**Universidad de Las Ciencias Informáticas**
**Facultad 4**


# Trabajo de Diploma

**Título: Diseño e implementación de un Sistema de Almacenamiento y Recuperación de Información (SRI)**

**Optando por el Título de Ingeniero en Ciencias Informáticas**

# SearchEngine

**Autor**
Hilda Lilia Moreno González
**Tutor**
Lic. Willibert Peña Vega

Junio del 2007

# Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo al _____ de la Universidad de Las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____ .

_____
Hilda Lilia Moreno González.
Ciudad de La Habana, 26 de Junio, 2007.

_____
Lic. Willibert Peña Vega.
Ciudad de La Habana, 26 de Junio, 2007.

# Dedicatoria

A mis padres y a mis tíos Hildita y Frank
por su confianza y por estar junto a mí siempre que los necesito....

A mi familia
por contar siempre con su apoyo....

Y muy especial a mi abuelo Rafael
que aunque ya no se encuentre entre nosotros se que le hubiese gustado acompañarme en este momento.

# Agradecimientos

A mi novio por tener su apoyo en los momentos importantes y por ser ese alguien que al entrar en mi vida la ha cambiado por completo, ese alguien que me hace reír sin cesar, ese alguien que me ha hecho ver que en el mundo existen realmente cosas maravillosas....

Hilda Lilia Moreno González.
Ciudad de La Habana, 26 de Octubre, 2006.

# Resumen

El desarrollo y crecimiento masivo de las redes de computadoras y medios de almacenamiento a lo largo de los últimos años, ha motivado la aparición de un creciente interés por los sistemas de clasificación automática de documentos, los cuales realizan "la clasificación" basándose en el análisis del contenido del texto de los documentos que procesan.

En la Universidad de Las Ciencias Informáticas **(UCI)** en la actualidad, se hace cada vez más difícil encontrar la información que buscamos, ya que fluye el proceso de digitalización de los documentos, así como, el desarrollo de nuevas tecnologías de la información tanto en su creación, distribución, como en su acceso, facilitando estos recursos a un número ilimitado de usuarios.

La **UCI** no se encuentra excenta del problema de la localización de la información en la Intranet y es la razón de ser de este proyecto de tesis, diseñar e implementar un **SRI** para centralizar los esfuerzos de la búsqueda de contenidos en la Intranet Universitaria.

# Índice general

**1. Introducción al tema y fundamentación matemática** **2**
1.1. Hipótesis . . . . . 2
1.2. Objetivos . . . . . 2
1.3. Conceptos básicos . . . . . 3
1.4. Descripción del modelo de RI Vectorial . . . . . 4

**2. Descripción del sistema** **5**
2.1. API Lucene . . . . . 5
2.2. Componentes del SRI SearchEngine . . . . . 6
2.2.1. Base de datos documental . . . . . 6
2.2.2. Subsistema de consulta . . . . . 7
2.2.3. Subsistema de evaluación . . . . . 9
2.3. Evaluación del SRI SearchEngine . . . . . 9

**3. Implementación del SRI** **11**
3.1. Arquitectura . . . . . 11
3.2. Aplicación servidor . . . . . 14
3.3. Subsistema de *plugins* . . . . . 14
3.4. Aplicación web cliente . . . . . 15

**Conclusiones** **15**

**Recomendaciones** **17**

**Documentación de clases** **18**

**Bibliografía** **22**

# Capítulo 1

# Introducción al tema y fundamentación matemática

En este capítulo se plantea la hipótesis, se definen los objetivos de la tesis, además se presentarán un conjunto de conceptos que se han ido desarrollando en el campo de la Recuperación de Información **(RI)**, se abordará el concepto de **SRI** así como los distintos modelos en que se basan.

## 1.1. Hipótesis

¿Existirá en la **UCI** un **SRI** para almacenar y brindar una solución al problema de encontrar la información que necesitamos de manera eficiente?

## 1.2. Objetivos

### General

Diseñar e implementar un **SRI** para almacenar y manipular la información de manera eficiente en la Universidad.

### Específicos

- Realizar un detallado estudio del arte de las principales herramientas de **SRI** existentes en la actualidad.
- Analizar las posibles variantes de almacenamiento de información.
- Estudiar los principales modelos matemáticos de **RI**, sus ventajas y desventajas.
- Definir un modelo matemático por el cual se regirá el motor de búsqueda.
- Diseñar e implementar un modelo robusto de objetos donde esté centrada toda la lógica del **SRI**.

## 1.3. Conceptos básicos

### ¿Qué es un SRI?

**Baeza–Yates (1999):** Parte de la informática que estudia la recuperación de la información (no datos) de una colección de documentos escritos. Los documentos recuperados pueden satisfacer una necesidad de información de un usuario expresada normalmente en lenguaje natural.

**Korfhage (1997):** La localización y presentación a un usuario de información relevante a una necesidad de información expresada como una pregunta.

**Salton (1989):** Un sistema de recuperación de información procesa archivos de registros y peticiones de información e identifica y recupera de los archivos ciertos registros en respuesta a las peticiones de información.

### Clasificación de los modelos de RI

La propuesta más acertada hasta ahora en la literatura es la expresada por Baeza en [4] y reflejada por Dominich – consultar en [3] – en la Figura 1.1, donde se puede apreciar que están agrupados según el metamodelo[1] y sucesivamente el modelo en cuestión.

| Modelo | Descripción |
|---|---|
| Clásicos | Booleanos, Probabilísticos y basados en el Espacio Vectorial. |
| Alternativos | Basados en la Lógica Fuzzy. |
| Lógicos | Basados en la Lógica Formal. |
| Basados en la interactividad | Posibilidades de expansión del alcance de la búsqueda y uso de retroalimentación por relevancia. |
| Basados en la Inteligencia Artificial | Redes neuronales, bases de conocimiento, algoritmos genéticos y procesamiento de lenguaje natural. |

Figura 1.1: Clasificación de los **SRI**.

### Pasos para desarrollar un SRI

Para realizar un diseño de un **SRI** se debe utilizar un modelo, en el que se definirá cómo se obtienen las representaciones de los documentos y de las consultas. La estrategia para evaluar la relevancia de un documento respecto a una consulta, los métodos para establecer la relevancia u orden de los documentos de salida y los mecanismos que permiten una realimentación por parte del usuario.

[1] Agrupación de un conjunto de modelos matemáticos de **RI** según su orden cronológico en la historia.

## 1.4. Descripción del modelo de RI Vectorial

En este modelo los documentos están representados por vectores de términos y cada componente del vector $w_{ij}$ representa el peso del término $t_j$ presente en el documento $d_i$. De esta manera, la representación lógica de cada documento será un vector de pesos $d_i = (w_{i1}, w_{i2}, w_{i3}, \ldots, w_{in})$, donde $w_{ij}$ indicará el grado de relevancia de que el término $t_j$ este presente en el documento $d_i$, ver Figura 1.2. Este peso suele estar asociado a la frecuencia de aparición del término en el documento en cuestión.

| | $t_1$ | $t_2$ | $t_3$ | ... | $t_j$ | ... | $t_m$ |
|---|---|---|---|---|---|---|---|
| $d_1$ | $w_{11}$ | $w_{12}$ | $w_{13}$ | ... | $w_{1j}$ | ... | $w_{1m}$ |
| $d_2$ | $w_{21}$ | $w_{22}$ | $w_{23}$ | ... | $w_{2i}$ | ... | $w_{2m}$ |
| .. | .. | .. | .. | .. | .. | .. | .. |
| $d_i$ | $w_{i1}$ | $w_{i2}$ | $w_{i3}$ | ... | $w_{ij}$ | ... | $w_{im}$ |
| .. | .. | .. | .. | .. | .. | .. | .. |
| $d_n$ | $w_{n1}$ | $w_{n2}$ | $w_{n3}$ | ... | $w_{nj}$ | ... | $w_{nm}$ |

Figura 1.2: Matriz de pesos de términos en el modelo vectorial.

Una consulta se representa de igual manera que un documento, asignándole un vector de pesos asociados a los términos, representando así la importancia de los términos en la consulta: $q_k = (q_{k1}, q_{k2}, q_{k3}, \ldots, q_{kn})$ con $w_{iq} = (0{,}5 + \frac{0{,}5 \times tf_{iq}}{\max_l tf_{lq}}) \times idf_i$.

Se definen las siguientes propiedades para los términos:

$tf_{ij}$ = Es la frecuencia de aparición del término $t_j$ en el documento $d_i$.
$df_j$ = Indica el número de documentos en los que aparece el término $t_j$.

Y de aquí se define el peso $w_{ij}$ como sigue: $w_{ij} = tf_{ij} * idf_j$, donde $idf_j$ *es la frecuencia inversa de* $df_j$, o sea, $idf_j = \log N/df_j$, siendo N el número total de documentos en el sistema.

El modelo vectorial se basa en la suposición de que la proximidad relativa entre dos vectores es proporcional a la distancia semántica entre dos documentos. Esta es medida de diferentes maneras, pero la más difundida es la **medida del coseno** y sobre esta profundizaremos en el **Capítulo #2**.

# Capítulo 2

# Descripción del sistema

## 2.1. API Lucene

Lucene es una novedosa herramienta que permite tanto la indexación como la búsqueda de documentos. Creada bajo una metodología orientada a objetos e implementada completamente en Java[1], no se trata de una aplicación que pueda ser descargada, instalada y ejecutada sino de una API flexible, muy potente y realmente fácil de utilizar, a través de la cual se pueden añadir, con pocos esfuerzos de programación, capacidades de indexación y búsqueda a cualquier sistema que se esté desarrollando.

Originalmente escrita por Doug Cutting, en Septiembre de 2001, pasó a formar parte de la familia de código abierto de la fundación Jakarta.

Existen otras herramientas, a parte de Lucene, que permiten realizar la indexación y búsqueda de documentos pero dichas herramientas han sido optimizadas para usos concretos, lo que implica que el intentar adaptar dichas herramientas a un proyecto específico sea una tarea realmente difícil. La idea que engloba Lucene es completamente diferente, ya que su principal ventaja es su flexibilidad, que permite su utilización en cualquier sistema que lleve a cabo procesos de indexación.

A continuación se detallan algunas características que hacen de ella una herramienta flexible y adaptable:

**Indexación incremental vs indexación por lotes:** El término de indexación por lotes se utiliza para referirse a aquellos procesos de indexación, en los cuales, una vez que ha sido creado el índice para un conjunto de documentos, el intentar añadir algunos documentos nuevos es una tarea difícil por lo que se opta por reindexar todos los documentos. Sin embargo en la indexación incremental se pueden añadir documentos a un índice ya creado con anterioridad de forma fácil. Lucene soporta ambos tipos de indexación.

[1]Escrita originalmente en Java, se utilizó la versión para Net.

**Origen de datos:** Muchas herramientas de indexación sólo permiten indexar ficheros o páginas web, lo que supone un serio inconveniente cuando se tiene que indexar contenido almacenado en una base de datos. Lucene permite indexar tanto documentos y páginas web como el contenido procedente de una base de datos.

**Contenido etiquetado:** Algunas herramientas, tratan los documentos como simples flujos de palabras. Pero otras como Lucene permiten dividir el contenido de los documentos en campos y así poder realizar consultas con un mayor contenido semántico. Esto indica, que se pueden buscar términos en los distintos campos del documento concediéndole más importancia según el campo en el que aparezca. Por ejemplo, si se dividen los documentos en dos campos, título y contenido, puede concederse mayor importancia a aquellos documentos que contengan los términos de la búsqueda en el campo título.

**Técnica de indexación:** Existen palabras tales como a, unos, el, la ...etc. que añaden poco significado al índice, son palabras poco representativas del documento. Al eliminar estas palabras del índice se reduce considerablemente el tamaño del mismo así como el tiempo de indexación. Estas palabras están contenidas en lo que se denomina lista de parada, que es la técnica de indexación contemplada por Lucene.

**Concurrencia:** Lucene gestiona que varios usuarios puedan buscar en el índice de forma simultánea así como también que un usuario modifique el índice al mismo tiempo que otro lo consulta.

**Elección del idioma:** Tal y como ya se indicó con anterioridad Lucene trabaja con listas de parada, las cuales son proporcionadas por el desarrollador que está utilizando Lucene, esto permite escoger el idioma a utilizar.

## 2.2. Componentes del SRI SearchEngine

Un SRI está compuesto por tres componentes principales: ***la base de datos documental***, ***el subsistema de consulta*** y ***el mecanismo de emparejamiento o evaluación***. Las tres secciones siguientes están dedicadas a estudiar la composición de cada uno de ellos.

### 2.2.1. Base de datos documental

Aunque la evolución tecnológica ha propiciado la aparición de documentos multimedia y la variedad en cuanto a documentos se refiere está aumentando, tanto en soportes, como en el carácter de su contenido, se centró este trabajo en los que tienen naturaleza únicamente textual[2]. La arquitectura del sistema es flexible y extensible para añadir *plugins* para tratar nuevos formatos. Ahora la construcción de los vectores asociados a cada documento se realiza mediante el proceso de *indizado*, de la colección de documentos, ver Figura 2.1.

[2]Trataremos diferentes formatos (.pdf, .doc, .rtf, .ppt, .xls, .xml, .html, .txt).

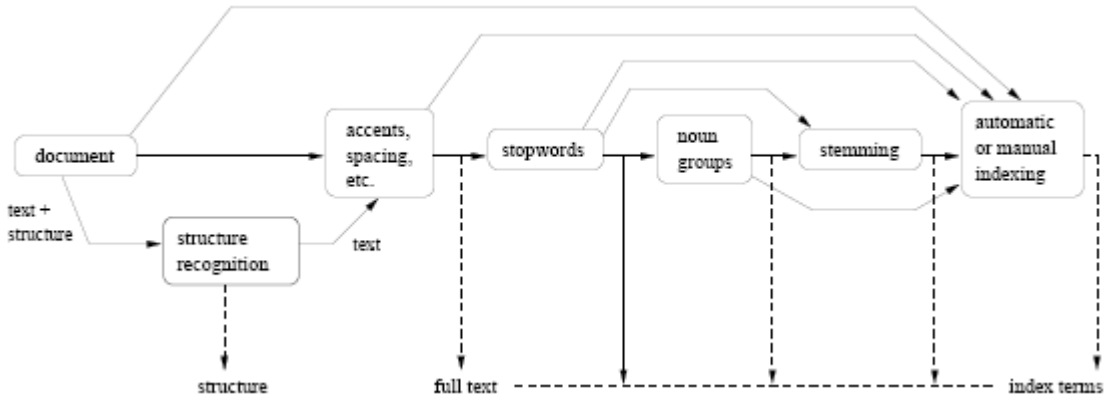


Figura 2.1: Proceso de indizado.

**Indización:** Creación de las estructuras de datos adecuadas para permitir un acceso eficiente y eficaz a los documentos.

Se diseñó la base de datos documental en función de carpetas físicas en el sistema de ficheros por la flexibilidad que esta desprende de poder elegir la manera de incluir los documentos[3], así en el sistema lo que importa es que se "introduzcan" los documentos en estas, sin importar la manera en cuestión. De hecho la recomendación de este trabajo es implementar un *spider* para analizar la red y así alimentar la base de datos documental y por consiguiente automáticamente actualizar el índice.

### 2.2.2. Subsistema de consulta

Como hemos indicado, en este modelo tanto las consultas como los documentos tienen la misma representación, es decir, vectores n–dimensionales, donde $n$ es el número de términos índice considerados. Cada una de las posiciones del vector contienen un peso $w_{t_{ij}}$, el cual indica la importancia relativa del término concreto de la consulta o del documento. Este peso es un número real positivo que en nuestro caso va a estar normalizado[4].

Cuando un usuario formula una pregunta, la mayoría de los pesos de la misma serán 0, con lo que bastará con proporcionar los términos con un peso distinto de 0 para poder definirla. El sistema se encargará de representar la consulta completa en forma de vector n–dimensional de modo automático.

Ahora mostraremos todos los elementos que pueden ser utilizados para formular una determinada *query*:

[3]La aplicación servidor es un servicio de windows que monitorea estas carpetas y actualiza el índice.
[4]Valor real positivo que alcanza valores en el intervalo [0,1].

**Soporta *wildcars:*** O sea patrones de búsquedas, estos son ? y *.

```
Ejemplo: doc?ment matchea con document, dociment y docum*
matchea con document, documentation.
```

**Soporta frases:** O sea búsquedas de una frase como término con “ ”.

```
Ejemplo: "important document".
```

**Permite repesado de los términos:** En el siguiente ejemplo por defecto el valor es 1 dentro de la multiplicación en el vector y le asignamos a important el valor 4. Le brindamos una mayor relevancia a dicho término.

```
Ejemplo: important^4 document.
```

**Permite todas las operaciones booleanas:** Tratamiento de agrupaciones de AND, OR, –[5].

**Permite búsquedas por campo de contenido etiquetado:** O sea buscar cosas por título, autor, etc.

```
Ejemplo: author:{Einstein}.
```

**Rango de valores:** Permite buscar entre rangos.

```
Ejemplo: date:{20050101 TO 20050201}, busca documentos
publicados entre las dos fechas.
```

**Permite aplicar búsquedas basadas en lógica *fuzzy:*** Esto es para utilizar directamente *thesaurus*.

```
Ejemplo: cerveza~, buscaría sinónimos, tales como, bucanero, cristal,
solo por citar dos.
```

[5]Símbolo para el operador NOT.

### 2.2.3. Subsistema de evaluación

El mecanismo de evaluación de los SRI "empareja" la consulta $Q$ contra la representación (el vector) asociado a cada documento de la base documental, $d_i \in D$, para obtener el grado de relevancia $RSV_i$ del documento $d_i$ con respecto a la consulta. El ***RSV*** toma un valor real que será tanto mayor cuanto más similares sean los documentos y las consultas.

En nuestro caso se aplicó el modelo de RI **Vectorial** con una pequeña modificación[6]. Existen diferentes funciones para medir la similitud entre documentos y consultas. Todas ellas están basadas en considerar ambos como puntos en un espacio n–dimensional, se utilizará la medida ampliamente difundida del coseno:

$$RSV(q,d) = \frac{\sum_{j=1}^{n} w_{q_j} * w_{d_j}}{\sqrt{\sum_{j=1}^{n} w_{q_j}^2 * \sum_{j=1}^{n} w_{d_j}^2}}$$

donde:

$$tf_{ij} = \frac{n_{ij}}{\max_l n_{lj}}$$

$$idf_i = \log \frac{N}{n_i}$$

$$w_{ij} = tf_{ij} \times idf_i$$

$$w_{iq} = (0{,}5 + \frac{0{,}5 \times tf_{iq}}{\max_l tf_{lq}}) \times idf_i$$

## 2.3. Evaluación del SRI SearchEngine

La evaluación de un **SRI** puede enfocarse desde dos puntos de vista, por una parte se tendrán una serie de medidas orientadas a analizar el acceso físico a los datos, y por otra, existen medidas que pretenden analizar la pertenencia o no del contenido. Seguidamente se muestran las medidas según Cleverdom.

**1:** La cobertura de una colección.

**2:** El intervalo de tiempo transcurrido en que el sistema recibe la consulta del usuario y presenta las respuestas.

**3:** La forma de presentación de los resultados de la búsqueda, la cual influye en la habilidad del usuario para utilizar la información recuperada.

**4:** El esfuerzo, intelectual o físico, requerido por el usuario en la formulación de las consultas, en el manejo de la búsqueda y en el proceso de examinar los resultados.

[6]Se ha realizado una extensión para discriminar índices de título, resumen y cuerpo del documento en cuestión.

**5:** La exhaustividad o habilidad del sistema para presentar todos los términos relevantes.

**6:** La precisión o habilidad del sistema para presentar solamente términos relevantes.

Según el autor las cuatro primeras medidas son fácilmente estimables e intuitivas, y las dos últimas, la *exhaustividad* y la *precisión*, son las que medirán verdaderamente la efectividad del sistema. En el sistema **SearchEngine** se satisface una reducción de aproximadamente el 30 % de la base de datos documental con muy buenos índices de precisión. La interfaz para presentar los resultados de una determinada búsqueda es amigable, sencilla y muy fácil de utilizar.

# Capítulo 3

# Implementación del SRI

Para diseñar e implementar el **SRI**, este se programó de manera modular con la menor cantidad de dependencias, con el objetivo de brindar en la solución final mayor reusabilidad y escalabilidad.

## 3.1. Arquitectura

El flujo de operaciones del **SRI** se muestra a continuación en la Figura 3.1.

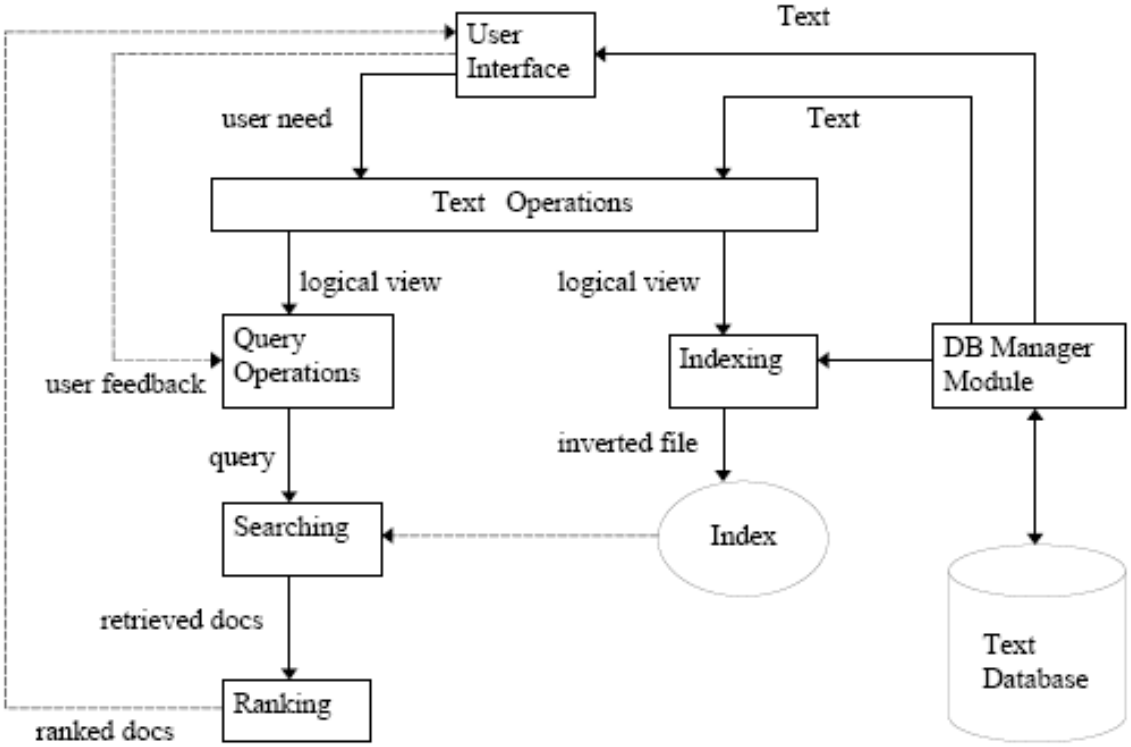


Figura 3.1: Flujo del **SRI**.

Aquí se puede ver que dado una necesidad de información del usuario en lenguaje natural se procesa mediante operaciones de texto,[1]los términos resultantes que identifican a la *query* son inmediatamente cargados según la lógica del sistema[2] y se utiliza el índice para devolver el conjunto de documentos relevantes utilizando la norma del coseno vectorial. Finalmente son presentados estos en la aplicación Web.

La interfaz es una aplicación Web desarrollada sobre .Net y mantiene el típico "front–end" de un buscador Web – ver Figura 3.2 – y la misma se basa en el módulo del subsistema de consultas.


Figura 3.2: Aplicación Web.

[1]Se le aplica filtrado contra la **StopWordList** y acto seguido **Stemming** para reducir cada término resultante a la raíz.

[2]Se forma el documento *query* con un formato vectorial idéntico a la representación de los documentos.

En la solución se detallan tres aspectos fundamentales:

**El software final es completamente configurable:** Todos los parámetros se almacenan en ficheros de configuración basados en XML[3]. En la Figura 3.3 se muestra el fichero config.xml donde se configura la ruta de acceso tanto de la base de datos documental como del índice del sistema **SearchEngine**.

**El servicio de indexación es concurrente y flexible:** El sistema gestiona que varios usuarios puedan buscar en el índice de forma simultánea, así como, también que un usuario modifique el índice al mismo tiempo que otro lo consulta. La flexibilidad del sistema **SearchEngine** viene dada por un sistema de *plugins* que se incorporó al mismo, siguiendo la idea de separar la lógica del documento en cuestión, del formato. Así pues, se filtra dado la interfaz **IFilter** el tipo de fichero asociado a las extensiones y lo más cómodo es que mediante esta vía se puede configurar el sistema tanto por implementaciones nuestras, como por implementaciones de terceros[4].

**La aplicación web es sencilla pero potente:** Intranet Search Engine consta de dos plantillas de tipo buscador: **Search.aspx**, ver Figura 3.2 y **AdvancedSearch.aspx**, ver Figura 3.6, la primera es un formulario de un buscador clásico y la segunda es el típico formulario de asistente[5].

```
<?xml version="1.0" encoding="utf-8" ?>
- <IndexConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <Watch>
      <FolderPath>C:\Documents and Settings\Administrator\My
        Documents\Latex Projects\src\Seekafile.Web\DocDirectory</FolderPath>
    </Watch>
    <IndexPath>C:\Documents and Settings\Administrator\My Documents\Latex
      Projects\src\Seekafile.Web\Index</IndexPath>
  </IndexConfig>
```

Figura 3.3: Relación entre el índice y la base de datos documental.

[3]Tecnología estándar para el intercambio de información estructurada entre diferentes plataformas.

[4]Se implementó un filtro para los formatos enriquecidos y se le asoció la extensión .rtf, se bajó el filtro para extraer el texto de los ficheros .pdf, ver Figura 3.5.

[5]Donde se brinda una interfaz completa de todo el subsistema de consulta.

## 3.2. Aplicación servidor

La aplicación servidor sigue el siguiente esquema:

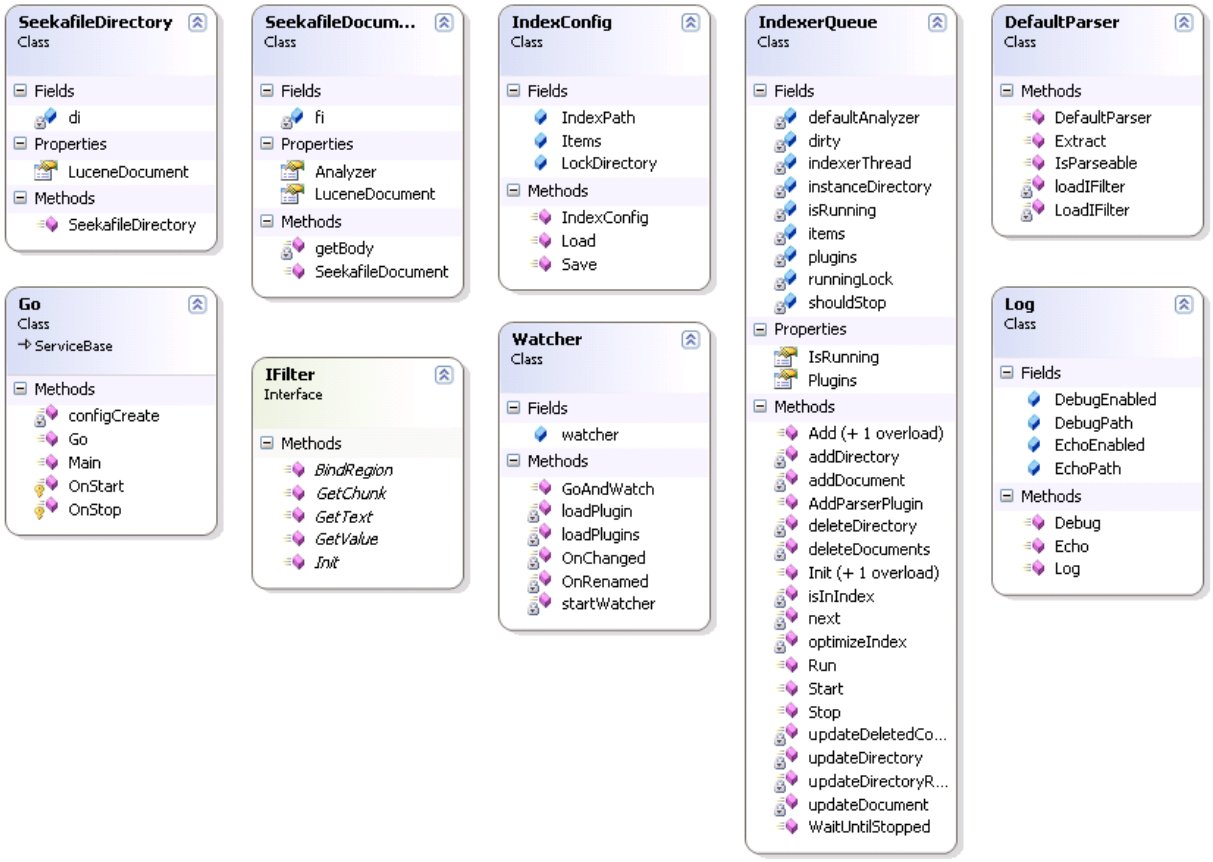


Figura 3.4: Esquema de la aplicación servidor.

El núcleo del servidor es la clase **Go** que implementa un servicio de Windows que se encarga de cargar la configuración – ver Figura 3.3 – de la base de datos documental y el índice relacionado. La misma utiliza la cola **IndexerQueue** para atender las solicitudes[6] para actualizar el índice. Se programaron otras utilidades como un sistema de logging para registrar todas las acciones en el sistema[7].

## 3.3. Subsistema de *plugins*

El subsistema de *plugins* se basa en la interfaz **IParserPlugin**, ver Figura 3.5. La idea es proporcionar un nivel más alto de abstracción, separar lo físico de lo lógico y tratar el documento como un contenido etiquetado abstrayéndonos del formato en cuestión.

[6]Se tratan las agregaciones, las actualizaciones y las eliminaciones de nuevos ficheros a la base de datos documental mediante el filtro asociado y se programó en la clase **Watcher**.

[7]Esto se codificó en la clase **Log**.

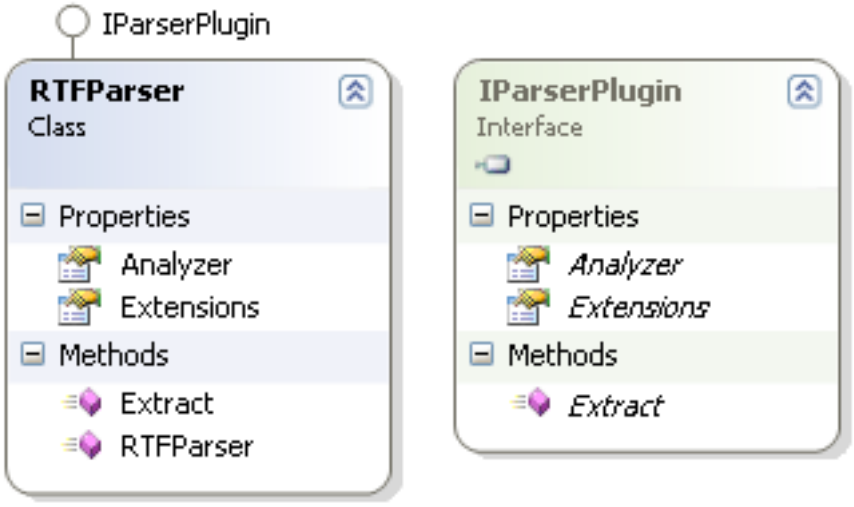


Figura 3.5: Esquema del subsistema de *plugins*.

## 3.4. Aplicación web cliente

La aplicación Web es un típico buscador programado bajo la tecnología ASP.NET que encapsula todas las funcionalidades del subsistema de consulta. La misma está dividida en dos formularios: Search.aspx que se encarga de toda la lógica de recuperación de la información y AdvancedSearch.aspx que es un asistente para *parsear* y devolver la petición al primero para operar a recuperar la información relevante a la *query*. La misma cumple con las especificaciones de la W3C y soporta un mecanismo multicultural, o sea, se captura la cultura proveniente del *browser* para desplegar en el idioma correspondiente la interfaz de usuario.

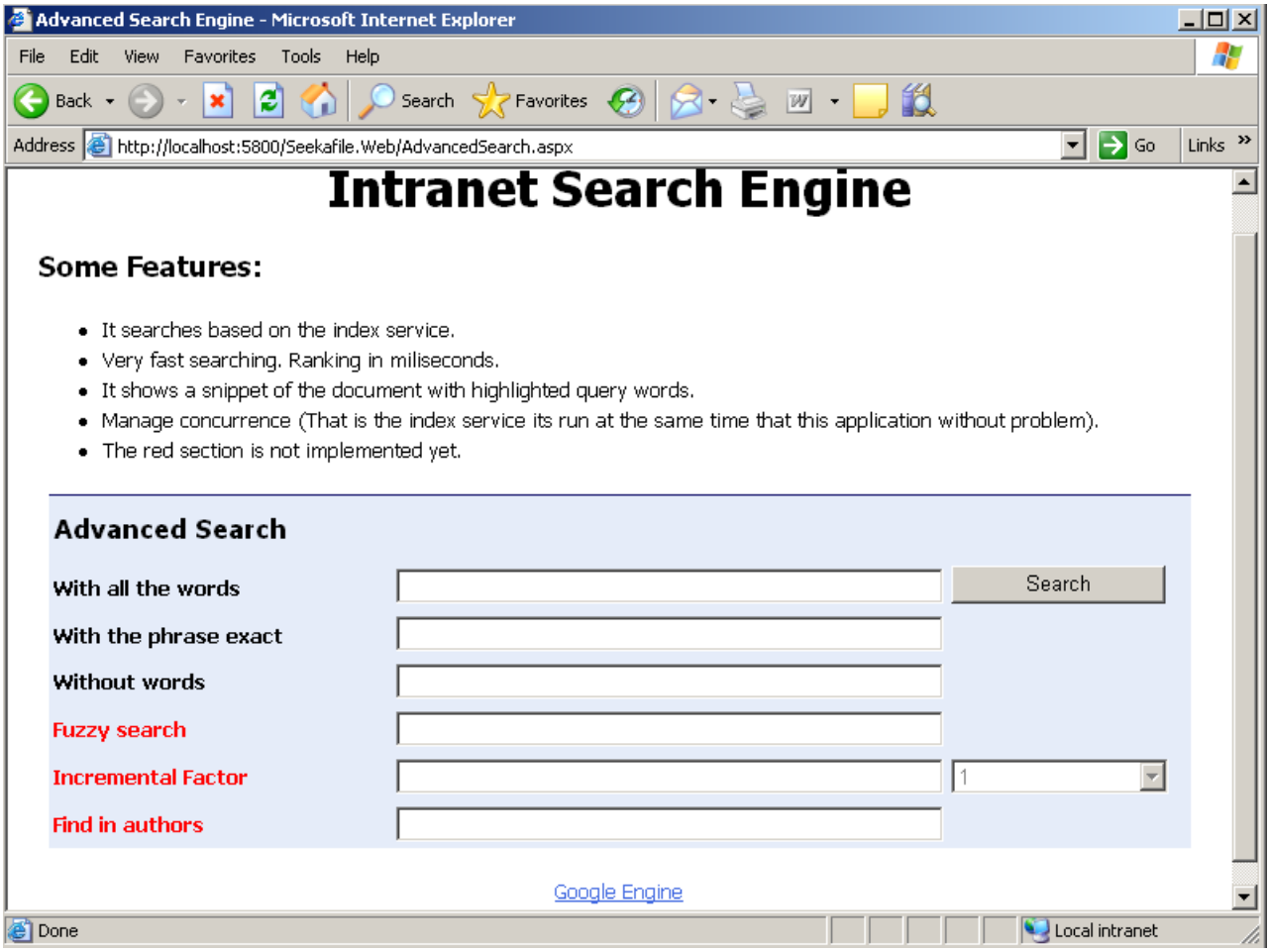


Figura 3.6: Formulario asistente del buscador.

# Conclusiones

El resultado de este trabajo ha sido la construcción del **SearchEngine**, que brinda una interfaz con las facilidades para cumplir con los requerimientos planteados en la etapa de análisis del software inicial, que no fue otro, a grandes rasgos, que modelar e implementar un sistema de recuperación de información acorde a las necesidades en la intranet universitaria de la **UCI**.

# Recomendaciones

Se espera que se realice un despliegue de la aplicación para hacerla funcional dentro de nuestra intranet universitaria, y con ello, hacer de la búsqueda de la información que disponemos en la misma y nos sea relevante a nuestros intereses, una operación rápida y eficiente.

Se recomienda trabajar en función de diseñar e implementar un *spider* para recopilar toda la información contenida en nuestra web universitaria y actualizarla en nuestra base de datos documental con fin de poner en producción nuestra aplicación, así como, trabajar en función de un módulo de retroalimentación vía interactiva, desde el usuario, para ganar en precisión de la lista de los documentos recuperados por el sistema.

# Documentación de clases

## Server

```
public class IndexConfig
{
    public static IndexConfig Load(string location);
    public void Save(string location);
}

public class IndexerQueue
{
    public static Hashtable Plugins
    {
        get;
    }
    public static bool IsRunning
    {
        get;
    }
    public static void Add(string path);
    public static void Add(string path, bool startIndexing);
    public static void Stop();
    public static void WaitUntilStopped();
    public static void Start();
    public static void Run();
    public static void Init(Directory directory);
    public static void Init(Directory directory, Analyzer analyzer);
    public static void AddParserPlugin(IParserPlugin plugin);
}
```

```
public class SeekafileDirectory
{
    public SeekafileDirectory(DirectoryInfo fi);
    public Document LuceneDocument
    {
        get;
    }
}

public class SeekafileDocument
{
    public SeekafileDocument(FileInfo fi);
    public Document LuceneDocument
    {
        get;
    }
    public Analyzer Analyzer
    {
        get;
    }
    private string getBody();
}

public class Watcher
{
    public void GoAndWatch(IndexConfig cfg);
}
```

# Plugins

```
public interface IParserPlugin
{

    string[] Extensions {get;}
    string Extract(string path);

    Analyzer Analyzer
    {
        get;
    }
}
```

```
public class RTFParser : IParserPlugin
{
    public string Extract(string path);

    public Lucene.Net.Analysis.Analyzer Analyzer
    {
        get;
    }

    public string[] Extensions
    {
        get;
    }
}
```

## Aplicación Web

```
public partial class Search : System.Web.UI.Page
{
    private void search();
}

public partial class AdvancedSearch : System.Web.UI.Page
{
    protected void ButtonSearch_Click(object sender, System.EventArgs e);

    protected string GetAllWordQuery(string _info);

    protected string GetWithoutWordQuery(string _info);

    protected string GetPhraseQuery(string _info);

    protected string GetFuzzySearch(string _info);
}
```

# Bibliografía


[1] B. Dervin, M. Nilan. *Information needs and uses.* Annual Review of Information Science and Technology, 1986.

[2] D. Ellis. *The physical and cognitve paradigms in information retrieval research.* Journal of Documentation, 1992.

[3] Dominich. *A unified mathematical definition of classical information retrieval.* Journal of the American Society for Information science, 2000.

[4] R. Baeza-Yates B. Ribeiro–Net. *Modern information retrieval.* Addison Wesley, 1999.