que permite a los usuarios acceder a sus bibliotecas digitales personales desde una amplia gama de dispositivos de cómputo, que van desde las computadoras personales, portátiles, Tablet PC, PDA's y hasta celulares. De estos tipos de dispositivos, las PDA's y los celulares son los que ofrecen un mayor reto debido a sus limitaciones de recursos disponibles (procesamiento, memoria, visualización, etc).

La *motivación* de este trabajo de tesis es la integración de éstas tres áreas de la computación (Bibliotecas Personales Digitales, Cómputo Móvil y Adaptación de Contenido) en un mismo proyecto. Nuestra *hipótesis* es que haciendo uso de técnicas de adaptación de contenido se puede proporcionar información adicional de los documentos de una biblioteca personal digital que, al acceder a ellos desde un dispositivo móvil (e.j: PDA), puedan ayudar en el proceso de discriminación de los mismos. Esta *hipótesis* nos llevó a definir el *objetivo* de diseñar e implementar servicios eficientes de extracción automática de *keyphrases* en un sistema de bibliotecas personales digitales existente (proyecto PDLib), para después hacer uso de este servicio en un prototipo para clientes móviles (e.j: PDA) que proporcione alternativas de presentación de información adicional de los documentos obtenidos en el resultado de una búsqueda en la biblioteca personal digital. Para la realización de este objetivo, se identificó que se tenían que desarrollar principalmente las siguientes actividades:

1. La extracción automática de *keyphrases* eficiente en términos de tiempo de ejecución y su integración con PDLib. Debido a que PDLib es un sistema diseñado para proporcionar los servicios de biblioteca personal digital a un gran número de usuarios concurrentes, es importante realizar la extracción automática de *keyphrases* en el menor tiempo posible.

2. El desarrollo de un prototipo para dispositivos móviles (PDA's) que haga uso de este servicio de extracción automática de *keyphrases* para proporcionar alternativas de presentación de información adicional de los documentos en el resultado de una búsqueda.

Para resolver la primer actividad, se utilizó un algoritmo de extracción automática de *keyphrases* denominado KEA[60], el cual está implementado en *Java* y tiene el código fuente disponible para descarga en *http://www.nzdl.org/Kea/*. A esta implementación de KEA, se le hicieron modificaciones con el objetivo de paralelizar, mediante el uso de *threads*, el algoritmo de extracción automática de *keyphrases*, además de agregarle conectividad con base de datos debido a que los documentos de las bibliotecas personales digitales de PDLib se almacenan siguiendo un modelo estructurado, por lo que se requiere un manejador de bases de datos. Se realizaron experimentos con la finalidad de evaluar el desempeño del algoritmo de KEA en paralelo en términos de rendimiento en tiempo de ejecución, encontrándose un *speedup superlineal* de un 3.12 y una *eficiencia* de 1.54, lo que indica una mejora considerable con respecto al algoritmo original (secuencial).

Se realizó también experimentación para evaluar el algoritmo paralelo en términos de precisión exacta y precisión subjetiva, encontrándose una pérdida promedio de un 13.72 % y 0.43 %, respectivamente. En la precisión exacta se evalúa la cantidad de *keyphrases* que son iguales a las que el autor del documento seleccionó, pero no evalúa si las que no son iguales son correctas o no, como en el caso de la precisión subjetiva, la cual es determinada por personas con conocimientos en el área que evalúan si dichas *keyphrases* son de relevancia para el documento. Debido a estos resultados y tomando como referencia la evaluación subjetiva, podemos determinar que la pérdida de precisión del algoritmo paralelo es mínima (0.43 %) comparado con la ganancia en términos de tiempo de ejecución del mismo.

La versión de KEA en paralelo se integró a PDLib haciendo las modificaciones necesarias a el componente principal, el cual puede ser usado por cualquier cliente registrado en el sistema (PC's, Laptop's, PDA's, etc).

Para el uso de las *keyphrases* extraídas automáticamente, se desarrolló un prototipo para PDA's en el cual se proponen dos alternativas de visualización de resultados de una búsqueda de documentos en una biblioteca personal digital. En la primer alternativa (denominada vista *List + Keyphrases*), se presentan los resultados de la búsqueda con una lista de documentos, en donde para cada documento, se agrega un renglón con las *keyphrases* del mismo.

La segunda alternativa (denominada vista *List + Metadata*), presenta una lista de documentos utilizando sólo una parte de la pantalla del dispositivo, dejando el resto de ella para mostrar los metadatos del documento, iniciando por las *keyphrases*.

El prototipo tiene también la opción de mostrar los resultados de la búsqueda de forma tradicional, esto es, usando una lista plana con los nombres de los documentos. En esta vista tradicional, el usuario puede ver los metadatos de un documento seleccionado mediante el uso de un menú tradicional o un menú contextual.

Las contribuciones de este trabajo son la disminución en el tiempo de ejecución del algoritmo de extracción automática de *keyphrases* de KEA, la cual se logró con la integración de *threads*. La segunda contribución de este trabajo de tesis es la implementación de un cliente móvil para *Pocket Client* en el proyecto PDLib, el cual hace uso de las *keyphrases* extraídas automáticamente en PDLib por KEA, para proponer opciones de adaptación de contenido basándose en el uso de *keyphrases*. En esta implementación de cliente móvil, se resalta el diseño de un framework basado en capas (Model-View-Controller), el cual permite aplicar el uso de estándares de desarrollo para equipos de cómputo con mayor poder de procesamiento (e.j: PC) en dispositivos con menos recursos (e.j: PDA).

El resto de este trabajo de tesis está organizado de la siguiente manera: en el capítulo dos se describen los antecedentes y el trabajo relacionado; el capitulo tres describe las modificaciones realizadas para la paralelización del algoritmo de extracción automática de *keyphrases* de KEA, además se presentan los resultados de los experimentos realizados para evaluar el rendimiento del mismo; el capítulo cuatro describe la arquitectura de PDLib y la integración de KEA con el *Data Server*, y presenta el prototipo de cliente móvil para PDA's; por último, en el capítulo cinco se presentan las conclusiones y el trabajo futuro.

Capítulo 2

Antecedentes

En este capítulo se describen las bibliotecas digitales, sus características y los servicios que proporcionan. Se presentan algunos trabajos existentes sobre bibliotecas digitales y bibliotecas digitales personales, para después describir el proyecto PDLib (*Personal Digital Library*), en el cual se desarrolló este trabajo de tesis. Se describen también algunos trabajos sobre sistemas de extracción automática de *keyphrases*, ya que forman una parte importante del desarrollo del presente trabajo. Por último se presentan los trabajos relacionados con el uso de bibliotecas digitales en ambientes móviles y la utilización de *keyphrases* en estos dispositivos.

2.1 Bibliotecas digitales

Una Biblioteca Digital proporciona acceso en línea a un gran número de recursos digitalizados (texto, imágenes, video, sonido, etc) de una manera integrada. Las bibliotecas tradicionales están limitadas por el espacio, las bibliotecas digitales, por el contrario, tienen el potencial para almacenar mucha más información por el simple hecho de que la información digital requiere menos espacio de almacenamiento físico [57]. Algunos de los servicios que ofrecen las bibliotecas digitales son [2]:

- *Creación de documentos digitales.* Este servicio permite la creación de un documento digital a partir de la conversión del mismo documento almacenado en otro formato. Esto se hace con la finalidad de tener disponible diferentes versiones (formato) del mismo documento (e.j: pdf, txt, doc).

- *Clasificación e indexamiento.* Los documentos almacenados (en sus diferentes formatos) en la biblioteca digital deben ser clasificados (al igual que en una biblioteca normal) e indexados periódicamente, esto con la finalidad de mantener los servicios de búsqueda con las información más actualizada (cada que se agrega un nuevo documento, este debe ser indexado).

- *Búsqueda y recuperación.* Una biblioteca digital debe proporcionar servicios de búsqueda y recuperación de documentos de manera fácil e intuitiva para el usuario. Los mecanismos de búsqueda pueden ser variados, pero en la mayoría de las bibliotecas digitales se permite buscar por palabras contenidas en el documento (búsqueda básica) y en los metadatos del mismo (búsqueda avanzada).

- *Distribución.* Los usuarios de la biblioteca digital deben disponer, de manera rápida y segura, de los documentos almacenados en la biblioteca digital.

- *Administración y control de acceso.* Una biblioteca digital debe contar con un sistema de control de acceso a los documentos, así como una manera fácil de administrar y configurar usuarios y características de la biblioteca digital.

- *Personalización.* Las bibliotecas digitales deben proporcionar de personalización para satisfacer las preferencias de los usuarios y/o grupos de usuarios.

Algunos ejemplos de bibliotecas digitales son: *ACM Portal*[15], *Phronesis*[28], *Perseus Digital Library*[56], *New Zealand Digital Library*[58], entre otras. Estas bibliotecas digitales proveen interfaces uniformes para proporcionar a los usuarios con un acceso a la información de manera transparente (e.j: acceso mediante un navegador Web).

Existen también las bibliotecas personales digitales, las cuales proporcionan los servicios tradicionales de una biblioteca digital, con la diferencia de que a cada usuario se le proporciona un repositorio de almacenamiento de documentos personales, se le permite personalizar la clasificación de documentos (crear, recuperar, renombrar y eliminar colecciones) e interactuar con otras bibliotecas digitales (colectivas o personales).

En la siguiente sección se describe PDLib[20], la cual es una biblioteca personal digital con acceso universal.

2.1.1 PDLib

Una biblioteca personal digital está compuesta por colecciones, las cuales a su vez contienen otras colecciones y/o documentos. Los usuarios interactúan con estas bibliotecas personales digitales creando y eliminando colecciones, subiendo documentos (cargarlos a su biblioteca personal digital), moviéndolos, copiándolos o descargándolos. Además los usuarios puede definir el conjunto de metadatos que será usado para describir el contenido de cada colección.

PDLib (*Personal Digital Library System*) [20] es un sistema de biblioteca personal digital de acceso universal que permite a los usuarios personalizar la estructura de la biblioteca personal digital y accederla desde casi cualquier dispositivo de cómputo (desde computadoras personales hasta PDA's y celulares). Algunas de las características de PDLib son[4]:

- *Administración flexible de colecciones y metadatos.* Las colecciones son un mecanismo de clasificación de documentos digitales. Los usuarios de PDLib pueden crear, recuperar, actualizar y eliminar sus colecciones, además de definir el conjunto de metadatos que se usará para describir el contenido de cada colección.

- *Carga de documentos digitales.* El usuario puede agregar cualquier objeto digital (*.pdf, *.doc, *.jpg, *.mp3) a su biblioteca personal digital.

- *Búsqueda y recuperación.*Los mecanismos de búsqueda y recuperación de documentos se adaptan al esquema personalizado de clasificación definido por cada usuario en el sistema.

- *Administración y control de acceso.* Proporciona un mecanismo de identificación/autenticación de usuario, ya que cada usuario tiene acceso a su biblioteca personal digital sin restricciones. El usuario tiene la posibilidad de definir contenido público, el cual puede ser accedido por cualquier otro usuario del sistema PDLib.

- *Interoperabilidad.* Proporciona mecanismos de comunicación e intercambio de información con otros sistemas de bibliotecas digitales a través de protocolos estándar de interoperabilidad (e.j. OAI).

- *Acceso universal.* Esta es una de las características principales del sistema PDLib. PDLib proporciona diferentes tipos de aplicaciones cliente para permitir el acceso universal a los documentos y servicios de las bibliotecas personales digitales.

En la figura 2.1 se muestra un esquema general del sistema PDLib, el cual se divide en tres capas: *Client*, *Server* e *Interoperability*, separadas según la funcionalidad que ofrecen[1]:

1. *Client Tier.* En esta capa se incluyen la variedad de dispositivos con los que un usuario de PDLib puede interactuar: *PDA*, *Tablet PC*, Teléfono móvil, computadora portátil y de escritorio.

2. *Server Tier.* Infraestructura servidor que provee los servicios a los clientes. Esta capa está compuesta por el *Data Server* (DS), el *Mobile Connection Middleware* (MCM) y el *Web Front-end.*

3. *Interoperability tier.* Incluye los servicios de interoperabilidad con otros servidores PDLib y sistemas de bibliotecas digitales que cumplen con el estándar *OAI-MPH*[23].

Los dispositivos de la capa cliente se comunican con la capa del servidor para obtener acceso a los diferentes servicios de la biblioteca digital PDLib. El tipo de acceso de los

[1]Nombres tomados en inglés por consistencia con las figuras de la arquitectura de PDLib

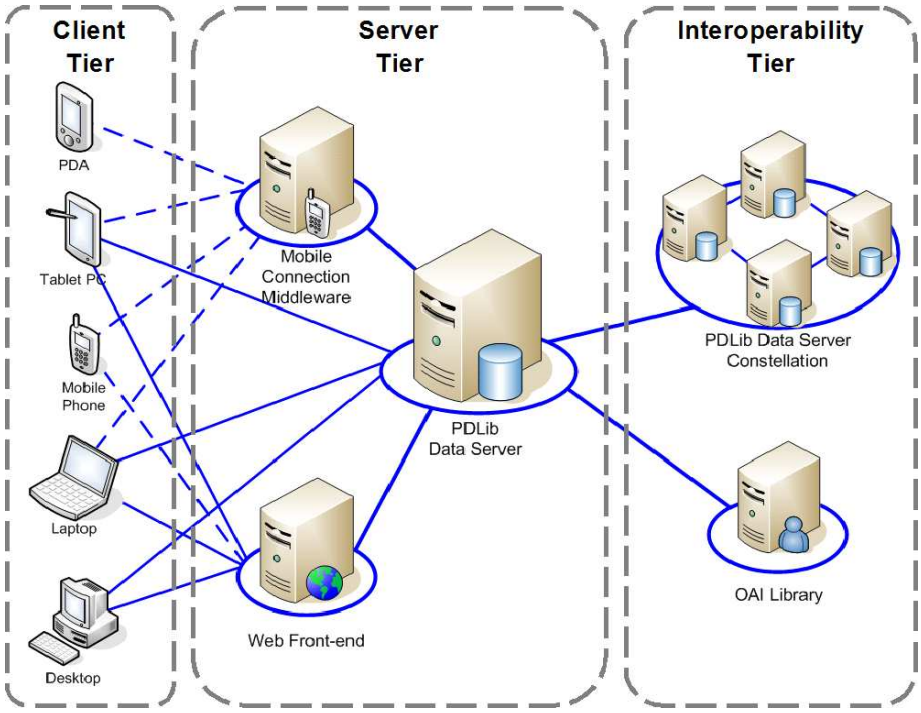

Figura 2.1: Esquema general del sistema PDLib. Figura tomada de [20].

clientes con el servidor varía de acuerdo a las características del dispositivo, esto con la finalidad de proporcionar un acceso universal a los clientes. Los accesos pueden ser de cualquiera de los siguientes tipos:

Acceso por Middleware. Da soporte a dispositivos móviles, especialmente aquellos con recursos de cómputo limitados (e.j: PDA, teléfonos móviles con soporte HTTP). Estos accesos se representan mediante las lineas punteadas entre los clientes y el *MCM* de la figura 2.1.

Acceso por Web. Provee acceso mediante HTTP a cualquier dispositivo que incluya un navegador Web (p.e: WML/HTML micronavegadores)

Acceso directo. Proporciona acceso directamente con el *Data Server* a aplicaciones con requerimientos muy particulares. Los accesos directos están representados por las líneas continuas entre los clientes y el *Data Server*.

PDLib es descrita ampliamente por [4], sin embargo, debido a que el presente trabajo de tesis está muy relacionado con uno de los principales componentes del sistema PDLib, en la sección 4.1 se describe más a detalle su arquitectura.

2.2 Sumarización de contenido

El objetivo de la sumarización es tomar un texto fuente, extraerle automáticamente el contenido y presentar al usuario lo más relevante del contenido (resumen) de una manera condensada y sensitiva que cumpla con las expectativas del usuario o aplicación[42]. El resumen puede ser de dos clases:

1. *Enfocado al usuario* (user-focused). Se enfocan en los requerimientos de un usuario o grupo de usuarios en particular, también pueden ser enfocados en tópicos (topic-based) o enfocados en consultas (query-focused).

2. *Genérico*. Generalmente, estos tipos de resúmenes sirven como sustitutos en ambientes de acceso a texto completo y son escritos por los autores de los documentos o por extractores (abstractors) profesionales.

Debido a los sistemas de búsqueda en texto completo, navegación y filtrado personalizado de información, el resumen *enfocado al usuario* es el que ha tenido mayor importancia y crecimiento.

Un resumen puede ser un extracto de la información, el cual contiene solo información tomada del texto fuente; o abstracto, en el cual se tiene material (texto) que no está presente en el texto fuente.

La sumarización de contenido realizada en este trabajo de tesis es *enfocada al usuario* y está basada en la extracción automática de palabras clave. Debido a que estas palabras clave están compuestas en su mayoría de los casos por más de una palabra, usualmente se les denominan *keyphrases*.

2.2.1 Extracción automática de *Keyphrases*

Una *keyphrase* usualmente está compuesta por dos o más palabras clave (*keywords*) y se usan principalmente en los artículos científicos para dar una idea general de los temas relevantes tratados en dicho artículo. El uso de *keyphrases* tiene diferentes objetivos, entre otros [55]:

1. *Sumarización*. Cuando son usadas en la primer hoja de un artículo científico, permiten al lector determinar rápidamente si el artículo en cuestión es de determinada área de interés.

2. *Indexamiento*. Cuando son usadas en el índice de un *journal*, permiten al lector encontrar rápidamente un artículo de relevancia para una necesidad específica del lector.

3. *Búsqueda*. Cuando son usadas en lugar de una búsqueda en texto completo, permiten al lector realizar búsquedas más específicas.

La *extracción automática de keyphrases* es un caso particular de una tarea más general denominada *generación automática de keyphrases*, en la cual las *keyphrases* generadas no necesariamente aparecen en el cuerpo del texto fuente dado [55]. En la *extracción automática de keyphrases*, las *keyphrases* generadas están contenidas en el cuerpo del texto fuente.

A pesar de que las *keyphrases* son muy útiles, solo una pequeña parte de los documentos disponibles en línea (e.j: a través de bibliotecas digitales) tienen *keyphrases*, por lo que la necesidad de sistemas de extracción automática de *keyphrases* se ha incrementado. Actualmente existen algoritmos de extracción automática de *keyphrases* en productos de software comerciales, algunos de ellos son: *Word* de Microsoft (a partir de la versión 97), *Search* de Verity (desde la versión 97) y *Metabot* y *Wisebot* de Tetranet.

Existen también algoritmos de extracción automática de *keyphrases* no comerciales, algunos de los cuales usan heurísticas ([6, 18]) para extraer las *keyphrases* de un documento fuente con base en reglas sintácticas (uso de letras en cursivas, la presencia de frases en encabezados, etc); otros usan algoritmos de aprendizaje no supervisados (redes neuronales basadas en ART-*Adaptive Resonance Theory*-) [1] para descubrir *keyphrases* con longitud de dos palabras; y algoritmos con aprendizaje supervisado: *GenEx* [53, 54], en el que usan técnicas de algoritmos genéticos para el entrenamiento y la extracción de *keyphrases*, y *KEA* [17], en el que para el entrenamiento y la extracción de *keyphrases* se usa un algoritmo Bayesiano. A continuación se describen ambos algoritmos:

GenEx

GenEx es un algoritmo genético híbrido para la extracción automática de *keyphrases*. *GenEx* tiene dos componentes:

1. *Genitor*. Es un algoritmo genético que se usa sobre los documentos de entrenamiento para configurar los parámetros de entrada del algoritmo de extracción de *keyphrases* denominado *Extractor*.

2. *Extractor*. Este algoritmo toma un documento como entrada y produce una lista de *keyphrases* como salida.

Una descripción detallada y resultados de experimentos de este algoritmo se presentan en [52]. En [55] se presentan resultados de experimentos realizados con la unión de *GenEx* y un algoritmo de extracción de *keyphrases* de propósito general denominado *C*4.5.

KEA

KEA es un sistema de extracción de *keyphrases* que usa un algoritmo de aprendizaje automático conocido como *Naive Bayes* para el entrenamiento y la extracción

automática de *Keyphrases*. El mecanismo de extracción de *Keyphrases* se compone de dos fases:

1. *Entrenamiento*. En esta fase se utiliza un algoritmo para la construcción de un modelo, el cual requiere documentos de entrenamiento con las *keyphrases* previamente asignadas por el autor. La salida de esta fase es un *modelo* que se utiliza en la fase de extracción.

2. *Extracción*. En esta fase se toman como parámetros de entrada el modelo generado por la fase de entrenamiento y algunos otros parámetros opcionales (e.j: la cantidad de *keyphrases* a extraer), produciendo como salida una lista de *keyphrases*.

KEA se describe ampliamente en [60, 17]. En el capítulo 3 se presenta una descripción más detallada de las fases de entrenamiento y extracción de *keyphrases* de KEA.

De los sistemas no comerciales para la extracción automática de *keyphrases* (*GenEx* y *KEA*), se decidió utilizar *KEA* debido a que se tiene acceso al código fuente y a que en [17] realizaron experimentos que comparan el desempeño de ambos algoritmos, obteniendo resultados similares.

2.3 Cómputo móvil

El término de cómputo móvil se usa para describir aplicaciones para dispositivos pequeños, portables y con capacidades de comunicación inalámbricas. El cómputo móvil incluye, entre otros, dispositivos como computadoras portátiles con tecnología inalámbrica, teléfonos móviles, Asistentes Personales Digitales (PDAs), etc. El cómputo móvil está basado en el desarrollo de sistemas distribuidos con arquitectura cliente-servidor, en donde los clientes móviles (e.j: PDA) realizan tareas diferentes a las de los clientes fijos (e.j:PC). En [43] se definen 4 bloques para caracterizar la esencia del cómputo móvil, los cuales se describen brevemente a continuación:

1. *Restricciones de Movilidad*.

   Existen cuatro restricciones básicas que caracterizan el cómputo móvil:

   - *Uso limitado de recursos*. Debido a consideraciones como peso, energía, tamaño y portabilidad de los dispositivos, los recursos de cómputo de los dispositivos móviles siempre serán menores a los recursos disponibles en los dispositivos estáticos.
   - *Riesgos inherentes a la movilidad*. Debido a el tamaño de los dispositivos, es más fácil que se pierdan, sufran daños o sean robados.

- *Conectividad variable en desempeño y confiabilidad.* La conectividad en algunos edificios puede ser confiable y con buen ancho de banda, pero no es una garantía de las redes inalámbricas.
- *Fuente de energía limitada.* La energía proporcionada por las baterías siempre estará limitada en tiempo debido a las restricciones inherentes a los tamaños de las mismas.

2. *Necesidad de Adaptación.*

   La movilidad aumenta la tensión entre las características de autonomía e independencia de los sistemas distribuidos, si a esto se le agrega la relativa pobreza y poca robustez y confiabilidad de los dispositivos móviles, se tiene la necesidad de pedir la confianza en los servidores estáticos.

   Cualquier aproximación al cómputo móvil debe tener un balance entre el funcionamiento no confiable y el consumo de energía. Este balance no puede ser estático, por lo que los dispositivos móviles deben ser *adaptativos*, de manera que permitan reasignar las responsabilidades entre el cliente y el servidor.

3. *Taxonomía de Adaptación*

   El rango de estrategias para la adaptación de aplicaciones está limitado por dos extremos:

   - *No intervención.* En este extremo, la adaptación es responsabilidad de una sola aplicación, lo cual elimina la necesidad de soporte del sistema y evita el uso de un intermediario central (e.j: *middleware*) que resuelva las demandas de recursos incompatibles. Este tipo de aplicaciones son más complicadas de codificar y por consiguiente el costo de mantenimiento es mayor.
   - *Adaptación transparente.* Este tipo de aplicaciones le da la responsabilidad de la adaptación al sistema (usualmente mediante un *proxy* o *middleware*). Este tipo de aplicaciones son más sencillas de codificar y el costo de mantenimiento es menor, pero pueden resultar contraproducentes si el número de usuarios móviles del sistema se incrementa a un punto en el que el sistema deje de realizar su labor por realizar la adecuación.

4. *Modelo Cliente-Servidor extendido*

   En este modelo, un pequeño número de servidores confiables constituyen los repositorios de datos. Es posible acceder a estos datos de forma eficiente y segura desde una gran cantidad de clientes no confiables. Se usan técnicas de *caching* y *prefetching* para proporcionar un buen desempeño, además que se puede utilizar la autenticación y el cifrado de datos para proveer seguridad.

Tomando en cuenta las restricciones de la movilidad, es necesario replantear este modelo. Las limitaciones en los dispositivos móviles pueden requerir que las tareas que normalmente se hacían del lado del cliente se tengan que hacer del lado del servidor o incluso dividir la tarea de manera que se haga un parte en el servidor y otra en el cliente.

2.3.1 Asistentes Personales Digitales

El término de Asistente Personal Digital (PDA) es usado para hacer referencia a cualquier dispositivo móvil de tamaño pequeño que permite almacenar y procesar información para uso personal o de negocio. Originalmente fueron diseñados como organizadores personales, pero con el tiempo se han vuelto más versátiles y han crecido sus capacidades y por lo tanto sus posibles usos. Una PDA básica incluye, entre otras, las siguientes aplicaciones: reloj, agenda, libreta de direcciones, lista de tareas, bloc de notas, un reconocedor de texto y/o voz y una calculadora. La mayoría de los modelos recientes de PDA's incluyen también servicios de comunicación inalámbrica, entre los cuales se encuentran: *IrDA* [5], *Bluetooth* [21] y *Wi-Fi* [3]. Las PDAs son conocidas también como *palmtops*, *handheld computer* y *pocket computers*.

Los sistemas operativos más comúnmente usados en este tipo de dispositivos son:

- Microsoft Windows Pocket PC (basado en el Windows CE) [34]
- Palm OS [37]
- BlackBerry (de Research In Motion) [30]
- Linux, en diferentes distribuciones: GPE (Basado en GTK+/X11) [48], Qtopia (desarrollado por Trolltech) [51] y OPIE [49]
- Symbian OS (Ericsson, Panasonic, Nokia, Samsung, Siemens y Sony Ericsson) [31]

En la siguiente sección se describen las herramientas existentes para el desarrollo de aplicaciones para dispositivos móviles con sistema operativo *Windows Pocket PC*.

2.3.2 Herramientas de desarrollo para Pocket PC

Las herramientas de desarrollo existentes para la plataforma *Pocket PC* se puede catalogar de la siguiente forma:

- Microsoft .Net Compact Framework
- Java J2ME

- Microsoft eMbedded Visual Basic
- Microsoft eMbedded Visual C++
- Otras herramientas de desarrollo de terceros (p.e: Symbol SDK)

Además de Microsoft y Java, existen otras alternativas de desarrollo, como los kits de desarrollo de *BlackBerry* [30], las librerías para desarrollo en linux: Qtopia [51], OPIE [49], GPE (Basado en GTK+/X11) [48]; y las proporcionadas por Symbian para la gama de celulares de diferentes compañías (Ericsson, Panasonic, Nokia, Samsung, Siemens y Sony Ericsson). Debido a que las principales herramientas de desarrollo para Pocket PC son el *.Net Compact Framework* y *Java J2ME*, se describen a continuación.

1. *Java J2ME*. La arquitectura *Java 2 Micro Edition* descrita ampliamente en [27, 50], mejor conocida como J2ME (ver figura 2.2), define combinaciones de configuraciones y perfiles para el desarrollo de aplicaciones de acuerdo a las características de hardware de los dispositivos móviles. Actualmente las configuraciones existentes son: *Connected Limited Device Configuration* (CLDC) y *Connected Device Configuration* (CDC).

   CLDC. Esta configuración está diseñada para dispositivos con procesadores lentos y memoria limitada, además de que pueden tener una conexión a la red intermitente.

   CDC. La configuración CDC incluye una máquina virtual completa, un conjunto de librerías mayor al existente en CLDC y está pensada para dispositivos móviles con mayor poder de procesamiento, mayor memoria y conexión a red de mejor calidad.

2. *Microsoft .Net Compact Framework*

   Microsoft .Net Compact Framework (.Net CF) ofrece diferentes lenguajes para el desarrollo de aplicaciones para dispositivos móviles que cuentan con el sistema operativo Windows CE (para dispositivos con características de hardware potentes, sin llegar a tener la capacidad de una PC de escritorio) y Windows Pocket PC (*high-end* PDA's y celulares *smartphone*). Algunas de estos lenguajes de desarrollo para el .Net Compact Framework son, entre otros:

   - C#
   - Visual Basic
   - J#

   Las aplicaciones desarrolladas en .Net Compact Framework se pueden redistribuir fácilmente (sin importar el lenguaje de programación) por ser aplicaciones *byte code* que usan el CLR (*Common Language Runtime*) similar a las aplicaciones .Net de windows. La arquitectura del .Net CF se muestra en la figura 2.3.

   Al desarrollar en esta arquitectura .Net CF, se tienen las características inherentes a la plataforma del sistema Windows Pocket PC, las cuales son, entre otras:

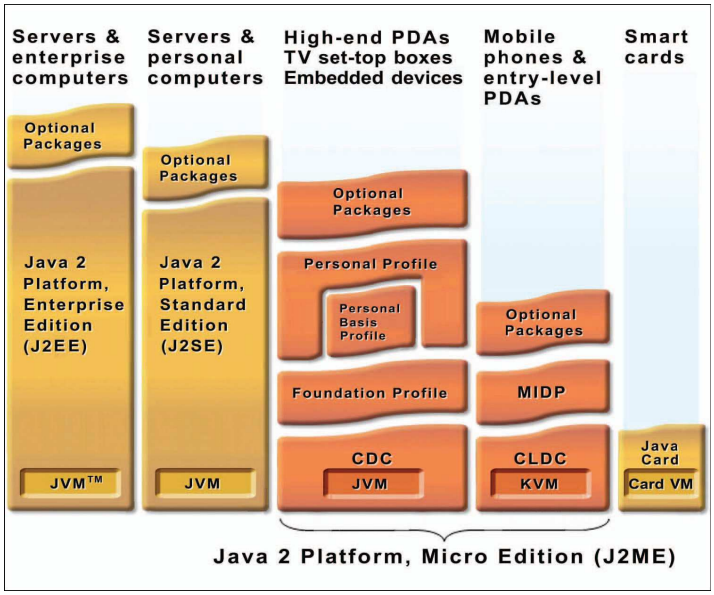

Figura 2.2: Arquitectura Java J2ME. Imagen tomada de [27].

- Soporte multitarea de hasta 32 aplicaciones corriendo simultáneamente.
- Almacenamiento de archivos con estructura jerárquica de directorios y archivos similar a la que proveen los sistemas Windows.
- Un mínimo de 64 MB de memoria.
- 4 arquitecturas de procesadores.
- Soporte multimedia para *Direct X* y *Media player*.
- Soporte de red 802.11, Bluetooth, IrDA.

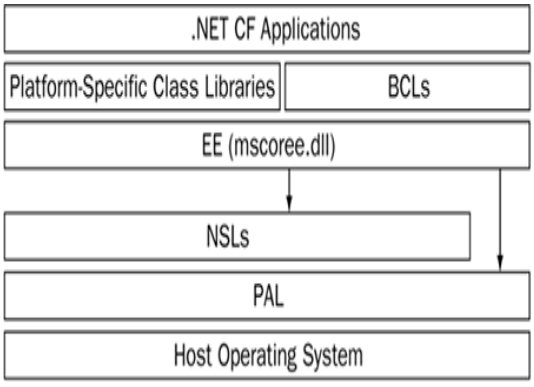

Figura 2.3: Arquitectura .Net CF. Figura tomada de [16].

2.4 Trabajo Relacionado

Debido a que el presente trabajo de tesis tiene como objetivo el uso de técnicas de sumarización de contenido como un enfoque de adaptación a contenido para bibliotecas digitales en ambientes móviles, podemos clasificar los trabajos relacionados en dos grupos:

1. El acceso a bibliotecas digitales desde dispositivos móviles.
2. Con el uso de técnicas de sumarización de contenido para dispositivos móviles.

A continuación se describen brevemente algunos trabajos relacionados de acuerdo al grupo en el que podemos clasificarlos.

2.4.1 Acceso a bibliotecas digitales desde dispositivos móviles

Con el creciente uso de los dispositivos móviles (PDAs, celulares, etc), han aumentado también la diversidad de tareas que se pueden realizar con ellos. Una de esas tareas es el poder acceder a los recursos de las bibliotecas digitales. Algunos trabajos relacionados con esta tarea de acceder a bibliotecas digitales desde dispositivos móviles son:

- *Exploring Small Screen Digital Library Access with the Greenstone Digital Library.*

  En este trabajo, George Buchanan et al [32] desarrollaron dos prototipos para acceder a la biblioteca digital de *Greenstone*. Uno de los prototipos denominado *LibTwig*, es una interfaz *HTML* que accede a la biblioteca digital desde una *PDA* (el prototipo se desarrolló para Pocket PC 2002) usando el protocolo interno de *Greenstone* basado en CORBA. La navegación se hace a través de una estructura con estilo jerárquico de categorías y subcategorías, las cuales contienen a los documentos relacionados con la búsqueda realizada.

  El segundo prototipo se utiliza para acceder a la biblioteca digital de *Greenstone* desde celulares a través del protocolo WAP. El resultado de las búsqueda se presenta con el mismo estilo jerárquico de categorías y subcategorías.

- *PoPS: Mobile Access to Digital Library Resources.*

  En este trabajo Nohema Castellanos y Alfredo Sánchez [13] desarrollaron un framework que produce interfaces que accedan a los recursos de una biblioteca digital desde PDAs y celulares vía ambientes Web personalizados. Con este framework se desarrollaron dos prototipos para el acceso a la biblioteca digital de la *Universidad de las Américas*, uno para *PDAs* y otro para celulares.

El prototipo para PDAs fue desarrollado usando *PersonaJava*^{TM}*Runtime Environment* para el sistema operativo Microsoft Windows CE 1.1. El prototipo fue probado en una *PDA* iPaq *H*3870 Pocket Pc y accede a la biblioteca digital a través de Internet.

Un segundo prototipo fue desarrollado para celulares usando *Java J2ME* y solo se ha probado en un emulador que se ejecuta en una PC y accede a la biblioteca digital por la intranet de la universidad.

- *Customizing Digital Libraries for Small Screen Devices.*

Dynal Patel y Gary Marsden [39] desarrollaron un sistema que está compuesto por componentes principales: la biblioteca digital de *Greenstone*, una herramienta de personalización y una *PDA*.

El módulo de visualización de la biblioteca digital de *Greenstone* genera el código *HTML* al vuelo, basándose en archivos de configuración que especifican como se debe visualizar el contenido de la biblioteca.

La herramienta de configuración y personalización se encarga de generar los archivos de configuración que son utilizados por el módulo de visualización de *Greenstone*. Esta herramienta provee un editor gráfico del tipo *WYSIWYG* que permite a los usuarios personalizar las páginas *HTML* (*home page*, *search page*, *document page*, etc) mostrando una pre-visualización de la página en el tamaño requerido para una *PDA*. Los archivos de configuración generados por esta herramienta son enviados via *FTP* a la instalación de la biblioteca *Greenstone*.

La *PDA* es un cliente ligero (*thin client*) que accede a las páginas *HTML* generadas por la biblioteca *Greenstone* en el formato especificado por la herramienta de configuración. El acceso se hace a través del navegador web de la *PDA* y se conecta usando una red inalámbrica para la transferencia de datos usando el protocolo *HTTP*.

2.4.2 Sumarización de contenido para dispositivos móviles

La adaptación de contenido mediante el uso de técnicas de sumarización de contenido ha sido explorada ampliamente en diferentes trabajos, algunos de los cuales son:

- *Exploring Small Screen Digital Library Access with the Greenstone Digital Library.*

En este trabajo, Orkut Buyukkokten et al [11] proponen cinco métodos para la sumarización. Estos métodos se implementaron en un browser para *PDAs* desarrollado anteriormente por ellos, el cual denominaron *Power Browser* [9]. Los

métodos se sumarización propuestos en [11] se aplican a la navegación en páginas Web, las cuales son particionadas en unidades textuales semánticas o STUs (*Semantic Textual Units*). Las STUs son fragmentos de páginas identificados por *TAGs* de *HTML* (parrafos <p>, listas /<lu> ó tags *ALT* que describen imágenes). Proponen las siguientes cinco formas de sumarización para mostrar las STUs:

1. *Incremental*. En este método, cada STU es mostrado gradualmente, usando tres estados: la primer línea, las primeras tres líneas y el STU completo.
2. *All*. En este método, el texto de la STU se muestra completamente, sin tener diferentes estados para la visualización.
3. *Keywords*. Para este método se utilizan también tres estados, los cuales se despliegan de la misma manera que en la forma *Incremental*, con la diferencia que el texto del STU son *keywords* o palabras clave que son extraídas usando técnicas de agrupación (*clustering*) con el atributo $TF \times IDF$, el cual es calculado para cada frase de un documento e indica la frecuencia de aparición.
4. *Summary*. Este método consta solo de dos estados. En el primer estado se muestra la sentencia más significativa del STU, en el segundo estado se muestra por completo el STU. Este método requiere de un módulo para hacer un *Ranking* de Frases, el cual extrae las frases más importantes de cada STU.
5. *Keyword/Summary*. Este método es una combinación de los dos anteriores y consta de tres estados. En el primer estado muestra *keywords*, el segundo estado muestra las STUs más significativas y en el tercer estado se muestra por completo el contenido de la STU.

Presentan resultados de experimentos que demuestran que el uso de *keywords* combinadas con resúmenes de una sola frase proporcionan mejoras significativas en el tiempo de acceso y en el número de acciones con la pluma en la PDA.

- *WEST: A Web Browser for Small Terminals.*

Staffan Björk et al [8] presenta en este trabajo un navegador Web para dispositivos con limitaciones de tamaño de pantallas (e.j: PDAs): *WEST*. La propuesta de WEST es que los usuarios de dispositivos móviles puedan acceder a páginas Web usando una combinación de técnicas de reducción de texto y visualización de tipo *thumbnail*. La solución a esta propuesta consta de dos elementos centrales:

1. Un *servidor proxy*, el cual se ejecuta en un servidor y toma la página en HTML estándar y en tiempo real la transforma en un formato que pueda ser procesado por la PDA para su visualización. El proceso del servidor proxy consta de tres fases:

a) *Chunking stage*. En esta fase, la página HTML es dividida en un número de diferentes páginas más pequeñas denominadas *cards*, las cuales posteriormente son agrupan en *dekcs*.

b) *Text reduction stage*. En esta fase, se extraen del texto un conjunto de *keywords* que sumaricen cada *card*.

c) *Link extraction stage*. En esta fase son extraídos todos los *hyper-links* de cada *card*.

Las *cards* resultantes de estas tres fases (con soporte de *keywords* y links) son enviadas a la aplicación cliente.

2. Una *aplicación cliente* para PDA, la cual permite al usuario ver e interactuar con las páginas Web procesadas por el servidor proxy. Esta aplicación cliente soporta los siguientes modos de visualización:

   a) *Thumbnail view*. En esta forma de visualización basada en *focus+context* se presentan vistas en miniatura para cada *card* de la página web enviada por el servidor proxy.

   b) *Keyword view*. Para cada *card*, se presenta una vista en miniatura que contiene *keywords* extraídas de dicha *card*.

   c) *Link view*. Esta forma de visualización es similar a la anterior, con la diferencia de que en lugar de *keywords*, las *cards* contienen los *links* disponibles para cada una de ellas.

Cada vista propuesta permite al usuario hacer un *zoom* completo para cada *card*, lo que provee una vista legible para leer todo el contenido de la *card*.

Además de los trabajos mencionados, existe un prototipo en el proyecto PDLib, el cual fue desarrollado con la finalidad de acceder a PDLib desde clientes móviles (e.j: PDAs). Dicho prototipo se implementó con tecnología *Java J2ME* y se describe brevemente en [4]. Este prototipo se tomó como base para este trabajo, por lo que en la sección 4.3.1 se detallan las diferencias entre el prototipo para Palm OS y el desarrollado en este trabajo de tesis.

Este trabajo se relaciona con los presentados en el sentido de el uso de alguna técnica de adaptación de contenido, en la que, al igual que en Staffan Björk et al [8] y Orkut Buyukkokten et al [11], se utiliza la extracción de *keyphrases* para obtener información relevante relacionada a un documento, pero se diferencian de estos trabajos por aplicarse en el acceso a bibliotecas digitales y no a la navegación de páginas Web, además de que los algoritmos de extracción automática de *keyphrases* son diferentes y en nuestro caso, las *keyphrases* se extraen dinámicamente, es decir, cada que se agrega un documento a la biblioteca personal digital, se le extraen las *keyphrases*. En los siguientes capítulos se describe el desarrollo de la investigación del presente trabajo, así como la experimentación realizada, los resultados obtenidos y las conclusiones a las que se llegaron.

Capítulo 3

Extracción automática de Keyphrases

La extracción automática de *keyphrases* es una de las técnicas de sumarización de contenido (ver sección 2.2) más utilizadas. Algunas de las actividades en las que los algoritmos de extracción automática de *keyphrases* se utilizan son, entre otras: la *Sumarización*, el *Indexamiento* y la *Búsqueda y recuperación* de información[55]. El problema de la *Sumarización* basado en la extracción automática de *kayphrases* puede ser resuelto con diferentes algoritmos existentes, tanto comerciales como no comerciales. En este trabajo de investigación se atacó este problema usando KEA, un algoritmo de extracción automática de *keyphrases* (no comercial) desarrollado por Ian H. Witten *et al* [60] en la Universidad de Waikato, Nueva Zelanda.

Existe también un algoritmo de extracción automática de *keyphrases* basado en algoritmos genéticos denominado *GenEx* desarrollado por Peter D. Turney [52]. La decisión de cual de los dos algoritmos utilizar (KEA o *GenEx*) se tomó considerando los resultados de los experimentos realizados por Eibe Frank *et al* [17], en donde demuestran que el rendimiento en términos de precisión de ambos algoritmos es similar, sin embargo, en la fase de entrenamiento, KEA es más eficiente en términos de tiempo de ejecución que *GenEx*, lo que cumple con uno de los principales objetivos propuestos en este trabajo de tesis.

KEA ha sido ampliamente descrito en [60, 17], por lo que en este capítulo solo se resumirá su funcionamiento. Primero se describirá brevemente el algoritmo original de *KEA*, el cual tiene un funcionamiento secuencial. A este algoritmo de KEA secuencial se le hicieron modificaciones para su paralelización -mediante el uso de *Threads*- con la finalidad de mejorar el rendimiento del algoritmo de extracción automática de *keyphrases* en términos de tiempo de ejecución. Se presentarán también los resultados de los experimentos realizados con el algoritmo KEA *threads*, en los que se evalúa el rendimiento en términos de tiempo de ejecución y en términos de precisión. Para el primer caso se presentarán dos evaluaciones, una de manera automática: comparando las *keywords* asignadas por el autor con las generadas automáticamente por el algoritmo KEA *threads*; y de una manera subjetiva, en donde la calidad de las *keyphrases*

obtenidas automáticamente es determinada por el criterio humano. Para finalizar se presentan gráficas del comportamiento en rendimiento y precisión del algoritmo paralelo cuando se incrementa el número de *threads* para su ejecución.

3.1 KEA

KEA es un sistema de extracción de *keyphrases* que usa un algoritmo de aprendizaje automático conocido como *Naive Bayes* para el entrenamiento y la extracción automática de *Keyphrases*. Una implementación en Java de KEA se encuentra disponible en la biblioteca digital de Nueva Zelanda [35]. El mecanismo de extracción de *Keyphrases* se compone de dos fases, entrenamiento y extracción automática de keyphrases.

En la figura 3.1 se muestra el proceso de entrenamiento y de extracción automática de *keyphrases* de KEA. Primero se construye un modelo usando documentos con *keyphrases* asignadas por el autor, después de construido el modelo, puede ser usado por el proceso de extracción automática de *keyphrases* para generar una lista *keyphrases* como resultado, por cada documento fuente.

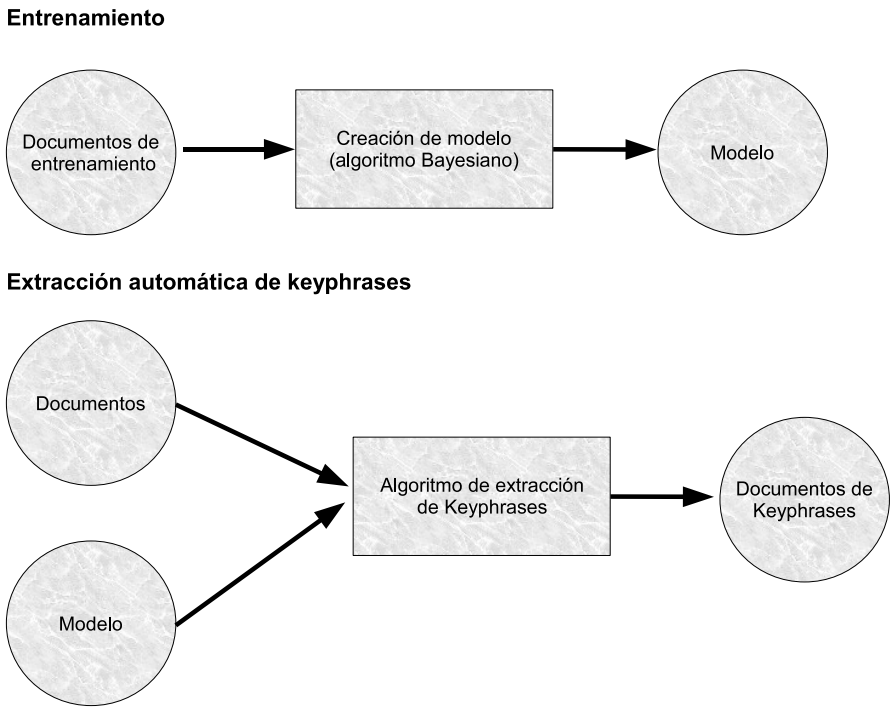

Figura 3.1: Fase de entrenamiento y extracción automática de *keyphrases* con KEA.

3.1.1 Entrenamiento

En la fase de entrenamiento se crea un *modelo* para identificar *keyphrases*, usando un conjunto de documentos de entrenamiento, para los cuales las *keyphrases* son conocidas (pueden ser asignadas por el autor o escritas manualmente). El *modelo* generado es un clasificador Bayesiano que además de calcular una probabilidad asociada con cada frase, les asigna una de las dos posibles clases: *keyphrase* o *non-keyphrase*.

KEA usa dos atributos de las frases para construir el modelo:

1. *TF×IDF*: Mide la frecuencia con la que aparece una frase en un documento, comparado con los demás documentos de entrenamiento. Las variables involucradas en el cálculo de este atributo se muestran en la ecuación 3.1,

$$TF \times IDF = \frac{freq(P, D)}{size(D)} \times -\log_2 \frac{df(P)}{N} \tag{3.1}$$

   en donde:

   - P es una frase
   - D es un documento
   - $freq(P, D)$ es el número de veces que P ocurre en D
   - $size(D)$ es el número de palabras en D
   - $df(P)$ es el número de documentos de entrenamiento que contienen P
   - N es el número de documentos de entrenamiento

2. *primer ocurrencia*:Es la *distancia* (medida en el número de palabras) entre el inicio del documento y la primera aparición de la frase en el mismo, dividido entre la cantidad total de palabras en el documento.

Si al construir el modelo, se desea usar *dominios de información específicos* [17], entonces KEA usa un atributo más para la construcción del modelo: *keyphrase-frecuency*. Este atributo refleja el número de veces que un frase dada P de un documento D, ocurre como una *keyphrase* asignada por el autor en todos los demás documentos de entrenamiento diferentes de D.

Cada documento es procesado por una serie de filtros (*KEAPhraseFilter* y *NumbersFilter*) que "limpian"el texto (se eliminan signos de puntuación, apóstrofos, números, etc). Se utiliza también una lista de *stopwords* (palabras comunes que carecen de significado propio y son omitidas, e.j: la, para, de, yo, etc) y un *stemmer* (algoritmo que extrae la raíz de una palabra, eliminando prefijos y sufijos, e.j: correr, corre, corrió y corriendo tienen como raíz "corr").

El número de documentos a usar para el entrenamiento no está establecido, pero los experimentos presentados en [60] y en [17] demuestran que después de 50 documentos,

la calidad del modelo construido no varía (o la variación es mínima), pero si se usan menos de 20 la variación en calidad es muy notoria. Una vez que se ha construido el modelo, puede ser usado en la fase de extracción automática de keyphrases.

3.1.2 Extracción de keyphrases

Para la extracción de *keyphrases* de un documento, se utiliza el *modelo* generado en el proceso de entrenamiento descrito en la sección 3.1.1 . El algoritmo de extracción de keyphrases calcula, mediante el algoritmo bayesiano descrito en [60], la probabilidad que tiene cada frase de ser una *keyphrase*. La salida de este algoritmo es una lista de *keyphrases* ordenadas de acuerdo a su probabilidad asociada. El tamaño de la lista de salida depende de un parámetro de entrada del algoritmo (la opción por defecto es 5), en [17], recomiendan usar un rango de entre 5 y 12 *keyphrases* de salida, ya que a medida que crece el número de *keyphrases* de salida, disminuye la precisión de las mismas.

En [17] demuestran también que el uso de modelos relacionados con el documento al que se le extraen las *keyphrases* genera una lista de *keyphrases* más precisas, comparado con la lista que se genera usando un modelo no relacionado. A esta relación entre el modelo y el documento le denominan *dominio específico del modelo*. Esto implica que si se requiere extraer *keyphrases* de documentos relacionados con matemáticas, se genere un modelo usando documentos de entrenamiento del mismo dominio, en este caso, documentos de matemáticas.

3.2 Mejoras al sistema KEA

Debido a diversas necesidades que se tenían para la correcta elaboración de este trabajo de investigación, fue necesario hacer algunas modificaciones a la implementación en JAVA del sistema KEA. Dichas modificaciones se listan a continuación:

- Integración de un *stemmer* y un algoritmo de *stopwords* en español.
- Mejoras en desempeño del algoritmo de extracción automática de *keyphrases*, haciendo una paralelización del mismo.
- Integración con base de datos, usando MySQL.

En las siguientes secciones se describe a detalle las modificaciones mencionadas. Debido a que las modificaciones para la integración con base de datos están relacionadas con la arquitectura de PDLib, será hasta el capítulo 4 donde se detallen.

3.2.1 Integración de *stemmer* y *stopwords* en español

Debido a que los documentos con los que trabajaremos son en su mayoría Tesis en español y KEA no tiene soporte de algoritmos de *stemmer* y *stopwords* para documentos en español, se realizó la implementación de ambos algoritmos para dar el soporte a la extracción automática de *keyphrases* en documentos escritos en dicho idioma.

El algoritmo *stemmer* en español es una implementación en Java del algoritmo de Porter [41], el cual se obtuvo de las librerías *Snowball* [40].

Para el caso de el algoritmo de *stopwords* en español, se implementó una nueva versión basada en la que utiliza KEA para documentos en inglés (*StopwordsEnglish*) y se cambió la lista de palabras *stopwords* en inglés por la lista correspondiente en español (*StopwordsSpanish*).

3.2.2 Mejoras en el desempeño de KEA

La paralelización del algoritmo de extracción automática de *keyphrases* es importante en el contexto de PDLib debido a que se tendrán un gran número de usuarios concurrentes en el sistema, cada uno de ellos realizando alguna de las operaciones disponibles para las diferentes versiones de clientes PDLib (ver la arquitectura de PDLib en el capítulo 4). Cada que un usuario agregue un nuevo documento a su biblioteca personal (en cualquiera de los clientes que lo haga), se extraerán automáticamente las *keyphrases* del mismo, lo que implicará una utilización de recursos en el servidor PDLib importante (e.j: procesador, operaciones de E/S, entre otras), sin tomar en cuenta el tiempo de transferencia en la red y el indexado del nuevo documento en la base de datos. Debido a dicha utilización de recursos de procesamiento, resulta indispensable liberar pronto lo más que se pueda de ellos. Una forma de contribuir en esto es reduciendo el tiempo de ejecución del algoritmo de extracción automática de *keyphrases*.

Debido a que el tiempo de ejecución del algoritmo de extracción automática de *keyphrases* es proporcional al número y al tamaño de los documentos fuente y, teniendo como motivación la reducción del uso de recursos de procesamiento, se buscó mejorar el desempeño en tiempo de ejecución paralelizando (mediante el uso de *threads*) la fase de extracción automática de *keyphrases*. Se logró una disminución del tiempo de ejecución de un 62 % (ver sección 3.3.3) con respecto al algoritmo original de KEA (secuencial), consiguiendo así una mejora importante en el rendimiento del algoritmo.

En las sección 3.2.3 y 3.2.4 se detallan, respectivamente, los algoritmos de extracción automática de *keyphrases* en secuencial y paralelo.

3.2.3 Algoritmo KEA secuencial

En la figura 3.2 se muestra el proceso de extracción automática de *keyphrases* (*KeyPhraseExtraction*) del algoritmo KEA secuencial. El algoritmo recibe co-

mo parámetros, , entre otros, la ruta de documentos para los que se les extraerán las *keyphrases*, el nombre del *modelo* (previamente generado) a usar, el número de *keyphrases* a extraer (k), el nombre del algoritmo de *stemmer* a usar (Porter ó Livins) y además el nombre del algoritmo de *stopwords* (español o inglés) que se aplicará.

Una vez que se validan los parámetros de entrada del algoritmo, se lee un documento y se pasa por un filtrado de caracteres y números que lo "limpian". Se eliminan números, signos de puntuación, apóstrofos y llaves; el resultado de este proceso de filtrado del documento es una lista de frases, las cuales son procesadas, usando el algoritmo Bayesiano descrito en [60], para determinar si es una *keyphrase* y calcular el *ranking* de cada una de ellas. De la lista de *keyphrases* generadas, se toman las de mayor *ranking* (k, en orden descendente), las cuales son la salida del algoritmo.

Cuando se termina de procesar el documento, se toma el siguiente y se repite el proceso de filtrado, identificación y *ranking* de *keyphrases*, dando como resultado un listado de las primeras k *keyphrases* por cada documento fuente.

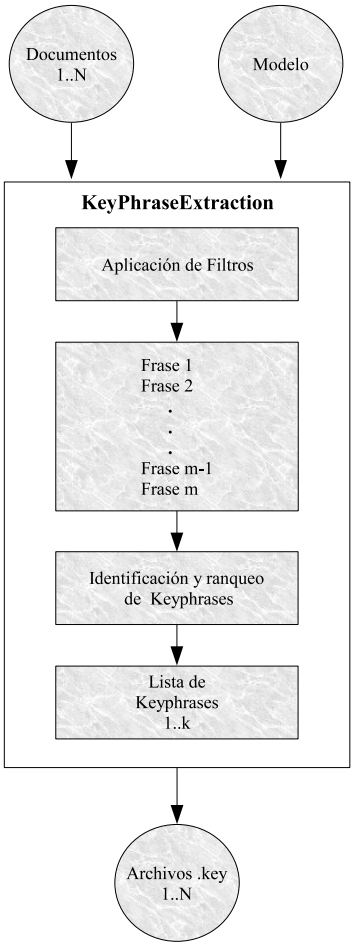

Figura 3.2: Proceso de extracción automática de *keyphrases* con KEA, de forma secuencial.

Proceso	Tiempo ejecución (seg)
Aplicación de Filtros	7.078
Identificación y ordenamiento de Keyphrases	138.749

Tabla 3.1: Tiempo de ejecución de los procesos clave del algoritmo de extracción automática de *Keyphrases*. Estos tiempos se calcularon para una colección de 49 documentos (tesis) del área de MCT del ITESM, Campus Monterrey

.

3.2.4 Paralelización del algoritmo

Como se mencionó en la sección 3.2.2, uno de los objetivos principales de la paralelización del algoritmo de extracción automática de *keyphrases* es la reducción del uso recursos en el servidor PDLib. Siguiendo este objetivo, se realizó un análisis al algoritmo secuencial descrito en la sección 3.2.3. El análisis consistió en determinar y calcular los tiempos de ejecución de los procesos clave del algoritmo, sin tomar en cuenta los tiempos de lectura (E/S) del documento fuente, ya que estos tiempos dependerán de las características de *hardware* (p.e: velocidad del disco duro) del servidor y de los algoritmos de almacenamiento y recuperación de información del manejador de base de datos que se esté usando (actualmente MySQL)[1]. En la figura 3.2 se puede apreciar que los procesos clave del algoritmo de extracción automática de *keyphrases* son la *aplicación de filtros* y la *identificación y ordenamiento de keyphrases*. A continuación se detallan ambos procesos:

1. *Aplicación de Filtros*. Este es un proceso de filtrado del texto fuente en el que se aplican básicamente dos algoritmos: *KEAPhraseFilter* y *NumbersFilter*. El primer filtro se encarga de quitar los signos de puntuación y caracteres especiales, mientras que el segundo filtro elimina los números.

2. *Proceso de identificación y ordenamiento de keyphrases*. Este proceso se encarga de calcular, para cada frase, los atributos *TF×IDF* y *primer ocurrencia* descritos en la sección 3.1.1, con lo que determina si cada una de las frases de la lista es una *keyphrase*.

En la tabla 3.1, se muestran los resultados obtenidos para el cálculo de los tiempos de ejecución de dichos procesos. Como se puede apreciar, el tiempo total de ejecución del algoritmo de extracción automática de *keyphrases* es de 145.83 segundos (2.43 minutos). De este tiempo total de ejecución, el proceso de *aplicación de filtros* solo utiliza 7.078 segundos, lo que representa el 5.10 % del tiempo total de ejecución del algoritmo, mientras que el proceso de *identificación y ordenamiento de keyphrases* utiliza un 94.89 % del tiempo de ejecución total del mismo.

[1]Ver la arquitectura de PDLib en el capítulo 4

Debido a que el proceso de *identificación y ordenamiento de keyphrases* es el que utiliza mayor tiempo de procesamiento del algoritmo de extracción automática de *keyphrases*, se tomó la decisión de reducir el tiempo de ejecución de este utilizando la paralelización.

En la figura 3.3 se muestra el nuevo proceso de extracción automática de *keyphrases* usando *Threads*. El algoritmo recibe los mismos parámetros de entrada que el algoritmo KEA en secuencial, más el número de *threads* a usar.

Una vez que se validan los parámetros de entrada del algoritmo, se ejecuta el mismo proceso de filtrado que en la versión en secuencial; el resultado de este proceso de filtrado del documento es una lista de frases, las cuales son separadas en j bloques del mismo tamaño (ecuación 3.2), siendo j un parámetro de entrada que define la cantidad de *threads* a usar.

$$block = \left\lfloor \frac{m}{j} \right\rfloor \tag{3.2}$$

en donde m es el número de frases generadas por los filtros (1 frase por línea), y j es el número de *threads* a usar (parámetro de entrada).

A cada *thread* se le asigna un bloque de frases. Cada bloque de frases es procesada por el *thread* para calcular sus atributos ($TF{\times}IDF$ y *primer ocurrencia*) y determinar si son *keyprases*. Cada *thread* genera su propia lista de *keyphrases* de acuerdo al bloque que se le asignó y se toman las de mayor *ranking* (k, en orden descendente) de los j *threads* usados.

A continuación se presentan los resultados obtenidos de la comparación de los algoritmos de extracción automática de *keyphrases* secuencial y paralelo. Las métricas a comparar son: el *rendimiento* en términos de tiempo de ejecución (para lograr una *reducción* del uso de recursos) y el *rendimiento* en términos de *precisión* de ambos algoritmos.

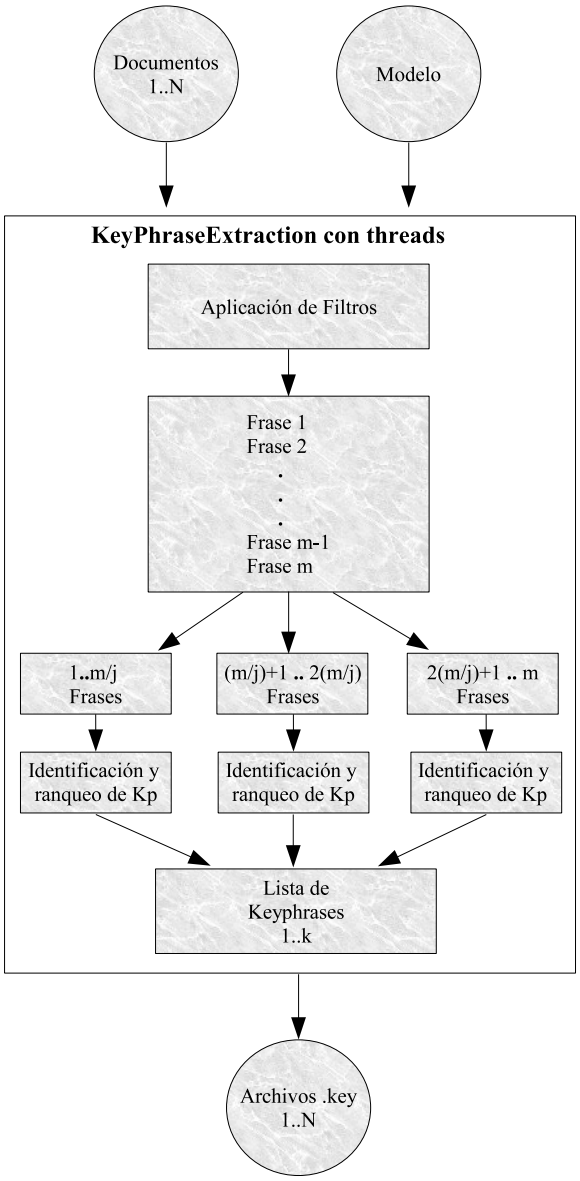

Figura 3.3: Proceso de extracción automática de *keyphrases* con la implementación de *threads* en KEA.

3.3 Experimentos

Para poder evaluar el *rendimiento* del algoritmo de extracción automática de *keyphrases* en paralelo (usando *threads*), en términos de tiempo de ejecución y de *precisión*, se diseñaron algunos experimentos, para los cuales se definieron las siguientes métricas:

- *Rendimiento en términos de tiempo de ejecución.* Es el tiempo de ejecución del algoritmo de extracción automática de keyphrases, sin tomar en cuenta el tiempo de E/S del archivo fuente ni el tiempo de aplicación de los filtros de preprocesamiento del documento descritos en la sección 3.2.4.

- *Rendimiento en términos de precisión*, se calcula en base a el número de *keyphrases* identificadas automáticamente que son iguales a las palabras clave (*keywords*) previamente definidas por el autor[2]. Esta métrica también será evaluada en otro experimento similar, pero usando un criterio cualitativo (evaluación subjetiva), en donde a un grupo de personas del área de tecnología informática, se le pedirá que identifique de una encuesta de *keyphrases* las que según el criterio de cada persona, son de relevancia para cada uno de los documentos evaluados.

En las siguientes secciones se describe el entorno experimental y los resultados de las evaluaciones del algoritmo de extracción automática de *keyphrases* en secuencial y en paralelo, para las métricas definidas anteriormente.

3.3.1 Entorno experimental

Los experimentos se realizaron en un servidor *HP Proliant* con las siguientes características:

- Procesador Dual Intel Xeon de 3.0 GHz. con 1 MB de memoria cache
- 3.0 GB de memoria RAM
- 1 Arreglo de discos RAID 5 de 1 TB.
- Sistema Operativo Windows Server 2003
- Base de datos MySQL ver 4.1.X
- Java 1.5.1

[2]Se compara la raíz de las palabras usando el algoritmo de *stemmer* en español de Porter

Los documentos usados tanto para la fase de entrenamiento, como para la fase de extracción, fueron tomados del repositorio de tesis de maestría de la DTIE del ITESM, Campus Monterrey [14]. Dicho repositorio está compuesto por documentos en español (tesis), de los siguientes programas de posgrado:

- Maestría en Ciencias en Ingeniería Electrónica con especialidad en Telecomunicaciones (MSET), con 41 trabajos de tesis.
- Maestría en Ciencias en Tecnología Informática (MCT), con 47 trabajos de tesis.
- Maestría en Administración de Telecomunicaciones (MTL), con 38 trabajos de tesis.
- Maestría en Administración de Tecnologías de Información (MTI), con 152 trabajos de tesis.

3.3.2 Fase de entrenamiento

Basándonos en [17], se construyó un modelo para cada colección de documentos (5 modelos en total), para formar dominios de información y obtener así mejores resultados con respecto a la exactitud de las *keyphrases*. Como se explicó en la sección 3.1.1, la calidad del modelo (y por consiguiente, de las *keyphrases* extraídas) no varía después de 50 documentos (según experimentos presentados en [17] y [60]), pero debido a que nuestras colecciones de documentos no son tan grandes (solo MCT y MTL tienen más de 50 documentos), se determinó usar para cada colección un total de 25 documentos de entrenamiento, los cuales son elegidos aleatoriamente por el algoritmo de construcción del modelo (*KEAModelBuilder*).

Para construir el modelo, se utilizaron los algoritmos de *stemmer* [40] y *stopwords* en español, así como también se activó la opción para que el algoritmo de construcción del modelo use el atributo *keyphrase-frecuency* descrito en la sección 3.1.1 y crear *dominios de información específicos*.

En la tabla 3.2 se muestra el tiempo de ejecución del algoritmo de construcción del modelo para cada colección. Como se puede apreciar, a pesar de que para cada modelo se tomaron 25 documentos de entrenamiento, los tiempos de construcción de cada uno de ellos es diferente, esto es debido a la variedad de tamaños en los documentos fuente.

No se modificó a la versión original (en modo secuencial) del algoritmo de construcción del modelo (*KEAModelBuilder*) debido a que este proceso será una tarea programada para ejecutarse de acuerdo al crecimiento que se tenga en las bibliotecas personales de los usuarios, por lo que se hará de manera esporádica y controlada. Debido a esto, se usará el mismo modelo, según el programa (MCT, MTI, etc), para las versiones secuencial y paralelo del algoritmo de extracción automática de *keyphrases*.

A continuación de presentan los experimentos realizados para evaluar el rendimiento en términos de tiempo de ejecución y de su precisión.

Colección	Documentos de entrenamiento	Tiempo de ejecución (min)
MSET	25	1.66
MIT	25	3.28
MCT	25	5.3
MTL	25	8.55
MTI	25	4.74

Tabla 3.2: Tiempo de ejecución del algoritmo de construcción de los modelos del repositorio de tesis en español de la DTIE

.

3.3.3 Rendimiento en términos de tiempo de ejecución

En esta sección se describen los experimentos realizados con el algoritmo de extracción automática de *keyphrases* para medir el rendimiento en términos de tiempo de ejecución. Se ejecutaron las dos versiones del algoritmo (secuencial y paralelo) usando los modelos específicos generados en la fase de entrenamiento (sección 3.3.2) y 2 *threads* para el caso del algoritmo en paralelo.

Ambas versiones del algoritmo (secuencial y paralelo) fueron configuradas (de la misma forma que en la fase de entrenamiento) para usar los algoritmos de *stemmer* y *stopwords* en español y el atributo *keyphrase-frecuency*. Se estableció un máximo de 10 *keyphrases* como salida por documento fuente.

En la tabla 3.3 se muestran los resultados obtenidos. Como se puede apreciar, el tiempo de ejecución del algoritmo de extracción automática de *keyphrases* depende de la cantidad de documentos a procesar y del tamaño de los mismos. En el caso de la colección $MSET$, la cual contiene solo 41 documentos, el tiempo de ejecución del algoritmo en secuencial es de 2.51 minutos, contra 0.89 segundos que se tarda en la versión en paralelo. Esto representa el menor tiempo de ejecución del algoritmo de nuestras 4 colecciones. La colección con un mayor tiempo de ejecución es MTI, con 24.68 minutos en secuencial y 10.29 minutos en paralelo; esto es debido a que dicha colección es la que contiene un mayor número de documentos (152). Las colecciones MCT y MTL (con 47 y 38 documentos, respectivamente), tienen una diferencia en tiempo de ejecución secuencial de solo 1.09 minutos, (8.10 y 7.01 minutos); mientras que en paralelo esta diferencia es de la mitad de ese tiempo: 0.53 minutos (2.37 y 1.84 minutos).

La diferencia promedio del rendimiento en términos de tiempo de ejecución de las 4 colecciones, entre el algoritmo de extracción automática de *keyphrases* en secuencial y en paralelo, es de un 63.62 % (26.91 minutos).

Otra forma de medir el rendimiento en términos de tiempo de ejecución del algoritmo es mediante dos métricas muy utilizadas en el área de cómputo paralelo o cómputo de alto nivel, para las cuales tomaremos las definiciones de [19]:

Colección	**Num. Docs**	**Tiempo de ejecución (min)**		*Sp*
		Secuencial	**Paralelo**	
MSET	41	2.51	0.89	2.82
MCT	47	8.10	2.37	3.42
MTL	38	7.01	1.84	3.82
MTI	152	24.68	10.29	2.40
Total:	278	42.30	15.39	3.12

Tabla 3.3: Tiempo de ejecución del algoritmo de extracción automática de *keyphrases* para el repositorio de tesis en español de la DTIE

.

- *Speedup*. El speedup es una medida que determina que tan rápido es un algoritmo al ser ejecutado en una máquina con p procesadores, comparado con la ejecución secuencial del mismo algoritmo en un solo procesador. El *speedup* se calcula con la ecuación:

$$Sp = \frac{Ts}{Tp}$$

  donde Ts es el tiempo de ejecución del algoritmo secuencial y Tp es el tiempo de ejecución del algoritmo en paralelo ejecutado en p procesadores.

- *Eficiencia*. La eficiencia es la medida de la fracción de tiempo para el cual un elemento de procesamiento es empleado provechosamente. La eficiencia se puede calcular con la ecuación:

$$E = \frac{Sp}{p}$$

  donde Sp es el *speedup* del algoritmo en paralelo y p es el número de procesadores usado por el algoritmo en paralelo.

En sistemas paralelos ideales, el *speedup* es igual a p y la eficiencia es igual a 1. En la práctica, el *speedup* es menor a p y la eficiencia está entre 0 y 1, dependiendo de la efectividad con la que los elementos de procesamiento son utilizados.

Cuando en la práctica se llega a obtener un *speedup* mayor a p, se le conoce como un *speedup superlineal*. Usualmente esto pasa cuando el trabajo realizado por un algoritmo secuencial es mayor al de su formulación en paralelo ó es debido a características en el *hardware* que ponen en desventaja al algoritmo secuencial.

Siguiendo estas métricas del cómputo paralelo, el *speedup* (columna Sp de la tabla 3.3) obtenido en nuestro algoritmo en paralelo es *superlineal*, ya que en promedio tenemos un $Sp = 3.12$, y el número de procesadores (p) del servidor experimental es

de 2. Este *speedup* superlineal se debe a que la información que procesa cada *thread* (la lista de frases del documento fuente) es menor que en el algoritmo secuencial, ya que el documento es dividido en bloques de frases de tamaño m/j, donde m es el tamaño total de frases del documento fuente, y j es el número de *threads* a usar por cada documento. Esta diferencia en la cantidad de frases a procesar es muy importante en el rendimiento del algoritmo, ya que la lista de frases es almacenada en un *Vector* (con algoritmos de inserción y borrado del orden *O(n)*), el cual se accede constantemente para extraer sus elementos (frases), calcular los atributos correspondientes a cada frase (con los que determina si la frase es o no una *keyphrase* descritos en la sección 3.1.1) y guardarlos nuevamente con los valores de los atributos calculados. Estos operaciones de acceso al *Vector* consumen más tiempo de ejecución cuando la información almacenada es mayor, como en el caso del algoritmo en secuencial, que cuando se almacena menos información, como se hace en el algoritmo en paralelo. Por estas diferencias entre los bloques de información asignados a los *threads* (paralelo), con el bloque completo de un documento (secuencial), se reflejan directamente en el *speedup* superlineal obtenido.

Para el caso de la eficiencia, si se utiliza el *Sp* promedio, la eficiencia del nuevo algoritmo de extracción automática de *keyphrases* es de 1.54. Con lo que se demuestra que el algoritmo paralelizado es mejor en rendimiento en términos de tiempo de ejecución y en eficiencia que el algoritmo secuencial. En la siguiente sección se mostrarán los resultados obtenidos con respecto al rendimiento en términos de precisión del algoritmo.

3.3.4 Rendimiento en términos de Precisión

El rendimiento en términos de precisión es una de las principales métricas usadas para evaluar algoritmos de extracción automática de *keyphrases*. La precisión puede ser evaluada de diferentes formas. En [55], se usa el criterio humano para evaluar los resultados de la extracción automática de *keyphrases* del algoritmo *GenEx*, obteniendo hasta un 80 % de *keyphrases* consideradas como buenas. En [25], se usa también el criterio humano para evaluar *links* generados automáticamente usando *keyphrases* con KEA, obteniendo hasta un 58.9 % de *links* relacionados con documentos del mismo tema, usando búsquedas de texto completo en los documentos fuente. La precisión del algoritmo KEA (en su versión original), evaluada ampliamente en [17, 60], es de hasta un 11 % de las *keyphrases* que hacen *match* exacto con las *keyphrases* definidas por el autor. En este trabajo de tesis, se evaluará la precisión de KEA usando un algoritmo de comparación automática de *keyphrases*, con lo que obtendremos la precisión en términos de *match* exacto y como una evaluación de precisión adicional, realizaremos experimentos usando un criterio subjetivo, en donde un grupo de usuarios evaluará si las *keyphrases* son acertadas o no.

Área	**Num.**	**Total**		**Secuencial**		**Paralelo**	
	Docs	**Kw**	**Kp**	#	%	#	%
MSET	41	170	410	19	11.18	19	11.18
MCT	47	216	470	38	17.59	34	15.74
MTL	38	234	380	43	18.38	41	17.52
MTI	152	732	1520	150	20.49	110	15.03
Total:	278	1352	2780	250	18.49	204	15.09

Tabla 3.4: Match exacto del algoritmo KEA secuencial y paralelo para el repositorio de tesis en español de la DTIE

.

Precisión exacta

Para poder evaluar la precisión *exacta* de KEA, se construyó un algoritmo que compara las *keywords*[3] de los documentos contra las *keyphrases* obtenidas automáticamente por KEA secuencial y paralelo. Dicho algoritmo utiliza la raíz de la palabra (utilizando el mismo algoritmo de *stemmer* en español que KEA) para determinar si son o no iguales (e.j: correr = corre). Este criterio de evaluación de las *keyphrases* es similar al utilizado en [17, 55].

En la tabla 3.4 se muestran los resultados obtenidos de la evaluación de la precisión exacta de el algoritmo de KEA en secuencial y paralelo, usando el repositorio de tesis de la DTIE. En la columna *Área* se indica el área de posgrado de los documentos fuente (tesis); la columna *Docs* indica el número de documentos usados en el experimento; las columnas *Kw* y *Kp* indican el total de *Keywords* y *Keyphrases*[4], respectivamente. Las columnas *Secuencial (# y %)* indican el número de *match* exactos y su equivalente porcentual (número de *matches* exactos entre el total de *keyphrases* extraídas) para el algoritmo secuencial, mientras que las columnas*Paralelo (# y %)* indican lo mismo, pero en el caso del algoritmo en paralelo. Como se puede apreciar, en la colección $MSET$ se obtiene la misma precisión en ambos algoritmos; en el caso de las colecciones MCT y MTL se tiene una diferencia en la precisión menor a 5 *keyphrases* (el equivalente a un máximo de 11.75 por ciento global). En la colección MTI hay una mayor pérdida de precisión del algoritmo en paralelo con respecto al secuencial, esta diferencia en *keyphases* que hacen *match* exacto es de 40, lo que equivale a un 36.33 por ciento global. Esta diferencia es debido a que el programa de posgrado de MTI tiene una amplia gama de temas de investigación, por lo que la construcción del modelo específico requiere de un mayor número de documentos de entrenamiento, con la finalidad de cubrir una mayor parte de los temas de investigación que abarca el programa de *Administración de Tecnologías de Información* (MTI).

[3]Las *keywords* en el contexto de los experimentos son las *keyphrases* definidas por el autor de la tesis.

[4]Se extrajo la misma cantidad de *keyphrases* con ambos algoritmos.

En el renglón de totales se puede apreciar una diferencia de 46 *keyphrases*, lo que representa una pérdida de precisión en un 22.53 por ciento global del algoritmo paralelo con respecto al algoritmo secuencial. Esta pérdida de precisión del algoritmo paralelo es debido a que cada *thread* genera una lista de k *keyphrases* correspondientes al bloque del documento fuente que se le asignó, en donde k es igual al número de *keyphrases* que se desean extraer del documento fuente. De la lista de salida que genera cada *thread*, se van tomando las *keyphrases* de mayor peso (con base en los atributos $TF{\times}IDF$ y *primer ocurrencia*) de una en una, hasta completar una sola lista que contenga las k *keyphrases* de mayor peso en orden de aparición de las listas de cada *thread*. Este proceso de selección de las *keyphrases* finales de mayor peso por cada lista de salida de los *threads*, es el que provoca que algunas de las *keyphrases* que aparecen en la lista generada por KEA secuencial obtengan un menor peso en su bloque correspondiente, por lo que son omitidas en el algoritmo en paralelo, y en su lugar se ponen otras que hayan obtenido un mayor peso en esa misma lista.

Un aspecto que hay que tomar en cuenta al evaluar de esta forma es que las palabras clave que define el autor (*keywords)* del documento no necesariamente son las que mejor describen al mismo, debido a que el autor puede enfatizar cierto aspecto de su trabajo que para él tenga mayor relevancia como aportación de la investigación. Sin embargo, este método de evaluación es simple y puede ser calculado automáticamente, además de tener la flexibilidad de evaluar documentos en inglés y español.

Precisión subjetiva

Además de los experimentos de *match* exacto, se hicieron los experimentos para evaluar la precisión de una manera subjetiva, usando personas (a los cuales llamaremos usuarios) del área de tecnologías de información. Este tipo de experimentos es similar al realizado por [26], con la diferencia de que en nuestros experimentos no se leerá el documento completo, ya que los documentos a evaluar son Tesis de maestría. Cada usuario indicará que *keyphrases*, según su criterio y sus conocimientos en el área, son relevantes al documento evaluado. Esta evaluación tiene la finalidad de saber la cantidad de *keyphrases acertadas* que genera el algoritmo de extracción automática de *keyphrases* en secuencial y paralelo, de manera que nos proporcione una idea general de la precisión del algoritmo paralelo con respecto al secuencial. Se espera que como trabajo futuro se realice experimentación de campo con un mayor número de usuarios.

En éste experimento se utilizó la colección de tesis de maestría del área de tecnología informática (MCT) que se usó para la evaluación de la *precisión exacta* descrito en la sección 3.3.4. Se seleccionó un grupo de 6 personas, los cuales llamaremos usuarios, para realizar el experimento. Los usuarios 1, 2 y 3 son estudiantes de posgrado del programa MCT; los usuarios 4 y 5 son estudiantes de licenciatura y son miembros del equipo de desarrollo de PDLib; el usuario 6 es graduado de la maestría en sistemas inteligentes (MIT). A estos usuarios se les proporcionó, por cada documento a evaluar, la siguiente información:

Usuario	**KEA Secuencial**		**KEA Paralelo**	
	#	%	#	%
1	153	36	143	34
2	110	26	103	25
3	153	36	148	35
4	202	48	241	57
5	240	57	219	52
6	155	37	151	36
Promedio:	168.83	40.00	167.5	39.83

Tabla 3.5: Evaluación subjetiva del algoritmo de extracción automática de *keyphrases* para la colección de tesis de la DTIE en el programa *MCT*

.

- Nombre del documento (Título de la tesis).
- *Abstract* o resumen de la tesis.
- *Keywords*. Son las *keyphrases* definidas previamente por el autor.
- *Keyphrases 1*. Son las *keyphrases* extraídas automáticamente por el algoritmo KEA secuencial.
- *Keyphrases 2*. Son las *keyphrases* extraídas automáticamente por el algoritmo KEA en paralelo.

Los usuarios no tenían conocimiento de cuales *keyphrases* eran del algoritmo en secuencial o del algoritmo en paralelo. A cada usuario se le indicó leer el título, *abstract* y *keywords* de cada documento a evaluar. Con base en lo leído, se le indicó que subrayara las palabras definidas en *Keyphrases 1* que, según su criterio, son de relevancia para el documento evaluado. Al terminar de subrayar las palabras, el usuario tenía que contarlas y apuntar el total de ellas, para después repetir el mismo procedimiento con las palabras definidas en *Keyphrases 2*. Está evaluación se hizo para un total de 47 documentos.

En la tabla 3.5, se muestran los resultados obtenidos. Como se puede apreciar, la evaluación subjetiva tiene mejores resultados que la precisión exacta, ya que en el peor de los casos, la evaluación subjetiva del algoritmo en paralelo es de 103 *keyphrases* relevantes (un 25 % de las 470 *keyphrases exraidas* para la colección *MCT*). Los usuarios 4 y 5 son los determinaron la mayor precisión de ambos algoritmos: 202 *keyphrases* en secuencial y 241 *keyphrases* en paralelo para el usuario 4; el usuario 5 evaluó con 240 *keyphrases* en secuencial y 219 *keyphrases* en paralelo. Para los usuarios 1, 2, 3, 5 y 6, el algoritmo secuencial tiene mayor precisión que el paralelo, con una diferencia máxima de 21 *keyphrases* (para el usuario 5) y 4 *keyphrases* (usuario 6) como diferencia mínima.

3.3.5 Comportamiento del algoritmo KEA *threads*

En la Figura 3.4 se muestra el comportamiento del rendimiento en términos de tiempo de ejecución (a) y de precisión exacta (b) del algoritmo KEA en paralelo cuando se incrementa el número de *threads* usados para su ejecución. El experimento consistió en ejecutar el algoritmo de extracción automática de *keyphrases* bajo el mismo entorno experimental descrito en la sección 3.3.1, variando sólo el número de *threads* en cada ejecución.

En la Figura 3.4 (a) se muestra una reducción en el tiempo de ejecución del algoritmo. Como se puede apreciar, el tiempo de ejecución del algoritmo con 4 *threads* es menor en un 83.3 % del tiempo de ejecución en secuencial, pero sólo un 60 % menor con respecto a la ejecución con 2 *threads*, lo que significa que el algoritmo en paralelo es casi tres veces más eficiente en términos de tiempo de ejecución a medida que se incrementa el número de *threads* a usar (e.j: 2, 4, 6).

A pesar de este buen desempeño en tiempo de ejecución del algoritmo paralelo, se presentan problemas cuando se utilizan más de 4 *threads*, ya que el algoritmo no concluye satisfactoriamente el cálculo del tiempo de ejecución en todas las colecciones y, por consiguiente, la extracción automática de *keyphrases* en todos los documentos. Esto es debido a que los múltiples accesos simultáneos que se hacen al modelo (cada *thread* tiene acceso independiente al modelo correspondiente de la colección en proceso) ocasionan excepciones de E/S, dificultando el cálculo del tiempo de ejecución por la terminación anticipada del algoritmo. Este comportamiento inicia a partir de 6 *threads* y se presenta también (aunque en diferentes colecciones) en las ejecuciones con 8, 10 y 13 *threads*.

En la Figura 3.4 (b) se muestra también una mayor pérdida de precisión exacta en el algoritmo de extracción automática de *keyphrases* a medida que se incrementa el número de *threads* a usar. Como se puede apreciar, existe una pérdida de precisión en la ejecución del algoritmo con 2, 4 y 6 *threads* de un 31.25 %, 37.5 % y 81.25 % respectivamente, comparada con la precisión del algoritmo secuencial en la colección MTI (la de mayor número de documentos). Esta pérdida de precisión es diferente para las demás colecciones, por ejemplo MCT, en donde la pérdida con 2, 4 y 6 *threads* es de un 20 %, 30 % y 24 % respectivamente. Esto refleja que (consistentemente con las demás colecciones) la mejor precisión para el algoritmo en paralelo es cuando se usan sólo 2 *threads* para su ejecución.

Considerando estos resultados de los experimentos, se puede concluir que el número de *threads* a usar dependerá de si se requiere o no un mayor rendimiento en términos de tiempo de ejecución (lo que ocasiona también una mayor pérdida de precisión), pero se recomienda manejar la ejecución del algoritmo de extracción automática de *keyphrases* sólo con 2 o 4 *threads*, de manera que la precisión exacta del algoritmo no se reduzca demasiado.

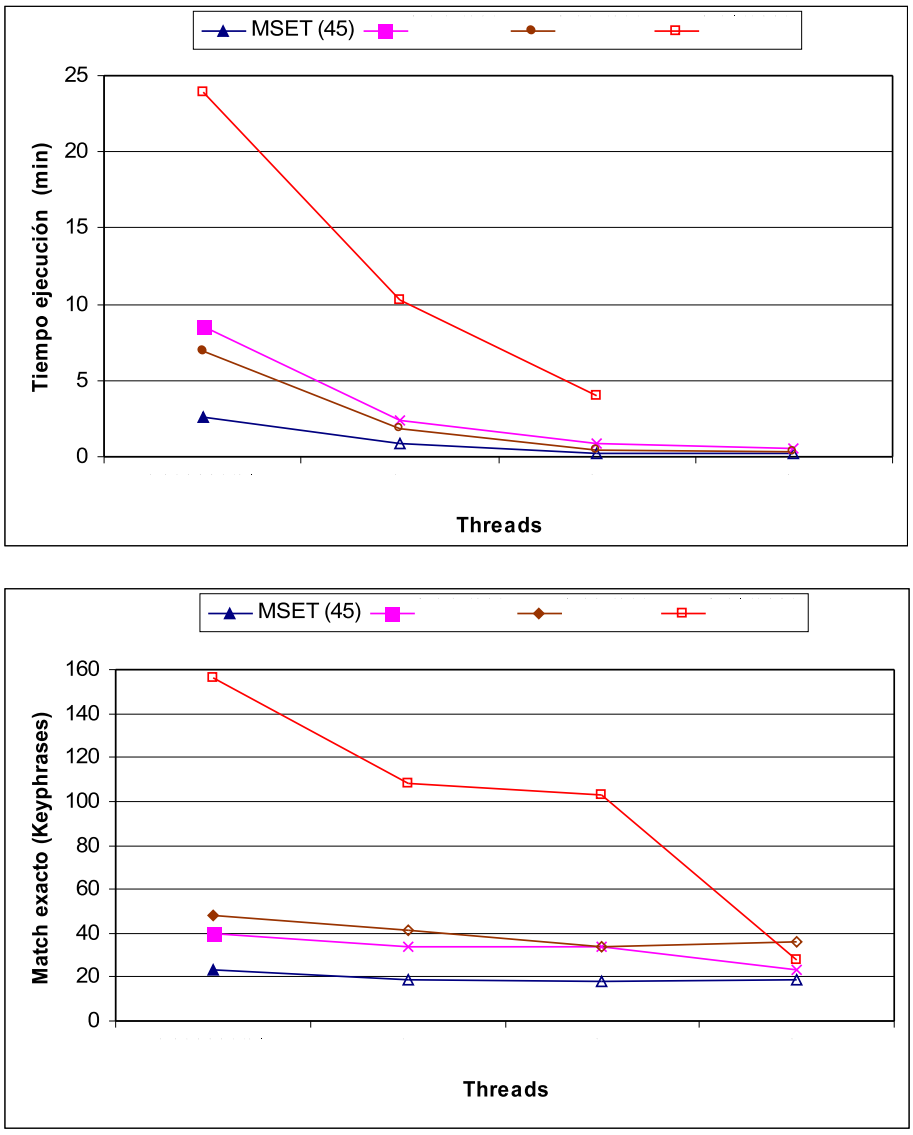

Figura 3.4: Comportamiento en rendimiento y precisión del algoritmo de extracción automática de *keyphrases* con respecto al número de *threads* utilizados para su ejecución en las colecciones de documentos de tesis de la DTIE.

3.4 Conclusiones

En este capítulo se describió KEA, un algoritmo para la extracción automática de *keyphrases*. Se describió la paralelización del algoritmo de extracción automática de KEA y se realizaron algunos experimentos para evaluar el rendimiento en términos de tiempo de ejecución y en términos de precisión del algoritmo con respecto al algoritmo secuencial original de KEA.

Los experimentos realizados se enfocaron al rendimiento del algoritmo paralelo cuando el número de *threads* usados es de 2. Estos experimentos muestran un mejor rendimiento del algoritmo en términos de tiempo de ejecución en promedio de un 63.62 % más rápido que el algoritmo secuencial, obteniendo así un *speedup* superlineal de 3.12. Con respecto al rendimiento en términos de precisión exacta, los resultados muestran una ligera pérdida de precisión, que en promedio es de un 13.72 %. Está pérdida de precisión del algoritmo en paralelo se ve disminuida en los resultados de la evaluación subjetiva, en dónde en promedio, la precisión del algoritmo paralelizado es menor por 0.43 %.

Con base en los resultados de los experimentos, podemos concluir que el algoritmo de extracción automática de *keyphrases* en paralelo con 2 y 4 *threads* es más eficiente en términos de tiempo de ejecución que la versión original en modo secuencial, por lo que puede ser utilizado para los propósitos de este trabajo de tesis, ya que la pérdida de precisión resultante debido a las modificaciones realizadas al algoritmo original para su paralelización puede no ser tan importante comparada con el rendimiento obtenido. Se propondrá como trabajo futuro un análisis más profundo de ambos algoritmos para determinar si dicha pérdida en la precisión puede ser eliminada.

En el siguiente capítulo se explicará la arquitectura de PDLib y se mostrará el prototipo que se construyó para hacer uso de las *keyphrases* en un ambiente de bibliotecas digitales en dispositivos móviles. Se justificará el hecho de darle mayor importancia a la métrica *desempeño* que a la de *precisión* del algoritmo de KEA *threads*.

Capítulo 4

Integración Data Server - KEA

Una parte de este trabajo de tesis consistió en desarrollar un prototipo para bibliotecas digitales en ambientes móviles, con la finalidad de proponer el uso de las *keyphrases* como un mecanismo que facilite la pronta localización de documentos en una biblioteca digital personal (específicamente en PDLib).

En este capítulo se describirá la arquitectura de PDLib y la integración de KEA con uno de sus principales componentes: el *Data Server*. Se definirá la arquitectura propuesta para el desarrollo de un cliente móvil (*thick client*) de PDLib, con el cual se propondrá el uso de *keyphrases* como facilitadoras en el proceso de agilizar la localización de documentos relevantes para el usuario de la biblioteca digital en el dispositivo móvil.

4.1 Arquitectura PDLib

En esta sección se describirá brevemente la arquitectura de PDLib[1] y las modificaciones realizadas a uno de los componentes principales de la arquitectura: el *data server*. Las modificaciones se realizaron con la finalidad de implementar la extracción automática de *keyphrases* utilizando KEA y poder así implementar diferentes alternativas de visualización de pantallas de resultados de las búsquedas realizadas en un prototipo de cliente móvil.

En la figura 4.1 se muestra un esquema general del sistema PDLib, el cual se divide en tres capas: *Client*, *Server* e *Interoperability*, separadas según la funcionalidad que ofrecen[2]:

1. *Client Tier*. En esta capa se incluyen la variedad de dispositivos con los que un usuario de PDLib puede interactuar: *PDA*, *Tablet PC*, Teléfono móvil, computadora portátil y de escritorio.

[1] La información de la arquitectura de PDLib que se presenta a continuación está basada en los trabajos de [4] y [46].

[2] Nombres tomados en inglés por consistencia con las figuras de la arquitectura de PDLib

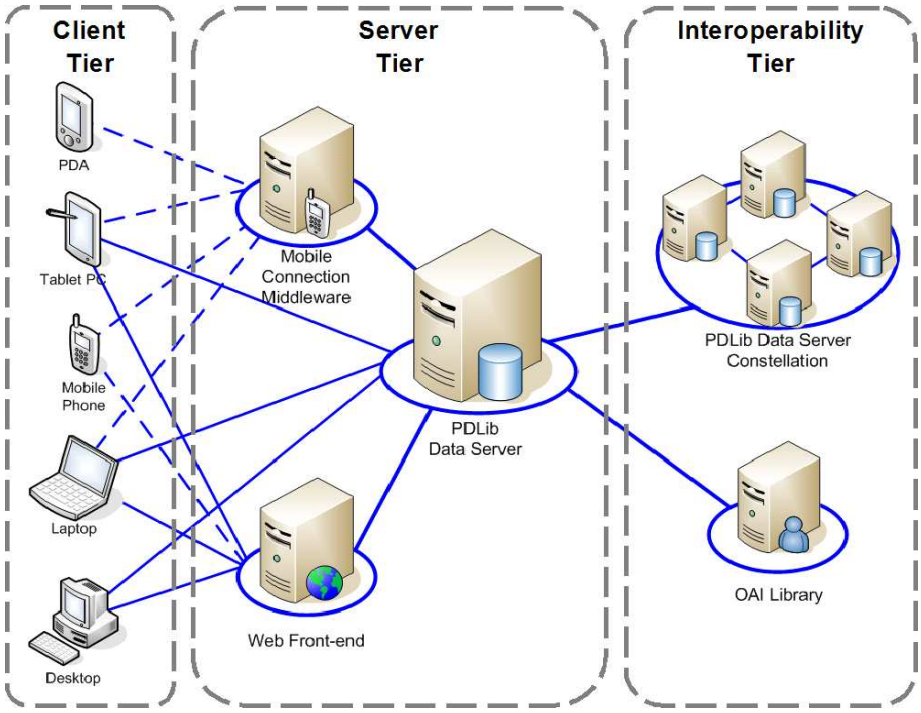

Figura 4.1: Esquema general del sistema PDLib. Figura tomada de [20].

2. *Server Tier*. Infraestructura servidor que provee los servicios a los clientes. Esta capa está compuesta por el *Data Server* (DS), el *Mobile Connection Middleware* (MCM) y el *Web Front-end*.

3. *Interoperability tier*. Incluye los servicios de interoperabilidad con otros servidores PDLib y sistemas de bibliotecas digitales que cumplen con el estándar *OAI-MPH* [23].

Los dispositivos de la capa cliente se comunican con la capa del servidor (*server tier*) para obtener acceso a los diferentes servicios de la biblioteca digital PDLib. El tipo de acceso de los clientes con el servidor varía de acuerdo a las características del dispositivo. Los accesos pueden ser de cualquiera de los siguientes tipos:

Acceso por Middleware. Da soporte a dispositivos móviles, especialmente aquellos con recursos de cómputo limitados (e.j: PDA, teléfonos móviles con soporte HTTP). Estos accesos se representan mediante las lineas punteadas entre los clientes y el *MCM* de la figura 4.1.

Acceso por Web. Provee acceso mediante HTTP a cualquier dispositivo que incluya un navegador Web (p.e: WML/HTML micronavegadores)

Acceso directo. Proporciona acceso directamente con el *data server* a aplicaciones con requerimientos muy particulares. Los accesos directos están representados por las líneas continuas entre los clientes y el *Data Server*.

En la figura 4.2 se muestra la arquitectura del sistema PDLib definida en [4]. Como se puede apreciar, el componente principal del sistema PDLib es el *Data Server*. Una de las funciones primordiales del *Data Server* (DS) es almacenar los objetos de las bibliotecas digitales personales (colecciones, documentos, metadatos, etc). Debido a que el DS tiene contacto directo con los documentos de las bibliotecas personales, se decidió integrarle KEA para aprovechar todas las funcionalidades que este proporciona para el acceso a dichos objetos. En las siguientes secciones se describirán los componentes de la arquitectura de PDLib, iniciando con los clientes, el *Web Front-end*, el *MCM* y por último el *Data Server* y la integración de KEA.

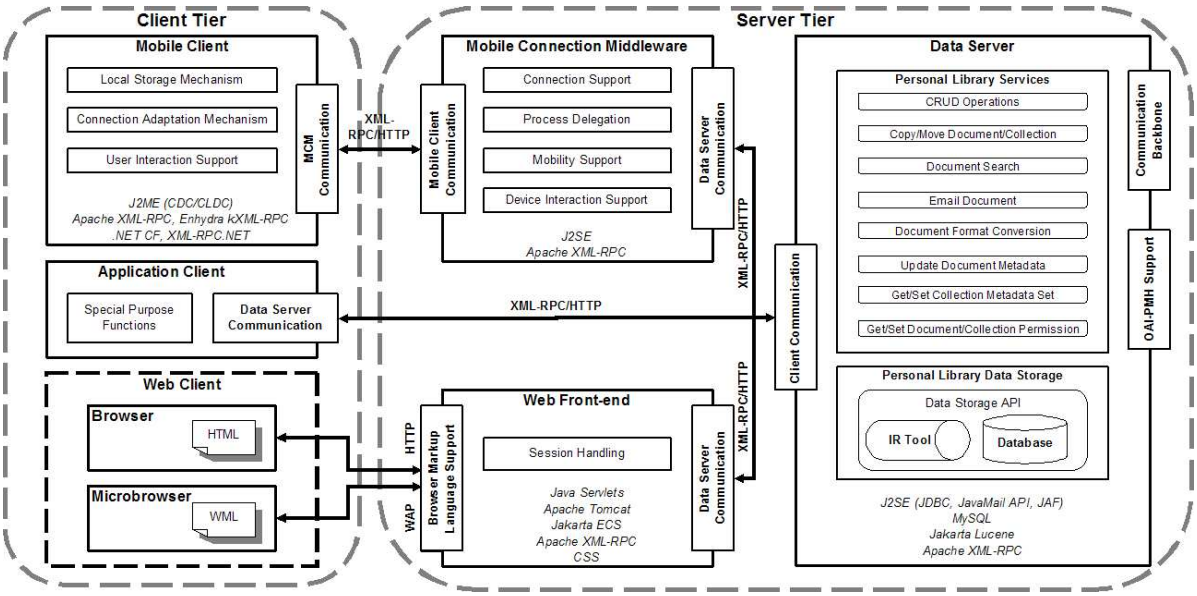

Figura 4.2: Arquitectura del sistema PDLib. Figura tomada de [20].

4.1.1 Clientes

Uno de los objetivos primordiales del proyecto PDLib es que cada usuario tenga las facilidades necesarias para acceder y organizar su librería digital en cualquier lugar usando casi cualquier dispositivo electrónico (acceso universal). Para proveer a los usuarios con dicho acceso, en [4] definen diferentes tipos de aplicaciones cliente, las cuales pueden ser clasificadas de acuerdo a su *arquitectura del lado cliente* en: clientes Web (*thin clients*) y clientes pesados (*thick clients*); y de acuerdo a su *movilidad* en: clientes fijos (*fixed clients*) y clientes móviles (*mobile clients*). Esta clasificación de clientes se puede apreciar mejor en la figura 4.3 propuesta por [4].

- *Clientes Web* (clientes ligeros fijos y móviles). En esta categoría se encuentran los dispositivos capaces de mostrar el contenido de una página en formato HTML o

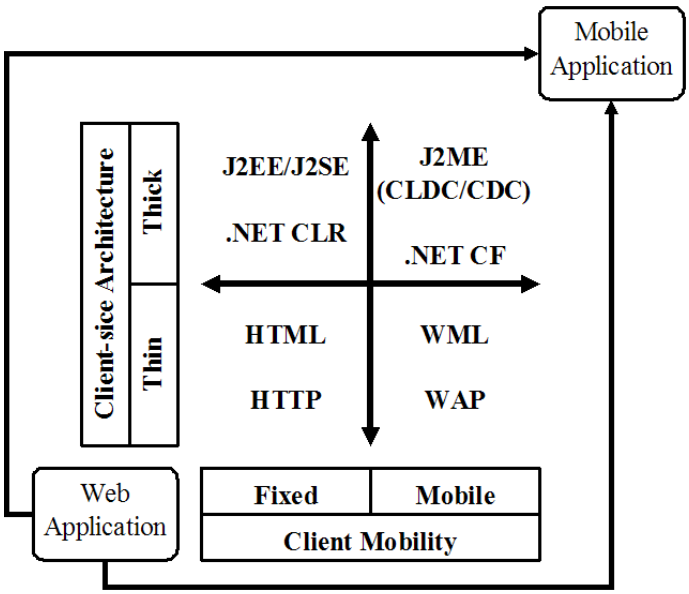

Figura 4.3: Clasificación de clientes en PDLib. Imagen tomada de [4].

WML. A pesar que estos clientes no cuenten con todas las características de un cliente pesado, proveen soporte de interacción básica con PDLib. Estos clientes Web se comunican con el *Data Server* a través del *Web Front-end*.

- *Aplicación cliente* (clientes pesados fijos). Este tipo de aplicación está diseñada para ejecutarse en computadoras de escritorio o portátiles que tiene pocas limitaciones de recursos de cómputo. Estas aplicaciones tienen la capacidad de comunicarse directamente con el *Data Server* y presentar interfaces gráficas más ricas que las de un cliente Web.

- *Clientes móviles* (clientes pesados móviles). Estos clientes fueron diseñados para lidiar con las limitaciones inherentes al ambiente móvil. Esto implica una redefinición de las funcionalidades que proporcionan los clientes pesados fijos, con la finalidad de proveer una abstracción de una biblioteca digital en el dispositivo móvil. Estos clientes requieren de un *middleware* (MCM) para la comunicación con el *Data Server*. Una de las tareas del presente trabajo de tesis fue elaborar un prototipo de cliente móvil que aproveche las *keyphrases* obtenidas por KEA para proponer esquemas de presentación de resultados obtenidos a partir de una búsqueda de documentos. Este prototipo se detalla en la sección 4.3.1. Las tareas principales de los clientes móviles son:

  1. Tener *mecanismos de almacenamiento local* que permitan visualizar documentos en el dispositivo móvil sin necesidad de tener conexión al *Data Server* (*trabajo off-line*).
  2. Proveer *mecanismos de adaptación a la conexión* que permitan tener un tiempo de respuesta constante a pesar de la variabilidad de la conexión

inalámbrica. En [46] se presenta una técnica para proporcionar tiempos de respuesta constantes mediante el cálculo de la ventana de transferencias de información y predicción del estado de la red.

4.1.2 Web Front-end

El *Web Front-end* es el componente encargado de proporcionar los servicios de biblioteca personal digital a la aplicación Web. El *Web Front-end* genera código WML o HTML de acuerdo al dispositivo que lo solicita, es decir, si la petición la hace un *microbrowser* desde un dispositivo ligero (*thin client*), el *Web Front-end* responderá la petición en formato WML, mientras que si la petición la hace un navegador Web desde una computadora de escritorio, la respuesta que enviará el *Web Front-end* será en formato *HTML*.

Para mantener la interacción de un cliente Web con el *Web Front-end* a través de una petición HTTP, se usa el mecanismo de manejo de sesiones.

4.1.3 Mobile Connection Middleware

Uno de los principales objetivos del sistema PDLib es proporcionar a los usuarios un *acceso universal* a su biblioteca digital personal. Este objetivo ocasiona que el proporcionar los servicios del *data server* sea uno de los principales problemas a resolver, ya que dentro de la variedad de dispositivos clientes disponibles para PDLib (Tablet PC, PDA, celulares, PCs, Laptops), los de menor disponibilidad de recursos de cómputo, pero mayor tendencia a la movilidad, son los que van ganando popularidad (e.j. PDA, celular). Debido a esto, fue necesaria la creación de un *middleware* que funcione como mediador de la comunicación entre el *data server* y los clientes móviles. A este *middleware* se le denominó *Mobile Connection Middleware* (MCM), el cual es responsable de proveer las siguientes funcionalidades:

- *Soporte a la Conexión*. Realiza tareas que proporcionen la adaptabilidad que el cliente móvil necesita para contrarrestar la variabilidad y limitaciones de ancho de banda que caracterizan a las conexiones inalámbricas.

- *Delegación de Procesos*. Ejecutar las tareas de procesamiento más demandantes, de manera que los dispositivos móviles no agoten sus recursos en este tipo de tareas si pueden ser evitadas.

- *Soporte a la movilidad*. Aplicar técnicas de *prefetching* para predecir comportamientos del usuario y adelantar la recuperación de documentos desde el *data server* y almacenarlos en servidores de *cache* cercanos al usuario. Esto implica que se proporcione también un soporte al cambio de ubicación del usuario mediante la migración de información entre diferentes instancias del *MCM*.

- *Soporte a la interacción con dispositivos.* Aplicar técnicas de adaptación de contenido que adecuen a las características del dispositivo mediante el cual se esté accediendo.

En [46], se presenta una descripción más detallada de la arquitectura del MCM y las características y servicios que proporciona.

4.1.4 Data Server

El *data server* (DS) es uno de los principales componentes del sistema PDLib. El DS provee los servicios de una biblioteca digital personal, almacena la información de la biblioteca digital y provee la interoperabilidad mediante el protocolo OAI-MHP [23]. El DS proporciona las siguientes funcionalidades[4]:

- *Servicios de biblioteca digital personal.* El DS permite las operaciones *CRUD* (*Creation, Retrival, Update, Delete*) sobre los objetos de la biblioteca (colecciones, documentos, metadatos) almacenados en el espacio personal de información asignado para cada biblioteca digital personal. Además de los servicios *CRUD*, el DS proporciona, entre otros, los servicios para copiar o mover colecciones y/o documentos, buscar documentos en colecciones públicas y privadas, enviar documentos a las bibliotecas de otros usuarios PDLib y a cualquier otro usuario vía correo electrónico y realizar conversiones de documentos a diferentes formatos.

- *Almacenamiento personal de información.* Este servicio permite almacenar e indexar, mediante el uso de un motor de búsqueda de texto, el contenido de los documentos de las bibliotecas digitales personales. Se utiliza una base de datos para almacenar los objetos de las bibliotecas digitales personales, ya que se requiere un modelo estructurado para representarlos y relacionarlos entre ellos con la finalidad de realizar búsquedas de texto completo en los documentos y en sus metadatos.

- *Soporte OAI.* El DS expone los metadatos de los documentos de las bibliotecas personales digitales mediante el protocolo OAI-MHP[23], por lo que usuarios de otras bibliotecas digitales que cumplan con este protocolo pueden acceder a los documentos PDLib (que se tengan con permiso de acceso público) y viceversa.

Parte del presente trabajo de tesis consistió en implementar al DS la capacidad de extraer automáticamente las *keyphrases* de los documentos que se agreguen a las bibliotecas personales digitales. En la siguiente sección se describirá la integración de KEA con el DS.

4.2 Arquitectura de integración Data Server - KEA

Debido a las limitaciones de tamaño de pantalla que se tienen en los clientes móviles, es importante proponer alternativas que permitan presentar al usuario PDLib, información relevante que le permita identificar a simple vista si un determinado documento le es relevante o no. El uso de *keyphrases* en este proceso de identificación de documentos relevantes para usuarios PDLib que accesen desde clientes móviles es el objetivo principal de este trabajo de tesis.

Con la finalidad de implementar la extracción automática de *keyphrases* como un servicio que permita presentar información relevante a los usuarios PDLib, se decidió integrar KEA al Data Server, ya que es el componente encargado de proporcionar la mayoría de los servicios de PDLib, además de tener acceso directo a los objetos de las bibliotecas digitales personales.

Para explicar de una manera más entendible la integración *Data Server* con KEA, se mostrará la arquitectura del *Data Server* y del *Data Server-KEA*, para finalizar con un diagrama de secuencia de ejemplo de la creación de un documento desde un cliente móvil para ambas arquitecturas.

4.2.1 Arquitectura de servicios del Data Server

En la figura 4.4 se muestra un diagrama con la arquitectura actual del *Data Server* (DS). Como se puede apreciar, existen dos formas de interactuar con el DS, la más común es mediante el protocolo XML-RPC, la cual es usada por la mayoría de los clientes del DS (*Mobile Connection Middleware* (MCM), *Web Front-End*) y permite usar todos los servicios del DS. Una segunda forma de acceso al DS es mediante la Sindicación Web, la cual puede accederse directamente desde cualquier navegador Web y permite el uso de los servicios de navegación de colecciones y búsqueda de documentos. Esta versión de *Data Server* proporciona, además de los servicios de biblioteca personal digital mencionados en la sección 4.1.4, los siguientes servicios:

- *Sindicación*. Permite integrar bibliotecas digitales públicas de PDLib con programas denominados *agregadores* mediante el protocolo *RSS* [29].
- *Comunicación XML-RPC*. Este es el servicio que permite al Data Server exponer todos sus servicios de biblioteca personal digital para que sean accedidos por los diferentes tipos de clientes (p.e. Web Front-end) a través del protocolo de comunicación XML-RPC[61].
- *Soporte OAI*. Este servicio proporciona la interoperabilidad con otras bibliotecas digitales que cumplan con el protocolo OAI-MHP[23]. Como se puede apreciar en la figura 4.4, este servicio está montado sobre el protocolo de transporte XML-RPC[61].

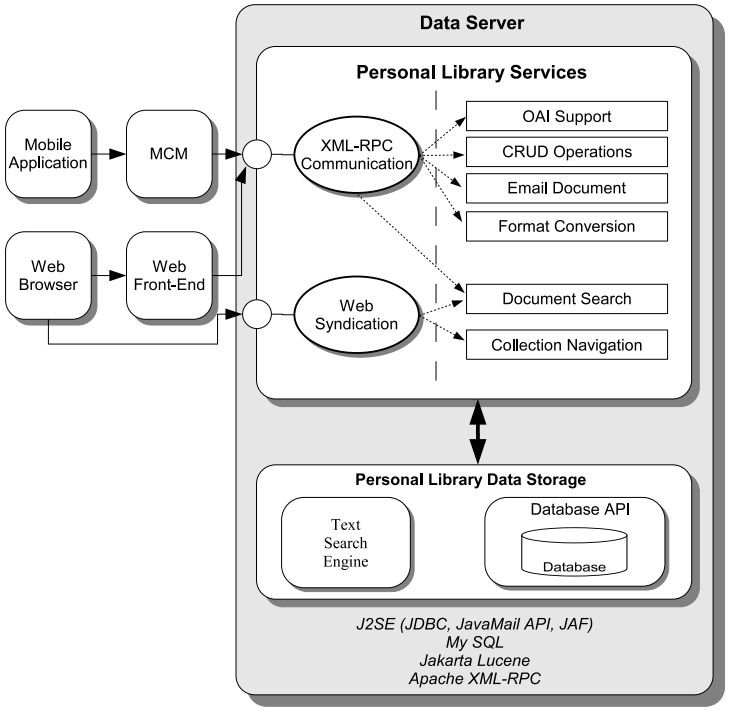

Figura 4.4: Arquitectura del *Data Server* sin la integración de KEA

4.2.2 Arquitectura de servicios del Data Server - KEA

Una vez finalizadas las modificaciones en KEA para paralelizar el algoritmo e implementar las adecuaciones requeridas para el soporte a conexión con base de datos[3], se procedió a la integración con el *Data Server*, la cual denominamos *DS-KEA*.

En la figura 4.5 se muestra la arquitectura de servicios de la integración *Data Server*-KEA. Como se puede apreciar, se agregó el nuevo servicio *KEA Operations* que depende de los servicios *CRUD* (Create, Retrieve, Update, Delete) de las colecciones/documentos, esto con la finalidad de tener una sincronía de la información que requiere KEA sobre los documentos y colecciones existentes en cada biblioteca personal digital: por ejemplo, cuando se agrega un nuevo documento a la biblioteca personal digital, KEA guarda una referencia a ese nuevo documento para poder extraerle las *keyphrases* y/o en su caso, considerarlo para la construcción del modelo (tomando en cuenta las consideraciones que se explican a continuación en la parte de creación de documentos).

Los servicios que proporciona *KEA* en la integración con el *Data Server* son accedidos a través del servicio de comunicación por XML-RPC, lo que implica que cualquier cliente del *Data Server* que se comunique mediante este servicio puede acceder a los servicios de KEA de manera transparente. Los servicios proporcionado por *KEA Operations* son, entre otros:

[3]La integración se realizó para MySQL 4.1.X

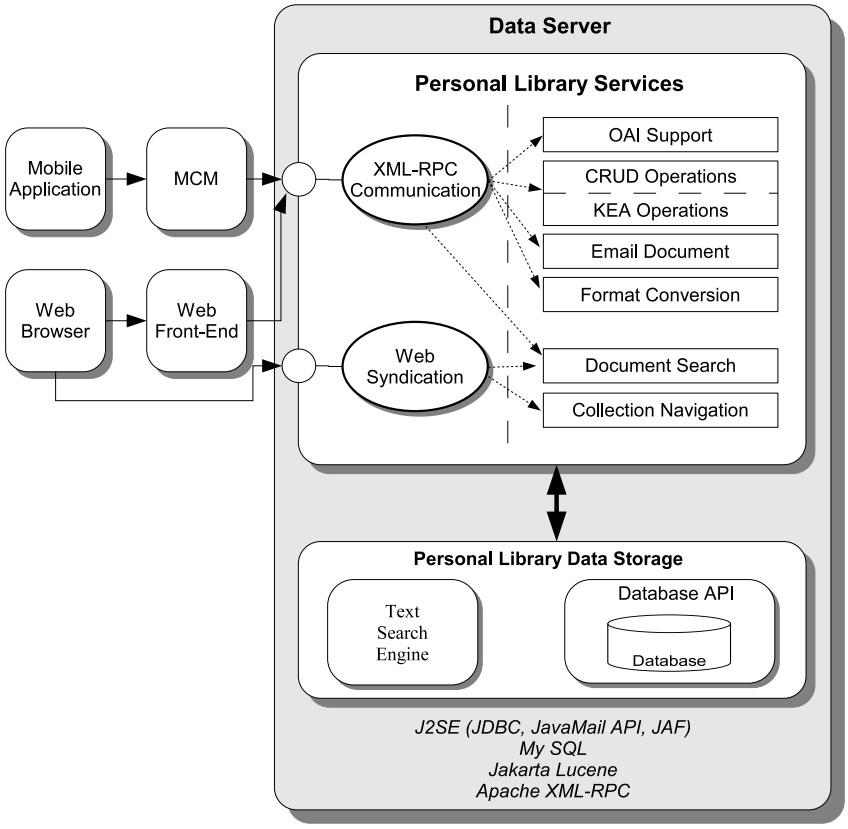

Figura 4.5: Arquitectura de servicios de la integración Data Server-KEA

- *Activar/Desactivar KEA*. Este servicio permite definir si se activa KEA para una biblioteca digital personal dada. Cuando KEA se encuentra activo, la generación de modelos y la extracción de *keyphrases* se ejecutan de manera automática para las colecciones y documentos de cada biblioteca personal digital.

- *Creación de modelos*. Permite crear un modelo para cada colección de las bibliotecas digitales personales. Esto permite tener dominios de información, con la finalidad de explotar la característica de utilización de modelos específicos en la extracción automática de *keyphrases*.

- *Creación de documentos*. Este servicio es dependiente de las operaciones *CRUD* del *Data Server*. Después de crear el nuevo documento de manera tradicional por el servicio *Create*, se determina si este documento puede ser agregado a la lista de documentos de entrenamiento de KEA, esto se hace tomando en consideración lo siguiente:

  1. El nuevo documento debe pertenecer a una biblioteca/colección para la cual KEA esté activo.
  2. El conjunto de metadatos del nuevo documento debe ser igual al conjunto de metadatos asignado para KEA.

3. El nuevo documento no debe tener el metadato *keywords* vacío, ya que este metadato contiene las *keyphrases* definidas por el autor y son usadas en el proceso de construcción del modelo.

Si el nuevo documento cumple con estos tres criterios, entonces es agregado a la lista de documentos de entrenamiento de KEA para ese par *biblioteca-colección*, de lo contrario, solo se le extraen las *keyphrases* para el caso de existir ya un modelo para la colección a la que se está agregando el documento, en el caso contrario, las keyphrases serán extraídas la próxima vez que se haga la construcción de modelos y la extracción de *keyphrases* programadas por el administrador del servidor PDLib.

- *Extracción automática de keyphrases*. Permite extraer automáticamente las *keyphrases* de un documento. Este servicio también se puede utilizar para extraer las *keyphrases* de todos los documentos de una colección ó para todas las colecciones de una biblioteca digital personal, por lo que se puede ejecutar por el administrador del servidor PDLib para correr periódicamente (e.j. una vez por semana, en la noche) como una tarea programada.

En la figura 4.6 se muestra un diagrama de secuencia con el ejemplo de la creación de un documento desde un cliente móvil. En la figura 4.6(a), el cliente móvil solicita agregar de un documento a su biblioteca personal digital (*createDocument*), esta solicitud de creación de documento es enviada al *MCM* el cual a su vez, le indica al *DS* que realice las operaciones requeridas para agregar el documento a la biblioteca personal digital del usuario. El protocolo utilizado para la comunicación *Cliente Móvil-MCM-DS* es *XML-RPC*.

En la figura 4.6(b) se muestra el mismo proceso de creación de documento, con la diferencia que ahora se tiene la implementación *DS-KEA*, lo que provoca que después de que el documento es agregado a la biblioteca personal digital (por uno de los servicios *CRUD*), se ejecuten también los servicios requeridos para agregar dicho documento a KEA (*AddDocument* y *Extract Keyphrases*), dichos servicios se determinan con base en las validaciones explicadas anteriormente en esta sección, en el servicio de *creación de documentos*.

Con la integración DS-KEA, se brindan nuevas alternativas de presentación de información relevante en los clientes PDLib. En la siguiente sección se describe una forma de aprovechar el servicio de extracción automática de *keyphrases* de la integración DS-KEA para proponer alternativas de visualización de resultados de búsquedas en dispositivos móviles (*thick clients*), de manera que se agilice la localización de un documento de interés para el usuario PDLib.

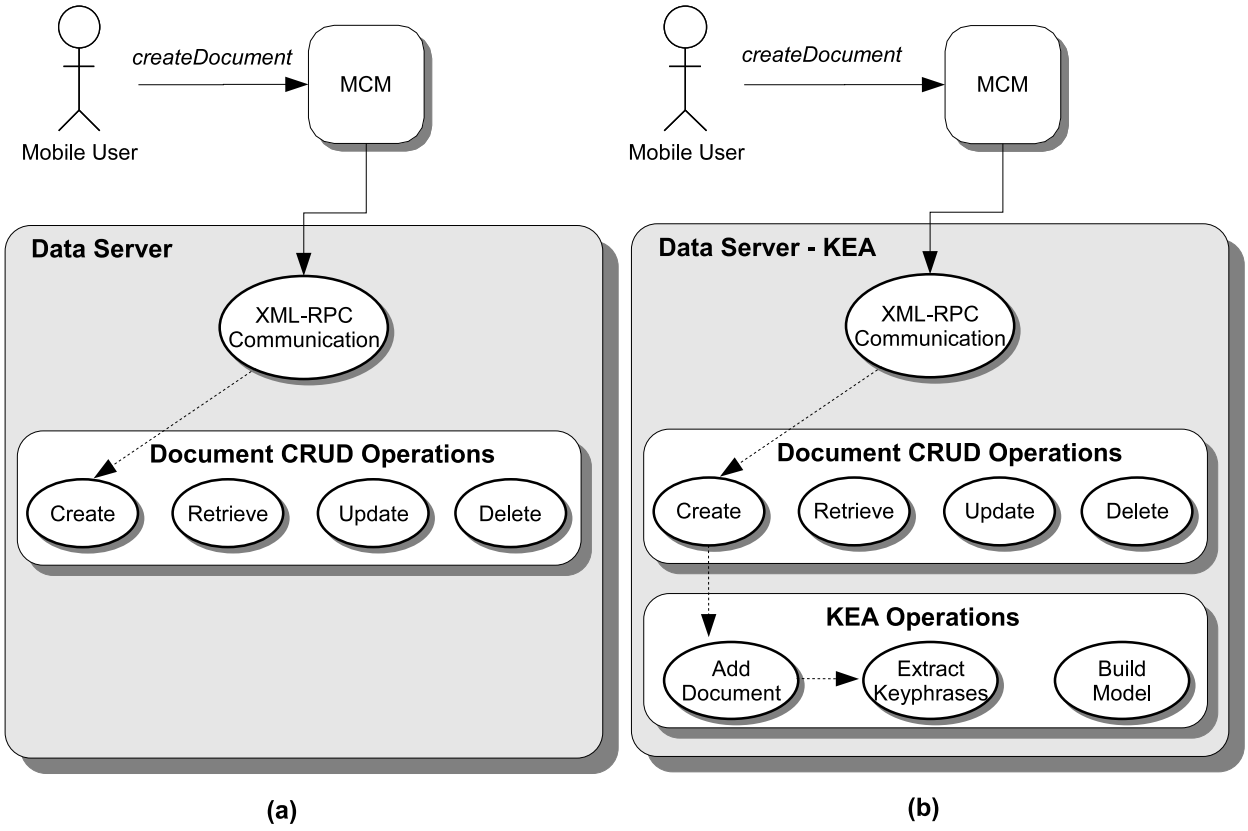

Figura 4.6: Diagrama de secuencia para crear un documento desde un cliente móvil en (a) *Data Server* y (b) *Data Server-KEA*

4.3 Prototipo cliente móvil PDLib.

Además de las adecuaciones realizadas a KEA para su implementación con el Data Server (lo que llamamos DS-KEA), una gran parte de este trabajo de tesis es desarrollar un prototipo de cliente móvil para PDA (*thick client*). El objetivo de este prototipo es implementar diferentes pantallas de visualización de resultados para las búsquedas realizadas por un usuario PDLib es su dispositivo PDA. Dichas pantallas pretenden presentar información relevante de cada documento encontrado, de manera que el usuario pueda, de forma fácil y rápida, determinar si alguno de los documentos de la lista de resultados es el que requiere en ese momento.

Para implementar este prototipo de cliente móvil, se tenían principalmente dos alternativas: *Java J2ME* y *Microsoft .Net Compact Framework*, las cuales se describen en la sección 2.3.2. Cuando se inició el desarrollo de este prototipo, no existía una máquina virtual para J2ME gratuita que ofreciera desarrollo de interfaces de usuario ricas en componentes (como Java Swing en computadoras de escritorio), además de que las pruebas realizadas con la máquina virtual J9 de IBM nos ayudaron a identificar

que sólo se cumplía con la implementación para algunos de los componentes de Java AWT. Debido a esto, se eligió desarrollar en Microsoft .Net Compact Framework, por ser la arquitectura que ofrece las mejores características visuales para el desarrollo de interfaces gráficas ricas en componentes, además de que se puede redistribuir fácilmente con instaladores similares a los existentes en aplicaciones de windows. El lenguaje que se utilizó para realizar la implementación de este prototipo es *C#*. Se eligió este lenguaje por su gran parecido al lenguaje Java, lo que facilita la curva de aprendizaje y el aprovechamiento de su potente arquitectura orientada a objetos.

En las siguientes secciones se describirá la arquitectura y los servicios del cliente móvil de PDLib para Pocket PC, al cual denominamos *Pocket Client*.

4.3.1 Arquitectura Pocket Client

El cliente móvil de PDLib para Pocket PC (PDLib Pocket Client) se desarrolló con las finalidad de proponer el uso de las *keyphrases* (generadas automáticamente por la integración DS-KEA descrita en la sección 4.2) como una forma de presentar información relevante que facilite el proceso de localización de documentos en el resultado obtenido al realizar una búsqueda desde un cliente PDLib móvil. Debido a que el prototipo de cliente móvil para PDLib ([4]) desarrollado para Palm OS en J2ME (configuración CLDC MIDP 2.0), no soporta el uso de interfaces ricas en componentes gráficos, se realizó un prototipo con funcionalidades básicas de un cliente móvil PDLib, además de las requeridas para el uso de las *keyphrases*, para lo cual se requirió definir una arquitectura que facilitará dicha implementación.

Debido a las limitaciones inherentes al desarrollo de aplicaciones para dispositivos móviles descritas en [43], es importante tener una arquitectura que garantice la correcta funcionalidad de la aplicación a pesar de las limitantes de los dispositivos. Teniendo en cuenta estas limitantes, se decidió usar una arquitectura tipo MVC (*Model-View-Controller*), que nos permite tener las siguientes ventajas:

- *Claridad en el diseño*. Los métodos públicos del modelo se consideran como un *API* de comandos para manipular el estado y la información de la aplicación. Con un listado de los métodos públicos del Modelo, es fácil entender, a simple vista, como controlar el comportamiento del modelo.

- *Múltiples vistas*. Es fácil agregar/eliminar vistas (formas) de la aplicación sin afectar las ya existentes, ya que cada vista maneja su propia lógica de negocio, y usa el Modelo para los métodos comunes de todas las vistas. Esto también permite que la aplicación sea escalable.

- *Reusabilidad y Modularidad*. Debido a que el Modelo está completamente desacoplado de las vistas, permite diseñar e implementar el Modelo considerando reusabilidad y modularidad.

Existen diferentes implementaciones de arquitecturas MVC para .Net [33, 38], pero cuando se inició el desarrollo de este trabajo, no había alguna compatible con el .Net Compact Framework, por lo que se diseñó e implementó una. En la figura 4.7 se muestra la arquitectura definida. Además de las características/ventajas mencionadas anteriormente, se implementó también un *cache* de formas, que permite almacenar una referencia a las formas ya instanciadas y evita crear nuevamente una del mismo tipo (solo actualiza su contenido), por lo que se ahorra tiempo de construcción en formas ya existentes. Esta arquitectura MVC facilita la separación de código mediante el uso de la modularidad por capas [45]. Cada capa realiza tareas específicas y proporciona diferentes servicios. A continuación se describen algunos de los servicios proporcionados por cada capa:

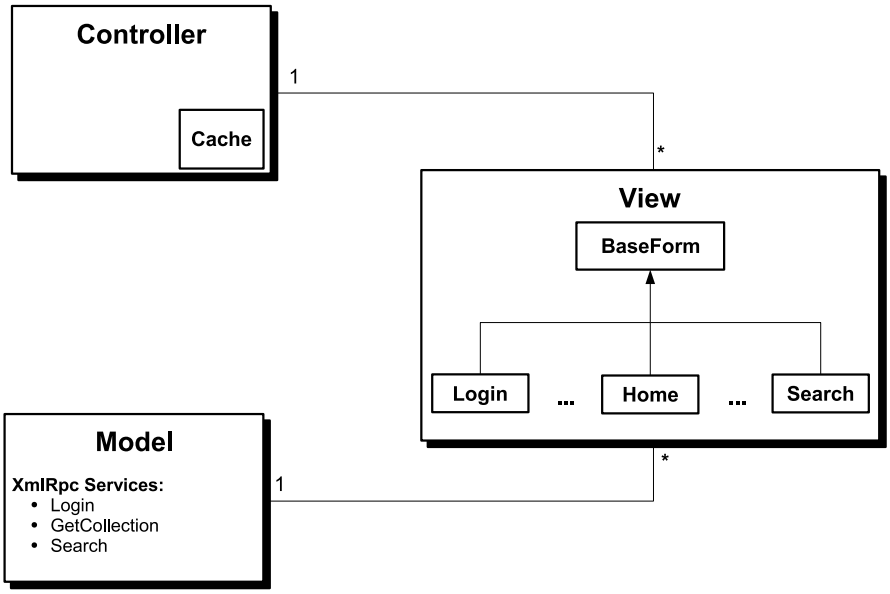

Figura 4.7: Diseño de la arquitectura MVC para .Net Compact Framework

1. Capa *Model*. Esta capa proporciona los servicios de conectividad con el *Data Server* a través del MCM (*Mobile Connection Middleware*) mediante el protocolo de transporte XML-RPC. En la figura 4.8 se representa esta capa mediante la clase *Data Pool*. Algunos ejemplos de los servicios que proporciona la capa *Model* son:

   - *Login*. Realiza la identificación del usuario PDLib proporcionando el correo electrónico y la contraseña de acceso al sistema PDLib.
   - *getLibraries*. Proporciona una lista de las bibliotecas disponibles para el usuario.
   - *getCollections*. Devuelve las colecciones y documentos asociados a una biblioteca digital personal.

- *Search*. Ejecuta el servicio de búsqueda del *Data Server*. Devuelve la lista de documentos que cumplen con un criterio de búsqueda.

2. Capa *View*. Esta capa proporciona la interfaz gráfica con la cual el usuario puede interactuar e invocar, de manera transparente, los servicios proporcionados por las capas *Model* y *Controller*. La interfaz gráfica desarrollada consta de varias pantallas ó formas, algunas de ellas son: *Login*, *Home* y *Search* (ver figura 4.8).

3. Capa *Controller*. Esta capa se encarga del flujo de la aplicación. La capa *controller* también proporciona un *cache* de formas, lo que permite crear la forma solo la primera vez que se utiliza. Si se vuelve a mostrar una pantalla que ya se ha creado, solo se actualiza la información que requiere la pantalla a mostrar, evitando así el tiempo de construcción de la misma.

En la figura 4.8 se muestra un modelo conceptual de la implementación de la arquitectura MVC para el *Pocket client*. La capa *Model* está representada por la clase *DataPool*, la cual contiene los diferentes objetos que proporcionan los servicios de conectividad con el *Data Server* a través del MCM (*Mobile Connection Middleware*. La capa *Controller* se representa por un objeto de la clase *Controller* heredado en todas las formas y permite a cada una controlar el flujo de la aplicación: por ejemplo, la forma *Login* puede hacer que se muestre la forma *Home* mediante el objeto *Controller* usando el método *show*, de la misma manera, la forma *Home* puede hacer regresar el flujo hacia la forma *Login* utilizando el método *close* del objeto *Controller*, o puede mostrar la forma *Search* utilizando el método *show*. La capa *View* está representada por cada forma: *Login, Home, Search, etc.*

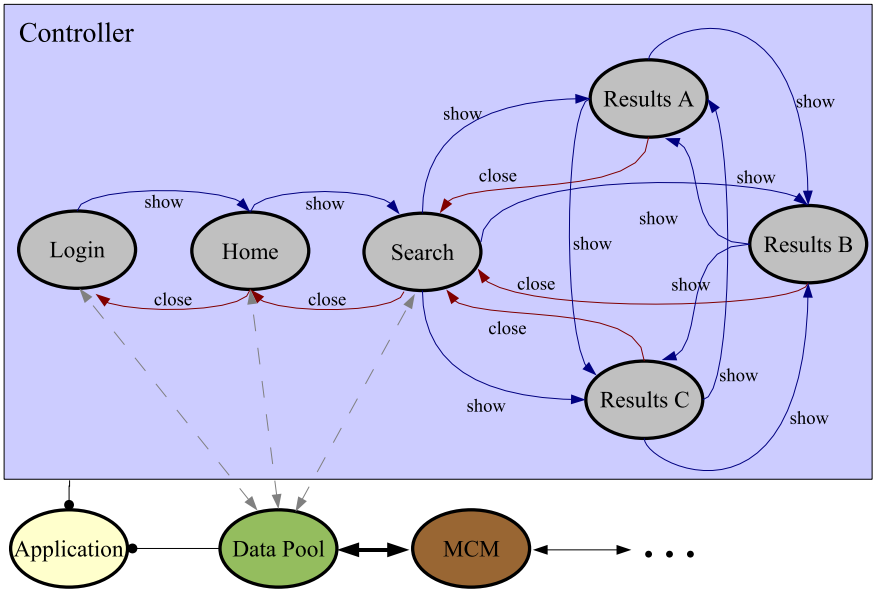

Figura 4.8: Modelo conceptual de la arquitectura MVC implementada en el PDLib Pocket Client

Característica	**Palm OS**	**Pocket PC**
Identificación de usuario (en biblioteca personal)	X	X
Navegación de colecciones con paginación	X	X
Crear/Eliminar de colecciones	X	X
Crear Documentos	X	
Eliminar Documentos	X	X
Copiar/Mover Documentos	X	
Ver metadatos de un documento	X	X
Editar metadatos de un documento	X	
Enviar documento por correo	X	
Búsqueda de documentos	X	X
Descarga y apertura de documentos		X
Uso de keyphrases en resultados de búsquedas		X

Tabla 4.1: Funcionalidades disponibles para clientes Palm OS y Pocket PC.

4.3.2 Servicios del Pocket Client

En la tabla 4.1 se muestran las características del cliente móvil de Palm OS y el de Pocket PC. Como se puede apreciar, el prototipo para Palm OS tiene todas las funcionalidades básicas requeridas para la abstracción de una biblioteca personal digital en la PDA: Servicios CRUD (*Create, Retrieve, Update, Delete*) para documentos y colecciones, autenticación de usuario, obtención y actualización de metadatos de un documento, envío de documentos por correo electrónico y búsqueda de documentos.

El cliente Pocket PC por el contrario, no cuenta con las funcionalidades de enviar documento por correo, editar metadatos y crear documentos, pero estas funcionalidades están fuera del alcance de este trabajo de tesis. Lo relevante del nuevo prototipo de Pocket PC son las funcionalidades para descargar y abrir un documento seleccionado (ya sea en la navegación de colecciones o en el resultado de un a búsqueda), dicho documento es abierto con la aplicación correspondiente de acuerdo a su tipo (un .pdf con el Pocket Acrobat Reder, un .doc con Pcoket Word, etc), siempre y cuando la aplicación se encuentre instalada en el dispositivo; también cuenta con la funcionalidad de presentar diferentes alternativas de visualización de resultados de una búsqueda mediante el uso de las *keyphrases* extraídas automáticamente por KEA, lo cual representa el objetivo principal de este trabajo de tesis.

En la figura 4.9 se muestran algunas pantallas del prototipo. En la figura 4.9(a) se muestra la pantalla de *Login* (identificación de usuario) al sistema PDLib. La figura 4.9(b) muestra la navegación con paginación en las colecciones de una biblioteca digital personal. La figura 4.9(c) muestra el servicio de descarga y apertura de un documento en el dispositivo (PDA) del usuario. La descarga de un documento se hace con un *thread* de baja prioridad, lo que permite seguir navegando entre las colecciones mientras se efectúa la descarga. La apertura del documento deseado se hace de acuerdo al tipo de

documento descargado (e.j. Un .pdf se abre con el Pocket Acrobat Reader, un .doc se abre con el Pocket Word). Por último, la figura 4.9(d) muestra la pantalla para realizar una búsqueda de documentos de la biblioteca digital personal.

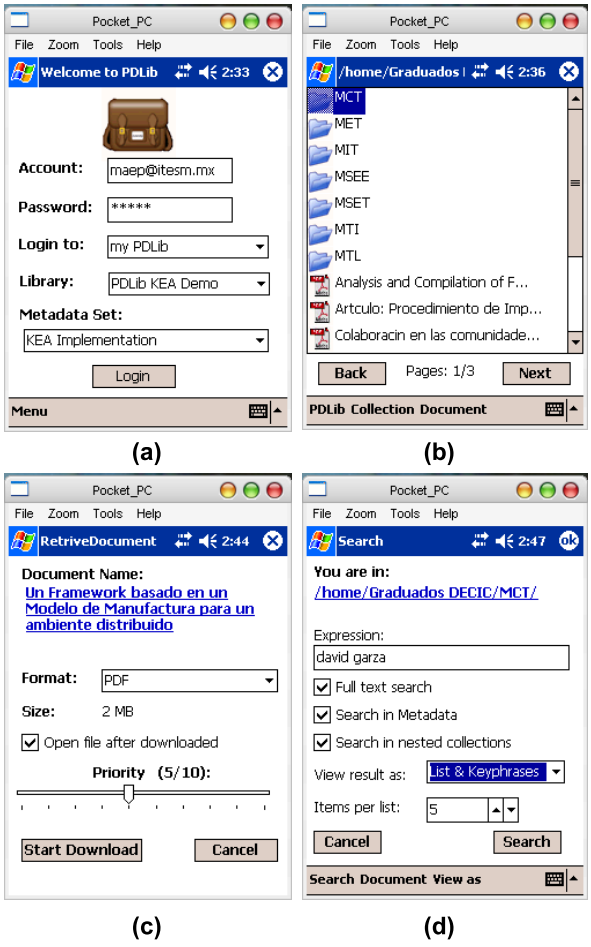

Figura 4.9: PDLib Pocket Client (C# .Net CF)

En la siguiente sección se describirán las pantallas definidas para el uso de *keyphrases* en los resultados de una búsqueda de documentos.

4.3.3 Uso de *keyphrases* en PDLib Pocket Client

El uso de *keyphrases* presentadas como información adicional que ayuden en la localización de documentos relevantes en el resultado de una búsqueda desde un cliente PDLib móvil es una de las principales motivaciones de este trabajo de tesis. Este servicio del *Pocket client* es una de las primeras formas de aprovechar la característica de extracción automática de *keyphrases* descrita en la sección 4.2.

Para usar las *keyphrases*, se diseñaron 2 pantallas que muestran los resultados de la búsqueda de documentos:

1. *List + Keyphrases*
2. *List + Metadata*

En la figura 4.10(a) se muestran los resultados de la búsqueda en formato *List + Keyphrases*. Como se puede apreciar, las *keyphrases* asociadas a cada documento se muestran abajo del título del mismo, lo que facilita que el usuario vea al mismo tiempo el título y las *keyphrases* e identifique rápidamente de qué se trata el documento sin necesidad de abrirlo. Esta pantalla también cuenta con un mecanismo de paginación, lo que permite saber si es necesario un refinamiento de el criterio de búsqueda introducido, ya que si se tiene un número de paginas totales muy grande, es muy probable que se requiera de un refinamiento en el criterio de búsqueda.

En la figura 4.10(b) se muestran los resultados de la búsqueda con el formato *List + Metadata*. Este formato permite visualizar los nombres de los documentos y al mismo tiempo ver los metadatos del mismo, iniciando por las *keyphrases* del documento. El conjunto de metadatos definidos en el *DS-KEA* son:

1. *Título*. Define el título del documento.
2. *Keywords*. Se utiliza para guardar las *keyphrases* definidas por el autor del documento.
3. *Autor*. Contiene el nombre del autor del documento.
4. *Abstract*. Guarda el resumen del documento.
5. *Lenguaje*. Especifica el lenguaje en el que se escribió el documento (e.j. Español).
6. *Editor*. Indica el editor del documento (e.j. ITESM).
7. *Fecha*. Fecha de publicación del documento.
8. *Tipo*. Contiene el formato fuente del documento (e.j. Pdf, Doc, etc).
9. *Keyphrases*. Este metadato guarda las *keyphrases* generadas automáticamente por el *DS-KEA*.

Esta pantalla de visualización de resultados (4.10(b)) es la más completa, ya que presenta toda la información disponible para cada documento. Si la información proporcionada por el título y las *keyphrases* del documento no son suficientes para saber si es lo que se está buscando, se puede utilizar fácilmente cualquier otro metadato para discriminarlo, basta con ajustar la barra de desplazamiento vertical al nivel en el que aparezca el metadato deseado.

Además de estas dos pantallas de visualización de resultados, se cuenta con una tercer pantalla (ver figura 4.10(c)), de visualización, la cual solo muestra los títulos de los documentos encontrados. El usuario puede cambiar la vista de resultados a cualquiera de las tres (*List*, *List + KP* y *List + Metadata*) desde el menú *View as* que aparece en la parte inferior de la pantalla. Si el usuario encuentra el documento deseado, puede descargarlo en su dispositivo (siempre que tenga espacio de almacenamiento disponible) para abrirlo posteriormente sin necesidad de usar el cliente PDLib o tener conexión a la red.

En [24] se realizaron diferentes experimentos con usuarios de un *microbrowser* (*thin client*) en el que evalúan el uso de las *keyphrases* extraídas automáticamente por KEA como sustituto del título del documento en los resultados de una búsqueda. Dichos experimentos reportados por [24] indican que no hay mejoras significativas si se usan las *keyphrases* en lugar del título del documento para mostrar los resultados de la búsqueda. Sin embargo, en este trabajo de tesis el interés del uso de las *keyphrases* extraídas automáticamente no es usarlas como sustituto del título, sino usarlas como información adicional al documento, de manera que con el título y las *keyphrases*, un usuario PDLib móvil identifique rápidamente si determinado documento le es o no relevante.

Con la utilización de *keyphrases* en el *Pocket Client* se pretende realizar experimentos futuros con usuarios PDLib para identificar si efectivamente presentar las *keyphrases* como información adicional (además del título) facilita la localización de documentos, pero debido a limitaciones de tiempo, queda como trabajo futuro realizar estos experimentos.

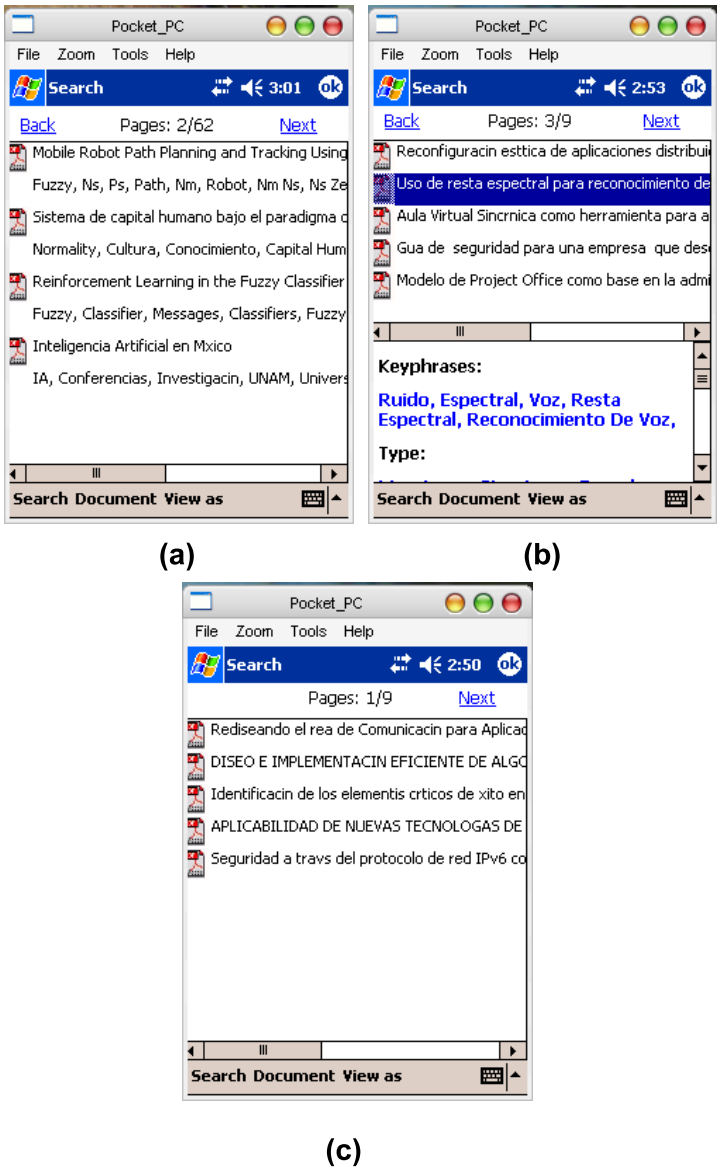

Figura 4.10: PDLib Pocket Client: Uso de *keyphrases* en los resultados de las búsquedas

4.4 Conclusiones

En este capítulo se describió la integración de KEA a uno de los componentes principales de PDLib, el *Data Server*. Esta integración, a la cual denominamos *DS-KEA*, permite que en cada ocasión que se agregue un nuevo documento a una biblioteca digital personal se le extraiga automáticamente las *keyphrases* del mismo, de una manera transparente para el usuario.

La integración *DS-KEA* es deseable para la utilización de las *keyphrases* en los resultados de búsquedas para el prototipo denominado PDLib Pocket Client, el cual se desarrollo para *high-end* PDA's que utilizan el sistema operativo Microsoft Pocket PC. Para la realización de dicho prototipo, se diseño una arquitectura tipo MVC, la cual se basa en tres capas que le dan una gran modularidad a la aplicación, lo que permite que los cambios de un componente tengan un mínimo impacto en los otros.

Debido a las limitaciones de tiempo, se propone como trabajo futuro la realización de un experimento con usuarios que determine la efectividad de las *keyphrases* como facilitadoras en el proceso de discriminación de de documentos relevantes en el resultado de una búsqueda en un cliente PDLib móvil (*thick client*).

Capítulo 5

Conclusiones y Trabajo futuro

En este último capítulo del documento se presentan las conclusiones que se obtuvieron y las alternativas de trabajo futuro que se dieron con la realización de este trabajo.

5.1 Conclusiones

La proliferación de los ambientes de comunicación inalámbrica y los dispositivos de cómputo móvil como Asistentes Personales Digitales (PDA) y celulares con capacidades de comunicación inalámbrica han ocasionado un interés de desarrollo por parte de la comunidad científica y tecnológica. El cómputo móvil ofrece retos importantes [43] debido a las restricciones que impone el ambiente, dentro de las cuales se encuentran, entre otras: la variabilidad en el ancho de banda de la conexión, las limitantes en recursos de cómputo y almacenamiento, las limitaciones de espacio para desplegar información.

En este trabajo de tesis se buscó una manera de proporcionar información adicional a los usuarios que acceden a una biblioteca personal digital desde su dispositivo móvil (PDA). Esta información adicional se encuentra en el contexto de los resultados de una búsqueda de documentos. La idea es proporcionar al usuario una breve descripción del documento que le ayude a identificar si es relevante o no, sin necesidad de leerlo por completo.

Teniendo en cuenta las limitaciones de espacio de visualización en los dispositivos móviles (específicamente las PDA's), se decidió aplicar técnicas de sumarización de contenido que nos permitan extraer de los documentos de cada usuario, información suficiente para describirlo brevemente. Existen diferentes técnicas de sumarización de contenido, una de ellas es la extracción automática de *keyphrases*, para la cual ya se han desarrollado diferentes herramientas [6, 18, 1, 53, 17], comerciales y gratuitas, que permiten hacer uso de la extracción automática de *keyphrases* para diferentes aplicaciones, dentro de las cuales se encuentran, entre otras: la sumarización, el indexamiento y la búsqueda.

Una vez identificada la herramienta a utilizar para la extracción de *keyphrases*, se procedió a realizar las principales aportaciones de este trabajo de tesis:

- La paralización del algoritmo de extracción automática de *keyphrases* de KEA y su integración con un sistema de biblioteca personal digital existente, PDLib [20].
- El desarrollo de un prototipo para dispositivos móviles (PDA's) que haga uso de este servicio de extracción automática de *keyphrases*.

La integración de KEA se realizó con el componente del sistema PDLib denominado *Data Server* (DS). A esta integración le denominamos *DS-KEA* y permite que la extracción automática de *keyphrases* sea un servicio más de los que proporciona el DS, por lo que cualquier cliente de PDLib puede hacer uso de él de manera transparente.

Los experimentos realizados en la integración *DS-KEA* con este algoritmo en paralelo demuestran que se obtuvo una mejora en el tiempo de ejecución del mismo de hasta un 63.62 % (usando 2 *threads*) con respecto del algoritmo secuencial. A pesar de este buen desempeño en términos de tiempo de ejecución, se hicieron experimentos para evaluar también el desempeño del algoritmo paralelo en términos de precisión exacta y precisión subjetiva, en los cuales se obtuvo una reducción promedio del algoritmo paralelo de un 13.72 % en precisión exacta y una reducción promedio de 0.43 % en precisión subjetiva. En el caso de la precisión exacta, un 13.72 podría considerarse alarmante, pero tenemos que considerar que la precisión exacta evalúa sólo la cantidad de *keyphrases* que son iguales a las que el autor del documento seleccionó y no evalúa si las que no son iguales son correctas o no, por lo que esta pérdida de precisión exacta se puede justificar considerando la ganancia obtenida en términos de tiempo de ejecución.

Podemos concluir entonces que la implementación en paralelo del algoritmo de extracción automática de *keyphrases* fue modificado de una manera que lo hizo muy eficiente en términos de tiempo de procesamiento, pero el hecho de separar el documento en n bloques causa que los atributos que se le asignan a cada frase para determinar si es o no una *keyphrase* se ven afectados por dicha separación. Una posible solución a este problema es mejorar el algoritmo final que selecciona la lista de las k *keyphrases* a mostrar como resultado, ya que actualmente solo considera las de mayor *peso* de cada bloque y no busca por coincidencias para hacer una suma de *pesos* de *keyphrases* iguales de diferentes bloques. Se requiere además un análisis más detallado para corregir los problemas detectados en los casos en que el número de *threads* usados para el algoritmo es mayor a 4.

La segunda aportación de este trabajo de tesis es la realización de un prototipo de cliente móvil de PDLib que hace uso de las *keyphrases* extraídas automáticamente por la integración *DS-KEA*. En este prototipo se proponen dos pantallas de visualización de resultados de una búsqueda de documentos:

1. *List + Keyphrases*. En esta pantalla se presenta un listado de documentos y *keyphrases*. Las *keyphrases* asociadas a cada documento se muestran abajo del

título del mismo, lo que permite que el usuario vea el título y las *keyphrases* del documento al mismo tiempo.

2. *List + Metadata*. Este formato de visualización muestra los nombres de los documentos al mismo tiempo que los metadatos del mismo, iniciando por las *keyphrases* del documento, lo que permite al usuario tener disponible una mayor cantidad de información relevante del documento sin requerir más acciones con la pluma del dispositivo (el peor caso se presenta cuando el usuario tenga que hacer un *scroll* por los metadatos para ver alguno de los que se listan al final de la pantalla).

Adicional a las propuestas de visualización, se diseñó e implementó una arquitectura tipo MVC que permite una mayor claridad en el diseño, agregar/eliminar vistas sin afectar las ya existentes y una modularidad y reusabilidad en el sistema. Esta arquitectura MVC permitirá realizar las modificaciones necesarias en el prototipo para trabajos futuros de una manera sencilla y eficiente.

5.2 Trabajo futuro

En esta sección se proponen algunas mejoras que se pueden realizar a esta investigación. Estas mejoras que se proponen se pueden clasificar en dos secciones:

1. Trabajo futuro con KEA en paralelo.
2. Trabajo futuro para el prototipo Pocket Client.

5.2.1 Trabajo futuro con KEA en paralelo

.

Las modificaciones realizadas al algoritmo de extracción automática de *keyphrases* de KEA pueden ser mejoradas con la finalidad de disminuir la pérdida de precisión exacta del algoritmo en paralelo.

Una de las principales mejoras que se le pueden hacer a esta investigación es modificar el algoritmo que se encarga de hacer el *merge* de los resultados de cada *thread*, ya que actualmente este algoritmo va tomando una a una *keyphrase* de mayor *peso* de cada bloque de resultados (*n bloques/threads*) hasta llenar una lista con las *k keyphrases* a mostrar como resultados. Esta forma de seleccionar la lista es óptima en cuanto a tiempo de ejecución se refiere, pero pierde precisión debido a que no hay una suma de *pesos* de las *keyphrases* iguales que se encuentren en bloques diferentes. Realizar un experimento que sume los *pesos* de las *keyphrases* iguales de los diferentes bloques podría determinar si efectivamente esto es un factor de pérdida de precisión del algoritmo paralelo.

Si se requiere incrementar el rendimiento en términos de tiempo de ejecución en un porcentaje mayor al obtenido con el uso de 2 y 4 *threads*, se tendrá que realizar un análisis más detallado sobre el comportamiento de los *threads* con respecto al acceso del modelo correspondiente para la colección en proceso, ya que las múltiples operaciones de E/S realizadas por los *threads* sobre un mismo recurso (el modelo) provocan comportamientos no esperados (excepciones en tiempo de ejecución) en la ejecución del mismo. Una posible solución a este problema puede ser el uso de algún mecanismo de sincronización (p.e: semáforos).

Según algunos experimentos realizados al inicio de esta investigación, nos dimos cuenta de que KEA tiene mejor precisión para documentos en inglés que para documentos en español, esto es debido a que el algoritmo de *stemmer* que se usa para documentos en inglés (*Lovins* con iteraciones) es más agresivo y preciso que el algoritmo en español (*Porter*). El desarrollar un algoritmo en español para el *stemmer* basado en el algoritmo *Lovins* puede ser un trabajo de investigación futuro.

Otra mejora que podría ayudar mucho para evitar la pérdida de precisión de ambos algoritmos de extracción de *keyphrases* es implementar un mecanismo que configure dinámicamente las opciones del idioma para el *stemmer* y los *stopwords*, ya que actualmente es de forma estática a través de un archivo de configuración. Un forma de automatizar esto puede ser en base a los metadatos del documento, o incluso de la colección, de manera que se pueda identificar cuando usar el idioma español o cuando el inglés para la construcción del modelo y la extracción de *keyphrases*.

5.2.2 Trabajo futuro para el prototipo Pocket Client

.

En este prototipo de cliente móvil hay mucho por hacer como trabajo futuro, ya que las pantallas propuestas para el uso de *keyphrases* son solo una de muchas alternativas que se tienen. Un ejemplo de pantalla de visualización interesante es hacer uso de uso de *thumbnails*, permitir un *zoom dinámico* (al estilo del Acrobat Reader para Pocket), presentar *thumbnails* de las *keyphrases* de cada documento en dónde se marquen las que estén en el criterio de búsqueda, entre otras.

Existen ya algunos trabajos que tienen diferentes formas de presentación de información con similares características, algunos de ellos son, entre otros: Orkut Buyukkokten et al [11, 10], Staffan Björk et al [8], Christopher C. Yang y Fu Lee Wang [62].

Algunas propuestas que se tenían pensadas para este prototipo de *Pocket Client* pero que quedaron fuera de alcance son:

1. Presentar una vista con la estructura en $HTML$ de un documento, en la que se permita navegar entre las secciones y subsecciones del mismo como si fuera una página Web.

2. Presentar visualizaciones de *thumbnail* de documentos, en donde al seleccionar alguno, se muestren las *keyphrases* del mismo en un cuadro de dialogo tipo ventana sobrepuesta al documento. Estas visualizaciones de *thumbnail* de los documentos podrían hacer uso de *templates* que asemejen las estructuras de documentos más comunes (e.j: dos columnas, dos columnas con imagen, etc, etc), lo que facilitaría su programación.

3. Mostrar *thumbnails* de *keyphrases* de los documentos, en donde se marquen en negritas u otro color las que hagan *match* con las introducidas en la cadena de búsqueda.

4. Hacer agrupaciones de los resultados por tipo de documento, representando a estas agrupaciones con alguna imagen que ejemplifique el tipo de documentos asociados, por ejemplo, .pdf, .txt, etc. Creemos que esta agrupación es útil porque usualmente cuando un usuario hace una búsqueda de algún documento personal, lo asocia con un tipo de documento (ppt, pdf, etc), por lo que puede ser un factor importante en la discriminación de resultados.

Además de estas propuestas de trabajo futuro presentadas, creemos que sería útil una evaluación de campo similar a la realizada por Steve Jones and Gordon W. Paynter [26], pero aplicada a el uso de las *keyphrases* en aplicaciones de acceso a bibliotecas personales digitales en ambientes móviles, la cual nos ayude a identificar si efectivamente nuestras alternativas implementadas y propuestas para la visualización de resultados de una búsqueda, son o no de ayuda para el usuario final.

Apéndice A

Arquitectura de integración Data Server - KEA

En este apéndice se muestran diagramas UML con la arquitectura de KEA y del *Data Server* de PDLib, esto con la finalidad de explicar a detalle la integración de KEA con el *Data Server*. Se muestran también las modificaciones realizadas para la paralelización del algoritmo de extracción automática de *keyphrases* de KEA.

A.1 Arquitectura Data Server

En la figura A.1 se muestra un diagrama UML [36] con la arquitectura actual del *Data Server*. Como se puede apreciar, se tiene una implementación de capas basada en herencia, la cual permite agregar fácilmente nuevos servicios al *Data Server*. Esta versión de *Data Server* proporciona, además de los servicios de biblioteca personal digital mencionados en la sección 4.1.4, los siguientes servicios:

- *Sindicación*. Permite integrar bibliotecas digitales públicas de PDLib con programas denominados *agreagadores* mediante el protocolo *RSS* [29].

- *Comunicación XML-RPC*. Este es el servicio que permite al *Data Server* exponer todos sus servicios para que sean accedidos por los diferentes tipos de clientes (p.e. Web Front-end) a través del protocolo de comunicación XML-RPC[61].

- *Soporte OAI*. Este servicios proporciona la interoperabilidad con otras bibliotecas digitales que cumplan con el protocolo OAI-MHP[23]. Como se puede apreciar en la figura 4.4, este servicio está montado sobre el protocolo de transporte XML-RPC[61].

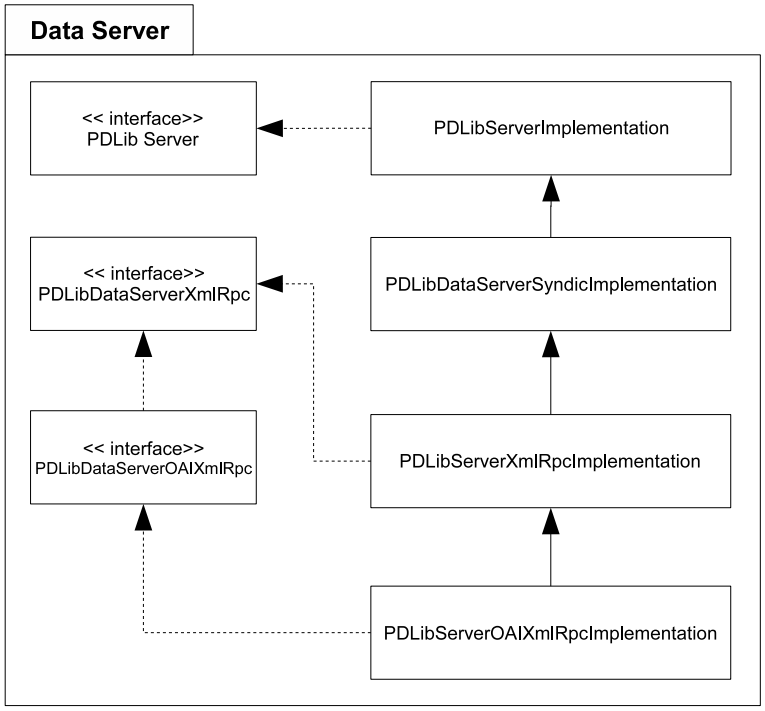

Figura A.1: Arquitectura actual del *Data Server*

A.2 Arquitectura de KEA en Secuencial

En la sección 3.1 se describe el funcionamiento de KEA en su versión original (secuencial), en esta sección describiremos su arquitectura. En la figura A.2 se muestra un diagrama UML[36] con la arquitectura general del algoritmo original de KEA. Como se puede apreciar, el algoritmo de construcción del modelo y el de extracción automática de *keyphrases* son independientes, por lo que las modificaciones que se implementaron para la versión en paralelo de KEA fueron transparentes en la mayoría de los casos, es decir, los algoritmos no paralelizados no sufrieron modificación alguna.

Se puede observar en la figura A.2 que, para integrar los algoritmos *PorterSpanish-Stemmer* y *StopwordsSpanish*, solo se tuvieron que implementar, respectivamente, las interfaces *Stemmer* y *Stopwords* definidas previamente en el sistema KEA.

A.3 Arquitectura de KEA en Paralelo

En la figura A.3 se muestra un diagrama UML[36] con la implementación de KEA en paralelo. Los bloques con líneas en negritas son las nuevas interfaces y clases agregadas. Como se mencionó en la sección 3.1, el algoritmo de creación del modelo no fue paralelizado, por lo que solo se agregaron métodos para la interacción con la base de datos.

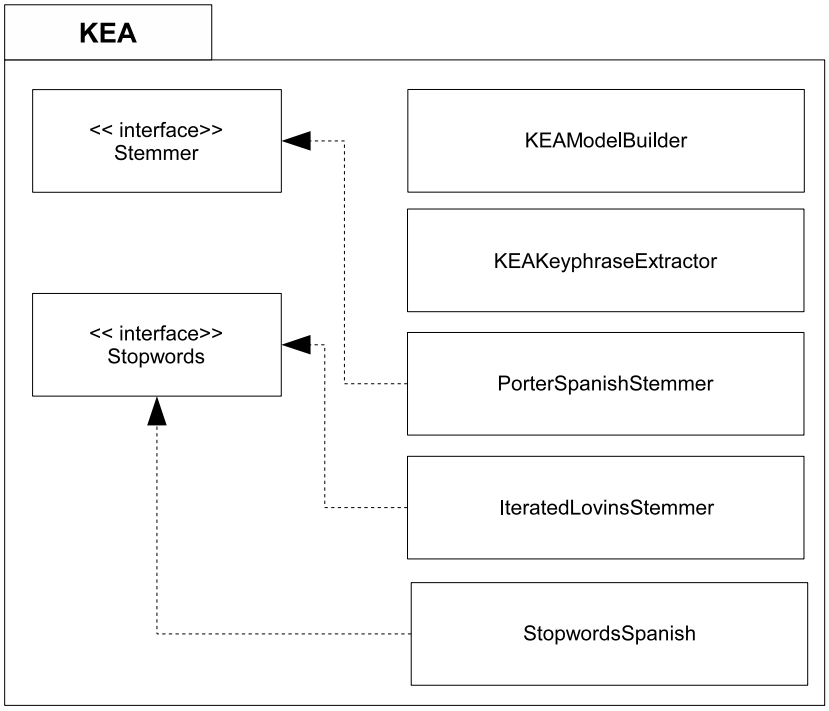

Figura A.2: Arquitectura KEA secuencial

Las modificaciones más relevantes son la creación de las nuevas interfaces *KEAThreadObservable* y *KEADataAccess*, y la creación de las clases *KEAThread*, *KEAFileSystemAccess* y *KEADataBaseAccess*. A continuación se describe la funcionalidad de cada una de ellas:

- *KEAThreadObservable*. Esta interfaz es implementada por el algoritmo de extracción automática de *keyphrases* (*KEAKeyphraseExtractor*) para que el algoritmo *KEAThread* pueda notificarle que ha finalizado su ejecución.

- *KEADataAccess*. La interfaz *KEADataAccess* se utiliza para que los algoritmos *KEAModelBuilder* y *KEAKeyphraseExtractor* puedan implementar de manera transparente el acceso al sistema de archivos o el acceso a la base de datos. Esta interfaz se diseño con la finalidad de seguir teniendo la funcionalidad original de KEA para procesar los documentos desde un sistema de archivos (NTFS, FAT, ext2, etc), además de poder procesar también los documentos almacenados en una Base de Datos. Con esta nueva arquitectura diseñada (ver figura **??**), se logra configurar, mediante parámetros, la manera en la que los algoritmos de KEA acceden a los documentos para procesarlos: Sistema de Archivos ó Base de Datos.

- *KEAThread*. Esta clase se encarga de determinar, para cada frase del bloque del documento asignado si es o no una *keyphrase*. Esto es lo que en la versión

secuencial del algoritmo se realiza en la clase KEAKeyphraseExtractor. Cada uno de los *threads* que se utilizan, al terminar de procesar todas las frases, envían un mensaje a la clase *KEAKeyphraseExtractor* para indicarle que han terminado, la cual a su vez, al recibir este mensaje de los *threads* involucrados, genera una sola lista de *keyphrases* y continúa con el siguiente documento.

- *KEAFileSystemAccess*. Implementa los servicios de *KEADataAccess* que proveen el acceso, de manera transparente, al sistema de archivos. Esta clase es utilizada en los algoritmos *KEAModelBuilder* y *KEAKeyphraseExtractor* para obtener los documentos a procesar.

- *KEADataBaseAccess*. Implementa los servicios de *KEADataAccess* que proveen el acceso, de manera transparente, a los objetos de las bibliotecas digitales personales en la base de datos del *Data Server*. Esta clase es utilizada en los algoritmos *KEAModelBuilder* y *KEAKeyphraseExtractor* para obtener los documentos a procesar.

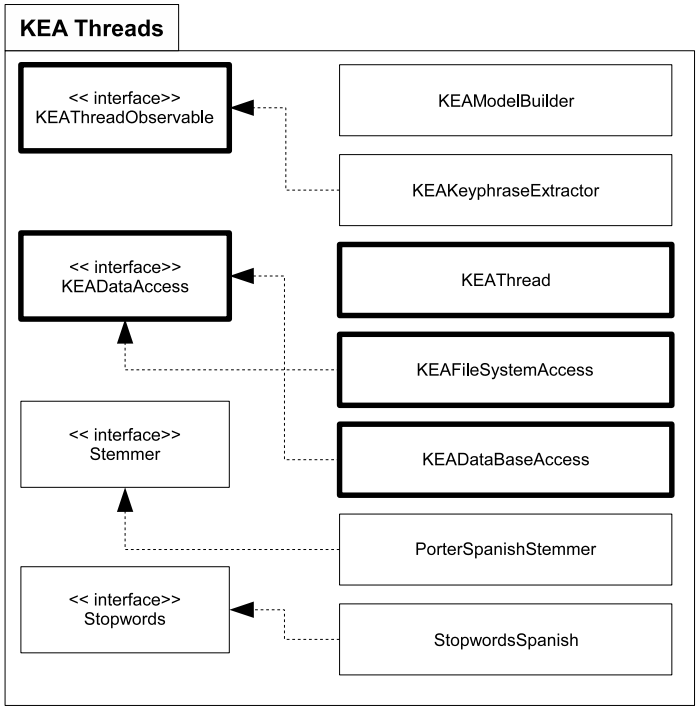

Figura A.3: Arquitectura KEA en paralelo

A.4 Arquitectura de integración Data Server - KEA

Una vez finalizadas las modificaciones en KEA para paralelizar el algoritmo e implementar las adecuaciones requeridas para el soporte a conexión con base de datos[1], se procedió a la integración con el *Data Server*, la cual denominamos *DS-KEA*.

En la figura A.4 se muestra la nueva arquitectura de servicios de la integración *Data Server-KEA*. Como se puede apreciar, la extracción automática de *keyphrases* se implementó como un servicio más que el *Data Server* ofrece a las bibliotecas personales digitales de los clientes PDLib.

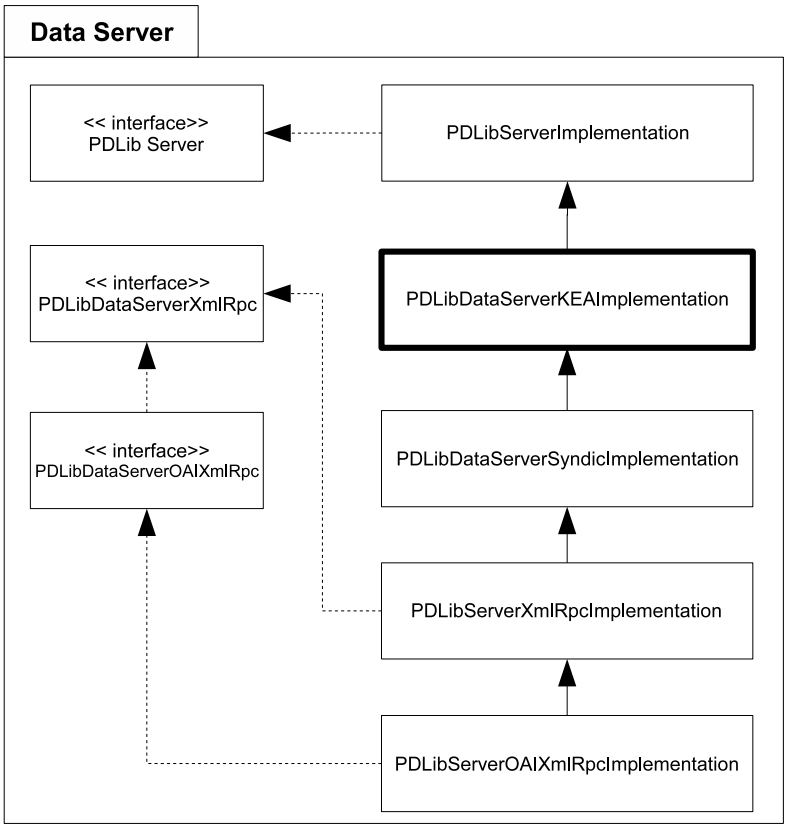

Figura A.4: Arquitectura KEA-*Data Server*

Como se puede apreciar, la diferencia entre esta nueva arquitectura del *Data Server* y la mostrada en la sección 4.2.1 radica principalmente en la nueva clase *PDLibDataServerKEAImplementation* y en el cambio de herencia de la clase *PDLibDataServerSyndicImplementation*. La clase *PDLibDataServerKEAImplementation* utiliza objetos de las clases *KEAKeyphraseExtractor* y *KEAModelBuilder* para delegar, respectivamente, las tareas referentes a la creación de modelos para bibliotecas digi-

[1]La integración se realizó para MySQL 4.1.X

tales personales y la extracción automática de *keyphrases*. Algunos de los servicios que proporciona la clase *PDLibDataServerKEAImplementation* son:

- *Activar/Desactivar KEA*. Este servicio permite definir si se activa KEA para una biblioteca digital personal dada. Cuando KEA se encuentra activo, la generación de modelos y la extracción de *keyphrases* se ejecutan de manera automática para las colecciones y documentos de cada biblioteca digital.

- *Creación de modelos*. Permite crear un modelo para cada colección de las bibliotecas digitales personales. Esto permite tener dominios de información, con la finalidad de explotar la característica de utilización de modelos específicos en la extracción automática de *keyphrases*.

- *Creación de documentos*. Este servicio es una implementación de la clase padre *PDLibDataServerImplementation*. Después de crear el nuevo documento de manera tradicional por la clase padre, se determina si este documento puede ser agregado a la lista de documentos de entrenamiento de KEA, esto se hace tomando en consideración lo siguiente:

  1. El nuevo documento debe pertenecer a una biblioteca/colección para la cual KEA esté activo.
  2. El conjunto de metadatos del nuevo documento debe ser igual al conjunto de metadatos asignado para KEA.
  3. El nuevo documento no debe tener el metadato *keywords* vacío, ya que este metadato contiene las *keyphrases* definidas por el autor y son usadas en el proceso de construcción del modelo.

  Si el nuevo documento cumple con estos tres criterios, entonces es agregado a la lista de documentos de entrenamiento de KEA para ese par *biblioteca-colección*, de lo contrario, solo se agrega normalmente al *data server*.

- *Extracción automática de keyphrases*. Permite extraer automáticamente las *keyphrases* de un documento. Este servicio también se puede utilizar para extraer las *keyphrases* de todos los documentos de una colección o para todas las colecciones de una biblioteca digital personal, por lo que se puede ejecutar por el administrador del servidor PDLib para correr periódicamente (e.j. una vez por semana, en la noche) como una tarea programada.

Con esta integración *DS-KEA*, se brindan nuevas alternativas de presentación de información relevante en los clientes PDLib.

Bibliografía

[1] Muñoz A. Compound key word generation from document databases using a hierarchical clustering art model. *Intelligent Data Analysis*, 1(1), 1996.

[2] Nabil R. Adam, Richard Holowczak, Milton Halem, Nand Lal, and Yelena Yesha. Digital library task force. In *IEEE Computer*, pages 89–91, 1996.

[3] Wi-Fi Alliance. Wireless fidelity, 2005. http://www.wi-fi.org/.

[4] Francisco Alvarez-Cavazos, David A. Garza-Salazar, and Juan C. Lavariega-Jarquin. Pdlib: personal digital libraries with universal access. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 365–365, New York, NY, USA, 2005. ACM Press.

[5] Infrared Data Association. The official irda web site, 2005. http://www.irda.org/.

[6] Krulwich B. and Burkey C. Learning user information interests through the extraction of semantically significant phrases. In M. Hearts and H Hirish, editors, *AAAI 1996 Spring Symposium on Machine Learning in Information Access*, California, 1996. AAAI Press.

[7] Bharat Bhargava, Melliyal Annamalai, and Evaggelia Pitoura. Digital library services in mobile computing. *SIGMOD Rec.*, 24(4):34–39, 1995.

[8] Staffan Björk, Lars Erik Holmquist, Johan Redström, Ivan Bretan, Rolf Danielsson, Jussi Karlgren, and Kristofer Franzén. West: a web browser for small terminals. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 187–196. ACM Press, 1999.

[9] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Focused web searching with pdas. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):213–230, 2000.

[10] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Accordion summarization for end-game browsing on pdas and cellular phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 213–220. ACM Press, 2001.

[11] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. Seeing the whole in parts: text summarization for web browsing on handheld devices. In *Proceedings of the tenth international conference on World Wide Web*, pages 652–662. ACM Press, 2001.

[12] Orkut Buyukkokten, Oliver Kaljuvee, Hector Garcia-Molina, Andreas Paepcke, and Terry Winograd. Efficient web browsing on handheld devices using page and form summarization. *ACM Transactions on Information Systems (TOIS)*, pages 82 – 115, 2002.

[13] Nohema Castellanos and J. Alfredo Sánchez. Pops: mobile access to digital library resources. In *Proceedings of the third ACM/IEEE-CS joint conference on Digital libraries*, pages 184–185. IEEE Computer Society, 2003.

[14] DTIE. División de tecnologías de información y electrónica, 2004. http://dtie.mty.itesm.mx/.

[15] The Association for Computing Machinery. The acm portal, 2005. http://portal.acm.org/dl.cfm.

[16] Dan Fox and Jon Box. *Building Solutions with the Microsoft .NET Compact Framework: Architecture and Best Practices for Mobile Development.* Addison Wesley, 2003.

[17] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. In *IJCAI*, pages 668–673, 1999.

[18] Orlando Julián Piña González. Extracción automática de metadatos de un documento digital. Tesis de maestría, ITESM, Campus Monterrey, 2000.

[19] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing.* Addison Wesley, second edition, January 2003.

[20] PDLib Group. Pdlib: The personal digital library project, 2004. http://copernico.mty.itesm.mx/pdlib/.

[21] The Bluetooth Special Interest Group. The official bluetooth web site, 2005. http://www.bluetooth.com/.

[22] Hilda Hardy, Nobuyuki Shimizu, Tomek Strzalkowski, Liu Ting, Xinyang Zhang, and G. Bowden Wise. Cross-document summarization by concept classification. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 121–128. ACM Press, 2002.

[23] OAI Initiative. Open archives initiative, 2005. http://www.openarchives.org/.

[24] Steve Jones, Matt Jones, and Shaleen Deo. Using keyphrases as search result surrogates on small screen devices. *Personal and Ubiquitous Computing*, pages 55 – 68, February 2004.

[25] Steve Jones and Gordon Paynter. Topic-based browsing within a digital library using keyphrases. In *Proceedings Of The Fourth ACM Conference On Digital Libraries*, pages 11–14, Berkeley, CA, August 1999.

[26] Steve Jones and Gordon W. Paynter. Human evaluation of kea, an automatic keyphrasing system. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 148–156, New York, NY, USA, 2001. ACM Press.

[27] Pradip Lamsal. J2me architecture and related embedded technologies, 2004.

[28] Juan Carlos Lavariega, Martha Sordia Salinas, and David A. Garza Salazar. Information retrieval and administration of distributed documents in internet. In Witold Abra, editor, *Knowledge Based Information Retrieval and Filtering from the Web*, 2003.

[29] Reuven M. Lerner. At the forge: syndication with rss. *Linux J.*, 2004(126):8, 2004.

[30] Research In Motion Limited. Blackberry web site, 2005. http://www.blackberry.com/.

[31] Symbian Ltd. Symbian os - the mobile operating system, 2005. http://www.symbian.com/.

[32] George Buchanan Matt. Exploring small screen digital library access with the greenstone digital library, 2003.

[33] Microsoft. User interface process application block for .net, 2003.

[34] Microsoft. Windows mobile, 2005. http://www.microsoft.com/windowsmobile/.

[35] The University of Waikato. The new zealand digital library, 2004. http://www.nzdl.org/.

[36] OMG. Object management group, 2005. http://www.omg.org/.

[37] Palm. Palm web site, 2005. http://www.palm.com.

[38] Nimesh Panchal. Implementing mvc design pattern in .net. http://www.c-sharpcorner.com/Code/2003/Feb/MVCDesign.asp, 2003.

[39] Dynal Patel and Gary Marsden. Customizing digital libraries for small screen devices. In *SAICSIT '04: Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 234–238, , Republic of South Africa, 2004. South African Institute for Computer Scientists and Information Technologists.

[40] Dr Martin Porter. a small string processing language designed for creating stemming algorithms for use in information retrieval, 2004. snowball.tartarus.org/.

[41] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1980.

[42] Mark Sanderson. Review of .ªdvances in automatic text summarization"by inderjeet mani and mark t. maybury. the mit press 1999. *Comput. Linguist.*, 26(2):280–281, 2000.

[43] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Symposium on Principles of Distributed Computing*, pages 1–7, 1996.

[44] WilliamÑoah Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, COLUMBIA UNIVERSITY, 1995.

[45] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, September 2000.

[46] Roberto García Sánchez. Técnicas de adaptación a la conexión para clientes móviles que accesan servicios de biblioteca difital. Tesis de maestría, ITESM, Campus Monterrey, Diciembe 2004.

[47] Min Song, Il-Yeol Song, and Xiaohua Hu. Kpspotter: a flexible information gain-based keyphrase extraction system. In *Proceedings of the fifth ACM international workshop on Web information and data management*, pages 50–53. ACM Press, 2003.

[48] Open Source. The gpe palmtop environment, 2005. http://gpe.handhelds.org/.

[49] Open Source. Opie - open palmtop integrated environment applications and libraries for mobile devices, 2005. http://opie.handhelds.org/cgi-bin/moin.cgi/.

[50] Java Sun. Java 2 platform, micro edition. http://java.sun.com/j2me/docs, 2000.

[51] Trolltech. Qtopia: comprehensive application platform for embedded linux based pdas, 2005. http://www.trolltech.com/products/qtopia/.

[52] P. Turney. Learning to extract keyphrases from text, 1999.

[53] Peter D. Turney. Extraction of keyphrases from text: Evaluation of four algorithms. Technical Report ERB-1051, National Research Council, Institution for Information Technology, 1997.

[54] Peter D. Turney. Learning to extract keyphrases from text: Evaluation of four algorithms. Technical Report ERB-1057, National Research Council, Institution for Information Technology, 1999.

[55] Peter D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

[56] Tufts University. The perseus digital library, 1998. http://www.perseus.tufts.edu/.

[57] Wikipedia. Digital library, 2005. http://en.wikipedia.org/wiki/Digital_library.

[58] Ian Witten. The new zealand digital library project, July 2003. Department of Computer Science, University of Waikato, New Zealand.

[59] Ian H. Witten, Rodger J. McNab, Stefan J. Boddie, and David Bainbridge. Greenstone: A comprehensive open-source digital library software system. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[60] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. Kea: Practical automatic keyphrase extraction. In *ACM DL*, pages 254–255, 1999.

[61] XML-RPC. Simple cross-platform distributed computing, based on the standards of the internet, 2005. http://www.xmlrpc.com/.

[62] Christopher C. Yang and Fu Lee Wang. Fractal summarization for mobile devices to access large documents on the web. In *Proceedings of the twelfth international conference on World Wide Web*, pages 215–224. ACM Press, 2003.