Informe Técnico - Technical Report
DPTOIA-IT-2007-003
julio, 2007

Librerías Lucene y dotLucene para Recuperación de Información. Estudio y desarrollo de casos prácticos

Vadim Paz Madrid Gorelov
Ángel F. Zazo
Carlos G. Figuerola
José Luis Alonso Berrocal

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por

Dr. D. Roberto Therón Sánchez
Departamento de Informática y Automática
Universidad de Salamanca

Dr. D. Luis Antonio Miguel Quintales
Departamento de Informática y Automática
Universidad de Salamanca

Aprobado en el Consejo de Departamento de 25 de julio de 2007.

Información de los autores:

D. Vadim Paz Madrid Gorelov
Estudiante de Doctorado
Departamento de Informática y Automática
Facultad de Ciencias. Universidad de Salamanca
Plaza de los Caídos, s/n. 37008 - Salamanca
pazmadrid@usal.es

Dr. D. Ángel F. Zazo
Dr. D. Carlos G. Figuerola
Dr. D. José Luis Alonso Berrocal
Grupo de investigación en Recuperación de Información Avanzada (REINA)
Departamento de Informática y Automática
Facultad de Traducción y Documentación. Universidad de Salamanca
C/ Francisco Vitoria, 6-16. 37008 - Salamanca
{afzazo,figue,berrocal}@usal.es
http://reina.usal.es

Resumen

En este informe técnico se describe la utilización de dos librerías[1] para Recuperación de Información. Después de una introducción a esta disciplina, se realiza un tutorial básico de utilización de la librería Lucene, bajo el lenguaje de programación Java, explicando en qué consiste, qué se puede hacer con ella, y poniendo ejemplo prácticos de su utilización. Se estudia el modelo de clases de Lucene, y se exploran los principales objetos para la indexación y búsqueda de información. Además del estudio e implementación de la librería Lucene, se describe la utilización de dotLucene, un puerto adicional de Lucene en .Net, con el que probar la versatilidad de Lucene en otras plataformas. Para ello se han elaborado y documentado dos ejemplos de búsqueda de información. En el primero se lleva a cabo una búsqueda de información en documentos almacenados en un árbol de directorios. Se pueden realizar búsquedas de información sobre cualquier fichero convertible a texto plano. El segundo va más allá y realiza la indexación, delimitación y búsqueda de información en documentos XML, permitiendo la búsqueda por campos concretos en este tipo de documentos.

Abstract

This technical report describes two libraries for Information Retrieval: Lucene and dotLucene. Lucene is a library developed for the Java program language. DotLucene is a port for Lucene in .Net. Our purpose is to make a basic tutorial to use both libraries. So, first we describe Lucene, its class model and the most important objects for the process of information indexing and searching. Next, we study dotLucene, in this case, using two examples for the description of this library. The former finds files in a folder containing the text of the search. The later can to index fields of XML files using different criteria. In this example, some searches can be done using one or more fields of the XML files.

[1] Hemos preferido utilizar la palabra "librería" en vez de "biblioteca" ya que es de uso habitual en ambientes informáticos. Además, el diccionario de la RAE en su vigésima segunda edición define la palabra "librería" como "biblioteca", con el sentido de conjunto de libros de un local (segunda acepción). Entendemos que un conjunto de libros relacionados entre sí reciben el nombre de librería, de ahí que por extensión también nosotros utilicemos la misma palabra para referirnos a un conjunto de funciones relacionadas entre sí.

Índice

Índice de figuras III

Indice de tablas III

1. Introducción 1

2. Recuperación de Información 2
2.1. Indización . . . 3

3. Lucene 5
3.1. Instalación de Lucene . . . 6
3.2. Integración de Lucene en las aplicaciones . . . 6
3.2.1. Clases para desarrollar el procedimiento de indexación . . . 6
3.2.2. Clases principales para desarrollar el procedimiento de búsqueda . . . 8
3.3. Un ejemplo de Lucene . . . 9
3.3.1. Creando un índice . . . 9
3.3.2. Indexando documentos de texto . . . 9
3.3.3. Realizando la búsqueda . . . 10

4. dotLucene 11
4.1. Ventajas que ofrece dotLucene . . . 11
4.2. Comenzando un nuevo proyecto con dotLucene . . . 11

5. Indexar y buscar en documentos 12
5.1. Definición de directorios o carpetas . . . 12
5.1.1. Carpeta donde se crea el índice . . . 13
5.1.2. Carpeta que contiene los documentos a indexar . . . 13
5.2. Creación de un índice . . . 13
5.3. Añadir documentos al índice . . . 15
5.4. Guardar el índice . . . 16
5.5. Realizando la búsqueda . . . 17

6. Indexar y buscar en documentos XML 19
6.1. Iniciando la aplicación . . . 19
6.2. Administración de parámetros de inicialización . . . 20
6.2.1. Opciones del fichero de configuración . . . 20
6.2.2. Fuente de datos . . . 21
6.2.3. Campos . . . 22
6.2.4. Obtención de datos . . . 22
6.2.5. Afinamiento en la creación del índice . . . 23
6.2.6. Analizador léxico . . . 24
6.3. Búsqueda de información . . . 25
6.3.1. Opciones de búsqueda . . . 25
6.3.2. Mostrando resultados . . . 25

7. Conclusiones 27

Referencias 28

Índice de figuras

1. Paradigma de la Recuperación de Información. 3
2. Arquitectura de Lucene 7
3. Definición de carpetas en nuestra aplicación. 12
4. Creando el índice en nuestra aplicación. 14
5. Proceso finalizado de creación del índice. 16
6. Información del directorio origen de documentos. 16
7. Resultados de la búsqueda. 18
8. Ubicación del archivo de configuración 19
9. Indexando y buscando en documentos XML. 20
10. Pantalla del valores del archivo de configuración. 21
11. Edición de Campos. 23
12. Seleccionando el tipo de campo de Lucene. 23
13. Obteniendo muestreo de datos. 24
14. Resultado de la búsqueda "+españa". 26
15. Mostrando un documento. 26

Indice de tablas

1. Tipos de campos en Lucene 8
2. Tablas de caracteres disponibles en .NET framework. 22
3. Analizadores de Lucene. 24
4. Opciones de búsqueda para todos los campos. 25
5. Opciones adicionales de búsqueda para el campo *Default*. 26

1. Introducción

Hoy día la información es pieza fundamental en todos los procesos de nuestra sociedad. El desarrollo de las tecnologías de la información y las comunicaciones ha sido la piedra angular en la creación de servicios y sistemas de información, requeridos cada vez con mayor celeridad. Este desarrollo está cambiando de manera decisiva nuestra forma de ver el mundo. El mejor ejemplo es ver como algunas empresas que sólo gestionan información, como Google o Yahoo!, son más importantes que grandes empresas convencionales. En los últimos años hemos asistido al cambio en los soportes y formatos de la información y en los mecanismos de tratamiento y difusión de la misma. El mayor exponente de este hecho es Internet, que permite que millones de usuarios accedan a una fuente de recursos de información y conocimiento compartido a escala mundial, siendo el instrumento que está produciendo la mayor y más sorprendente revolución en el mundo de la información.

Este gran volumen de información provoca también grandes problemas. La capacidad de digerir la información por parte de los usuarios no ha ido pareja a su producción, y es importante descartar la que no interesa, o lo que es lo mismo, seleccionar la información pertinente para una necesidad informativa dada. Evidentemente, la necesidad informativa no está limitada a Internet, sino a cualquier tipo de sistema que soporte o almacene información. Los ordenadores pueden almacenar cada día más y más información, en varios formatos (texto, imágenes, vídeo, sonido, etc.), de usuarios locales o remotos, y con diferentes características y objetivos, y la necesidad de localizar y organizar rápidamente esa información se está convirtiendo en pieza clave de tales sistemas. Debido a esta necesidad las opciones de búsqueda están implementadas en prácticamente todas las aplicaciones actuales.

Los sistemas de recuperación de información son la pieza clave para resolver el problema de la búsqueda rápida y eficiente de información. De hecho, nos encontramos en un momento crucial para estos sistemas. Los más avanzados tratan de agregar información semántica a las búsquedas, con una amplia actividad investigadora en torno a interfaces de usuario, nuevos métodos de búsqueda y nuevas formas de indización, para adelantarse a las necesidades de los usuarios. Hoy en día se debe tener la capacidad de encontrar información y datos de diversos tipos y formatos de manera flexible, libre de forma, que permitan realizar búsquedas requiriendo el mínimo esfuerzo posible.

En el presente trabajo realizaremos una introducción a la apasionante disciplina que es la Recuperación de Información (RI), y utilizaremos una de las más flexibles y poderosas librerías del mercado para crear aplicaciones de RI, denominada **Lucene**, que ha tenido gran éxito y escalabilidad en muchas aplicaciones. Esta librería fue creada inicialmente por Doug Cutting, y puesta como software abierto (*Open Source*) en marzo de 2000 a través de *SourceForge* [2]. En septiembre de 2001 se unió a la *Apache Software Foundation*, en concreto a la familia *Jakarta* de productos escritos en Java.

En este trabajo realizaremos un desarrollo práctico de Lucene, como ejemplo de su portabilidad y aplicación en otros entornos. Utilizaremos **dotLucene**[2], un puerto de desarrollo de Lucene en Microsoft Visual Studio .NET en el lenguaje de C# [15], basado en el API (*Application Programming Interface*, interfaz de programación de aplicaciones) de Lucene para Java, y cuyo desarrollo debemos a Dan Letecky. Se considera el código abierto más rápido y poderoso para realizar tareas de búsqueda e indexación bajo el mundo .NET [9, 19].

Lamentablemente, justo en la fecha de escribir este documento, nos hemos enterado de que Dan ha dejado de mantener dotLucene. No obstante, no hay mucho problema, pues desde abril de 2006 la versión de dotLucene que existía hasta ese momento se movió a la Apache Software Foundation como **Lucene.Net**, y Dan siguió trabajando para proporcionar recursos a DotLucene/Lucene.Net. Hoy día, Lucene.Net es el puerto de Apache Jakarta Lucene para .NET (C#) manteniéndolo actualmente George Aroush [3].

[2] `http://www.dotlucene.net` .

2. Recuperación de Información

La Recuperación de Información es una disciplina que desde hace unos pocos años está experimentando un renovado interés, habida cuenta del aumento de la disponibilidad de documentos en formato electrónico y de la necesidad consiguiente de obtener en cada momento aquellos que respondan a una necesidad informativa dada.

La RI tiene sus orígenes en las bibliotecas y centros de documentación, en los que se realizan búsquedas bibliográficas de libros y artículos de revista para satisfacer las necesidades de información de sus usuarios, proporcionándoles la información pertinente en el menor tiempo y coste posible. Los documentos en esos centros se representan utilizando un conjunto de palabras clave, que se incorporan a las fichas (manuales o electrónicas) en las que se rellenan una serie de campos con esa información. En esos campos se incluyen datos del documento en cuestión, como su título, autor, fecha de publicación, etc. También se incluyen habitualmente uno o varios términos que dan una indicación de su contenido, y que normalmente quedan reflejados en el campo *materia*. Tales términos reciben el nombre de *términos índice*. Los usuarios que consultan el sistema de recuperación para buscar información deben traducir su necesidad informativa en una consulta adecuada al sistema de recuperación. Esto supone utilizar un conjunto de términos que expresen semánticamente su necesidad. En sistemas tradicionales también es habitual utilizar operadores Booleanos para conectar varios criterios de búsqueda por campos diferentes.

Antes de continuar es necesario definir el alcance y los objetivos de la RI. Debemos dejar claro, en primer lugar, que en este trabajo trataremos con recuperación de información *automatizada*, es decir, aquella en la que los procedimientos utilizados en el proceso son fundamentalmente automáticos, sin ser ello óbice para que puedan utilizarse ocasionalmente procedimientos manuales en alguna de las etapas intermedias de la recuperación. Asimismo, entendemos que el concepto de recuperación de información también incluye todo el tratamiento para la preparación del documento y su almacenamiento. Entendemos por "preparación del documento" la asignación de una serie de características que permiten su posterior recuperación.

Por otro lado, la diferencia entre "recuperación de información" y "recuperación de datos" se utiliza normalmente para distinguir entre sistemas de RI y sistemas de gestión de bases de datos (SGBD). A este aspecto suele prestarse una especial atención en casi todas las monografías sobre el tema [21, 13, 4], aunque nosotros creemos que la distinción entre ambas es a menudo más teórica que práctica, pues algunos sistemas de RI, sobre todo si tratan con un volumen de información pequeño, se implementan sobre SGBD.

Para nosotros, la RI es el conjunto de acciones, métodos y procedimientos para la representación, almacenamiento, organización y recuperación de información. El objetivo fundamental de la RI es, dada una necesidad de información y un conjunto de documentos, obtener los documentos relevantes para esa necesidad, ordenarlos en función del grado de relevancia, y presentarlos al usuario. Podemos definir relevancia como la medida de cómo un documento se ajusta a una consulta.

La Figura 1 presenta de forma esquemática un sistema de RI, que acepta documentos y consultas, y que obtiene una salida, que es el conjunto de documentos que satisfacen la consulta. Una consulta es la expresión formal de la necesidad informativa del usuario. El problema principal en este sistema es obtener representaciones homogéneas, tanto de los documentos como de la necesidad informativa del usuario, y procesar convenientemente estas representaciones para obtener la salida. Se aprecian dos actividades: representación y búsqueda. Tampoco debemos perder de vista la evaluación de la salida, para determinar si ésta coincide con las necesidades informativas del usuario.

En el proceso de RI podemos distinguir las siguientes etapas:

1. Obtener representación de los documentos. Generalmente los documentos se representan utilizando un conjunto más o menos grande de términos índice. La elección de dichos términos es el proceso más complicado.

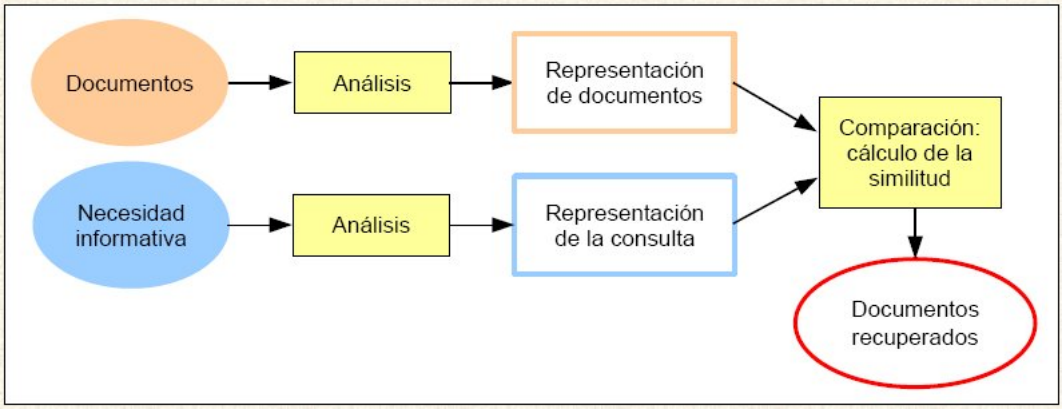

Figura 1: Paradigma de la Recuperación de Información.

2. Identificar la necesidad informativa del usuario. Se trata de obtener la representación de esa necesidad, y plasmarla formalmente en una consulta acorde con el sistema de recuperación.

3. Búsqueda de documentos que satisfagan la consulta. Consiste en comparar las representaciones de documentos y la representación de la necesidad informativa para seleccionar los documentos pertinentes.

4. Presentación de los resultados al usuario. Puede ser desde una breve identificación del documento hasta el texto completo.

5. Evaluación de los resultados. Para determinar si son acordes con la consulta.

El esquema de pasos aquí presentado se ha utilizado desde las primeras investigaciones en RI. Al finalizar la II Guerra Mundial, la gran producción de literatura científica y técnica hizo que los métodos tradicionales de almacenamiento y recuperación de información disminuyeran su efectividad. Coincidió en el tiempo el desarrollo de los primeros sistemas de computación electrónicos, y los dispositivos informáticos fueron utilizados entonces para el control de gran cantidad de información. Fueron importantes los trabajos sobre la teoría matemática de la información de autores como Shannon y Weaver, y estudios relacionados con la frecuencia de aparición de palabras y/o citas en los documentos, como los de Zipf, Bradford y Lotka [16]. Autores como Taube, Perry, Mooers y Luhn aprovecharon sus resultados para aplicarlos a una incipiente disciplina, la recuperación de información [18]. Sparck Jones [20] sugiere que la *International Conference on Scientic Information*, celebrada en Washington en 1958, marca el comienzo de la RI tal como la conocemos hoy. La mayoría de los principios de la RI que fueron establecidos en aquella época permanecen todavía vigentes.

2.1. Indización

Antes de continuar es conveniente indicar que es habitual utilizar el término *indización* cuando se habla de recuperación de información en entornos documentales. Sin embargo en entornos informáticos suele utilizarse más el término *indexación*. En realidad son sinónimos. Utilizaremos uno u otro indistintamente.

Hoy día con el incremento del número de documentos en formato electrónico se hace necesario contar con herramientas informáticas adecuadas para su recuperación. Actualmente los ordenadores hacen posible representar un documento utilizando su contenido completo, es lo que se denomina representación a texto completo. Efectivamente, es la forma más completa de representación, pero implica un coste computacional muy alto para colecciones grandes de documentos, por lo que se busca encontrar el equilibrio entre el número de términos utilizados para representar su contenido y el coste computacional asociado para obtener la salida en un tiempo razonable.

La mayoría de sistemas de RI utiliza términos índice para representar el contenido de los documentos. En sentido estricto un *término índice* es una palabra (o grupo de palabras) que posee significado propio [4] y que se utiliza para representar un concepto [1]. Los sistemas de recuperación basados en términos índice se apoyan en la idea fundamental de que tanto el contenido de documentos como la necesidad informativa del usuario pueden representarse utilizando el conjunto de términos índice. Esto es lo que se conoce como indización (o indexación).

Ahora bien, no todas las palabras de un documento poseen la misma utilidad para describir su contenido, y por tanto, de poder usarse como términos índice. Aunque existen palabras más importantes que otras en función de la categoría gramatical, no es tarea fácil decidir la importancia de cada una de ellas. Adverbios, artículos, preposiciones y conjunciones suelen ser palabras casi vacías de contenido semántico, y por tanto, poco útiles desde el punto de vista de la RI. No obstante, esto no está claro, pues muchas de palabras de este tipo incorpora un nivel semántico adicional al nombre o verbo que acompañan, y por tanto debieran tenerse en cuenta. Asimismo, también resultan poco útiles para las tareas de recuperación aquellas palabras que se repiten con mucha frecuencia en la colección de documentos, pues llevan a términos índice poco discriminatorios en relación con una consulta dada.

En conclusión, los sistemas de recuperación no sólo persiguen encontrar aquellos términos que mejor representen a los documentos, sino además aquellos que permitan diferenciar unos respecto de otros. Los diferentes modelos de recuperación se diferencian precisamente en el método elegido para representar los documentos y las consultas, y para realizar las búsquedas de información.

Así pues, el primer paso para la indización es analizar el texto del documento para determinar qué términos pueden utilizarse como términos índice. Es importante destacar que se parte de un documento a texto completo y se consigue al final del proceso un conjunto más o menos numeroso de términos índice que lo representan. Este proceso suele llevar la sucesión de los siguientes pasos [22]:

1. **Análisis léxico del texto**, con el objetivo de convertir la cadena de entrada en un conjunto de palabras, y determinar el tratamiento que se realizará sobre números, signos de puntuación, tratamiento de mayúsculas y/o minúsculas, nombres propios, etc. Este proceso que parece fácil no lo es tanto. En palabras de Greenstette [12, pág. 87]: *el proceso de separación de palabras se compone de un montón de preguntas espinosas, de las cuales solamente unas pocas tienen respuesta perfecta.* Solo para hacernos una idea, los separadores de palabras en español suelen ser el espacio y un conjunto más o menos reducido de signos de puntuación[3]. Pero, qué pasa en otros idiomas, como el alemán, que permite crear compuestos muy largos, o el chino, que prácticamente no contiene separadores.

2. **Eliminación de palabras vacías, muy frecuentes y muy poco frecuentes**, con el objetivo de reducir el número de términos con valores muy poco significativos para la recuperación. Es también una buena forma de reducir el tamaño de los ficheros de datos del sistema, y así aumentar la rapidez de respuesta del sistema.

3. **Aplicación de lematización** sobre los términos resultantes para eliminar variaciones morfosintácticas y obtener lemas. La lematización consiste en elegir convencionalmente una forma de una palabra para remitir a ella todas las de su misma familia por razones de economía[4].

4. **Selección de términos** o grupos de términos que serán considerados términos índice. Normalmente se realiza sobre la naturaleza morfo-sintáctica del término, pues, como ya se ha mencionado, los términos que actúan gramaticalmente como nombres suelen poseer un mayor contenido semántico que verbos, adjetivos o adverbios.

[3] También puede establecerse la separación como la frontera entre un carácter que sí forme parte de una palabra y otro que no.

[4] Tomado de la vigésimo segunda edición del diccionario de la RAE.

5. **Utilización de tesauros**. Muchas veces los sistemas de RI se muestran incapaces de resolver dos problemas importantes: la sinonimia y la polisemia[5], que evidentemente también afectan a la expresión de la necesidad informativa del usuario. Un tesauro es un vocabulario controlado que puede ayudar tanto en el proceso de indización como en el de búsqueda de información. También se pueden utilizar para expandir los términos de la consulta con otros términos relacionados, evitando en lo posible el problema de la sinonimía y polisemia.

La investigación en RI busca diseñar sistemas que acepten consultas en lenguaje natural y proporcionen documentos adecuados a tales consultas, ordenados según algún criterio del sistema, de acuerdo a las características de los documentos y a las necesidades informativas expresadas por el usuario en su consulta. En los últimos veinticinco años la RI se ha expandido mucho más allá de sus objetivos iniciales de indización de texto y búsqueda de documentos pertinentes en una colección. Hoy en día las investigaciones en RI incluyen modelado de información, clasificación y categorización de documentos, arquitectura de sistemas, interfaces de usuario, visualización de información, filtrado de información, búsqueda robusta, búsqueda de respuestas, recuperación interactiva, recuperación multilingüe, recuperación hipermedia, recuperación de documentos sonoros, de imágenes y de vídeo, extracción de información, resumen automático, etc. Como nota importante, resaltar que desde hace varios años existen conferencias internacionales como TREC[6], CLEF[7] o NTCIR[8], donde se tratan temas muy actuales de investigación en RI.

3. Lucene

Lucene es una librería escalable de alto rendimiento para la recuperación de información, que permite agregar capacidades de indexación y búsqueda a las aplicaciones. Puede indexar y buscar cualquier dato que pueda ser convertido a formato textual. Se trata de una API de indexación y búsqueda con simplicidad en su utilización que requiere básicamente que el usuario aprenda el uso de sus clases.

Lucene está implementado originalmente en el lenguaje de programación Java, y es miembro de la familia Apache Jakarta, siendo un proyecto de código abierto gratuito, muy maduro y fácil de utilizar. Durante muchos años ha sido la librería gratuita más utilizada bajo Java para recuperación de información. Lucene, siendo una librería, no es una aplicación finalizada lista para ser utilizada, como lo pudiera ser un programa buscador de archivos, un web crawler [7] o un buscador de sitios

[5]Estos dos figuras léxicas son las que más influyen en la recuperación, pero podemos identificar otras, como la homonimia [14], la antonimia, la hiperonimia o la hiponimia, por no hablar de los problemas que la anáfora provoca.

[6]Las TREC (*Text REtrieval Conferences*) comenzaron su andadura en 1992 patrocinadas por el *National Institute of Standards and Technology* (NIST) y la *Defense Advanced Research Projects Agency* (DARPA) del Gobierno de los EE.UU. con el propósito de establecer un foro para desarrollar la investigación en el campo de la recuperación de información, proporcionando a los investigadores la infraestructura necesaria para la evaluación de sus metodologías de recuperación sobre grandes colecciones de documentos, con el objetivo último de conseguir mayor rapidez en la transferencia de resultados. Muchos de los motores de búsqueda actuales tuvieron su origen en estas conferencias. Puede encontrar más información en `http://trec.nist.gov` .

[7]Las conferencias CLEF (*Cross-Language Evaluation Forum*) se iniciaron en Lisboa en 2000, enmarcadas en las *European Conferences on Digital Libraries* (ECDL), que son actividades que se desarrollan anualmente en Europa por el grupo de trabajo *DELOS Network of Excellence* del programa *Information Society Technologies* (IST) de la Comisión Europea, inicialmente fundado por el programa ESPRIT en 1996, para promover el desarrollo de tecnologías para la biblioteca digital. Uno de los fines primordiales de DELOS es proporcionar acceso a la información rompiendo las barreras idiomáticas o culturales, de ahí la necesidad de las actividades en recuperación de información multilingüe. CLEF se desarrolla en colaboración con las conferencias TREC para la evaluación de diferentes sistemas de recuperación de información multilingüe. Puede encontrar más información en `http://www.clef-campaign.org` .

[8]Las conferencias NTCIR (*NII-NACSIS Test Collection for IR Systems*) se desarrollan en Japón desde 1998, y se trata de talleres para la evaluación de sistemas de RI y técnicas de procesado automático de texto, tales como resumen automático, extracción de texto, etc., teniendo el japonés, y en menor medida el chino, como lengua principal. Las conferencias están patrocinadas por el *Japan Society for Promotion of Science* (JSPS), por el *National Center for Science Information Systems* (NACSIS) y por el *Research Center for Information Resources at National Institute of Informatics* (RCIR/NII). Más información en `http://research.nii.ac.jp/ntcir/` .

Web. Lucene es una librería de software, utilizada como herramienta para incorporar capacidades de búsqueda a las aplicaciones.

Remontándonos un poco en la historia de Lucene, esta librería fue escrita originalmente por Doug Cutting, creador de Nutch[9], primer buscador online de código abierto. Desde marzo de 2002 se dispuso para ser descargada desde SourceForge. En Septiembre 2001 paso a unirse a la familia Jakarta de la Apache Software Foundation, como código libre de alta calidad. Son muchas las empresas y sistemas que utilizan Lucene, puede encontrar una lista bastante amplia en `http://wiki.apache.org/lucene-java/PoweredBy`. Asimismo, el éxito del software de código abierto puede medirse por la cantidad de veces que ha sido traducido a otros lenguajes de programación. En el caso de Lucene ha sido traducido a Perl, Phyton, C++, C#.Net y Ruby.

3.1. Instalación de Lucene

Se puede obtener la última versión de Lucene en la sección de descargas de `http://jakarta.apache.org`. Actualmente la última versión es la 2.1.0, de febrero de 2007, aunque para este trabajo hemos utilizado la versión anterior, la 2.0.0, de marzo de 2006, pues es la que estaba disponible cuando empezamos el desarrollo. Todas las indicaciones que realicemos serán referidas a la versión utilizada, pues es sobre la que empezamos a trabajar, y el número de cambios realizados[10] apenas afecta para el seguimiento de este documento si desea utilizar la versión 2.1.0.

Puede descargar el código fuente o el fichero binario, en formato `.zip` o `.tar.gz` de la página web antes indicada. El archivo contiene una jerarquía de directorios, en cuya raíz está el directorio `lucene-2.0.0`. Es fácil extraer el contenido en el directorio `C:\` de Windows o en el directorio `home` de Unix. En Windows puede utilizar WinZip para descomprimir su contenido en `C:\`. Si está utilizando Unix, o quizás un servidor virtual como CygWin sobre Windows, descomprima el archivo `.tar.gz` en su directorio de usuario utilizando el comando `tar`, mediante la sentencia `tar zxvf lucene-2.0.0.tar.gz`.

En el directorio creado `lucene-2.0.0` encontrará el archivo `lucene-2.0.0.jar`. Este es el único archivo de Lucene que es necesario introducir dentro de las aplicaciones que se realicen. Por tanto, también debe incluirse en el momento de distribuir las aplicaciones. Por ejemplo, una aplicación Web que utilice Lucene debe incluir el fichero `lucene-2.0.0.jar` en el directorio `WEB-INF/lib`. Para aplicaciones de línea de comando asegúrese de que Lucene este referenciado en el *classpath* cuando este corriendo la máquina virtual de Java (JVM, *Java Virtual Machine*).

3.2. Integración de Lucene en las aplicaciones

En la Figura 2 se muestra la integración típica de una aplicación con Lucene, en la que el índice creado en Lucene contiene referencias de los documentos sobre los que se realizará la búsqueda.

Dado que Lucene es una libreria de programación orientada a objetos, se compone de una serie de clases que tienen métodos y propiedades. A continuación desglosaremos de qué clases está compuesta este API:

3.2.1. Clases para desarrollar el procedimiento de indexación

1. **IndexWriter**: Es el componente central del proceso de indexación. Esta clase crea un nuevo índice y añade documentos al índice existente. Es un objeto que da acceso de escritura al índice pero no de lectura o búsqueda.

[9] `http://lucene.apache.org/nutch/` .

[10] Puede ver la lista de cambios de las diferentes versiones en `http://svn.apache.org/repos/asf/lucene/java/tags/lucene_2_1_0/CHANGES.txt` .

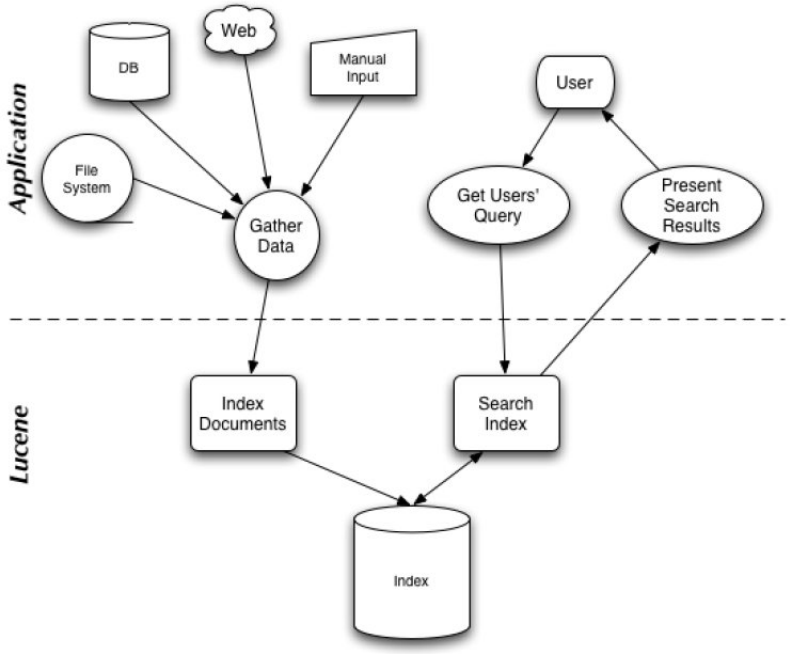

Figura 2: Arquitectura de Lucene. Tomada de [11, pág. 8].

2. **Directory**: Es la localización del índice de Lucene. En las aplicaciones es necesario almacenar el índice de Lucene, para lo cual se deben implementar las siguientes clases:

   a) **FSDirectory**: Mantiene la lista real de archivos en un directorio.

   b) **RAMDirectory**: Mantiene los datos en memoria. Para mayor rapidez de búsqueda, cuando un test finaliza, el índice es automáticamente destruido.

3. **Analyzer**: Se especifica en el constructor de la clase *IndexWriter*. Extrae los términos índice del texto que se va a indexar. Si el contenido a ser indexado no está en formato texto, debe convertirse primero.

4. **Document**: Es una colección de campos. Los campos de un documento representan los *metadatos* asociados a ese documento, y son los típicos campos de autor, título, materia, fecha de modificación, etc. Estos campos son indexados y almacenados en forma separada como campos de un documento.

5. **Field**: Es una porción de los datos que es consultada y localizada en el índice durante la búsqueda. Lucene ofrece 4 tipos de campos a escoger:

   a) **Keyword**: No es analizado pero sí indexado y almacenado en el índice *verbatim*. Se utiliza para campos cuyo valor original debe ser preservado, como por ejemplo, URL, rutas del sistema de ficheros, nombres, fechas, números de seguridad social, etc.

   b) **UnIndexed**: No es analizado ni indexado pero sí almacenado su valor en el índice. Se utiliza para campos que necesitan ser desplegados con resultados de búsqueda (como un URL o una llave primaria de una base de datos), pero cuyos valores no serán buscados directamente. No debe utilizarse para almacenar campos con valores muy largos si el tamaño del índice es importante.

   c) **Unstored**: El opuesto al *UnIndexed*. Es analizado e indexado pero no almacenado en el índice. Se utiliza para indexar una cantidad larga de texto que no necesita ser recuperado en su forma original.

   d) **Text**: Es analizado e indexado. Los campos pueden ser buscados, pero debe tenerse cuidado con el tamaño del campo.

Todo campo tiene un nombre y un valor, que son pasados como argumentos al definir el tipo de campo a crear. Estos son algunos ejemplos de cómo utilizar la clase *Field* y, según su tipo, si cada campo es analizado, indexado o guardado en el momento de crear el índice:

Método Field	**Analizado**	**Indexado**	**Guardado**	**Ejemplos**
Field.Keyword(String, String) Field.Keyword(String, Date)		x	x	Teléfonos y números de seguridad social, URLs, nombres de personas, fechas
Field.UnIndexed(String, String)			x	Tipo de documento (PDF, HTML, etc.), si no son usados como un criterio de búsqueda
Field.UnStored(String, String)	x	x		Títulos de documentos y contenidos
Field.Text(String, String)	x	x	x	Títulos de documentos y contenidos
Field.Text(String, Reader)	x	x		Títulos de documentos y contenidos

Tabla 1: Tipos de Campos en Lucene. Tomado de [11].

3.2.2. Clases principales para desarrollar el procedimiento de búsqueda

1. **IndexSearcher**: Se utiliza para buscar qué objetos *IndexWriter* están indexados. Es el enlace central al índice que ofrece muchos métodos de búsqueda. Es una clase que analiza e indexa la consulta en modo de solo lectura.

2. **Term**: Es la unidad básica para la búsqueda. Similar al objeto File, consiste en una dupla de cadena de caracteres: el nombre del campo y el valor del campo.

3. **Query**: Es la clase padre común y abstracta que se realiza en el proceso de búsqueda. Lucene cuenta con una serie de subclases del objeto *Query*. El Query más básico de Lucene es *TermQuery*, pero existen otros como el BooleanQuery, PhraseQuery, PrefixQuery, PhrasePrefixQuery, RangeQuery, FilteredQuery y SpanQuery.

4. **TermQuery**: Es el tipo de *Query* más básico soportado por Lucene. Se utiliza para encontrar documentos que contienen campos con valores específicos.

5. **Hits**: Es una colección de punteros para el ranking de los resultados de búsqueda.

6. **QueryParser**: Esta clase realiza un parsing de las consulta de los usuarios. Esta clase es una de las mas convenientes y rápidas en su implementación dentro de Lucene. Partiendo de la consulta del usuario, la analiza y crea internamente los objetos del tipo de la clase *Query* para satisfacer la búsqueda.

Con la utilización de clases de indización de documentos y de búsqueda de consultas listadas anteriormente, es posible crear el primer ejemplo de su utilización. Lo vemos en la siguiente sección.

3.3. Un ejemplo de Lucene

Existen muchos tutoriales básicos para crear aplicaciones sencillas utilizando la librería de Lucene. Uno de ellos es el de Steven Owens [17], en el que se incluyen varios ejemplos. Lo recomendamos por su fácil lectura. A continuación indicamos paso a paso la manera de utilizar la librería de Lucene.

3.3.1. Creando un índice

Hemos creado el ejemplo llamado *CreateIndex.java*, el cual crea un índice vacío mediante la creación de una nueva instancia del objeto *IndexWriter*. En el siguiente ejemplo, el nombre del directorio en el que se desea guardar el índice esta especificado en la línea de comando mediante el parámetro *args*.

```
public class CreateIndex
      {
   // modo de uso: CreateIndex index-directorio
         public static void main(String[] args) throws Exception
         {
            String indexPath = args[0];
            IndexWriter writer;
            // Un índice se crea creando una instancia del objeto IndexWriter
            // con el argumento "create" con el valor en verdadero.
            writer = new IndexWriter(indexPath, null, true);
            writer.close();
          }
}
```

Un constructor es el método para crear una clase. Los constructores definidos en la clase *IndexWriter* aceptan 3 parámetros o argumentos, el primer parámetro esta destinado a definir la ruta donde se creará el índice, el segundo define el tipo de analizador sintáctico utilizado, y el tercero si se creará el índice nuevalmente:

```
IndexWriter(FileInfo path, Analyzer a, bool create);
IndexWriter(Directory d, Analyzer a, bool create);
IndexWriter(string path, Analyzer a, bool create);
```

3.3.2. Indexando documentos de texto

El siguiente ejemplo, *IndexFile.java*, nos muestra la manera en que se agregan documentos a un índice. Los archivos de documentos son pasados a través de parámetros mediante la línea de comandos. Para cada archivo, la clase *IndexFiles* crea un objeto *Document*. Este objeto llama al método *IndexWriter.addDocument* para incluir el documento en el índice. Desde el punto de vista de Lucene, un documento (*Document*) es una colección de campos que están compuestos por la dupla *nombre-valor*. Un campo (*Field*) puede obtener su valor desde una cadena de caracteres (*String*) para campos cortos, o mediante la clase *InputStream* para campos largos. La utilización de los campos permite dividir el documento en secciones para búsqueda e indexación, y asociar metadatos (como el nombre, autor o fecha de modificación) con un documento.

En el código listado a continuación se guardarán dos campos de cada documento (*Document*). El primer campo es (*path*), el cual es utilizado para identificar la ruta del archivo, de manera que será posible recuperarlo posteriormente. El segundo es el cuerpo, *body*, utilizado para el contenido del archivo. El campo *path* es de tipo *UnIndexed*, lo cual significa que este valor es solamente guardado dentro del índice con el único propósito de obtener el valor en el futuro, pero no se indexa y analiza, ya que sobre él no se realizará ninguna búsqueda. El campo *body* es de tipo *Text*, y por tanto, se indexa y analiza. Puede ver estos tipos en la Tabla 1.

```
public class IndexFiles
      {
```

```
        // modo de uso: IndexFiles index-path file . . .
        public static void main(String[] args) throws Exception
        {
           String indexPath = args[0];
           IndexWriter writer;
           writer = new IndexWriter(indexPath, new SimpleAnalyzer(), false);
           for (int i = 1; i < args.length; i++)
           {
              System.out.println("Indexing file " + args[i]);
              InputStream is = new FileInputStream(args[i]);
              // Creando un Document con dos campos, uno contiene
              // la ruta del archivo, y el otro el contenido del archivo.
              Document doc = new Document();
              doc.add(Field.UnIndexed("path", args[i]));
              doc.add(Field.Text("body", (Reader) new InputStreamReader(is)));
              writer.addDocument(doc);
              is.close();
           };
           writer.close();
        }
     }
```

El texto original debe ser normalizado en términos por el analizador (*analyzer*), que realiza acciones como eliminar palabras vacias, convertir términos a minúsculas, aplicar lematización, insertar términos (procesamiento de sinónimos), quitar acentos, etc.

3.3.3. Realizando la búsqueda

La clase central utilizada para buscar documentos en un índice es *IndexSearcher*. Esta clase tiene muchos métodos de búsqueda, pero el más común es aquel que permite buscar un término específico en el índice. Es importante destacar que los términos pasados al objeto *IndexSearcher* deben ser consistentes, en términos de tratamiento de texto, con los términos producidos por el analizador de los documentos fuente. Como resultado de la búsqueda obtenemos un objeto *Hits*. Este objeto encapsula el acceso a los documentos, para ofrecer mejor rendimiento en cuanto al acceso a los mismos. Los documentos completos no son devueltos de manera inmediata, sino bajo demanda.

Lucene incluye una característica que transforma expresiones *Query* en clases *QueryParser*. Esta clae requiere un analizador (*analyzer*) para dividir la consulta en términos (*terms*), de hecho es el único elemento de búsqueda que utiliza un analizador. La clase *QueryParser* tiene un método estático, *parse()*, para uso simple. Este método es rápido y en general conveniente en la mayoría de búsquedas, pero no siempre es suficiente.

El archivo *Search.java* provee un ejemplo de cómo buscar documentos dentro de un índice. Lo explicamos brevemente. Se utiliza el objeto *Searcher* usando el objeto *QueryParser* para crear el objeto *Query*, y haciendo la llamada del método *Searcher.search* en la consulta. La operación de búsqueda devuelve el objeto *Hits*: una lista de objetos de tipo *Document*, donde cada documento satisface la consulta planteada, que además tiene asociado un factor de relevancia (score), de manera que la lista está ordenada por este factor.

```
public class Search
      {
         public static void main(String[] args) throws Exception
         {
            String indexPath = args[0], queryString = args[1];
            Searcher searcher = new IndexSearcher(indexPath);
            Query query = QueryParser.parse(queryString, "body",
                                 new SimpleAnalyzer());
            Hits hits = searcher.search(query);
            for (int i=0; i<hits.length(); i++)
            {
               System.out.println(hits.doc(i).get("path") + "; Score: " +
                                  hits.score(i));
            };
         }
      }
```

4. dotLucene

Basándonos en las clases explicadas y en el ejemplo básico de la sección 3, ahora es posible realizar este mismo ejemplo en un entorno de desarrollo como .NET, y probar así su portabilidad y eficiencia. Utilizaremos la librería **dotLucene**.

4.1. Ventajas que ofrece dotLucene

Se han tenido en cuenta la serie de ventajas que SourceForge [19] establece al desarrollador que utilice el API de dotLucene [8]:

1. Muy buen rendimiento.
2. Ranking al buscar resultados.
3. Resaltar los términos que cumplen el criterio de búsqueda.
4. Búsqueda tanto de datos estructurados como no estructurados.
5. Búsqueda de metadatos.
6. El tamaño del índice se reduce en aproximadamente un 30 % respecto del texto indexado.
7. Puede indexar y guardar documentos a texto completo.
8. Gestión pura en .NET como un simple *assembly*.
9. Tipo de licencia amigable (Apache Software License 2.0).
10. Internacionalización, por tanto es fácilmente localizable. El paquete DotLucene National Language Support Pack soporta Portugués, Checo, Chino, Holandés, Ingles, Francés, Alemán, Japonés, Coreano y Ruso.
11. Modificable y extensible (código abierto incluido).
12. El índice generado es compatible con el de la versión de Lucene en Java de Apache.

Siguiendo la misma filosofía de Lucene, dotLucene puede indexar cualquier documento que se traduzca a texto de manera muy eficiente. Vamos a verlo.

4.2. Comenzando un nuevo proyecto con dotLucene

La librería de dotLucene se concentra en un *assembly* simple de .NET (Lucene.Net.dll), que se debe incluir como referencia al proyecto creado. Este importará todas las clases y objetos de Lucene, estando accesibles para su programación. Una vez creada la referencia, se deben importar en la clase que desee hacer uso de las clases de dotLucene las siguientes librerias:

```
using Lucene.Net.Index;
using Lucene.Net.Documents;
using Lucene.Net.Analysis;
using Lucene.Net.Analysis.Standard;
using Lucene.Net.Search;
using Lucene.Net.QueryParsers;
```

Teniendo como base al ejemplo en Java del API de Lucene realizado en la sección 3.3, hemos realizado las funciones de creación del índice de Lucene, la indización de documentos de texto y, finalmente, hemos realizado una búsqueda sobre los documentos indexados. En la siguiente sección obtendremos la misma funcionalidad, pero utilizando el API de dotLucene.

5. Indexar y buscar en documentos

Una aplicación real de mucha utilidad hoy día es la que realiza la búsqueda de información concreta dentro de la gran cantidad de archivos almacenados en nuestro ordenador, organizados en directorios o carpetas. Este ha sido nuestro objetivo a la hora de implementar la librería de dotLucene. A continuación explicaremos todos los detalles de nuestra aplicación. Con ello pretendemos ofrecer al lector los mecanismos básicos de actuación con esta librería.

En general la extensión de los archivos nos indican su contenido. Así pues, de los existentes, intentaremos crear una lista de los que deben ser considerados para su indización. Es posible indexar los archivos en Lucene siempre que puedan ser tranformados en texto. Finalmente, se pretende obtener como resultado una lista de archivos que satisfagan el criterio de búsqueda planteado por el usuario.

La lista de tareas a realizar es la siguiente:

1. Definir los directorios donde (i) se creará el índice, (ii) donde se encuentran los documentos que serán indexados y sobre los que se realizarán búsquedas.
2. Creación de un índice.
3. Añadir documentos al índice.
4. Salvar el índice creado.
5. Realizar una búsqueda.
6. Obtener los resultados.

5.1. Definición de directorios o carpetas

Esta es la primera tarea que debe ser realizada, ya que se necesitan dos rutas o carpetas: aquella donde se creará el índice, y aquella donde están almacenados los documentos que se desea indizar. La Figura 3 muestra la interfaz en la que se especifican ambas carpetas en nuestro programa.

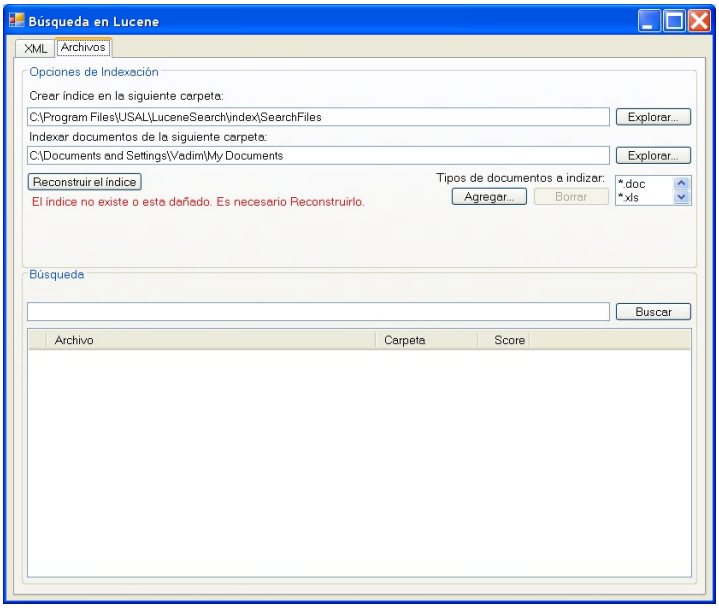

Figura 3: Definición de carpetas en nuestra aplicación.

5.1.1. Carpeta donde se crea el índice

En la Figura 3 se introduce primero el directorio donde se crea el índice. En nuestro caso, hemos asignado el directorio de instalación de nuestra aplicación, seguida del directorio por omisión para esa operación (`C:\Program Files\USAL\LuceneSearch\index\SearchFiles`). En esta carpeta Lucene se encargará de generar el índice.

Nuestra aplicación, cuando es ejecutada, intenta buscar en la carpeta predefinida información de algún índice que pueda existir con anterioridad. Si es así, mostrará el número de documentos indexados. En caso contrario, es decir, es la primera vez que es ejecuta el programa, mostrará un mensaje indicando que "*El índice no existe o esta dañado*". Es necesario, por tanto, reconstruirlo. La verificación de la existencia de un índice se realiza en Lucene mediante la siguiente función:

```
private void verIndex(TextBox pathText, Label statusLabel)
{
  try
  {
     searcher = new IndexSearcher(pathText.Text);
     searcher.Close();
  }
  catch (IOException)
  {
     status(statusLabel,
        "El índice no existe o esta dañado. Es necesario Reconstruirlo.", true);
     return;
  }
  string msg = String.Format("El índice ha sido creado. Contiene {0} documentos.", searcher.MaxDoc());
  status(statusLabel, msg);
}
```

En esta función se crea el objeto *IndexSearcher* que es el mismo utilizado posteriormente para la búsqueda, en el que se incorpora como parámetro el directorio donde se supone esta guardado el índice. En esta función se valida la existencia o no de un índice, y en caso de que no exista ningún error en su creación, mostrará el número de documentos indexados mediante la propiedad *MaxDoc()* de la clase *IndexSearcher*.

5.1.2. Carpeta que contiene los documentos a indexar

La segunda carpeta contiene todos los documentos que se desean indexar, y sobre los que se realizarán las búsquedas. Para nuestro ejemplo hemos introducido la carpeta `C:\Document and Settings\Vadim\My Documents`.

5.2. Creación de un índice

Para crear un índice es necesario utilizar la clase de Lucene denominada *IndexWriter*, mediante la creación de un objeto de esta clase de la siguiente manera:

```
//creando objeto IndexWriter. parametros(directorio , Analizador, crearlo)
indexWriter = new IndexWriter(this.textBoxIndex.Text, new StandardAnalyzer(), true);
```

Los parámetros enviados son los siguientes:

1. El directorio donde se creará el índice.
2. El objeto del analizador sintáctico que debe utilizar Lucene.
3. Si se creará nuevamente el índice (si este parámetro se envía con valor negativo se asume que ya esta creado el índice).

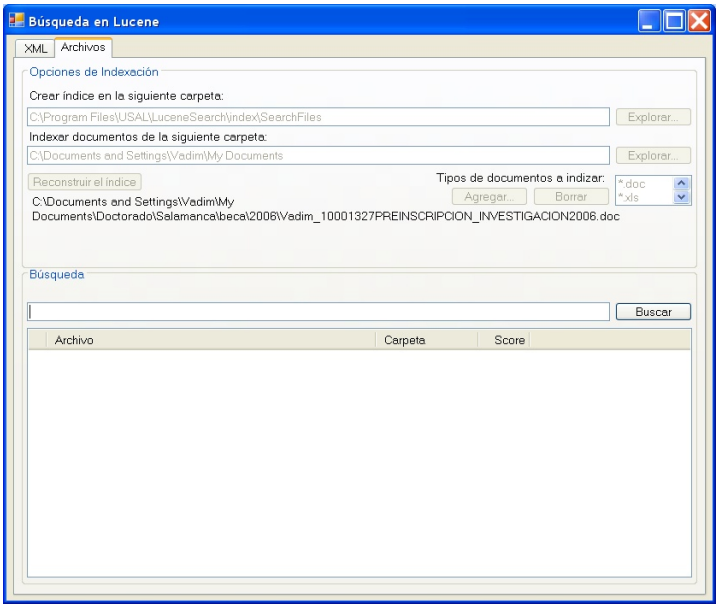

Figura 4: Creando el índice en nuestra aplicación.

El proceso de creación de un índice se inicia al pulsar el botón *Reconstruir el índice*, como puede verse en la Fig. 4. Este proceso realiza las siguientes tareas:

1. Recorrer todos los archivos del directorio definido que pertenezcan a los tipos especificados. Se filtran únicamente documentos convertibles a texto, aunque el programa permite añadir o quitar tipos de documentos. Esto es posible mediante los botones *Agregar* y *Borrar* situados justo a la derecha de la lista de los tipos de documentos a indizar. Inicialmente los documentos que se indizarán son los que poseen extensiones `.doc`, `.xls`, `.ppt`, `.htm` y `.txt`.

2. Añadir cada documento al índice.

3. Mostrar resultados.

Nuestra aplicación va recorriendo todos los documentos de la carpeta que contiene aquellos a indexar, y de manera recursiva también los subdirectorios que posea. Es necesario indicar que en el proceso se omiten archivos temporales, como por ejemplo los que comienzan con el símbolo `~`. Para ello se utiliza la siguiente función:

```
private void adicionarFolder(DirectoryInfo directory)
{
   // encontrando los archivos que hacen matching
   foreach (string pattern in patterns)
   {
      foreach (FileInfo fi in directory.GetFiles(pattern))
      {
         // omitir archivos temporales de office en el proceso
         if (fi.Name.StartsWith("~")) continue;

         try
         {
             //adicionando Documento de Office a Lucene
             adicionarOfficeDocument(fi.FullName);

             // actualizar contadores
             this.countTotal++;
             this.bytesTotal += fi.Length;

             // mostrar el archivo indexado que ha sido agregado
             status(this.labelStatusIndex, fi.FullName);
          }
```

```
            catch (Exception)
            {
               // parsing e indexado no se realizo satisfactoriamente. el archivo es omitido
               this.countSkipped++;
               status(this.labelStatusIndex, "Omitidos: " + fi.FullName);
            }
         }
      }

      // adicionando subfolders
      foreach (DirectoryInfo di in directory.GetDirectories())
      {
         adicionarFolder(di); //llamado recursivo
      }
   }
```

5.3. Añadir documentos al índice

Cada objeto que se desee incluir en el índice debe ser un objeto del tipo de la clase *Document* de Lucene, descrito anteriormente en la sección 3.2.1. Cada documento en el índice contiene campos con alguna información. Para cada campo se debe especificar de qué tipo es, para que se incorpore al índice de manera adecuada. Los parámetros necesarios para crear los tipos de campos en Lucene son:

1. **Stored**: Salvados en el índice. Es posible obtener cualquier valor de este tipo desde el índice en cualquier momento posterior; esto es aplicable para campos cortos como autor, título, etc.
2. **Indexed**: El indexado es requerido para todos los campos que se desean consultar.
3. **Tokenized**: Cuando se desea dividirlo en palabras antes de indexarlo.

La creación de campos se realiza mediante el constructor *Field* siguiente:

```
Field(string campoNombre, string campoValor, bool stored, bool indexed, bool tokenized);
```

Además del constructor, existen los métodos estáticos para la creación de nuevas instancias de campos que fueron mostrados en la Tabla 1 de la sección 3.2.1.

Con la función *adicionarOfficeDocument(path)* agregamos al índice de Lucene, representado en el objeto *indexWriter*, un documento:

```
private void adicionarOfficeDocument(string path)
{
   //creando objeto Documento
   Document doc = new Document();
   string filename = Path.GetFileName(path);

   //creando los Campos del Documento
   doc.Add(Field.UnStored("text", Parser.Parse(path)));
   doc.Add(Field.Keyword("path", path));
   doc.Add(Field.Text("title", filename));
   indexWriter.AddDocument(doc);
}
```

Para nuestro ejemplo hemos creado tres campos: *text*, *path* y *title* de tipo *UnStored*, *Keyword* y *Text*, respectivamente. A continuación se crea una clase definida por el usuario denominada *Parser* y un método *Parse* que extrae el contenido de cada documento, y así es guardado dentro del campo de tipo *text* de Lucene.

Lucene ofrece una alta flexibilidad para definir los tipos de campos, todo depende de la implementación que se desee dar. Lo importante es determinar qué tipo de campo es, y si se utilizará o no para buscar información en él (por lo que se deberá indexar), o si solo es un campo de despliegue o de referencia para ser mostrado una vez encontrada la información sobre otros campos.

5.4. Guardar el índice

Después de que el índice se ha creado, debe guardarse con todos los documentos indexados. Para hacerlo se ejecutan los siguientes métodos sobre el objeto *IndexWriter*:

```
IndexWrite.Optimize();
IndexWrite.Close();
```

Básicamente el método *Close* cierra y guarda el índice. Es recomendable realizar el método de optimización (*Optimize*) para mejorar el proceso de búsqueda, aunque tome un tiempo adicional de procesado en el momento de crear el índice. Al finalizar el proceso se muestra el estado de cuántos archivos se agregaron y omitieron, y el tiempo transcurrido, como se muestra en la Fig. 5.

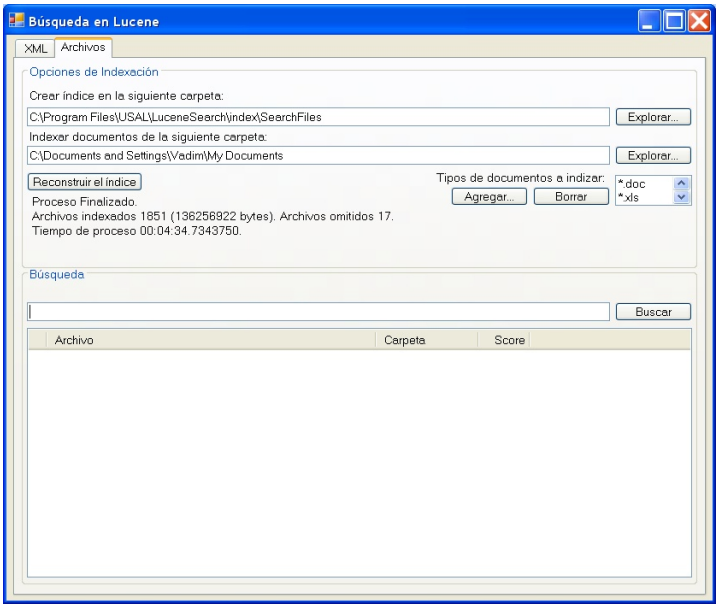

Figura 5: Proceso finalizado de creación del índice.

La Figura 6 muestra las propiedades de la indización: el tamaño total del directorio (18GB), donde existen un total de 14.608 documentos y 1.127 carpetas que se querían indexar. De todos estos documentos solamente se han considerado 1851, como puede verse en la Fig. 5. También vemos el tiempo que ha supuesto realizar la indexación (4 minutos 34 segundos). Hemos utilizado el objeto *FSDirectory* de Lucene visto en la sección 3.2.1.

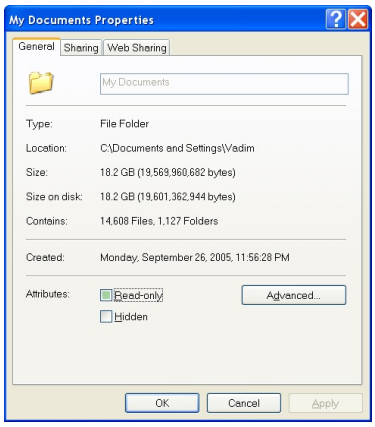

Figura 6: Información del directorio origen de documentos.

5.5. Realizando la búsqueda

En nuestro ejemplo la búsqueda se realizará hacia el objeto de tipo *IndexSearcher* que fue creado al comenzar a ejecutarse la aplicación, tal como se vio en la sección 3.2.2 si existía un índice o al reconstruirlo nuevamente.

```
searcher = new IndexSearcher("Directorio donde buscar");
```

La creación de la consulta se realiza precisamente al utilizar el objeto *QueryParser* de Lucene.

```
string consulta = "texto a buscar";
Query query = QueryParser.Parse(consulta, "text", new StandardAnalyzer());
```

Los parámetros utilizados en el método *Parse* son la cadena de caracteres que forma la búsqueda, el campo donde buscar la información, y el objeto analizador léxico de Lucene. Los resultados son después obtenidos en el objeto *Hits*, esto es, la lista de documentos encontrados ordenados por su ranking (score).

```
Hits hits = searcher.Search(query);
Console.WriteLine("Documentos Encontrados " + hits.Length() + ":\r\n");
for (int i = 0; i < hits.Length(); i++) {
   Document doc = hits.Doc(i);
   //agregar doc a una lista
}
```

Siempre se debe cerrar el objeto *IndexSearcher* al finalizar la consulta.

```
searcher.Close();
```

El proceso de búsqueda se implementa utilizando el siguiente código fuente:

```
//Proceso de Busqueda
private void search()
{
   //marcando inicio de tiempo
   DateTime start = DateTime.Now;
   //creando objeto de busqueda IndexSearcher. parametro(directorio donde esta el indice)
   searcher = new IndexSearcher(this.textBoxIndex.Text);

   //creando objeto QueryParser. parametros(consulta(termino), "campo", Analizador)
   Query query = QueryParser.Parse(this.textBoxQuery.Text, "text", new StandardAnalyzer());

   // Búsqueda
   Hits hits = searcher.Search(query);

   for (int i = 0; i < hits.Length(); i++)
   {
      // obteniendo el documento desde el índice
      Document doc = hits.Doc(i);

      // creando una nueva fila con los resultados de los datos
      string filename = doc.Get("title");
      string path = doc.Get("path");
      string folder = Path.GetDirectoryName(path);
      DirectoryInfo di = new DirectoryInfo(folder);

      ListViewItem item =
       new ListViewItem(new string[] { null, filename, di.Name, hits.Score(i) });
      this.listViewResults.Items.Add(item);
   }
   searcher.Close();
   //mostrando resultados
   string searchReport =
```

```
        String.Format("La búsqueda se realizo en {0}. Se encontraron {1} items.",
    (DateTime.Now - start), hits.Length());
    status(this.labelStatusResults, searchReport);
}
```

Una vez realizada la búsqueda se presentan los resultados en una lista, en la que se indica la cantidad de documentos encontrados y el tiempo que ha tomado su búsqueda, tal y como se muestra en la Fig. 7. En nuestra aplicación es posible ir viendo cada archivo de la lista sin más que hacer doble clic sobre él, ello abrirá un proceso relacionado con el programa que maneja el documento en cuestión, y que aparece como un icono a la izquierda del mismo.

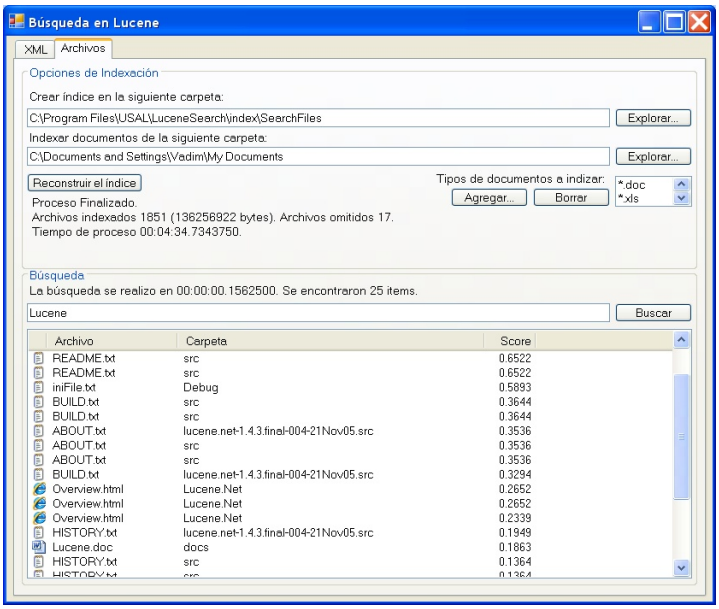

Figura 7: Resultados de la búsqueda.

6. Indexar y buscar en documentos XML

Después de realizar el ejemplo anterior, en el que se indexaban documentos de un árbol de directorios, obteniendo una lista de documentos que satisfacían un criterio de búsqueda, ahora intentaremos explorar aún más la librería de Lucene, para lo cual hemos desarrollado una aplicación que realiza una serie de tareas adicionales en la búsqueda de información sobre archivos XML.

Como es bien sabido, hoy en día el formato XML es un estándar muy utilizado para guardar, representar y transferir información, por lo que en el campo de la recuperación de información es de mucha importancia. De hecho muchas fuentes de datos, como por ejemplo las que se utilizan en las conferencias TREC, CLEF y NTCIR, utilizan este formato. Ello permite tener una diversidad de datos más amplia que los simples documentos de texto.

A continuación se detallan las tareas realizadas para poder obtener información de archivos XML, generar el índice basándose en campos definidos y buscar información en ellos. Hemos creado una aplicación que nos sirva de base para explicar todos los aspectos de la librería de Lucene relacionados con la búsqueda de información sobre archivos XML. Además hemos incluido comentarios de cómo funcionan otros aspectos de la aplicación, para dar al lector una visión más amplia de cómo utilizar las diferentes opciones de la misma, de manera que puedan serle útil en sus propios desarrollos.

6.1. Iniciando la aplicación

Nos propusimos que la aplicación fuera capaz de leer parámetros de configuración, como el lugar de dónde obtener los datos, la manera en que éstos se encuentran y dónde generar ficheros que representen el índice. Después de la instalación del programa se define su carpeta basándose en la opción escogida por el usuario, como se observa en la Figura 8. Dentro de esta carpeta de instalación existe el archivo de nombre *'LuceneSeach.var.txt'*, que guarda las variables de la última sesión de ejecución del programa, en este caso el nombre del archivo de configuración (*vector.ini*) y el analizador léxico utilizado (*SpanishAnalyzer*).

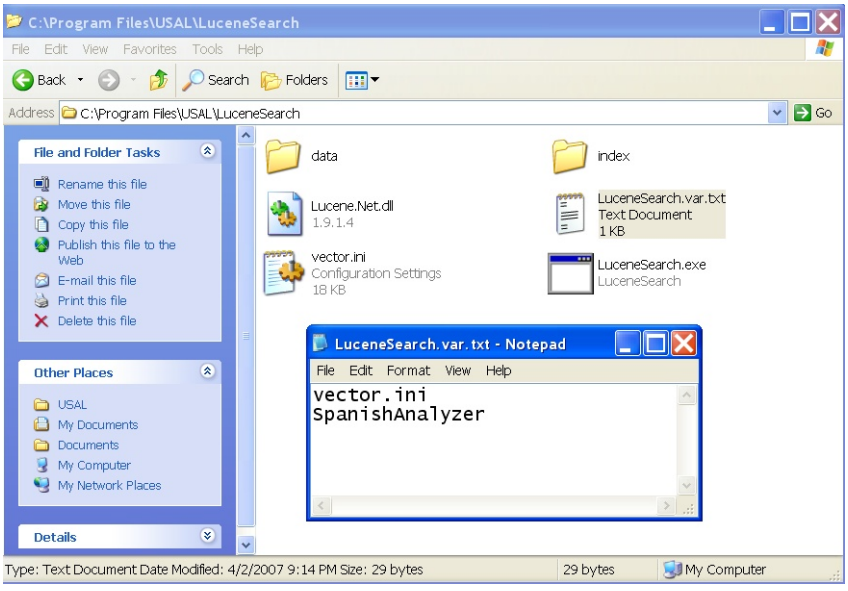

Figura 8: Ubicación del archivo de configuración

En el archivo `.ini` de configuración se guardan todos los parámetros relacionados con el arranque de la aplicación. Después de tomar los valores de secciones anteriores y del archivo de configuración, la aplicación mostrará si existe o no un índice creado. En la Figura 9 se muestra la carpeta donde se busca el índice creado, dando la opción de crearlo nuevamente (botón *Crear índice*), o la de explorar en otro directorio (botón *Explorar...*).

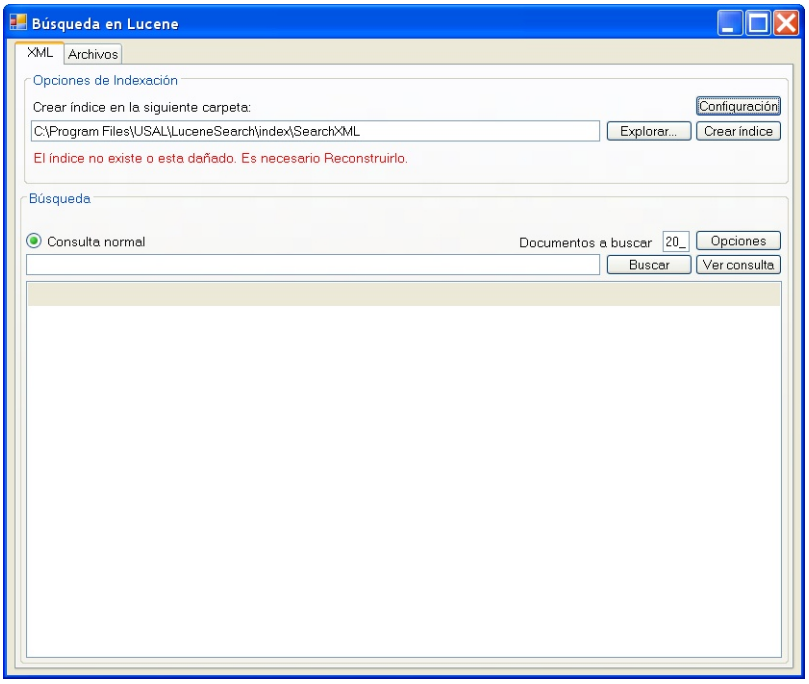

Figura 9: Indexando y buscando en documentos XML.

6.2. Administración de parámetros de inicialización

La administración del archivo de configuración se realiza al presionar el botón *Configuración* de la Fig. 9, mostrando una pantalla con todas las opciones parametrizables del archivo de configuración que son modificables.

En la Figura 10 se muestra la ventana de configuración de parámetros del archivo de configuración, el cual contiene las siguientes secciones:

1. **Opciones del fichero de configuración**, donde se permite definir la ruta y nombre del fichero de configuración.
2. **Fuentes de datos**, que obtiene la lista de documentos XML.
3. **Campos**, que obtiene la lista de campos definidos en el archivo de configuración dando la opción de agregar, editar o quitar alguno de ellos.
4. **Obtención de datos**, es un muestreo inicial para verificar la obtención de resultados de nuestra fuente de datos.
5. **Index tuning**, son opciones para mejorar el rendimiento en el momento de crear el índice.
6. **Analizador léxico**, igualmente son opciones que se utilizan para realizar de una manera especial tanto la indexación como la búsqueda, tal es el caso de implementación de búsquedas de términos en español que implican la realización de un analizador léxico para dicho idioma, *SpanishAnalyzer*.

6.2.1. Opciones del fichero de configuración

Este archivo está basado en la versión 2.0 del archivo de configuración de un sistema de recuperación de información desarrollado en Perl por el Grupo de investigación en Recuperación de Información Avanzada, del Departamento de Informática y Automática de la Universidad de Salamanca. Describimos los aspectos más importantes del mismo.

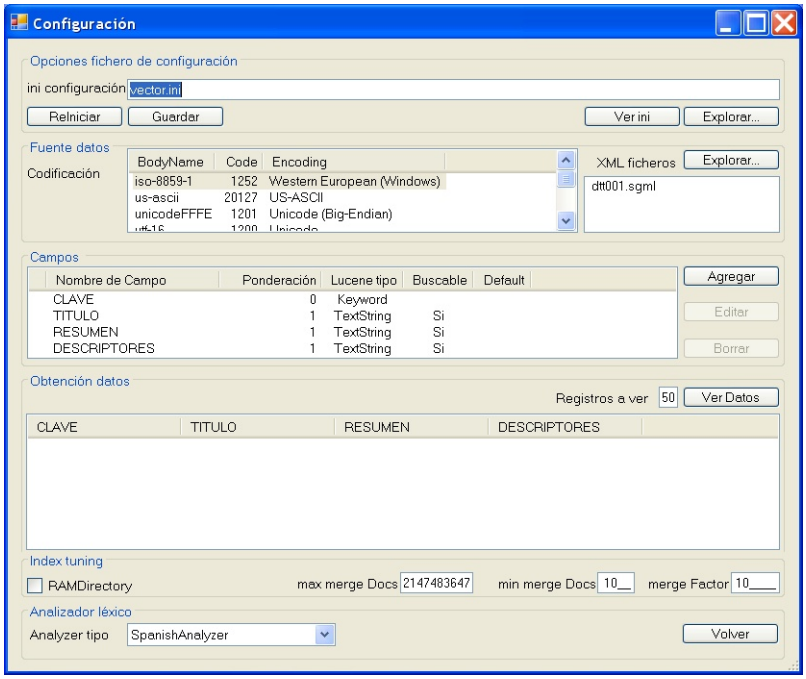

Figura 10: Pantalla del valores del archivo de configuración.

1. **Ruta de archivo de inicialización**. La ruta por omisión del archivo de configuración es la misma en la que se encuentra el programa ejecutable de nuestra aplicación, *LuceneSearch.exe*, teniendo el nombre de *vector.ini*.

2. **Visualización del archivo de inicialización**. En el listado siguiente se muestran las opciones que se guardan en el archivo de configuración, es posible visualizarlo al presionar el botón *Ver ini* de la Fig.10.

```
## Datos de la colección de documentos
DOC_PATH:  ./data                    # Ruta de los documentos
DOC_FICH:  dtt001.sgml               # Archivos con los documentos
DOC_SEP:   DOC                       # Etiqueta separadora de documentos
DOC_CLAVE: CLAVE                     # Etiqueta de clave de documentos
DOC_CAMPO: TITULO      ;1            # Etiqueta de campos, ponderación
DOC_CAMPO: RESUMEN     ;1
DOC_CAMPO: DESCRIPTORES;1
## Parámetros para el procesado léxico
FVACIAS: ./data/vacias_dura_efe.txt
```

3. **Guardar parámetros en el archivo de configuración**. Dado que es posible modificar los valores de todas las secciones de configuración, se permite guardar el nuevo fichero de configuración con los cambios realizados al presionar el botón *Guardar* de la Fig. 10.

4. **Reiniciar todos los valores de configuración**. Si se han realizado cambios, es posible obtener los valores iniciales de archivo de configuración presionando el botón *Reiniciar*, de este modo recrean todas las colecciones y variables internas; también se realiza el refresco de la ventana con los valores obtenidos del fichero `.ini` de configuración.

6.2.2. Fuente de datos

La fuente de datos esta determinada basándose en una lista de ficheros XML que son nombrados mediante la combinación del valor de la etiqueta `DOC_PATH` con múltiples valores de las etiquetas `DOC_FICH` para cada fichero de datos. A continuación describimos los aspectos que afectan al tratamiento de los ficheros XML.

1. **Codificación (*encoding*, tabla de caracteres)**. Cada fichero está codificado utilizando una tabla de caracteres, requisito necesario para poder leerlo correctamente. Al momento de abrir los documentos es necesario especificar la codificación que utilizan. Se han agregado los siguientes tipos de codificación que soporta el .NET framework [5]. De la lista de tipos de codificación mostrados en la Tabla 2 se establece por omisión el tipo *iso-8859-1*.

BodyName	Code	Encoding
iso-8859-1	1252	Western European (Windows)
us-ascii	20127	US-ASCII
unicode-FFFE	1201	Unicode (Big-Endian)
utf-16	1200	Unicode
utf-8	65001	Unicode (UTF-8)
utf-32	12000	Unicode (UTF-32)
utf-7	65000	Unicode (UTF-7)

Tabla 2: Tablas de caracteres disponibles en .NET framework.

La lectura de cada archivo XML se realiza mediante la clase *System.IO.StreamReader* del .NET framework, con la instrucción siguiente:

```
StreamReader sr = new StreamReader(Path + xfiles[i].ToString(), Encoding);
```

2. **Archivos de datos XML**. Los archivos XML utilizados contienen una lista de documentos delimitados entre sí por la etiqueta `DOC` e internamente están delimitados por campos, tal como se aprecia en el siguiente ejemplo:

```
<DOC>
<CLAVE>DTT001-0014</CLAVE>
<TITULO>  ROBERT LOWELL: PROCESO DE UNA VERSION
</TITULO>
<RESUMEN>  SE NARRA LA METODOLOGIA SEGUIDA PARA TRADUCIR A ROBERT LOWELL
TRATANDO DE REFLEJAR FIELMENTE SU ESTILO Y SU EVOLUCION
</RESUMEN>
<DESCRIPTORES>; TRADUCCION; LOWELL, ROBERT; POESIA NORTEAMARICANA
</DESCRIPTORES>
</DOC>
```

6.2.3. Campos

Los campos están definidos en los documentos XML por sendas etiquetas, entre las que es habitual encontrarse las correspondientes a los campos título, resumen, descripción, etc. En el fichero `.ini` de configuración se definen mediante las etiquetas `DOC_CAMPO`, que incluyen un valor de ponderación para dar más importancia a unos campos respectos de otros. En el momento de obtener los valores de los campos en nuestra aplicación, se recrea la lista de campos con sus características particulares: considerando su ponderación, tipo de campo de Lucene, si es tenido en cuenta para realizar la búsqueda y si es el campo por omisión (*default*) en el que buscar la información (Fig.10). Después es posible agregar, editar y eliminar campos, dependiendo en cual de ellos necesitemos buscar información. De esta manera se crea el índice. En la Figura 11 se muestran los datos correspondientes a cada campo. Evidentemente existe una relación directa con el API de Lucene para determinar el tipo de campo propio de Lucene, como se ve en la Fig. 12, que se corresponden con los mostrados en la Tabla 1 de la sección 3.2.1.

6.2.4. Obtención de datos

Teniendo la información en la fuente de datos y la manera en que está organizada en campos internos, es posible hacer un muestreo y ver una lista de resultados relacionados con los datos que maneja la aplicación.

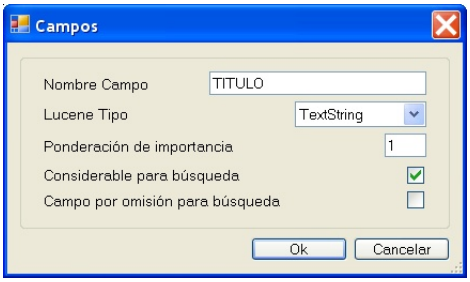

Figura 11: Edición de Campos.

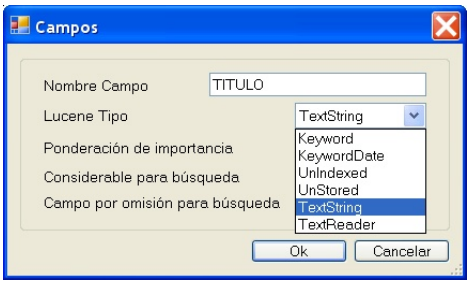

Figura 12: Seleccionando el tipo de campo de Lucene.

1. **Obtención de campos y datos de los archivos XML**. Los archivos de datos XML contienen una serie de documentos delimitados por etiquetas, cuyo valor está definido en el parámetro de configuración `DOC_SEP`, y también la información de cada campo dentro de éstos esta delimitada por etiquetas, que se corresponden con el parámetro de configuración `DOC_CAMPO`. Por tanto, se debe hacer uso de aplicaciones externas para extraer y leer estos ficheros. Una de ellas es *Digester* [10], pero existen otras muchas. En nuestro ejemplo, la obtención de los datos internos de los ficheros XML se obtiene mediante la clase *System.XML.XMLDocument* [6] del .NET framework, que permite obtener una colección de nodos en base a delimitadores, y así tener la lista de valores y campos de los archivos de datos.

2. **Mostrando una lista de datos obtenidos**. En la Figura 13 mostramos los primeros 50 registros de la fuente de datos XML correspondiente al fichero `dtt001.sgml`, para la codificación *iso-8859-1* utilizada en dicho fichero.

6.2.5. Afinamiento en la creación del índice

En este apartado se intenta agregar una lista de valores parametrizables con el objeto de mejorar el tiempo de creación del índice. Explicamos a continuación las distintas opciones incluidas en nuestro programa.

1. ***FSDirectory* versus *RAMDirectory***. La creación del índice necesita de la utilización de alguno de estos dos parámetros. Generalmente se utiliza *FSDirectory*, con lo cual el índice se crea en disco. En algunos casos, por optimización y para obtener resultados mas rápidos, en conveniente utilizar *RAMDirectory*, que procesa el índice en memoria. En nuestras pruebas de indexación obtuvimos una relación de la mitad del tiempo de proceso utilizando esta segunda opción.

2. **Merge Factor**. Esta es una propiedad de la clase *IndexWriter* de Lucene, y permite controlar cuántos documentos son guardados en memoria antes de escribir en el disco. El valor por omisión de este parámetro es de 10, esto significa que Lucene guarda 10 documentos en memoria antes de escribirlos a un segmento simple en el disco.

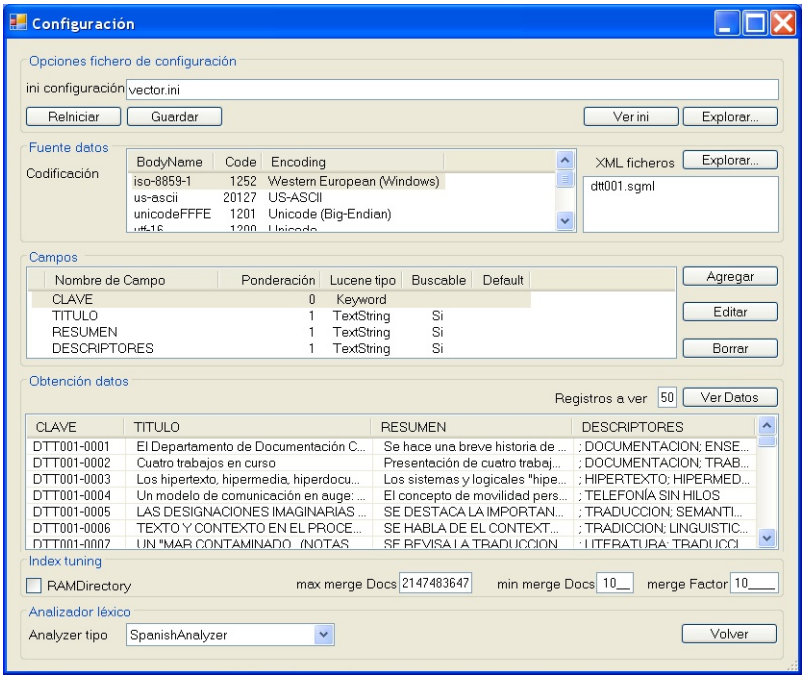

Figura 13: Obteniendo muestreo de datos.

3. **Max merge**. Esta es una propiedad de la clase *IndexWriter* de Lucene, y establece el límite máximo de documentos a guardar por segmento simple en disco.

4. **Min merge**. Esta es una propiedad de la clase *IndexWriter* de Lucene y establece el límite mínimo de documentos a guardar por segmento simple en disco.

6.2.6. Analizador léxico

El objeto analizador (*analyzer*) nos permite establecer la manera en que se procesarán y almacenarán internamente los términos en el índice, y la manera en que son tratadas las consultas a la hora de realizar las búsquedas. Es, por tanto, un parámetro requerido por los constructores de las clases *IndexWriter* y *IndexSearcher* de Lucene.

La Tabla 3 muestra los analizadores básicos que contiene la librería de Lucene internamente, y un ejemplo de cómo se realiza en cada uno de ellos el análisis basándose en los términos que indexa. En esta tabla cada término esta definido entre los símbolos paréntesis cuadrados, [ ]. De esta manera podemos ver las diferencias entre cada analizador basándose en qué términos son considerados.

Analizador	**Términos: The XY&Z cia - xyz@ciaxyz.com**
WhitespaceAnalyzer	[The] [XY&Z] [cia] [-] [xyz@ciaxyz.com]
SimpleAnalyzer	[The] [XY] [Z] [cia] [xyz] [ciaxyz.com]
StopAnalyzer	[XY] [Z] [cia] [xyz] [ciaxyz.com]
StandardAnalyzer	[The] [XY&Z] [cia] [-] [xyz@ciaxyz.com]

Tabla 3: Analizadores de Lucene.

En nuestra aplicación hemos incluido un analizador léxico definido por el usuario de nombre *SpanishAnalyzer*. Este analizador hereda todas las propiedades del objeto Analyzer básico de la librería de Lucene y maneja las nuevas reglas para el idioma español en cuanto a la aplicacón de lematización y la eliminación de palabras vacias.

1. **Stemmer**. Es el módulo lematizador, mediente la creación de la clase *SpanishStemmer.cs*, que se encarga de reducir las palabras a su raíz. Además no considera los acentos conviertiendo las vocales acentuadas a las correspondientes sin acentuar.

2. **Palabras vacías**. Se pueden incluir una lista de palabras para el proceso de indexación, de manera que no son considerados como términos para la búsqueda. Se permite al usuario definir esta lista de palabras, enviando como parámetro dicha lista en el momento de crear el objeto del analizador léxico. Internamente el analizador léxico define una colección por omisión de estas palabras, de nombre *SPANISH_STOP_WORDS*, que será utilizada en el caso de que el parámetro no se utilice.

6.3. Búsqueda de información

El algoritmo para realizar la búsqueda es similar al de la Sección 3.3.3, donde se buscaba en ficheros de un árbol de carpetas. La diferencia en este caso es que dependerá del analizador léxico que estemos utilizando para realizar la búsqueda.

6.3.1. Opciones de búsqueda

Las opciones de búsqueda que se pueden utilizar están listadas específicamente al presionar el botón *Opciones* del apartado de *Búsqueda* en la aplicación. En la Tabla 4 vemos las opciones válidas para consultas sobre todos los campos considerables para búsqueda.

Opción	**Descripción**
palabra	buscará los documentos que tienen esta palabra en cualquiera de sus campos
+palabra	buscará los documentos que tienen esta palabra en TODOS sus campos
palabra1 palabra2 ... palabraN	buscará los documentos que tengan cualquiera de estas palabras en cualquier campo
+palabra1 +palabra2 +palabraN	buscará los documentos que tengan todas las palabras y en todos sus campos
palabra1 -palabra2	busca específicamente documentos que contengan palabra1 y que no contengan palabra2
"frase con palabras"	busca exactamente la frase con palabras en los documentos
"palabra1 palabra5"∼5	busca los términos palabra1 y palabra5 separadas con 5 posiciones una de otra
java*	palabras que comienzan con "java", como "javaspaces", "javaserver", etc
java∼	busca palabras parecidas con "java", como por ejemplo "cava"

Tabla 4: Opciones de búsqueda para todos los campos.

En la Tabla 5 se listan una serie de opciones adicionales para buscar palabras o términos utilizando los operadores **AND** y **OR**, o los paréntesis () para la agrupación. Estas son posibles siempre que se tenga definido un campo por omisión para la búsqueda (*default*). La búsqueda en este caso se realiza basándose en el campo por omisión, así como en los campos considerables para búsqueda.

6.3.2. Mostrando resultados

La interfaz de usuario es otro factor muy importante a tener en cuenta en el diseño de un sistema de recuperación de información. En nuestra aplicación hemos desarrollado un interfaz de

Opción	Descripción
palabra1 OR palabra2...OR palabraN	buscará los documentos que tengan cualquiera de estas palabras en cualquier campo
palabra1 AND palabra2...AND palabraN	buscará encontrar todas las palabras en los documentos
(palabra1 OR palabra2) AND palabra3	buscará la existencia de palabra1 o palabra2 en cualquier campo y palabra3 en todos

Tabla 5: Opciones adicionales de búsqueda para el campo *Default*.

consulta bastante simple, pero efectivo. En dicho interfaz nos hemos centrado principalmente en ofrecer varias opciones de búsqueda, de acuerdo con las clases correspondientes que posee Lucene. Por ejemplo, podemos definir la consulta "+españa" para encontrar la lista de documentos que contienen el término "españa" en todos los campos definidos para la búsqueda.

En la Figura 14 se observa que se han encontrado 16 documentos que satisfacen nuestra consulta del total de 1177 que contiene el índice. El cálculo del tiempo de consulta es de lo mas óptimo de Lucene para este caso, fue prácticamente instantáneo. En la Figura 15 se muestra el primer documento recuperado. Con esto abríamos finalizado la explicación de nuestra segunda aplicación.

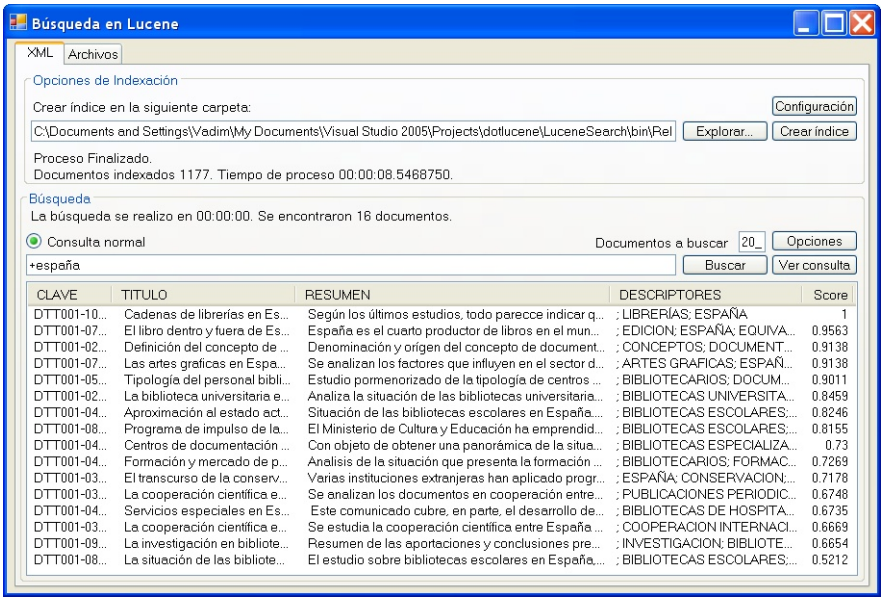

Figura 14: Resultado de la búsqueda "+españa".

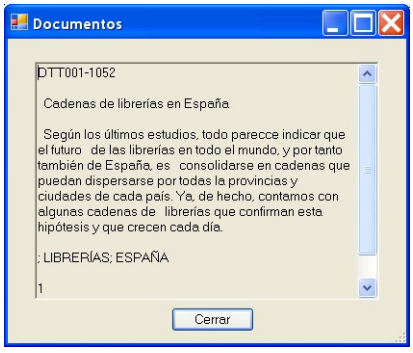

Figura 15: Mostrando un documento.

7. Conclusiones

En este informe técnico hemos puesto de manifiesto la facilidad en la utilización de Lucene y dotLucene. Para ello hemos desarrollado dos ejemplos diferentes de sistemas de indización y búsqueda de información. Para ambos ejemplos hemos documentado de manera detallada las diferentes clases y funcionalidades que nos permiten utilizar la librería.

Como resultado de todo nuestro desarrollo, de forma más específica, podemos realizar las siguientes conclusiones:

- Tanto Lucene como dotLucene son librerías fácilmente reutilizables para aplicaciones que requieran agregar la capacidad de búsqueda en aplicaciones que contengan una fuente de datos definida y su información se obtenga como texto.
- Una de las mejores ventajas de Lucene es que al ser un proyecto de código abierto y de rápida implementación, ofrece la posibilidad de agregar nuevas funcionalidades desarrollando nuevos componentes. En nuestro caso hemos agregado un analizador léxico que reconoce los patrones propios de la Lengua Española, permitiendo de esta manera la búsqueda de términos en este idioma.
- Las aplicaciones creadas deben considerarse puntos de partida para el estudio y ampliación de la potencia de Lucene como herramienta de recuperación de información, fortalecida fuertemente por su flexibilidad en la obtención y asignación de parámetros de configuración iniciales, lo que las hace portables y demostrables hacia diferentes ambientes y fuentes de datos que sigan las pautas de los ejemplos mostrados en nuestro trabajo.
- Finalmente, queremos resaltar que Lucene define un modelo de clases compacto y de fácil comprensión, por lo que una implementación inicial completa de búsqueda e indización se puede realizar con muy pocas líneas de código y pocas instancias de objetos de Lucene. A partir de este modelo se pueden ir agregando funcionalidades basándose en la exploración del modelo completo de clases de Lucene.

Referencias

[1] AENOR, editor. "Documentación. Tomo 2. Normas Fundamentales", tomo 56 de "Recopilación de normas UNE". AENOR, Madrid, Segunda edición (1997). [*Citado en pág. 4.*]

[2] Doug Cutting Apache Foundation's Jakarta. Apache Lucene: Overview [on-line]. http://lucene.apache.org/java/docs/ (mayo 2006). [visit. 15/12/2006] URL: `http://lucene.apache.org/java/docs/`. [*Citado en pág. 1.*]

[3] George Aroush. George Aroush's World [on-line]. [visit. 31/03/2007] URL: `http://www.aroush.net/`. [*Citado en pág. 1.*]

[4] Ricardo Baeza-Yates y Berthier Ribeiro-Neto. "Modern Information Retrieval". Addison-Wesley, Harlow, England (1999). [*Citado en págs. 2 y 4.*]

[5] Microsoft Corporation. Encoding Class MSDN Library [on-line]. [visit. 20/02/2007] URL: `http://msdn2.microsoft.com/en-us/library/system.text.encoding.aspx`. [*Citado en pág. 22.*]

[6] Microsoft Corporation. MSDN Library Online System.Xml Namespace [on-line]. [visit. 20/02/2007] URL: `http://msdn2.microsoft.com/en-us/library/system.xml.aspx`. [*Citado en pág. 23.*]

[7] Carlos G. Figuerola, José Luis Alonso Berrocal, Ángel F. Zazo Rodríguez y Emilio Rodríguez Vázquez de Aldana. Diseño de spiders. Informe técnico DPTOIA-IT-2006-002, Departamento de Informática y Automática - Universidad de Salamanca (marzo 2006). [*Citado en pág. 5.*]

[8] The Apache Software Foundation. DotLucene 1.9 API documentation [on-line]. [visit. 20/02/2007] URL: `http://www.dotlucene.net/documentation/api/1.9/index.html`. [*Citado en pág. 11.*]

[9] Brian Goetz. The Lucene search engine: Powerful, flexible, and free [on-line]. http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html (september 2000). [visit. 06/01/2007] URL: `http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html`. [*Citado en pág. 1.*]

[10] Otis Gospodnetic. Parsing, indexing, and searching XML with Digester and Lucene [on-line]. *developerWorks* (June 2003). [visit. 12/02/2007] URL: `http://www-128.ibm.com/developerworks/java/library/j-lucene/`. [*Citado en pág. 23.*]

[11] Otis Gospodnetic y Erik Hatcher. "Lucene in Action". Manning Publications (2004). GOS o 05:1 1.Ex. [*Citado en págs. 7 y 8.*]

[12] Gregory Grefenstette y Pasi Tapanainen. What is a word, What is a sentence? Problems of tokenization. En "Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX'94), Budapest", páginas 79–87 (1994). [*Citado en pág. 4.*]

[13] Gerald Kowalski. "Information Retrieval Systems. Theory and Implementation". Kluwer Academic Publishers, Boston (1997). [*Citado en pág. 2.*]

[14] Robert Krovetz. Homonymy and polysemy in information retrieval. En "36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, August 10-14, 1998, Université de Montréal, Montreal, Quebec, Canada", páginas 72–79. ACL / Morgan Kaufmann (1997). [*Citado en pág. 5.*]

[15] Joe Mayo. C# station tutorial [on-line]. http://www.csharp-station.com/Default.aspx (2005). [visit. 10/01/2007] URL: `http://www.csharp-station.com/Default.aspx`. [*Citado en pág. 1.*]

[16] José Antonio Moreiro González. Aplicaciones al análisis automático del contenido provenientes de la teoría metemática de la información. *Anales de Documentación* **5**, 273–286 (2002). [*Citado en pág. 3.*]

[17] Steven J. Owens. Lucene Tutorial [on-line]. http://darksleep.com/lucene/ (2001). [visit. 20/02/2007] URL: `http://darksleep.com/lucene/`. [*Citado en pág. 9.*]

[18] Tefko Saracevic. Relevance: A review of and a framework for the thinking on the notion in information science. *Journal of the American Society for Information Science* **26**(6), 321–343 (1975). [*Citado en pág. 3.*]

[19] SourceForge. Lucene.Net (formerly dotLucene) is the fastest open-source search engine for .NET [on-line] (May 2006). [visit. 20/02/2007] URL: `http://www.dotlucene.net`. [*Citado en págs. 1 y 11.*]

[20] Karen Sparck Jones. "Information Retrieval Experiments". Butterworth, London (1981). [*Citado en pág. 3.*]

[21] C. J. van Rijsbergen. "Information Retrieval". Dept. of Computer Science, University of Glasgow, Segunda edición (1979). [*Citado en pág. 2.*]

[22] Nivio Ziviani. Text operations. En "Modern Information Retrieval", capítulo 7, páginas 163–190. Addison-Wesley, Harlow, England (1999). [*Citado en pág. 4.*]