UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

TESINA DE LICENCIATURA

OBTENCIÓN AUTOMÁTICA DE METADATOS DE PÁGINAS WEB PARA MEJORAR LA ORDENACIÓN DE LOS RESULTADOS DE UNA BÚSQUEDA

Autor: Martín Burgués

Directora: M. Sc. Cristina Bender

Tesina presentada para obtener el título de
Licenciado en Ciencias de la Computación

Rosario, Argentina, 2009

Índice general

Resumen

Actualmente existe en Internet gran cantidad de información, la cual es recopilada e indexada por los buscadores. Esta tarea se perfecciona a cada momento, los motores de los buscadores son aplicaciones extremadamente complejas que evolucionan continuamente para acercarse más a los intereses de los usuarios.

La información está muy diversificada en páginas Web que carecen de estructura. Esta carencia implica que las computadoras no puedan comprender y acceder al significado de las páginas, por lo tanto resulta muy difícil la tarea de encontrar contenidos adecuados a los requerimientos de los usuarios.

Un área en la cual este problema es particularmente importante es la búsqueda de cursos por Internet, ya sea en sitios dedicados a la recopilación de cursos o en la Web en general. Esto es así porque cuando un usuario busca un curso necesita que el mismo cumpla ciertas características y se adapte a su perfil y necesidades específicas, lo cual hace que los resultados devueltos por los buscadores comunes no sean los más adecuados.

En virtud de armar espacios para recopilar contenidos de cursos o tutoriales Web existen sitios que acopian esta información. Para que la información recolectada contenga sentido se pide al usuario que indique aspectos básicos del material que se sube. Esta tarea delega prácticamente toda la responsabilidad de indexación a la información brindada por el autor. Estos sitios son un ejemplo claro de los beneficios de la estructura en los contenidos Web, dado que al tener todas las páginas características homogéneas y estar bien definida su estructura, es más fácil buscar de manera “inteligente” en ellos.

El agregado de estructura a los datos desestructurados que conviven en la Web es un trabajo que se viene desarrollando desde hace varios años. La falta de estructura provoca que la información sea más difícil de localizar y muchas veces sea inaccesible.

En este trabajo se presenta una propuesta para la extracción de información del contenido de las páginas Web para mejorar las formas de clasificación utilizadas actualmente. Dada una consulta de un usuario y ciertas elecciones que indican las preferencias del mismo, la arquitectura propuesta devuelve las páginas encontradas de acuerdo a las preferencias y a la configuración cargada, otorgándole además información de las características de estas páginas que puede ser utilizada en un análisis posterior de las mismas. Las búsquedas pueden hacerse en la Web en general, o en sitios específicos, como ser repositorios de cursos indicados por el usuario. Este enfoque le da contenido semántico a las búsquedas. Se propone crear un árbol XML con el contenido recolectado de las páginas y utilizar el mismo para darle mayor estructura a los resultados devueltos.

En resumen, lo que se busca con este trabajo es hacer las búsquedas en Internet más específicas y personales, añadiendo valor al contenido de cada página encontrada.

Un aspecto importante que se tuvo en cuenta en este trabajo es el cálculo de índices de legibilidad sobre el texto de cada página. Esto brinda una información heurística del nivel de dificultad de los contenidos. Estos índices se utilizan en otros países para clasificar textos académicos y para determinar los tiempos que requerirán los alumnos para comprender un texto, y son especialmente útiles en la búsqueda personalizada de material académico.

Finalmente, se mostrará con diversas experimentaciones que el prototipo implementado es una herramienta eficaz para ordenar resultados obtenidos en la Web según preferencias de los usuarios, y que realiza la extracción de características de las páginas de manera totalmente confiable. Además se dejan sentadas las bases para experimentaciones futuras.

Capítulo 1.

Introducción

1.1. Motivación

El motivo principal que me lleva a desarrollar este trabajo es mejorar la utilización de información específica en las búsquedas por Internet, para que los resultados obtenidos en dichas búsquedas sean más efectivos y personalizados, y para ayudar a los usuarios a obtener información más adecuada del tema que están buscando.

La búsqueda personalizada de datos por Internet según los requerimientos del usuario es un área de gran crecimiento en los últimos años, principalmente debido al continuo incremento de la cantidad de información existente en la Web, que dificulta la posibilidad de encontrar datos y páginas específicas, ya que al realizar una búsqueda es imposible saber cuál de las miles de páginas retornadas por los buscadores se ajusta más a las preferencias e intereses del usuario.

Una de las principales razones que me motivaron a realizar este trabajo es desarrollar un sistema para ser aplicado en la búsqueda de páginas con contenido académico, principalmente cursos y tutoriales.

En la Web hay gran cantidad de repositorios de cursos, como por ejemplo MERLOT[*], que se clasifican con los datos que da el autor de los cursos al momento de indexarlos.

La herramienta propuesta en esta tesina le da mayor eficacia a las búsquedas en Internet, agregando información para la indexación automática y extrayendo datos inherentes a las características específicas de cada página encontrada. Esto le da contenido semántico a la búsqueda, y por lo tanto se consigue mayor cantidad de información relevante para la clasificación y localización de los datos específicos que cada usuario buscó. Así mismo, una vez recuperados los contenidos específicos de las páginas, se utiliza esta información para realizar una ordenación de las mismas que sea adecuada a las preferencias del usuario.

A pesar de que esta herramienta fue desarrollada teniendo como referencia la búsqueda de cursos por Internet, podría ser fácilmente extensible a búsquedas en otras áreas, realizando algunas modificaciones pertinentes al área de aplicación.

1.2. Problemática

Un gran problema que surge con la búsqueda de información es la enorme cantidad de documentos disponibles en formato electrónico, lo que hace imposible su análisis total. Lenguajes como XML, RDF[†] u OWL[‡] facilitan el procesamiento automático de la información, pero aún así se necesita un ser humano para realizar las tareas de acceso, extracción e interpretación de la misma. Una posible solución para alcanzar la automatización de estas tareas sería darle mayor nivel semántico a las consultas para reconocer más precisamente los requerimientos del usuario. El tener un perfil definido del usuario que busca determinada información actúa como un filtro frente a la enorme cantidad de datos que devuelve una búsqueda en un buscador cualquiera, por ejemplo Google. Este filtro debe retornar idealmente los resultados que se adaptan al perfil y/o la solicitud del usuario.

---

[*] http://www.merlot.org

[†] RDF: del inglés *Resource Description Framework*, lenguaje de descripción de recursos.

[‡] OWL: del inglés *Ontology Web Language*, lenguaje para compartir datos usando ontologías.

Uno de los grandes desafíos de los sistemas de búsqueda y extracción de información es obtener los datos que contienen las características de una página. Por ejemplo en el caso de páginas de cursos los datos importantes serían los referentes al tema del curso, el idioma, el autor, los niveles intelectuales requeridos para entender el texto, etc.

Es posible obtener información de las características y metadatos de las páginas Web, y agregarla a las páginas como datos organizados al devolverlas como resultados de la búsqueda.

En la Web existen sitios que se dedican a recopilar material publicado directamente por el autor de las páginas, en los que el autor mismo anexa información adicional del material para clasificarlo en las bases de datos del sitio. A éstas características es posible adicionarle información obtenida del material automáticamente para mejorar la clasificación. Esto hace que el acceso a la información sea más satisfactorio para el usuario.

Uno de los objetivos de este trabajo es tratar de conseguir características específicas del contenido de las páginas o documentos recuperados por los buscadores y usar esta información para devolverle al usuario las páginas que más se acerquen a sus requerimientos.

Actualmente, la Web se ha convertido en un repositorio de información que poco a poco va relegando la intervención humana en pos de la clasificación automática, por lo que se están generando muchas herramientas para tal fin. En este trabajo se propone una de ellas.

1.3. Problemas con metadatos

Uno de los principales inconvenientes para tomar la información relevante de las páginas Web es que no hay estándares concretos respecto de ciertas características que las mismas poseen, principalmente las etiquetas `<Meta>` de los encabezados de los HTML, con lo cual no es posible decidir con certeza de qué trata la página o qué tipo de información contiene.

Otra dificultad que aparece relacionada a las etiquetas <Meta> es que las palabras que aparecen en estos campos en las páginas HTML muchas veces no describen el contenido real de las páginas. Al no haber ninguna organización que controle que las palabras descriptivas de los campos <Meta> correspondan al verdadero contenido de las páginas Web, puede aparecer información falsa en las mismas, como ser palabras populares o atractivas para engañar a los buscadores. Para resolver este tipo de inconvenientes la Web necesitará disponer de mecanismos de confianza y credibilidad, los cuales pueden ser dados por firmas y certificados digitales que protejan a los usuarios de usos malintencionados de la información.

1.4. Problemas con la estructura HTML

El principal problema para hacer una extracción efectiva de datos de una página es que los documentos HTML no están perfectamente estructurados. A raíz de esto es muy difícil realizar un parser que sea capaz de extraer completamente la información. Una consecuencia de este problema es que los motores de los navegadores son extremadamente complejos. Por tal motivo se está trabajando en estandarizar un nuevo lenguaje para las páginas Web llamado XHTML (*véase sección 2.5*), basado en el estándar XML de la W3C.

La idea fundamental de este nuevo estándar en su versión 1.0 es que para todo tag que se abre debe existir un tag que represente su cierre. Además no se pueden mezclar apertura y cierre de tags diferentes, los cuales sí pueden estar anidados. Esta es la base de la representación de un árbol que se puede conseguir con los documentos XML. El hecho de llevar un HTML a un formato más rígido como lo es el XHTML, es que la representación de las páginas como árboles permite algoritmos recursivos para renderizado y análisis de código más eficientes y con menor probabilidad de errores.

1.5. Propuestas de solución

Como solución a los problemas mencionados en esta tesina se presenta un prototipo que le otorga al usuario una página Web dinámica en la cual se puede ingresar la consulta a realizar en buscadores estándar, (en este caso se decidió usar Google, pero cualquier buscador podría ser utilizado) y explicitar sus preferencias para la búsqueda y el ordenamiento de las páginas devueltas. Una vez hecho esto, se analizan los resultados en varias etapas.

La primera etapa consiste en extraer características relevantes de las páginas Web, como ser meta tags, idioma, tamaños, cantidad de links, cantidad de caracteres, sílabas, palabras y oraciones, etc. Esta extracción se hace a través de diversos mecanismos de exploración de la página que se detallarán en los siguientes capítulos.

Como segunda etapa se evalúan los parámetros obtenidos y se realizan cálculos entre ellos que proporcionan índices tales como relación entre imágenes y texto, o diferentes indicadores de legibilidad.

En la última etapa se estructura toda la información relevada en un árbol representado en un XML, y se la ordena usando las características indicadas por el usuario.

1.6. Estado del arte

Se utilizan diversos métodos con el objetivo de solucionar el creciente problema de buscar y extraer información específica de Internet sin tener que leer miles de documentos y decidir cuáles son los más adecuados. Éstos métodos van desde la concordancia de palabras a técnicas basadas en la popularidad de los sitios, pero para la mayoría de los usuarios que buscan contenido específico este tipo de técnicas no basta. Los buscadores actuales realizan las búsquedas de información con más o menos suerte, mediante palabras clave que aparecen en el código HTML de las páginas Web dispersas en Internet.

En los últimos años algunas empresas están realizando anotaciones de datos introducidas dentro del código HTML con el objetivo de aliviar este problema,

siguiendo algún esquema de anotación común, normalmente basado en XML [Lozano Tello, 2001].

Muchos grupos de investigación de recuperación de la información se enfocaron en el problema de extraer datos estructurados de documentos HTML. Gran parte de la investigación se hace en el contexto de las bases de datos, trasladando consultas que se hacen en bases de datos a consultas en la Web. El sistema ANDES [Myllymaki, 2002] usa XML para la extracción de datos Web y es utilizado en sistemas de extracción de datos en IBM. El enfoque de este sistema está orientado a procesos por lotes (batch-oriented): utiliza un crawler* sobre los sitios Web objetivo, extrae los datos estructurados y las características de un dominio específico y resuelve datos conflictivos o faltantes, para luego hacer que los mismos estén disponibles en aplicaciones de bases de datos locales. ANDES es un framework que mezcla tecnología crawler con tecnología de extracción de datos basada en XML. Está basado en XHTML y XSLT.

Mientras que el software ANDES extrae los datos estructurados de las páginas que recorre con su propio crawler, el sistema desarrollado en esta tesina se basa en los resultados devueltos por crawlers existentes, como Google, luego ejecuta extracción de datos estructurados y no estructurados, como los indicadores de legibilidad, que se calculan sobre el texto mismo de la página, siendo éste un proceso heurístico fuera de toda estructura que pueda contener la página.

Una gran mayoría de los proyectos de investigación desarrollados hasta el momento para la extracción de información de Internet utiliza ontologías. Por ejemplo en [Embley, 1999] se aplica un algoritmo que usa modelado conceptual para extraer y clasificar datos automáticamente. El algoritmo está basado en una ontología que describe los datos de interés, incluyendo relaciones, apariencia léxica, y palabras claves del contexto. Haciendo un análisis de la ontología genera automáticamente un esquema de base de datos y reconocedores para constantes y palabras clave, luego invoca rutinas para reconocer y extraer datos de documentos sin estructura y

* Los crawlers, también llamados soft bots, son los recolectores de información que utilizan los buscadores de Internet para generar sus índices.

estructurarlos de acuerdo al esquema de base de datos generado. A diferencia del sistema propuesto en esta tesina, este algoritmo no usa tags HTML para extraer información sino los aciertos (hints) provistos por una ontología.

En otros trabajos se emplean agentes inteligentes y otras técnicas de Inteligencia Artificial asociadas al uso de ontologías. Un ejemplo puede encontrarse en [Ponce, 2006]. En este trabajo se utiliza una ontología cuyos ejemplares son enlaces a páginas Web de un dominio específico sobre el cual fue construida la ontología, donde los usuarios puedan encontrar la información que necesitan. Se construye la ontología y se realiza clasificación automática recurriendo a técnicas de modelado de espacio vectorial (para manejar los documentos y las palabras que estos contienen mediante una tabla, asignándoles pesos a cada palabra) y de aprendizaje supervisado (Naive Bayes, k-vecinos, etc.).

De la misma manera, el proyecto ARIADNE [Ashish, 1997] se centra en el desarrollo de herramientas para la construcción de agentes inteligentes, similares a los wrappers[*] para extraer, consultar e integrar datos procedentes de Internet. Permite hacer búsquedas en la Web como si se tratara de una base de datos convencional. Se estructura la fuente (identifica las secciones que interesan de una página), luego se construye un parser para analizar la estructura generada y se añaden capacidades de comunicación entre el wrapper, el integrador y las fuentes Web.

En [Tramullas, 1999] se usan ontologías en el campo de los agentes de software para recuperación de información en Internet. El contenido de los documentos encontrados debe representarse usando una ontología y se realizan las tareas de descubrimiento y selección de recursos de información mediante soft bots.

Para la generación de estas ontologías se utilizan distintos recursos, como por ejemplo el que se presenta en la tesis [Zhou, 2002]. En esa tesis se propone un sistema semiautomático de generación de ontologías. En el proceso de generación de las mismas, el usuario define un formulario y recolecta un pequeño número de páginas en diferentes sitios de las cuales obtendrá datos para rellenar el formulario.

[*] Wrappers: Programas que visitan páginas Web y recogen información de acuerdo a algunos criterios generales.

Luego el sistema genera la ontología de extracción basada en la información de la muestra de páginas, los formularios y algún conocimiento previo, tal como tesauros, patrones de extracción de datos, etc.

El problema que todos estos sistemas presentan es que en Internet no se ha extendido la utilización de ontologías. Además los documentos en Internet no son homogéneos como necesitan este tipo de programas, por lo que se hace necesario incorporar una ontología exclusivamente dedicada al tipo de documento. El sistema propuesto en este trabajo intenta solucionar estos problemas aplicando una solución más general que el uso de ontologías: usar los metadatos presentes en cada vez más páginas Web y nuevos metadatos generados a partir del texto contenido en la página para la generación de indicadores e índices de legibilidad y nivel de dificultad de comprensión del texto.

Otros métodos muy diferentes son usados en [Grumbach, 1999] y [Mecca, 1999], en el sistema ARANEUS. Se propone una herramienta para el manejo de datos procedentes de fuentes Web. Se utiliza un lenguaje llamado EDITOR, basado en editores de texto. Utiliza operaciones "search" para buscar y seleccionar fragmentos de texto dentro de un documento y "cut & paste" para reorganizar el texto seleccionado de acuerdo con la consulta realizada.

Un programa que presenta características similares al desarrollado en esta tesina, pero que sólo se encarga de recuperar los datos que el usuario marca en la página Web es el Content Extractor 1.0[*].

Este programa es una herramienta de data mining para extraer datos de páginas Web y guardarlos de forma estructurada. La extracción se realiza de forma manual. Recolecta información de páginas asociadas a una fuente específica, pero no las devuelve ordenadas según las preferencias del usuario.

[*] http://code.google.com/p/content-extractor

1.7. Estructura del trabajo

Este trabajo está dividido en cinco capítulos, en el primero se detalla la motivación que me llevó a realizar la investigación, los problemas existentes, una propuesta de la solución presentada y la recopilación del actual estado del arte en la materia.

En el segundo capítulo se despliegan los conceptos teóricos en los que se basa todo el desarrollo de la investigación. Entre los más importantes están las estructuras XML con toda su maquinaria para manipularla, las expresiones regulares como herramienta sumamente flexible para extraer información de los textos analizados y algunos índices que se usan actualmente para determinar la legibilidad de los documentos, estudiados por científicos del área cognitiva y lengua.

En el tercer capítulo se despliega por completo y en detalle la arquitectura y la implementación del prototipo que se originó como resultado de la investigación, y la metodología utilizada para su desarrollo.

El cuarto capítulo muestra un ejemplo de aplicación en la búsqueda de cursos y ejemplifica diversas formas de experimentación para corroborar el buen funcionamiento de los sistemas del prototipo dedicados a extracción de información y clasificación de las páginas encontradas según los requerimientos del usuario.

Finalmente, en el quinto y último capítulo se detallan las conclusiones obtenidas a partir del desarrollo realizado y aplicaciones futuras que se puedan desprender de la investigación realizada en esta tesina.

Capítulo 2.

Conceptos teóricos

En este capítulo se presentan algunas nociones básicas respecto a las herramientas utilizadas para el desarrollo de la investigación. Entre ellas se destacan las herramientas para extracción de información y la teoría de Anotaciones Web como base para darle estructura a las páginas consultadas. También se presenta una introducción al lenguaje de etiquetas XML usado para representar la información extraída y los lenguajes de consulta sobre XML utilizados: XQuery y XPath.

Se hace una introducción a las expresiones regulares que fueron utilizadas como motor de extracción de texto en todo el trabajo. Finalmente se detallan varios algoritmos utilizados para medir índices de legibilidad en textos.

2.1. Extracción de Información

Extracción de Información (EI, en inglés *information extraction*) es un término que se utiliza para referirse a la actividad de extraer automáticamente determinados tipos de información previamente especificados de textos de lenguaje natural. El objetivo de esta técnica es extraer conocimiento estructurado, dependiente del contexto, de la información existente (que generalmente consiste en texto no

estructurado), con el fin de mejorar el uso y la reutilización de esta información. Es decir, encontrar y enlazar la información relevante mientras se ignora la extraña e irrelevante [Cowie, 1996].

[Cunningham, 1999] define la extracción de información como un proceso que toma los textos como entrada y que produce formatos fijos, datos no ambiguos, como salida. La extracción de información también puede verse como la actividad de poblar una fuente estructurada de información desde una fuente de información no estructurada o de texto libre. Esta fuente estructurada de información (que puede ser una base de datos) se usa entonces para otros propósitos: para la búsqueda o el análisis usando consultas convencionales de bases de datos o técnicas de minería de datos, para generar resúmenes o informes, para construir índices en los textos fuente, etc.

Algunos autores consideran a la extracción de información como una etapa posterior de la recuperación de información. La principal diferencia entre ambas radicaría en que mientras que la primera proporciona la información específica que interesa, la segunda proporciona los textos en que aparece dicha información [Téllez Valero, 2005]. Sin embargo, algunas nuevas tecnologías tratan de superar las diferencias y utilizar ambas técnicas en conjunto, por ejemplo en la generación de wrappers para Internet usados para extraer información desde documentos HTML como se propone en este trabajo.

2.2. Metadatos

Etimológicamente, metadato significa “dato sobre dato”. Ejemplos de metadatos pueden ser los códigos de barra, que describen datos sobre un producto, por ejemplo precio, código, etc.; el catálogo de una editorial, que describe sus libros y revistas; las palabras de resumen de un artículo de una revista científica, el esquema de una base de datos, etc.

En general, los metadatos describen las características de los datos en sí, son, como lo dice su nombre, datos sobre los datos. En otras palabras, enriquecen semánticamente los datos haciendo más fácil su interpretación. Una de las

principales funciones de los metadatos es su uso para mejorar la búsqueda y recuperación de la información.

En las páginas Web, se utilizan metatags, que son etiquetas HTML que se incorporan en los encabezados de estas páginas. Su función es introducir metadatos de referencia sobre una determinada página Web, como ser autor, título, descripción y palabras clave, lo que ayudará a su ubicación en los motores de búsqueda. Los metatags son una forma de definir las páginas Web para el mundo. Otras funciones de los metatags podrían ser especificar que una página posee derechos de autor, con qué frecuencia la página será visitada por los motores de búsqueda, etc. Los metatags pueden dar muchos datos útiles para definir las características de una página.

Hay dos tipos de metatags:

- **HTTP-EQUIV:** Estas etiquetas son el equivalente de las cabeceras http. Al hacer clic en un hipervínculo "`http://`" se está pidiendo una página que se transferirá a su navegador utilizando el protocolo HTTP. Lo que sucede es lo siguiente.

    1. El servidor Web se asegura de que la página exista
    2. Se devuelve una cabecera HTTP bloque que contiene información sobre la página
    3. Se envía la página en sí.

    Los metatags HTTP-EQUIV definen la información adicional que se envía al navegador en la cabecera HTTP a nivel de la página. Esto le da al creador del sitio Web control adicional sobre estos datos.

- **NOMBRE:** El atributo NAME se utiliza para definir la información a la que se hace referencia fuera del documento. Esto incluye los datos transmitidos a los motores de búsqueda y directorios, spiders y otras entidades.

Los metatags se colocan entre los tags `<HEAD>` y `</ HEAD>`, antes de la etiqueta `<BODY>` del código HTML de la página Web.

2.3. Anotación de metadatos en páginas Web

Desde los comienzos de la Internet, las páginas Web fueron hechas a mano por los programadores. Estos documentos consisten en una enorme cantidad de texto, imágenes y sonidos, comprensibles para un ser humano, pero que carecen de significado para las computadoras. En la actualidad, dada la enorme cantidad de información existente en la Web, se hace imposible que un usuario analice, extraiga o interprete la información que necesita revisando todos los contenidos existentes. Por esta razón han surgido nuevas tecnologías que tienden a la búsqueda y recuperación de la información de manera automática, como así también a la generación semiautomática de documentos Web. Estas nuevas tecnologías se valen de XML, RDF u OWL para lograr sus objetivos, pero aunque consigan facilitar la tarea del procesamiento automático de la información, una computadora no puede llevar a cabo de manera automática tareas de reconocimiento e interpretación de la información relevante. Para tratar de solucionar estos inconvenientes se introdujo la anotación de metadatos en páginas Web, cuyo objetivo es hacer más explícito para las computadoras el significado de una página [Blanco Suárez, 2004].

Las herramientas de anotación permiten agregar contenido semántico a las páginas Web. Es decir, usando anotaciones de metadatos es posible estructurar la información publicada mediante su clasificación en base a conceptos semánticos, lo cual da el primer paso en el procesamiento automático de la información por parte de las máquinas [Murua, 2004].

Las herramientas de anotación pueden dividirse en dos grupos:

- Herramientas de anotación externa: Permiten agregar información a páginas Web existentes. La información no se agrega a la página Web sino que se almacena en un repositorio externo (como ser una base de datos RDF).
- Herramientas de anotación de autor: Permiten incluir la información dentro de las propias páginas usando lenguajes de marcado XML o RDF.

En esta tesis se extrae la información y se almacena en un archivo XML que puede ser incluido como una referencia dentro de la página Web, o almacenado de manera separada en una base de datos externa.

2.4. XML

XML (en inglés *Extensible Markup Language*), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Fue creado para estructurar, almacenar, y transportar información. XML es una simplificación y adaptación de SGML[*] y permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de los lenguajes que usan XML para su definición son XHTML, SVG[†], MathML.

El secreto del éxito de XML es que es muy sencillo de entender y de utilizar, e implementa algo de lo que HTML adolece: establece un estándar fijo al que atenerse. Sumado a esto, XML describe el contenido de lo que etiqueta. XML no fue desarrollado sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, etc.

XML es una tecnología sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande, dándole mayores posibilidades. Este metalenguaje tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

El XML brinda una representación de la información de manera estructurada, que mediante herramientas y lenguajes de consulta específicos puede ser fácilmente accedida. Un XML básico se ve como un árbol de nodos delimitados con etiquetas,

---

[*] SGML: del inglés *Standard Generalized Markup Language*, lenguaje de marcas generalizado

[†] SVG: *Scalable Vector Graphics*, lenguaje para describir gráficos vectoriales.

con la posibilidad de contener en cada nodo atributos valorizados representados en una cadena de caracteres. El árbol posee un único nodo raíz que se diversifica a lo largo de los elementos que cuelgan de él y acaba en nodos hoja, que contienen sólo texto, comentarios, instrucciones de proceso o incluso que están vacíos y sólo tienen atributos. Un documento XML es procesado por un analizador, que construye el árbol de nodos.

Hay dos tipos de documentos XML: válidos y bien formados. Los documentos bien formados son los que cumplen las especificaciones del lenguaje respecto a las reglas sintácticas sin estar sujetos a elementos fijados en un DTD. Los documentos XML deben tener una estructura jerárquica muy estricta. Los documentos válidos, además de estar bien formados, tienen su estructura y semántica determinadas por un DTD. Sus elementos y su estructura jerárquica deben ajustarse al DTD.

El DTD (en inglés *Document Type Definition*) es una definición de los elementos que puede haber en el documento XML, y su relación entre ellos, atributos, valores, etc. Cuando se procesa cualquier información formateada mediante XML, lo primero es comprobar si está bien formada, y luego, si incluye una referencia a un DTD, comprobar si sigue sus reglas gramaticales.

El modelo de objetos de documentos o DOM (en inglés *Document Object Model*) es una representación interna estándar de la estructura de un documento. Proporciona una interfase al programador (API) para poder acceder de forma fácil a los elementos, atributos y estilo de un documento. Es un modelo independiente de la plataforma del lenguaje de programación. El objetivo es que cualquier script pueda ejecutarse de forma homogénea en cualquier navegador que soporte dicho DOM. El DOM es el que muestra los documentos XML como una estructura de árbol. Todos los elementos pueden ser accedidos a través del árbol DOM, y su contenido (texto y atributos) puede ser modificado o borrado, así como también pueden ser creados nuevos elementos.

Para una descripción técnica más detallada y más información de las reglas sintácticas véase [Web, 1], [Web, 2], [Web, 3].

2.5. XHTML

XHTML, del inglés *eXtensible HyperText Markup Language* (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas Web. La necesidad de reemplazar HTML nace de su limitación de uso con las herramientas basadas en XML, cada vez más extendidas.

En su versión 1.0, XHTML es solamente la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones más estrictas de XML. La versión 1.1 es similar, pero parte a la especificación en módulos. En sucesivas versiones, la W3C planea romper con los tags clásicos traídos de HTML. En otras palabras, XHTML extiende HTML combinando la sintaxis de HTML, diseñado para mostrar datos, con la de XML, diseñado para describir los datos [Web, 4].

Su principal objetivo es avanzar en el proyecto del W3C de lograr una Web semántica, donde la información, y la forma de presentarla estén claramente separadas.

Una de las principales ventajas del XHTML sobre el HTML es que los parsers[*] pueden ser mucho más sencillos, debido a que este lenguaje es más estructurado, y a que su etiquetado más estricto permite una correcta interpretación de la información independientemente del dispositivo desde el que se acceda a ella. Además, XHTML puede incluir otros lenguajes como MathML[†], SMIL[‡], etc.

Al estar orientado al uso de un etiquetado correcto, se exigen una serie de requisitos básicos en XHTML, como ser una estructuración coherente dentro del documento. Por ejemplo los elementos deben estar correctamente anidados, etiquetas en minúsculas, elementos cerrados correctamente, atributos de valores entrecomillados, etc.

---

[*] Un parser es un programa que transforma una entrada de texto en una estructura de datos (usualmente un árbol) que es apropiada para ser procesada.

[†] MathML: del inglés *Mathematical Markup Language*, lenguaje para descripción matemática.

[‡] SMIL: del inglés *Synchronized Multimedia Integration Language*, lenguaje de integración multimedia sincronizada.

2.6. XQuery

XQuery es un lenguaje de consulta diseñado para ser usado en colecciones de datos XML. Es semánticamente similar a SQL, pero incluye algunas capacidades de programación. Una descripción más detallada puede encontrarse en [Web, 5].

XQuery proporciona los medios para extraer y manipular información de documentos XML, o de cualquier fuente de datos que pueda ser representada mediante una estructura XML, como por ejemplo Bases de Datos Relacionales o documentos ofimáticos.

XQuery utiliza expresiones XPATH para acceder a determinadas partes del documento XML. Añade además unas expresiones similares a las usadas en SQL, conocidas como expresiones FLWOR. Las expresiones FLWOR toman su nombre de los 5 tipos de sentencias de las que pueden estar compuestas: FOR, LET, WHERE, ORDER BY y RETURN.

También incluye la posibilidad de construir nuevos documentos XML a partir de los resultados de la consulta. Se puede usar una sintaxis similar a XML si la estructura (elementos y atributos) es conocida con anticipación, o usar expresiones de construcción dinámica de nodos en caso contrario. Todos estos constructores se definen como expresiones dentro del lenguaje, y se pueden anidar arbitrariamente.

El lenguaje se basa en el modelo de árbol de la información contenida en el documento XML, que consiste en siete tipos distintos de nodo: elementos, atributos, nodos de texto, comentarios, instrucciones de procesamiento, espacios de nombres y nodos de documentos.

El sistema de tipos usado por el lenguaje considera todos los valores como secuencias, asumiéndose un valor simple como una secuencia de un solo elemento. Los elementos de una secuencia pueden ser valores atómicos o nodos. Los valores atómicos pueden ser números enteros, cadenas de texto, valores booleanos, etc. La

lista completa de los tipos disponibles está basada en las primitivas definidas en XML Schema*.

XQuery 1.0 no incluye la capacidad de actualizar los documentos XML. Tampoco puede realizar búsquedas textuales. Estas dos capacidades están siendo objeto de desarrollo para su posible incorporación en la siguiente versión del lenguaje. XQuery es un lenguaje de programación funcional que consta en su totalidad de expresiones. No hay sentencias, aún cuando algunas de las palabras claves utilizadas pueden sugerir un comportamiento similar al de una sentencia. Para ejecutar una función, la expresión dentro del cuerpo de la misma se evalúa y se retorna el resultado obtenido.

2.7. XPATH

XPATH (*XML Path Language*) es un lenguaje creado para examinar la estructura de documentos XML que permite construir expresiones que recorren y procesan dichos documentos. Este lenguaje permite buscar, seleccionar y direccionar partes de los documentos XML teniendo en cuenta su estructura jerárquica.

XPATH modela un documento XML como un árbol de nodos. Hay distintos tipos de nodos, incluyendo el nodo raíz, nodos elemento, nodos atributo y nodos texto [Web, 6].

El nodo raíz se identifica por `/`. No se debe confundir al nodo raíz del árbol con el elemento raíz del documento. El nodo raíz del árbol contiene al elemento raíz del mismo. Es decir, si `A` es el elemento raíz del árbol, entonces `A` colgará del nodo raíz `/`. Los nodos elementos están compuestos por cualquier elemento del documento XML. Cada nodo elemento tiene un nodo padre, y puede tener a su vez nodos hijos. Los nodos elemento tienen propiedades, tales como su nombre, sus atributos, etc. Estos nodos pueden tener identificadores únicos, lo que permite referenciarlos más

* XML Schema: lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML

directamente. Los nodos texto son los distintos caracteres del documento que no están marcados con alguna etiqueta. Un nodo texto no tiene hijos.

Los nodos atributo son un caso especial de nodo. Un nodo elemento puede tener tantos atributos como se quiera, y por cada uno se le creará un nodo atributo. Los nodos atributos no son considerados hijos del nodo elemento, sino más bien etiquetas. Cada nodo atributo consta de un nombre, un valor y opcionalmente un espacio de nombres. Otro tipo de nodos que pueden existir son nodos con comentarios o con instrucciones de proceso.

XPATH utiliza expresiones de ruta para seleccionar nodos o conjuntos de nodos en un documento XML. Estas expresiones de ruta se asemejan mucho a las expresiones que se usan cuando se trabaja con sistemas de archivos.

La sintaxis básica es similar a la estructura de directorios. Un camino dentro del árbol XML se indica separando cada nodo con "/". Cuando se indica "//" todos los elementos del documento que cumplen con el criterio son seleccionados.

El "*" selecciona todos los elementos ubicados en el camino que lo precede, es como un comodín.

Las expresiones entre corchetes permiten precisar la especificación de un elemento, un número dentro de los corchetes indica la posición del elemento en el conjunto seleccionado. Los atributos se especifican con el símbolo "@"

La combinación de estas funcionalidades brinda un alto poder de direccionamiento y selección sobre los documentos XML.

2.8. HtmlAgilitypack

HtmlAgilitypack [Web, 7] es un software desarrollado para transformar documentos HTML en documentos XML válidos. Los documentos resultantes son árboles DOM de lectura escritura que pueden ser consultados fácilmente con XPATH. La facilidad que provee este software se basa en ser tolerante con los documentos HTML que hay en la Web, de modo de poder darle estructura a los documentos y luego poder analizarlos más fácilmente.

2.9. .Net reflection

Reflection es una facilidad de .Net que permite obtener información acerca de objetos en tiempo de ejecución. Esta información contiene datos de la clase del objeto. También se pueden obtener los nombres de los métodos que están dentro de la clase y los constructores del objeto. Esto es posible en el framework de .Net, porque los tipos de datos son objetos. Es decir que cada dato, tanto si es una clase que se escribió o una clase propia del framework, tiene métodos que se pueden invocar y propiedades que se pueden examinar. Esto permite obtener la lista de métodos expuestas por una clase, determinar los tipos de datos de una propiedad o iterar por los parámetros de un método dado. En los lenguajes administrados por el CLR* se sabe exactamente que tipo de datos hay en cada ubicación de memoria en tiempo de ejecución. Esto permite que se puedan descubrir los tipos de datos sin necesidad de tener el código fuente.

Una de las grandes ventajas de trabajar con *reflection* es que se pueden agregar de manera dinámica librerías de métodos al desarrollo muy fácilmente y por ende se puede aumentar la funcionalidad del proyecto de manera transparente al desarrollo original [Hoffman, 2006].

2.10. Expresiones regulares

Una expresión regular o patrón, es una expresión que describe un conjunto de cadenas sin enumerar sus elementos. Es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje. Específicamente, las expresiones regulares se construyen utilizando los operadores unión, concatenación y clausura de Kleene.

Las expresiones regulares permiten un procesamiento rápido y eficiente del texto. El texto que se procesa puede ser tan pequeño como una dirección de correo electrónico o tan amplio como un cuadro de entrada multilínea. El uso de las

* CLR del inglés *Common Language Runtime*, es el componente de máquina virtual del .Net framework de Microsoft.

expresiones regulares no sólo permite validar un texto con un modelo definido, sino que permite extraer datos del texto que corresponde a un modelo especificado.

Se trata sencillamente de ir cotejando un patrón (pattern) con una cadena (subject) y ver si dentro de ella existe la misma secuencia. Si existe, se dice que se encontró una coincidencia (*Match()*).

Un uso común de las expresiones regulares es el de analizar el texto según la expresión y utilizar eso para extraer datos desde la entrada de usuario, lo cual se denomina correspondencia de grupo. Las expresiones regulares incluyen una característica especial llamada grupos. Un grupo permite poner un identificador designado en una sección especial de la expresión regular.

Cuando se utiliza *Match()* para comparar los datos de entrada con el modelo, en realidad se están separando las correspondencias por grupo, lo que permite extraer la parte de la entrada que corresponde a cada grupo [Hoffman, 2006].

2.11. Índices de legibilidad

En las siguientes subsecciones se presentan los índices de legibilidad empleados para indicar el nivel de dificultad de cada página obtenida. Estos índices indican tanto el nivel de educación necesario para entender los contenidos de las páginas como la comprensibilidad o la dificultad de los mismos. Los índices de legibilidad son especialmente útiles en la búsqueda de cursos, ya que el usuario puede utilizar los valores de los índices para clasificar las páginas encontradas según la complejidad del curso o según su nivel educacional.

En las teorías lingüísticas existen varios índices de legibilidad, en esta sección se detallan los utilizados en este trabajo. Cabe destacar que estos índices no están correlacionados entre sí, es decir, si un documento da un resultado mayor que otro usando un determinado índice, al aplicar otro índice esto no necesariamente tiene que ocurrir de la misma manera. En esta tesina se utilizan los índices más comúnmente usados por los procesadores de texto, con el fin de brindarle al usuario una amplia gama de opciones.

Cabe agregar que algunos de estos índices utilizan separación en sílabas, lo cual no siempre es aplicable a todos los idiomas.

2.11.1. Índice automatizado de legibilidad

El índice automatizado de legibilidad (en inglés *Automated Readability Index* (ARI)) [Senter, 1967] es un test de legibilidad diseñado para medir la comprensibilidad de un texto. Su resultado es una representación aproximada del nivel de estudios requeridos para comprender el texto.

El algoritmo para calcular el índice automatizado de legibilidad es el siguiente:

1. Dividir el número de caracteres del texto por el número de palabras, y multiplicar por 4.71.
2. Dividir el número de palabras por el número de oraciones, y multiplicar por 0.5.
3. Sumar ambos valores y restarle 21.43.

$$ARI = 4{,}71\left(\frac{caracteres}{palabras}\right) + 0{,}5\left(\frac{palabras}{caracteres}\right) - 21{,}43$$

Ecuación 1: Automated Readability Index

El resultado aproxima el nivel de conocimiento mínimo necesario para entender el texto. Por ejemplo, un resultado de 8.2 (8º grado) debería ser fácilmente entendido por individuos de 14 años promedio.

2.11.2. Índice Gunning Fog

En lingüística, el índice Gunning Fog es un test diseñado para medir la legibilidad de una muestra de texto. El resultado es un indicador del número de años de educación formal que una persona requiere para entender fácilmente el texto en una primer lectura. Por ejemplo, si un texto tiene un índice Gunning Fog de 12, tiene un nivel que podría leer un graduado universitario. Este test fue desarrollado por Robert Gunning, un empresario americano, en 1952 [News, 1].

El índice Fog es usado por personas que quieren que sus escritos sean fácilmente leíbles por un gran segmento de la población. Los textos que son para ser leídos por una audiencia amplia generalmente requieren un índice Gunning Fog menor a 12.

El índice Gunning Fog se puede calcular con el siguiente algoritmo:

1. Encontrar el largo promedio de las oraciones.
2. Contar las palabras que tengan cuatro o más sílabas (palabras complejas).
3. Sumar el largo promedio de las oraciones y el porcentaje de palabras complejas.
4. Multiplicar el resultado por 0,4.

La fórmula completa del cálculo es:

$$GFI = 0{,}4\left(\left(\frac{palabras}{oraciones}\right)+100\left(\frac{palabras\_complejas}{palabras}\right)\right)$$

Ecuación 2: Gunning Fog

A pesar del hecho que este índice es un buen indicador de la dificultad de lectura de un texto, también tiene sus limitaciones. Por ejemplo, no todas las palabras de más de cuatro sílabas son complejas. Igualmente sigue siendo una buena estimación.

2.11.3. Medida simple Gobbledygook

La medida simple Gobbledygook (en inglés *Simple Measure of Gobbledygook* (SMOG)) [McLaughlin, 1969] es una fórmula de legibilidad que estima los años de educación necesarios para entender completamente un texto. La fórmula exacta de SMOG tiene una correlación de 0.985 con el nivel de educación de los lectores que tenían 100% de comprensión de los materiales de prueba.

SMOG fue publicado por G. Harry McLaughlin en 1969 como una manera más precisa y más fácil de calcular que el índice de Gunning Fog.

Los pasos para calcular SMOG son:

1. Contar el número de oraciones.

2. Contar las polisílabas (palabras de 4 o más sílabas).

3. Aplicar la fórmula:

$$SMOG = 5{,}717\sqrt{\frac{polisílabas}{oraciones}} + 3{,}129$$

Ecuación 3: Gobbledygook

2.11.4. Índice de Coleman-Liau

El índice de Coleman-Liau [Coleman, 1975] es un test de legibilidad diseñado por Meri Coleman y T. L. Liau para calibrar la comprensibilidad de un texto. Tal como en el test de Flesch-Kincaid (ver sección 2.11.5), el índice Gunning fog, el índice Smog, y el índice automatizado de legibilidad, la salida del índice de Coleman-Liau aproxima el nivel de educación necesario para comprender un texto.

Al igual que el ARI (Ecuación 1), pero distinto de los demás índices, Coleman-Liau usa los caracteres en lugar de las sílabas por palabra. A pesar que las opiniones varían en cuanto a su efectividad comparado con otros índices más complejos, los caracteres son más legibles y se pueden contar más fácilmente por programas de computación que las sílabas.

Para calcular el índice de Coleman-Liau:

$$CLI = 5{,}89\left(\frac{caracteres}{palabras}\right) - 0{,}3\left(\frac{oraciones}{palabras}\right) - 15{,}8$$

Ecuación 4: Coleman-Liau

2.11.5. Tests de legibilidad de Flesch-Kincaid

Los tests de legibilidad de Flesch y de Flesch–Kincaid [Flesch, 1948], [Kincaid, 1975], [Farr, 1951] son pruebas cuyo objetivo es indicar la dificultad de comprensión que puede tener un texto. Hay dos tests, el de facilidad de lectura de Flesch, y el de

nivel de grado de Flesch–Kincaid. Aunque ambos tests usen las mismas medidas de base (longitud de las palabras y de las oraciones), tienen distintos factores de peso, por lo que los resultados de ambos no se correlacionan: Al comparar dos textos, puede que uno tenga puntaje más alto que el otro con el test de facilidad de lectura, y más bajo con el test de nivel de grado. Ambos tests fueron ideados por Rudolf Flesch.

• Test de Facilidad de lectura de Flesch

En este test, un puntaje más alto indica que el material es más fácil de leer. Puntajes más bajos señalan textos más difíciles de leer. La fórmula para calcular el valor de este índice es la siguiente:

$$K = 206{,}835 - 1{,}015\left(\frac{palabras}{oraciones}\right) - 84{,}6\left(\frac{sílabas}{palabras}\right)$$

Ecuación 5: Flesch

Puntajes de 90 – 100 son considerados fácilmente entendibles por un estudiante de 11 años. Estudiantes de 13 a 15 años pueden entender fácilmente textos con un puntaje de 60 - 70, y textos con un puntaje entre 0 - 30 son mejor entendidos por graduados universitarios. Por ejemplo, la revista Reader's Digest tiene un índice de legibilidad de aproximadamente 65, en tanto que la revista Time de 52. El puntaje de legibilidad más alto (más fácil) posible es de 121 (cada oración consistente de una palabra de una sílaba); teóricamente no hay un límite inferior.

El uso de esta escala es tan ubicuo que está relacionado con programas de procesamiento de textos populares y servicios tales como KWord, Lotus WordPro, Microsoft Word, y Google Docs. Las palabras largas afectan esta medida de legibilidad mucho más de lo que lo hacen en el test de nivel de grado de Flesch-Kincaid.

- **Nivel de Grado de Flesch–Kincaid**

Un uso obvio para tests de legibilidad es el campo de la educación. La fórmula de nivel de grado de Flesch-Kincaid traslada un puntaje de 0 - 100 a un nivel de educación, haciendo más fácil para los maestros, padres, etc. decidir el nivel de legibilidad de libros y textos.

Este resultado también puede significar el número de años de educación requeridos para entender un texto, relevante sólo cuando la fórmula resulta en un número mayor que 12. El nivel de educación es calculado con la siguiente fórmula:

$$FK = 0{,}39\left(\frac{palabras}{oraciones}\right) + 11{,}8\left(\frac{sílabas}{palabras}\right) - 15{,}59$$

Ecuación 6: Flesch–Kincaid

El resultado es un número que corresponde con el nivel de educación, es la cantidad de años aproximada de estudio en el sistema educativo que se requiere para comprender un texto. Para la mayoría de los documentos el objetivo de esta medición debería ser entre 7.0 y 8.0.

El menor nivel que se puede obtener es, en teoría -3.4, pero como hay pocos párrafos reales consistentes de palabras de una sola sílaba, este resultado raramente ocurre en la práctica.

Capítulo 3.

Propuesta de solución

3.1. Descripción general

El objetivo de este trabajo es recolectar las páginas obtenidas como resultado de una búsqueda en Internet y explorarlas en busca de información, a fin de devolverle al usuario los resultados más adecuados a sus requerimientos y de almacenar la información extraída para su posterior utilización. Para las pruebas del prototipo generado se usó el motor de búsqueda de Google, y se tomaron los resultados devueltos por este buscador como punto de partida para el análisis de su contenido.

El proceso consta de dos etapas fundamentales, la primera es la etapa de búsqueda. Consiste en realizar la consulta solicitada al buscador y analizar la respuesta del buscador para extraer las direcciones de las páginas obtenidas. La cantidad de direcciones a extraer es un parámetro provisto por el usuario.

La segunda etapa es la de recopilación y ordenamiento de la información. Una vez obtenidos los resultados se procede al primer paso de esta etapa que es obtener las páginas de su enlace original. El HTML obtenido es analizado y pasado a formato XHTML para poder representarlo como un árbol. Este procedimiento es muy

importante para facilitar la tarea del análisis de la información, que puede llegar de fuentes heterogéneas de HTML.

Una vez que los datos están contenidos dentro de un árbol XHTML se realiza el segundo paso de esta etapa, que consiste en la extracción de la información inherente a la página (es decir se extraen sus características) por medio de consultas XQuery y con la ayuda de expresiones regulares que permiten la manipulación eficiente del texto.

Las diferentes características extraídas se vuelcan en un nuevo árbol XML en el que cada nodo del primer nivel corresponderá a una página analizada. Las características de cada una de estas páginas se agregan como nodos hijos en la rama de la página que se está analizando. Una vez encontradas todas las características de una página, se agrega otro nodo en este nivel que corresponde a los resultados de calcular índices sobre las características recuperadas como muestra la Figura 1. De acuerdo a estos índices se realiza el tercer paso de esta etapa: la ordenación lógica según las especificaciones del usuario.

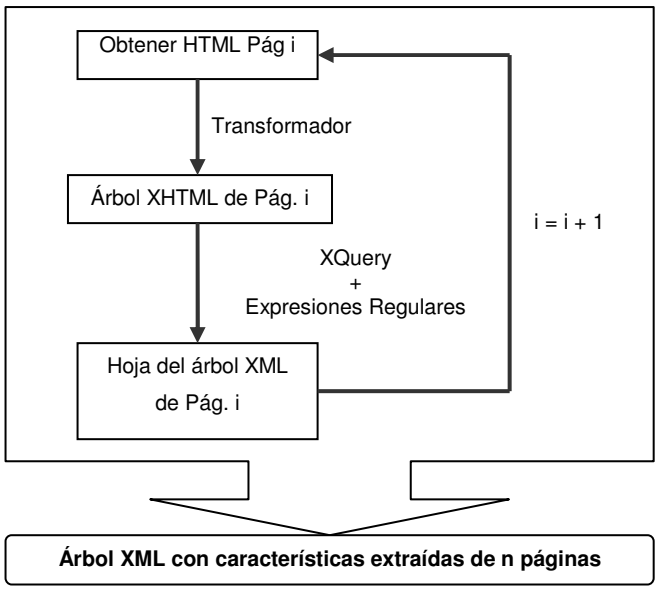

Figura 1: Diagrama de extracción

Como finalización del proceso se agrega un atributo más al nodo de la página en el árbol XML con la ubicación final de la página en cuestión luego de la ordenación.

Este proceso se realiza en forma iterativa para todos los resultados devueltos en el primer paso. Como resultado queda un árbol representado por un XML, que se envía como respuesta al pedido de la página principal del prototipo, y el usuario lo descarga como un archivo en su computadora

3.2. Diseño del XML

La generación de un árbol como resultado brinda la posibilidad de manipularlo con facilidad, obteniendo de él los datos necesarios para trabajos posteriores, como anexar la información a la página original o tomarla para indexarla en un repositorio. Éste trabajo se limita a obtener la información de las páginas y recopilarla en un árbol XML de fácil acceso posterior.

Como resultado del análisis se obtiene un XML final compuesto de un nodo inicial del que se desprende un nodo por cada página analizada. A su vez, en los nodos de cada página se desprende un nodo por cada característica y uno final llamado `<indices>` que contiene cálculos realizados sobre valores de las características extraídas.

El nodo inicial del árbol se llama `<paginas>`. Cada nodo hijo que se desprende de él, en el primer nivel de profundidad del árbol, es llamado `<pagina>`. Hay uno para cada página analizada y en cada uno de estos se encuentran los atributos que la definen. Ellos son:

- **ORDEN:** Es el orden resultante dentro de las páginas analizadas luego de ordenarlas por el criterio seleccionado. Se completa en el último paso del proceso.
- **ORDEN ORIGINAL:** Es el número de orden que se obtuvo de la búsqueda original en el buscador.
- **TÍTULO:** Es el título de la página obtenido de la etiqueta `<TITLE>` del código HTML analizado.
- **URL:** Es la dirección de donde se tomó el código fuente original de la página en cuestión.

Todos estos atributos definen el nodo principal de cada página. La definición de los atributos principales del esquema se puede ver en el fragmento XML de Código 1.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
           elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="paginas">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="pagina">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="general"> … </xs:element>
              <xs:element name="descripcion" /> … </xs:element>
              <xs:element name="clasificacion" /> … </xs:element>
              <xs:element name="flash"> … </xs:element>
              <xs:element name="keywords"> … </xs:element>
              <xs:element name="idioma" /> … </xs:element>
              <xs:element name="links"> … </xs:element>
              <xs:element name="imagenes"> … </xs:element>
              <xs:element name="meta"> … </xs:element>
              <xs:element name="indices"> … </xs:element>
            </xs:sequence>
            <xs:attribute name="orden" type="xs:unsignedByte"
                          use="required" />
            <xs:attribute name="ordenOriginal"
                          type="xs:unsignedByte" use="required" />
            <xs:attribute name="titulo" type="xs:string"
                          use="required" />
            <xs:attribute name="url" type="xs:string" use="required"
            />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 1: Fragmento de esquema de resultado final

Como se puede observar en el esquema, el nodo `<pagina>`, tiene una secuencia, que no se detalla, en la que están las características y los cuatro atributos anteriormente descriptos.

La generación del esquema es sólo a modo ilustrativo para explicitar mejor la estructura del XML, no siendo necesaria para la generación de resultados ya que el prototipo cumple con el requerimiento de forma nativa, pero el esquema permite validar que el XML obtenido finalmente cumple con la estructura impuesta.

La secuencia que se puede ver en el código dentro del nodo página contiene las características analizadas y extraídas de cada página. Hay un nodo por cada una y el contenido de cada una es particular.

3.3. Metadatos de interés

La arquitectura del prototipo permite que las funciones usadas para extraer y analizar los metadatos de las páginas estén fuera de la aplicación principal, dentro de una librería externa. Estas funciones se cargan dinámicamente de acuerdo a un archivo de configuración listado en el Apéndice A. Esto permite que este prototipo tenga un grupo de metadatos básicos que analizar de cada página. Estos metadatos extraídos son agrupados en nodos *Características* dentro del XML. Además el prototipo queda abierto a la posibilidad de agregar posteriormente ampliaciones y mejoras (como ser otros metadatos agrupados en otras *Características*) dentro de librerías nuevas independientes del funcionamiento del núcleo. Los metadatos extraídos y analizados se agrupan en las *Características* mostradas en la Tabla 1, que se detallan luego en esta misma sección.

Característica	Descripción
General	Metadatos generales de la página.
Descripción	Extracción del metadato descripción de la página
Clasificación	Extracción del metadato clasificación de la página
Flash	Extracción de los contenidos embebidos de la tecnología Flash.
Keywords	Extracción del metadato keywords de la página
Idioma	Extracción del metadato lenguaje de la página
Links	Extracción y análisis de los enlaces que contiene la página
Imágenes	Extracción y análisis de las imágenes que contiene la página
Meta	Extracción del resto de las etiquetas meta que tenga la página.

Tabla 1: Breve descripción de las características analizadas

Cada una de las características está concentrada en nodos de segundo nivel, hijos del nodo `<pagina>` que se está analizando, en el árbol XML resultado. En las siguientes subsecciones se describen detalladamente estas características.

3.3.1. Característica: General

Sobre la página Web se analizan metadatos básicos de índole general, estos se agrupan en una característica llamada general que contiene los detalles principales y un conteo de diferentes aspectos de la página. En el fragmento de Código 2 se muestra la definición de esquema que debe cumplir. El nodo dentro del XML de esta característica, llamado `<general>`, tiene un nodo hijo `<resultado>` con varios atributos que lo definen, ellos son:

- **DOMINIO:** de tipo cadena de caracteres, contiene la dirección principal de la página. La extracción de la cadena se realiza con una expresión regular que captura distintos grupos dentro de la URL de la página, entre ellos está el dominio de la misma.
- **BYTESTOTAL:** de tipo entero, contiene la cantidad total de bytes de la página, se obtiene contando la cantidad de bytes del stream originado en el pedido de la página al servidor.
- **BYTESTEXTO:** de tipo entero, contiene la cantidad de caracteres de la página, se obtiene contando la cantidad de caracteres del texto de la página luego de quitarle los scripts y los tags propios del HTML.
- **CARACTERES:** de tipo entero, es la cantidad de letras y números que tiene la parte visual de la página, este campo también se cuenta por medio de extracciones con expresiones regulares extrayendo solo los caracteres de letras y números.
- **PALABRAS:** de tipo entero, contiene la cantidad de palabras que hay en el texto visible de la página. Se realiza contando la cantidad de expresiones que concuerdan con la expresión regular `\w+`

- **ORACIONES:** de tipo entero, contiene la cantidad de oraciones que hay en el texto visible de la página, para contarlas se tienen en cuenta las palabras de al menos una letra que terminan con un punto o un retorno de carro. La expresión regular que concuerda con este requerimiento es `\w(\.|\n)`

- **POLISÍLABAS:** de tipo entero, contiene la cantidad de palabras que tienen al menos cuatro sílabas, esto se basa en las reglas se separación en sílabas desarrolladas en la tesis de [Figueroa, 1998] y se usa para los cálculos de los índices de legibilidad.

- **SÍLABAS:** de tipo entero, contiene la cantidad total de sílabas del texto visible de la página, es un cálculo que se desprende del cómputo del campo anterior, también se basa en el algoritmo propuesto por [Figueroa, 1998] en su tesis. La implementación de este algoritmo está explicada en la sección 3.5.

El fragmento de esquema del Código 2 describe como se debe armar el XML resultado de esta característica, en el fragmento de Código 3 se ve un ejemplo del XML final resultante.

```
<xs:element name="general">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="dominio" type="xs:string"
                        use="required" />
          <xs:attribute name="bytesTotal" type="xs:unsignedInt"
                        use="required" />
          <xs:attribute name="bytesTexto" type="xs:unsignedShort"
                        use="required" />
          <xs:attribute name="caracteres" type="xs:unsignedShort"
                        use="required" />
          <xs:attribute name="palabras" type="xs:unsignedShort"
                        use="required" />
          <xs:attribute name="oraciones" type="xs:unsignedShort"
                        use="required" />
          <xs:attribute name="polisilabas" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="silabas" type="xs:unsignedShort"
                        use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 2: Fragmento de esquema de la característica general

```
<general>
  <resultado dominio="www.sectormatematica.cl" bytesTotal="36802"
             bytesTexto="10498" caracteres="5663" palabras="1214"
             oraciones="138" polisilabas="12" silabas="746" />
</general>
```

Código 3: Ejemplo de característica General

3.3.2. Característica: Descripción

Una de las características que se deberían destacar en este tipo de trabajos es la que señala una descripción del contenido de la página Web. Esta característica se escribe como texto coloquial dentro de las etiquetas Meta del HTML. Debido a que no está estandarizado el requerimiento de agregar una etiqueta `<Meta description>` para detallar el contenido de la página, ésta no se encuentra en muchas páginas en las que sería útil para una correcta clasificación. En este trabajo se propone extraer esta etiqueta si existiera, con la expresión XPATH `//meta[@name='description']`.

```
<xs:element name="descripcion">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
          />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 4: Fragmento de esquema de la característica Descripción

Lo que se obtiene como resultado es un nodo XML llamado `<Descripcion>` con un nodo hijo llamado `<resultado>` que contiene un atributo *id* numerado correlativamente y un atributo valor con la trascripción completa del atributo de la página original. Un ejemplo del resultado se puede observar en el Código 5.

```
<descripcion>
  <resultado id="1" valor="Despu&amp;eacute;s de siete
             d&amp;iacute;as de deliberaciones, el empresario
             venezolano, ex socio de Antonini Wilson, fue encontrado
             culpable de conspirar y actuar ilegalmente en Estados
             Unidos como un agente del Gobierno de Hugo
             Ch&amp;aacute;vez para ocultar el origen y destino de
             la v" />
</descripcion>
```

Código 5: Ejemplo de extracción de Descripción

3.3.3. Característica: Clasificación

Al igual que la característica anterior, la clasificación también es un meta dato de las páginas Web, dentro de una etiqueta `<Meta clasification>` que contiene palabras clasificatorias del contenido de la página. Como esta etiqueta tampoco es un estándar, no todas las páginas la contienen, dificultando la tarea de clasificar correctamente según su contenido.

En el fragmento de Código 6 se puede apreciar el esquema que representa esta característica dentro del XML resultante. Se observa que la recopilación de este dato consta de una secuencia de resultados con un atributo *id* correlativo y el atributo *valor* tomado del campo del XHTML original de la página con la expresión XPATH

`//meta[@name='Classification']`, como se puede apreciar en el ejemplo mostrado en el Código 7.

```
<xs:element name="clasificacion">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
          />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 6: Fragmento de esquema de la característica Descripción

```
<clasificacion>
  <resultado id="1" valor="World, Español, Regional, Europa, España,
                           Noticias y medios, Revistas" />
</clasificacion>
```

Código 7: Ejemplo de extracción para Clasificación

3.3.4. Característica: Flash

Un atributo importante a tener en cuenta para la clasificación del contenido de una página Web son los aspectos multimedia que ésta contenga. Una de las formas más comunes que se utilizan para insertar multimedia en las páginas es la tecnología Flash de Macromedia. Saber cuántas inserciones Flash hay en una página puede resultar útil para el análisis actual o un análisis futuro. Esta característica se arma siguiendo las reglas del esquema listado en el Código 8, y se muestra un ejemplo en el Código 9 de un resultado aplicado.

La característica Flash está representada en el árbol XML por un nodo llamado `<flash>` que tiene dos nodos hijos: `<resultado>` y `<listado>`. Los nodos del árbol XHTML de la página Web recuperada que representan un Flash se obtienen con la expresión XPATH `//embed[@type='application/x-shockwave-flash']`.

```
<xs:element name="flash">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="cantidad" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="superficie" type="xs:unsignedInt"
                        use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="listado">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="item">
              <xs:complexType>
                <xs:attribute name="id" type="xs:unsignedByte"
                              use="required" />
                <xs:attribute name="url" type="xs:string"
                              use="required" />
                <xs:attribute name="ancho" type="xs:unsignedShort"
                              use="required" />
                <xs:attribute name="alto" type="xs:unsignedByte"
                              use="required" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 8: Fragmento de esquema de la característica Flash

Se cuenta cada uno de los nodos del XHTML analizados que contienen Flash para acumularlo en el atributo *cantidad* y se accede al nodo para obtener el tamaño que ocupa en la página en píxeles y acumularlo en el atributo *superficie*. Ambos atributos pertenecen al nodo `<resultado>` del XML resultante. Luego se listan como nodos `<item>`, hijos del nodo `<listado>`. Los nodos `<item>` tienen como atributos un *id* correlativo, una *url* con el nombre completo del Flash embebido, y los atributos *ancho* y *alto* que se corresponden con estas propiedades del objeto en el HTML origen, también medidas en píxeles.

```
<flash>
  <resultado cantidad="2" superficie="150000" />
  <listado>
    <item id="1" url="/diario/img/publicidades/vxv/283x238.swf"
          ancho="300" alto="250" />
    <item id="2" url="/diario/img/publicidades/vxv/durodedomar.swf"
          ancho="300" alto="250" />
  </listado>
</flash>
```

Código 9: Ejemplo de extracción de Flash

3.3.5. Característica: Keywords

Una característica importante a ser usada en la clasificación del contenido de las páginas Web son las palabras claves que el autor haya sugerido. Esto es comúnmente utilizado por los buscadores. Estas palabras clave están contenidas en una etiqueta `<meta name="Keywords">` dentro del código HTML de las páginas. Son palabras elegidas por el autor para clasificar su trabajo.

En el Código 10 se puede ver el esquema que representa como armar correctamente el nodo de la característica, llamado `<keywords>`, en el XML resultante. Las palabras listadas en la etiqueta `<meta name="Keywords">` de la página son agregadas al atributo *valor* del nodo `<resultado>`, que a su vez es hijo del nodo `<keywords>`.

```
<xs:element name="keywords">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
                        />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 10: Fragmento de esquema de la característica Keywords

El nodo `<resultado>` también tiene un atributo *id* numerado correlativamente para identificación.

Un ejemplo de esta extracción de keywords se puede observar en el Código 11. Estas palabras clave son muy usadas, porque se ha tomado como un estándar de facto que los buscadores las usan para clasificar el contenido de las páginas. De todas maneras, el abuso de esta característica para la manipulación de posicionamiento de resultados en los buscadores ha hecho que se tenga más cuidado a la hora de analizar su contenido.

```
<keywords>
  <resultado id="1" valor="Matemática,Semiprotección de
                           páginas,1202,1570,Algoritmo,Antinomia,
                           Análisis complejo,Análisis funcional,
                           Análisis matemático,Análisis numérico,
                           Análisis real" />
</keywords>
```

Código 11: Ejemplo de extracción de Keywords

3.3.6. Característica: Idioma

Para la clasificación de contenidos es necesario saber el idioma en el que está una página, para ello una de las características extraídas con este trabajo es justamente el idioma. Éste se obtiene de una etiqueta `<meta>` de la página original con la expresión XPATH `//meta[@http-equiv='Content-Language']`. Por la importancia de esta característica está planteada como una de las etiquetas `<meta>` estándar que propuso la W3C.

```
<xs:element name="idioma">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
          />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 12: Fragmento de esquema de la característica Idioma

En el Código 12 se puede ver como es el esquema necesario para armar correctamente esta característica dentro del XML resultante. Se ve que se desprende

un nodo <idioma>, con un nodo hijo <resultado>, que tiene como atributo un *id* numérico correlativo que representa cada resultado meta de idioma encontrado. Es de esperarse que sea uno solo, pero este prototipo esta preparado para recuperar varios si fuera el caso. En el Código 13 se puede ver el ejemplo de cómo queda en el XML resultante la extracción de esta característica. El atributo *valor* del nodo <resultado> esta representado por las letras estándar propuestas por el estándar ISO-639.

```
<idioma>
  <resultado id="1" valor="es" />
</idioma>
```

Código 13: Ejemplo de extracción de Idioma

3.3.7. Característica: Links

Algunos de los algoritmos más importantes que se usan en los buscadores se basan en los enlaces que la página tiene, por lo tanto, una de las extracciones de características se basa en los links contenidos en la página. De esta extracción se obtiene el listado de links, la cantidad, cuantos son links a páginas dentro del mismo servidor, y cuantos son a servidores externos.

Según se muestra en el Código 14, el nodo <links> correspondiente a esta característica en el árbol XML resultante tiene un nodo hijo llamado <resultado>, con los atributos *cantidad* que acumula todos los links de la página, *locales*, que acumula sólo los enlaces que se hacen dentro del mismo dominio, y finalmente *externos* que acumula los enlaces a páginas que están fuera del dominio original de la página.

Como se puede observar en el ejemplo del Código 15, además del nodo <resultado>, el nodo <links> tiene otro nodo hijo llamado <listado> que contiene tantos nodos hijos como links contenga la página. Cada link de la pagina está representado por un nodo <item>, que contiene como atributo un *id* que se numera correlativamente, un atributo *valor*, con la URL del enlace, y un atributo *título* con el título asignado en la página original.

```
<xs:element name="links">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="cantidad" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="locales" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="externos" type="xs:unsignedByte"
                        use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="listado">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="item">
              <xs:complexType>
                <xs:attribute name="id" type="xs:unsignedByte"
                              use="required" />
                <xs:attribute name="valor" type="xs:string"
                              use="required" />
                <xs:attribute name="titulo" type="xs:string"
                              use="required" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 14: Fragmento de esquema de la característica Links

```
<links>
  <resultado cantidad="177" locales="23" externos="154" />
  <listado>
    <item id="1" valor="http://la.wikipedia.org/wiki/mathematica"
          titulo="la:mathematica" />
    <item id="2" valor="http://worldcat.org/oclc/2014918"
          titulo="http://worldcat.org/oclc/2014918" />
    …
    <item id="176"
          valor="http://wikimediafoundation.org/wiki/Portada"
          titulo="" />
    <item id="177"
          valor="http://wikimediafoundation.org /wiki/Pol%C3%ADtica_
                 de_privacidad" titulo="wikimedia:Política de
                 privacidad" />
  </listado>
</links>
```

Código 15: Ejemplo de extracción de Links

3.3.8. Característica: Imágenes

Dentro de las características multimedia, una de las más importantes reclutadas en este trabajo es la extracción de las imágenes. Para ello en el Código 16 se puede ver el esquema que describe como se debe armar el nodo del XML resultante que representa las imágenes de la página. De manera similar a lo resuelto para los contenidos flash, en esta ocasión el nodo principal de la característica, llamado `<imagenes>`, contiene dos nodos hijos, el primero se llama `<resultado>` y contiene los atributos *cantidad*, que acumula la cantidad total de imágenes encontradas y *superficie* que acumula la cantidad de píxeles ocupados por imágenes en la página.

```
<xs:element name="imagenes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="resultado">
        <xs:complexType>
          <xs:attribute name="cantidad" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="superficie" type="xs:unsignedInt"
                        use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="listado">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="item">
              <xs:complexType>
                <xs:attribute name="id" type="xs:unsignedByte"
                              use="required" />
                <xs:attribute name="url" type="xs:string"
                              use="required" />
                <xs:attribute name="ancho" type="xs:unsignedShort"
                              use="required" />
                <xs:attribute name="alto" type="xs:unsignedShort"
                              use="required" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 16: Fragmento de esquema de la característica Imágenes

Como se puede ver en el ejemplo de Código 17 el segundo nodo hijo es `<listado>`, el cual contiene un nodo hijo llamado `<item>` por cada imagen

analizada y los atributos son comparables con los analizados para la característica *flash*, ellos son un *id* numerado correlativamente, un atributo *url* que contiene la dirección de la imagen, un atributo *alto* y un atributo *ancho* que dan la medida en píxeles de la imagen.

```
<imagenes>
  <resultado cantidad="13" superficie="74566" />
  <listado>
    <item id="1"
          url="http://upload.wikimedia.org/wikipedia/commons/thumb
               /f/fa/Padlock-silver-medium.svg/16px-Padlock-silver-
               medium.svg.png" ancho="16" alto="16" />
    <item id="2" url="http://upload.wikimedia.org/wikipedia/commons/
                      thumb/c/cf/Title_page_of_Sir_Henry_Billingsley
                      %27s_first_English_version_of_Euclid%27s_
                      Elements %2C_1570_%28560x900%29.jpg/180pxTitle
                      _page_of_Sir_Henry_Billingsley%27s_first
                      _English_version_of_Euclid%27s_Elements
                      %2C_1570_%28560x900%29.jpg"
          ancho="180" alto="289" />
    …
    <item id="12" url="/skins-1.5/common/images/poweredby_mediawiki_
                       88x31.png" ancho="15" alto="16" />
    <item id="13" url="/images/wikimedia-button.png" ancho="15"
          alto="16" />
  </listado>
</imagenes>
```

Código 17: Ejemplo de extracción de Imágenes

Para obtener el tamaño de las imágenes se recurre a los valores contenidos en los atributos del HTML *width* y *height* en el caso que estén presentes. Como buena práctica se recomienda que siempre estén explícitos estos atributos para que el navegador pueda hacer el render de la página de manera más eficiente antes de cargar efectivamente las imágenes de su fuente. En el caso que estos atributos no estén presentes se recurre a recuperar la imagen y medirla. Este procedimiento subsana el hecho de que el estándar de HTML permite obviar estos atributos, pero realiza un procedimiento más lento y con mayor consumo de ancho de banda.

3.3.9. Característica: Meta

Dentro de las características propuestas para extraer en este trabajo, ésta es la última, y de haber un orden estandarizado en la Web sería una de las más importantes. Las etiquetas `<meta>` se crearon para agregar metadatos al contenido

Web y que éste pueda ser analizado de forma automatizada, por lo tanto es una de las características que se extraen en este prototipo. Según el esquema mostrado en el fragmento de Código 18 se puede observar que el nodo `<meta>` tiene nodos hijos, llamados `<resultados>`, uno por cada etiqueta extraída de la página. Cada uno de estos nodos `<resultados>` consta de varios atributos, un atributo *id* numerado correlativamente, un atributo *nombre*, que contiene el nombre del meta y un atributo *valor* con su contenido como se puede apreciar en el ejemplo de Código 19.

```
<xs:element name="meta">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="resultado">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="nombre" type="xs:string"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
                        />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 18: Fragmento de esquema de la característica Meta

```
<meta>
  <resultado id="1" nombre="robots" valor="all" />
  <resultado id="2" nombre="distribution" valor="global" />
  <resultado id="3" nombre="rating" valor="general" />
</meta>
```

Código 19: Ejemplo de extracción de Meta

Desafortunadamente hay muchas de estas etiquetas que se van agregando porque los buscadores las requieren o por comodidad de algunos generadores de páginas, sin estar avalados por ningún estándar.

Para la extracción de la misma se utiliza la expresión XPath `//meta[@name!= 'description' and @name!='keywords' and @name!= 'clasification']` con la cual se evita extraer nuevamente los metadatos de las características *Descripción*, *Clasificación* y *Keywords* mencionadas anteriormente.

3.3.10. Índices

Como segundo paso de este trabajo se hace un análisis de los datos obtenidos de las características extraídas y se obtienen diversos indicadores o índices. Los resultados de este análisis se guardan dentro del mismo XML resultante, como una característica especial, en un nodo hermano del resto de las características llamado `<indices>`. Como se aprecia en el esquema mostrado en el Código 20, este nodo consta de una estructura que contiene un nodo hijo por cada índice calculado. En el nodo de cada índice, llamado `<indice>`, se observan los atributos *id* como un numerador correlativo, *indicador*, que es el nombre del índice en el atributo, y *valor* que contiene un número de doble precisión resultado del cálculo realizado en cada caso.

```
<xs:element name="indices">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="indice">
        <xs:complexType>
          <xs:attribute name="id" type="xs:unsignedByte"
                        use="required" />
          <xs:attribute name="indicador" type="xs:string"
                        use="required" />
          <xs:attribute name="valor" type="xs:string" use="required"
                        />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Código 20: Fragmento de esquema los Índices

```
<indices>
  <indice id="1" indicador="locales/cantidad" valor="0" />
  <indice id="2" indicador="externos/cantidad" valor="1" />
  <indice id="3" indicador="superficieMedios" valor="72066" />
  <indice id="4" indicador="cantidadMedios" valor="5" />
  <indice id="5" indicador="bytesTexto/bytesTotal"
          valor="0,205907844697975" />
  <indice id="6" indicador="ARI" valor="9,02904625966553" />
  <indice id="7" indicador="SMOG" valor="16,64173031801014" />
  <indice id="8" indicador="GFI" valor="12,92144098783193" />
  <indice id="9" indicador="CLI" valor="13,9551305970149" />
  <indice id="10" indicador="Flesh" valor="87,728435533178" />
  <indice id="11" indicador="FleshKincaid" valor="7,95191841994" />
</indices>
```

Código 21: Ejemplo de extracción de Índices

En el Código 21 se puede ver un ejemplo del resultado de este análisis para un caso particular.

En esta tesina se propone el cálculo de once indicadores, que pueden ser utilizados en la siguiente etapa de análisis. Los indicadores propuestos son:

1. **LOCALES/CANTIDAD:** es la relación entre los enlaces que apuntan al mismo dominio de la página sobre el total de enlaces que la misma posee.
2. **EXTERNOS/CANTIDAD:** es la relación entre los enlaces que apuntan a dominios externos de la página sobre el total de enlaces que la misma posee, junto con el indicador anterior deben sumar el 100% de los enlaces analizados.
3. **SUPERFICIEMEDIOS:** es la cantidad total de píxeles ocupados por elementos multimedia, en esta categoría se cuentan las *imágenes* y los contenidos embebidos de *flash* obtenidos en las características respectivas.
4. **CANTIDADMEDIOS:** es la cantidad de elementos multimedia contenidos en la página, se acumulan las cantidades de *imágenes* y *flash* contabilizadas en las características respectivas.
5. **BYTESTEXTO/BYTESTOTAL:** es la relación entre la cantidad de caracteres que contiene la página y la cantidad total de bytes.
6. **ARI:** este indicador, al igual que los siguientes, es un índice de legibilidad. Como se describió en el capítulo anterior, estos índices muestran el nivel intelectual que se requiere para comprender efectivamente el contenido de la página. Es el resultado de calcular el *Índice Automatizado de Legibilidad.*
7. **SMOG:** es el resultado de calcular el índice de la medida simple de Gobbledigook al contenido textual de la página.
8. **GFI:** Es el resultado de calcular el índice de Gunning Fog al contenido textual de la página.
9. **CLI:** es el resultado de calcular el algoritmo de Coleman-Liau al contenido de texto de la página analizada.
10. **FLESH:** es el resultado de aplicar el algoritmo del test de lectura de Flesh al contenido textual de la página bajo análisis.
11. **FLESHKINCAID:** es el índice resultante de aplicar el cálculo del test de nivel de grado propuesto por Flesh-Kincaid.

3.4. Arquitectura

En esta tesis se propone una arquitectura modular. Consta de varios módulos que representan las partes fundamentales del proyecto. La arquitectura básica se ve en la Figura 2. El usuario hace una consulta en la página principal del prototipo indicando además la cantidad de páginas de los resultados del buscador que desea analizar y el criterio por el cual ordenar la respuesta. La página envía el formulario Web al servidor cuando el usuario presiona sobre el botón Buscar. La consulta es reformateada para que sea comprendida por el buscador que se use.

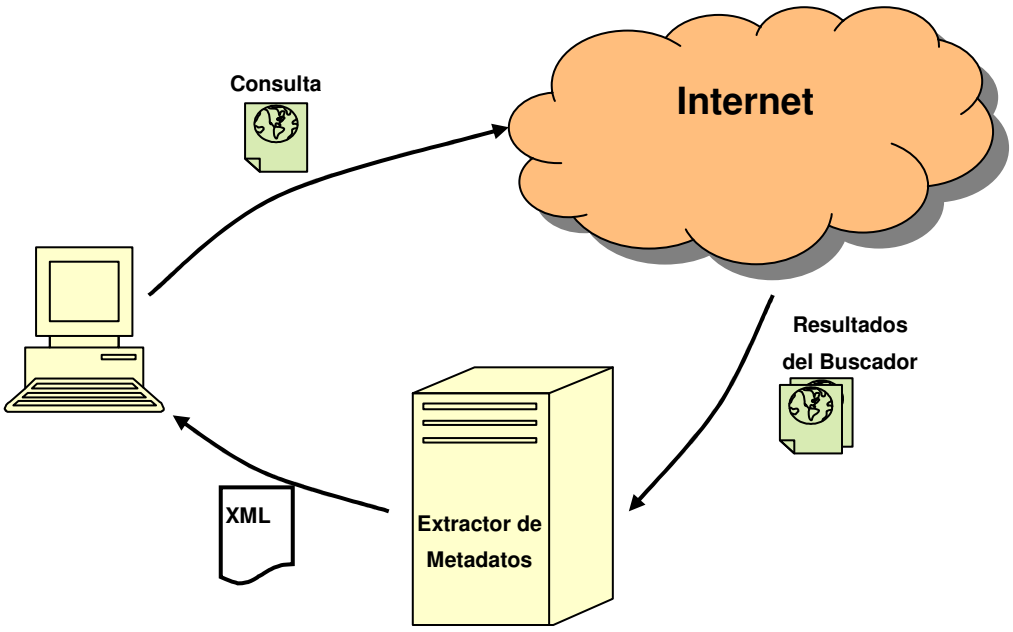

Figura 2: Funcionamiento básico

El formato de la consulta se compone de las opciones básicas que ofrece el buscador más la posibilidad de solicitar las búsquedas solamente dentro de un dominio especificado. Esto es necesario si se necesita traer la cantidad de resultados que el usuario requirió sólo de un dominio, pero puede omitirse. Una vez que el sistema obtiene todas las páginas de resultados, se extraen uno a uno los links obtenidos y se almacenan en una estructura de datos local.

La extracción de los links se realiza transformando en un XHTML la página con los resultados del buscador. Luego se consultan los nodos en el XHTML usando XPATH para obtener todos los links de las páginas de resultados del buscador.

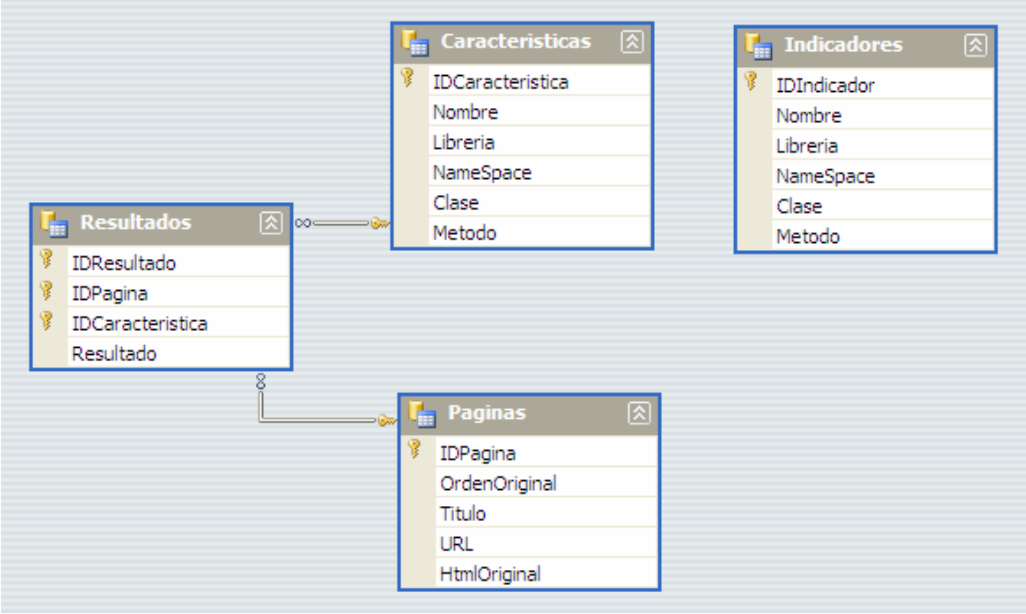

Figura 3: Documento principal de la aplicación usado como base de datos

Para administrar todos los datos obtenidos durante el proceso, se arma una estructura de datos en memoria, un DataSet* como el mostrado en la Figura 3. Este documento mantiene las tablas necesarias para todo el proceso, y es almacenado durante todo el ciclo de vida de la página.

Las tablas *Características* e *Indicadores* del DataSet son completadas en el comienzo del procesamiento usando un archivo de configuración† que describe las características que se pueden extraer según las librerías que dispone el sistema y los indicadores que ellas son capaces de calcular.

La idea fundamental que originó ese tipo de arquitectura es que se puedan agregar librerías con características programadas por cualquier desarrollador encapsuladas en una dll con indicadores eventuales que se puedan calcular. Para ello sólo se agregarían las dll al archivo de configuración y el sistema quedaría operando con la nueva funcionalidad. Esto origina una interacción más flexible entre los módulos como la mostrada en detalle en la Figura 4.

---

* Dataset: Se denomina así a cualquier conjunto de datos.

† En el Apéndice A se puede ver el listado completo del archivo de configuración usado para cargar inicialmente el DataSet.

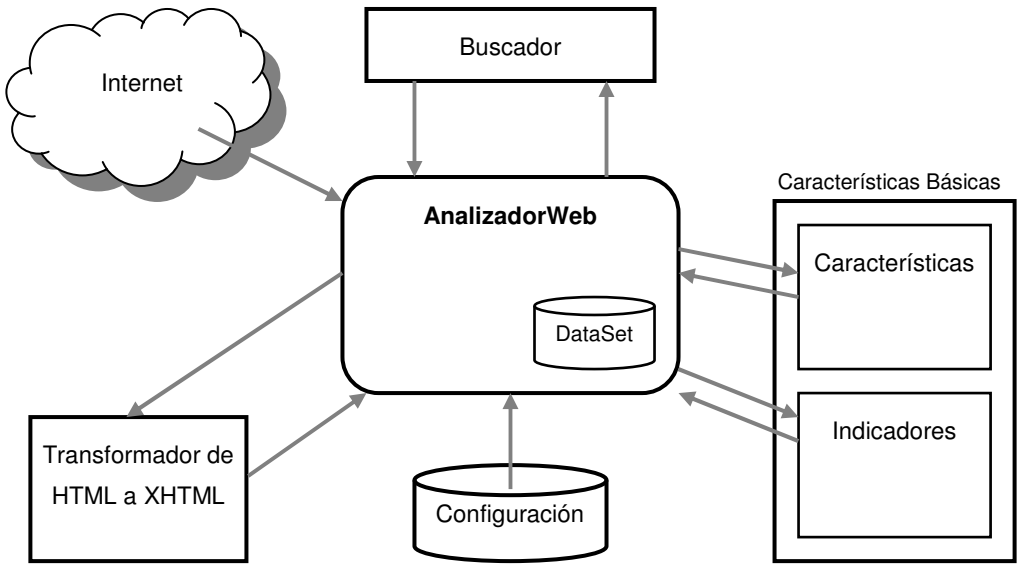

Figura 4: Diagrama de interacción entre los módulos

En las siguientes secciones se explicará más detalladamente el comportamiento particular de cada módulo.

El DataSet es el documento principal de la aplicación y sobre él trabajan todos los módulos, obteniendo y/o insertando datos recopilados. Cuando se obtienen las páginas resultantes del buscador éstas se almacenan en la tabla *Páginas* del DataSet, se le aplican todas las características disponibles listadas en la tabla *Características*, y por cada combinación de ellas se obtiene un registro que se inserta en la tabla *Resultados*.

El resultado insertado es una cadena de caracteres equivalente en texto al nodo XML que representa la característica sobre la página analizada. Luego en un proceso de recopilación se inserta esta cadena dentro del XML resultante para armar el resultado final del análisis.

3.4.1. Módulo Buscador

El primer módulo con el que se encuentra el usuario es el *Buscador.* La funcionalidad básica de este módulo es acumular el pedido del usuario, enviarlo al servidor cuando se presiona buscar y recibir el resultado como un archivo XML para

que el usuario lo almacene localmente. El diagrama de funcionamiento se puede apreciar en la Figura 5.

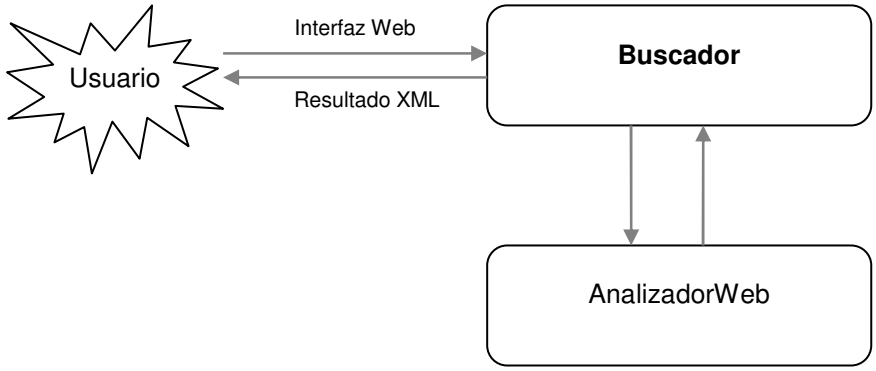

Figura 5: Modulo Buscador

La interacción con el usuario se hace por medio de una página Web que recauda la consulta y preferencias del usuario y la envía al AnalizadorWeb para que sea procesada. El AnalizadorWeb devuelve el XML con el resultado del proceso y el módulo Buscador lo devuelve al usuario como un archivo XML que se descarga desde el navegador.

3.4.2. Módulo AnalizadorWeb

El módulo más importante es el *AnalizadorWeb*, que tiene como objetivo administrar todos los movimientos de datos entre las librerías que analizan el contenido de las páginas, y generar el XML resultante de todo el proceso de análisis.

Este módulo carga las librerías que contienen las funciones para extraer y procesar las características de las páginas de acuerdo al archivo de configuración.

El procesamiento en este módulo comienza cuando el módulo *Buscador* le envía el pedido para analizar pasando como parámetro la consulta que se usa, la cantidad de resultados requeridos, y el indicador por el que se solicita ordenar el resultado.

Como segundo paso realiza la búsqueda en Internet a través de un buscador Web para obtener el conjunto de páginas a analizar. Para ello debe analizar las páginas de resultados obtenidas del buscador y extraer los resultados, estos resultados serán las páginas que luego serán analizadas por el resto de los módulos.

El tercer paso del *AnalizadorWeb* es enviar cada página obtenida al módulo transformador de HTML a XHTML para obtener una estructura fácilmente analizable.

En el siguiente paso se envían estas estructuras XHTML para ser procesadas por el módulo *CaracterísticasBásicas*, que extrae los metadatos según las funciones del archivo de configuración. Como consecuencia, estas funciones devuelven una porción de código XML que corresponde al resultado de su análisis. Estas porciones de XML son devueltas al módulo *AnalizadorWeb*, el cual las recopila para luego ser enviadas nuevamente al módulo *CaracterísticasBásicas* que se encarga de calcular los índices y devolverlos al *AnalizadorWeb*.

Luego el *AnalizadorWeb* recopila el XML armado para cada página y lo compagina en un único XML que es enviado al módulo *Buscador* para que sea devuelto como archivo al usuario.

3.4.3. Módulo Transformador HTML en XHTML

Este módulo cumple con un requerimiento básico para el correcto funcionamiento del desarrollo. Una de las actividades más complicadas de llevar a cabo en el análisis de código HTML es la diversidad de estilos de código aceptados por el estándar. Para facilitar las tareas de análisis se creo el estándar XHTML que encapsula todas las propiedades de un HTML pero en una estructura más rígida sin que por ello sea menos expresiva.

Este módulo es precisamente el encargado de parsear el HTML original de cada página analizada y rearmarlo en una estructura XHTML.

En la Figura 6 se puede ver el diagrama representativo que describe este procedimiento.

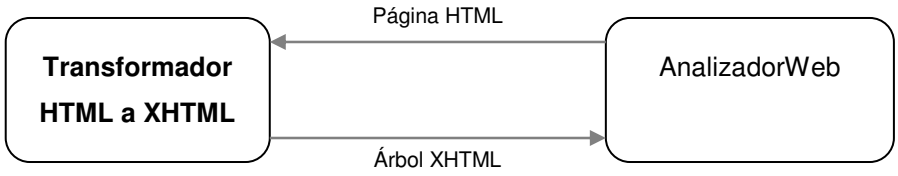

Figura 6: Módulo Transformador de HTML a XHTML

El resultado final de este módulo es un árbol XML, particularmente un XHTML que luego puede ser manipulado fácilmente por los siguientes procesos del desarrollo.

3.4.4. Módulo CaracteristicasBasicas

El módulo que contiene las características es el que aloja todo el poder de análisis y cálculo del desarrollo. Este módulo contiene la funcionalidad de cada característica analizada y de los índices calculados entre los resultados de las mismas.

El módulo está compuesto por una librería con las funciones para extraer y analizar cada una de las características propuestas. La interacción se lleva a cabo con el módulo *AnalizadorWeb* a través de llamados a las funciones que hacen cada cálculo. Para poder llamar a las funciones del módulo estas deben implementar una interfaz común a todas.

En la

Figura 7 se puede observar la interacción de este módulo con el resto del desarrollo.

La arquitectura del desarrollo permite que se agreguen más librerías de funciones para extraer otros metadatos diferentes, el módulo *CaracterísticasBásicas* presentado tiene la funcionalidad de extracción de los metadatos planteados por esta tesina. Más adelante en la subsección 3.5 (Implementación de Prototipo) se detalla cada una de las funciones implementadas en este módulo.

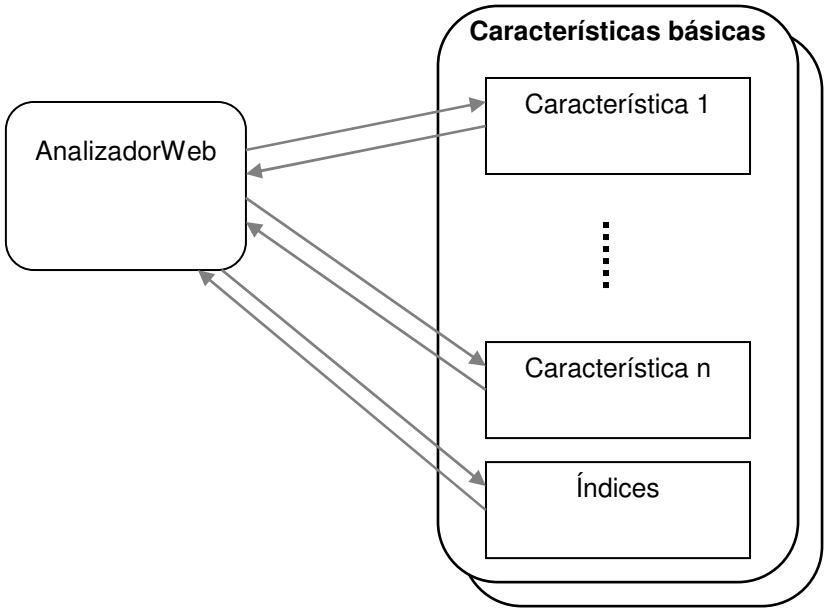

Figura 7: Módulo Características Básicas

3.5. Implementación de Prototipo

El prototipo desarrollado se presenta como una aplicación Web, creada con ASP.NET, servida por un *Internet Information Server*. Está desarrollado con Visual Studio 2005 y con el framework de .NET 2.0. Como lenguaje de programación se optó por C# por ser un lenguaje moderno de amplias facilidades. El diseño del prototipo se basa en una página Web que pide al usuario el ingreso de la consulta que desea realizar en el buscador Google, el sitio específico donde quiere hacer la búsqueda (o nada si quiere realizarla en la Web en general), la cantidad de páginas que se quieren analizar del total de los resultados obtenidos y el orden preferido para la devolución de los resultados de acuerdo a ciertos índices obtenidos, como se muestra en la Figura 8.

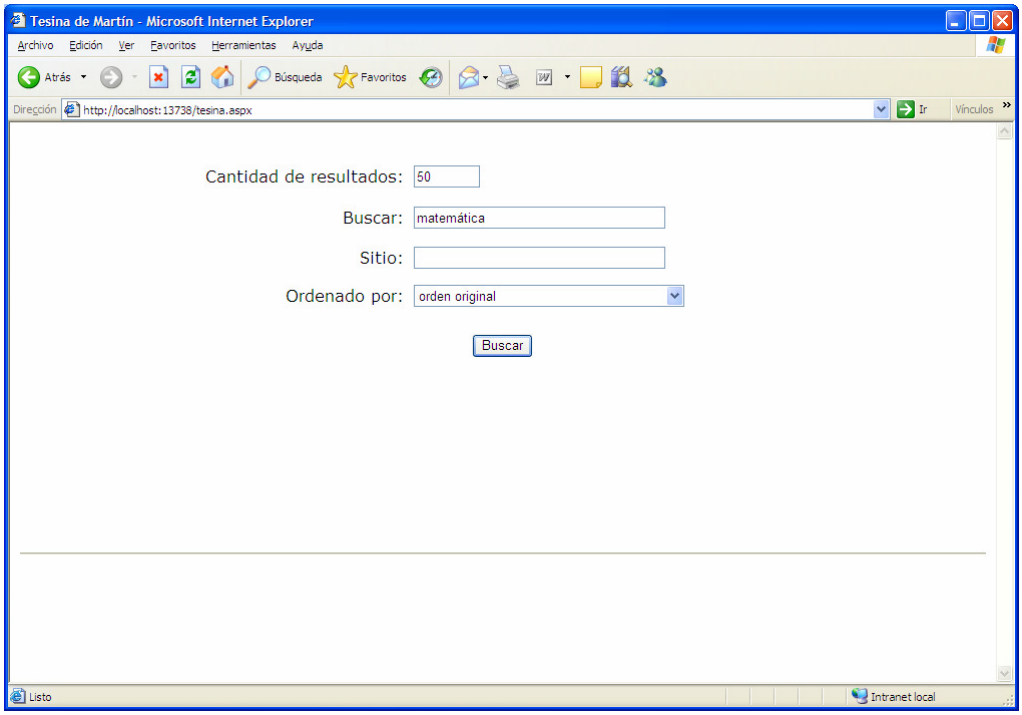

Figura 8: Página principal del prototipo

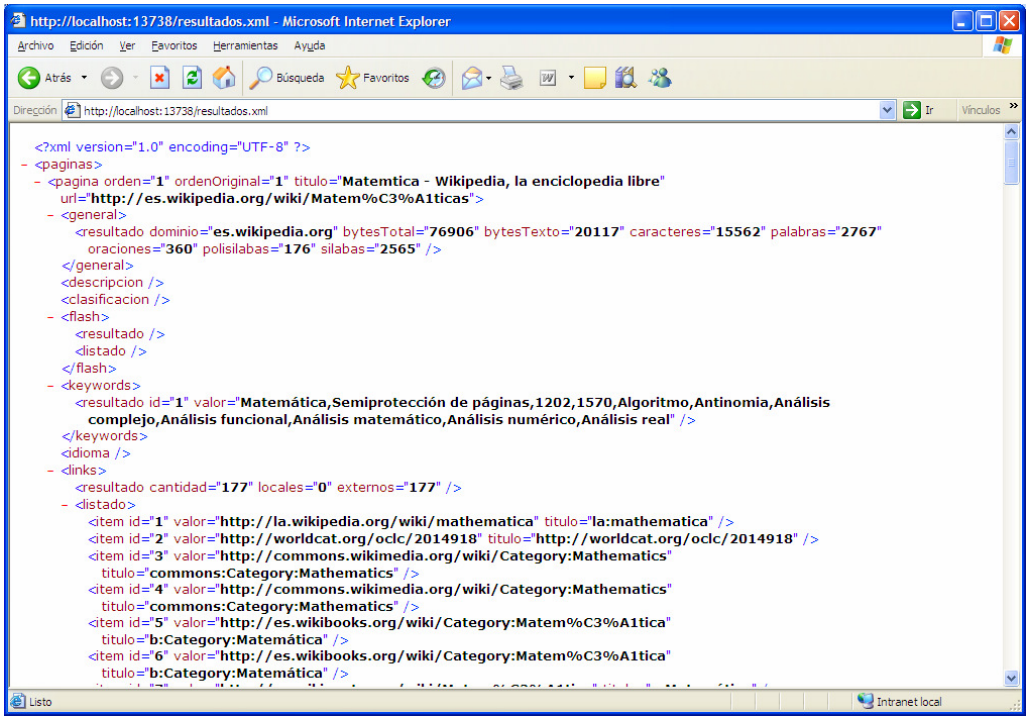

Figura 9: Resultado final

La página tiene métodos para capturar los eventos emitidos desde el navegador que está utilizando el usuario. El evento más importante de capturar y que desencadena todo el proceso es el clic del botón Buscar.

La solución usada para implementar el prototipo consta de cuatro proyectos a detallar:

1. **BUSCADOR:** proyecto Web que consta de una página inicial dinámica ASP.NET para la interacción del prototipo con el usuario. En ella se piden los requerimientos a través de un formulario Web, y como resultado desde el servidor se responde con un archivo `resultado.xml` con la información de peticiones solicitadas.
2. **ANALIZADORWEB:** proyecto de librería de objetos de .NET encargados de orquestar el análisis completo del prototipo. Es una dll[*] que contiene las clases principales del proyecto para administrar los datos y cálculos.
3. **HTMLAGILITYPACK:** proyecto de librería de objetos de .NET encargados en conjunto de modelar un XHTML armado desde el HTML original de la página analizada. Es un proyecto de código abierto que se utiliza para análisis de contenidos Web.
4. **CARACTERISTICASBASICAS:** proyecto de librería de objetos de .NET encargado de brindar la funcionalidad específica requerida por el analizador para procesar cada característica que el diseñador requiera. Esta librería se carga en tiempo de ejecución de la aplicación de forma dinámica, permitiendo que cualquier desarrollador pueda agregar más librerías por configuración.

En las siguientes subsecciones se describen en profundidad cada unos de los proyectos que componen el prototipo, con detalles de la implementación de cada proyecto que compone la solución de acuerdo a la arquitectura planteada en la sección 3.4.

[*] DLL: del inglés *Dynamic Linking Library*, es una Bibliotecas de Enlace Dinámico, término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa.

3.5.1. Proyecto Buscador

El proyecto buscador esta compuesto por las clases correspondientes a la página Web que se le ofrece al usuario para interactuar con el prototipo. El diagrama de clases mostrado en la Figura 10 es el correspondiente a la clase *_Default*, que contiene el comportamiento de la página Web del lado del cliente y del servidor cuando el operador interactúa con la página en el navegador.

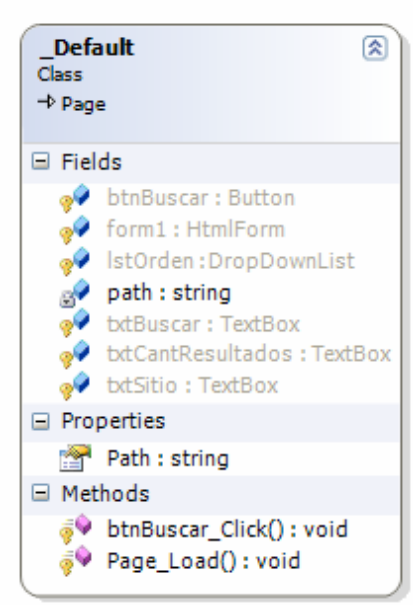

Figura 10: Diagrama de clase del módulo Buscador

El evento *Page_Load()* es utilizado para preparar la página con valores por defecto, y cargar el combo de indicadores y demás comportamientos anexos al núcleo del propósito del prototipo.

La propiedad *path* mantiene durante todo el ciclo de vida de la página el lugar físico donde se encuentra alojada la misma para poder ubicar los archivos de configuración y las librerías de comportamientos que sean configuradas para realizar el análisis.

Cuando el usuario pulsa el botón Buscar de la página se ejecuta el evento *btnBuscar_Click()*. Este evento genera una instancia del módulo *AnalizadorWeb* pasándole a su constructor los parámetros *cadenaBuscada*, *cantResultados, sitio* y *atributo*.

Los demás archivos que componen este módulo son de configuración propios de páginas dinámicas desarrolladas con .Net que no vale la pena mencionar en este trabajo ya que exceden los objetivos del mismo.

3.5.2. Proyecto Analizador Web

Cuando el módulo buscador instancia al *AnalizadorWeb* se reciben los parámetros que son almacenados en las propiedades *cadenaBuscada*, *cantResultados, sitio* y *atributo* respectivas de la clase Analizador. Los mismos se pueden observar en la Figura 11.

La instancia *Analizador* creada por el Buscador, llama a los métodos públicos *Buscar()*, *Extraer()*, *Analizar()* y *Ordenar()*, los cuatro pasos fundamentales que componen al analizador.

El primero de ellos, el método *Buscar()*, utiliza las clases del HtmlAgilityPack para armar un documento XHTML con la página del buscador (la página de Google que contiene las páginas encontradas). Una vez hecho esto, se hace una consulta sobre el XHTML armado en el paso anterior para extraer las URLs de los enlaces que devolvió el buscador.

En este caso el prototipo está preparado para hacer la consulta a Google pero como trabajo futuro se podría extender esto a cualquier otro buscador.

La consulta que se realiza es `http://www.google.com/search?hl=es&q={0}&start={1}` donde `{0}` es reemplazado por la cadena completa que el usuario ingresó en el formulario y `{1}` se reemplaza por el número de resultado a partir del cual el buscador devuelve las páginas. . En caso que el usuario haya especificado un sitio Web concreto donde buscar, éste se anexa a la consulta con la cadena `/site:{2}` donde `{2}` es el dominio del sitio especificado. La extracción de los links se realiza haciendo una transformación de las páginas del buscador obtenidas y trasformándolas en un XHTML usando HtmlAgilityPack. Luego se consulta en el XHTML los nodos que satisfagan la consulta XPATH `//*[@href][@class='l']`. Esta consulta asegura obtener todos los links de las páginas de resultados de Google

que están distinguibles por un `<DIV>` característico con el estilo asociado `class='l'`.

El resultado de este proceso se va almacenando hasta completar la cantidad de enlaces que el usuario requiere analizar y se guardan en la tabla *Páginas* del DataSet planteado en la sección 3.4, llamado *DocAnalizador*.

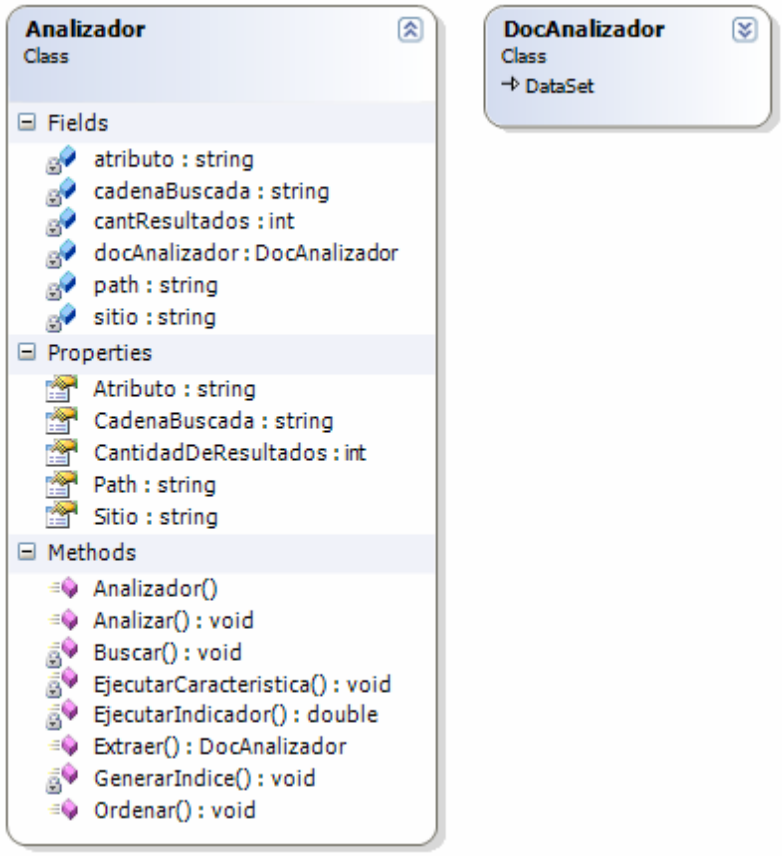

Figura 11: Diagrama de cases del módulo AnalizadorWeb

El método *Extraer()* es el encargado de calcular todas las características configuradas de las páginas encontradas. Para ello se utilizan las funciones del HtmlAgilityPack que arman un documento XHTML con el contenido de cada página, al que se accede mediante las URLs almacenadas en el paso anterior en el DataSet. A este XHTML se le aplican secuencialmente las funciones para extraer las características listadas en la tabla *Características* del DataSet.

Toda esta información recopilada se vuelca dentro del DataSet *DocAnalizador*. Cada característica provee como resultado una porción del XML final, y se va almacenando en éste, hasta completar todas las iteraciones.

El método *Analizar()* es el que cumple con la funcionalidad de crear los índices configurados en la tabla *Indicadores*. Los índices son los mencionados en la Sección 3.3.10. Estos se ejecutan de forma secuencial sobre los resultados almacenados en el XML armado hasta el momento por los procesos anteriores. Sobre el nodo de cada página se agrega cada uno de los índices ejecutados hasta completar todos los indicadores para todas las páginas existentes en el DataSet *DocAnalizador*.

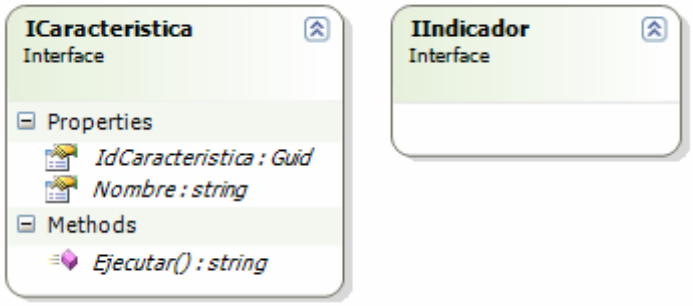

Figura 12: Diagrama de clases de las interfaces

El último método que provee el analizador es *Ordenar()*, que toma como parámetro el nombre del indicador que se va a utilizar como criterio de ordenación y lo lleva a cabo. La ordenación se hace sobre un nuevo atributo de cada nodo `<pagina>` llamado *Orden*. Para la ordenación de los nodos se usan las funciones provistas por el framework de .NET. Usando los tipos de datos Generics[*], se generan listas de nodos XML con la sentencia: `List<XmlNode> ordenado = new List<XmlNode>();`. Las listas tienen el algoritmo de ordenación incluido, solo hay que pasarle el criterio de comparación como un delegado[†] de una función que dé el orden de los elementos. En este caso, la función que realiza dicha acción toma el valor del indicador de dos nodos y devuelve el resultado de la comparación. El Código 22 muestra como se realiza esto.

[*] Los tipos de datos Generics son como las plantillas de C++

[†] Los delegados son punteros a funciones con un prototipo definido

```
ordenado.Sort(delegate(XmlNode x1, XmlNode x2)
{
    Return
double.Parse(x1.SelectSingleNode("indices/indice[@indicador='" +
this.atributo + "']").Attributes["valor"].Value).
    CompareTo(
double.Parse(x2.SelectSingleNode("indices/indice[@indicador='" +
this.atributo + "']").Attributes["valor"].Value));
});
```

Código 22: Acceso al atributo y ordenación

3.5.3. Proyecto HtmlAgilityPack

El módulo transformador de HTML a XHTML fue implementado en este prototipo por una librería de código abierto desarrollada para tal fin. La librería se llama *HtmlAgilityPack* y fue adaptada para incorporarla a la solución. Se desarrolló para encapsular el código HTML tomado de cualquier página Web y transformarlo en un XHTML, es decir, en un XML bien formado que puede ser navegado como un árbol y ser manipulado con mayor comodidad.

Este módulo brinda facilidades similares a las ofrecidas por el espacio de nombres *System.XML* propios del framework de .NET, para comodidad del usuario. Entre ellas la propiedad más importante por la que se tomó esta librería es porque permite navegar los árboles generados con una herramienta fundamental para la extracción de datos de un árbol, las expresiones XPATH.

Mediante una cadena de texto se puede representar un criterio para seleccionar y buscar un nodo o una lista de nodos que cumplan con el criterio requerido.

Lo más importante de esta herramienta es que es flexible y se adapta a los HTML reales que hay en Internet, ya que las exigencias para el código HTML son mucho más flexibles en términos sintácticos que lo que se requiere para un XML. Por este motivo la W3C se movilizó para crear otro estándar que abarque al HTML y brinde la posibilidad de crear *parsers* y navegadores Web más eficientes basados en recursiones aplicadas sobre árboles. De todas maneras, hasta que no sea un estándar generalizado, siguen conviviendo páginas HTML que se pueden interpretar por los navegadores actuales pero son muy difíciles de parsear. Este tipo de obstáculos hacen

que *HtmlAgilityPack* sea una herramienta ideal para tratar documentos buscados en la Web y por ese motivo fue utilizada en este prototipo.

3.5.4. Proyecto Características Básicas

En este prototipo, en la librería `CaracteristicasBasicas.dll` están contenidos las características y los indicadores mencionados en la sección anterior, los cuales se listan en el archivo de configuración del Apéndice A.

Como se puede observar en la Figura 13 el diagrama de clases del módulo *CaracteristicasBasicas* contiene una clase por cada característica que se implementó para analizar en la dll. Todas las clases que analizan la página deben implementar la interfase `ICaracteristica` definida en el módulo *AnalizadorWeb* para poder ser ejecutadas en el ciclo del análisis.

De acuerdo con la Interfase `ICaracteristica` las clases deben tener principalmente un método llamado *Ejecutar()* que toma como parámetro la URL de la página y el documento XHTML generado por la librería *HtmlAgilityPack* y devuelve la cadena de caracteres del XML del nodo que se está generando.

Este mismo proceso ocurre con cada clase contenida en la librería. Y cada método *Ejecutar()* realiza las tareas particulares para lo que fue diseñado en cada clase.

También hay una clase particular llamada *Indicadores* que implementa un método por cada *indicador* que se puede calcular. Estos métodos deben tener una firma* particular para que puedan ser llamados desde el módulo *AnalizadorWeb*. La firma de estas funciones debe recibir el documento XML por si necesitara todo el documento completo y el nodo del documento XML de la página que se esta analizando y como resultado devuelve un número de doble precisión.

* En .Net la firma de una función es el prototipo de la misma

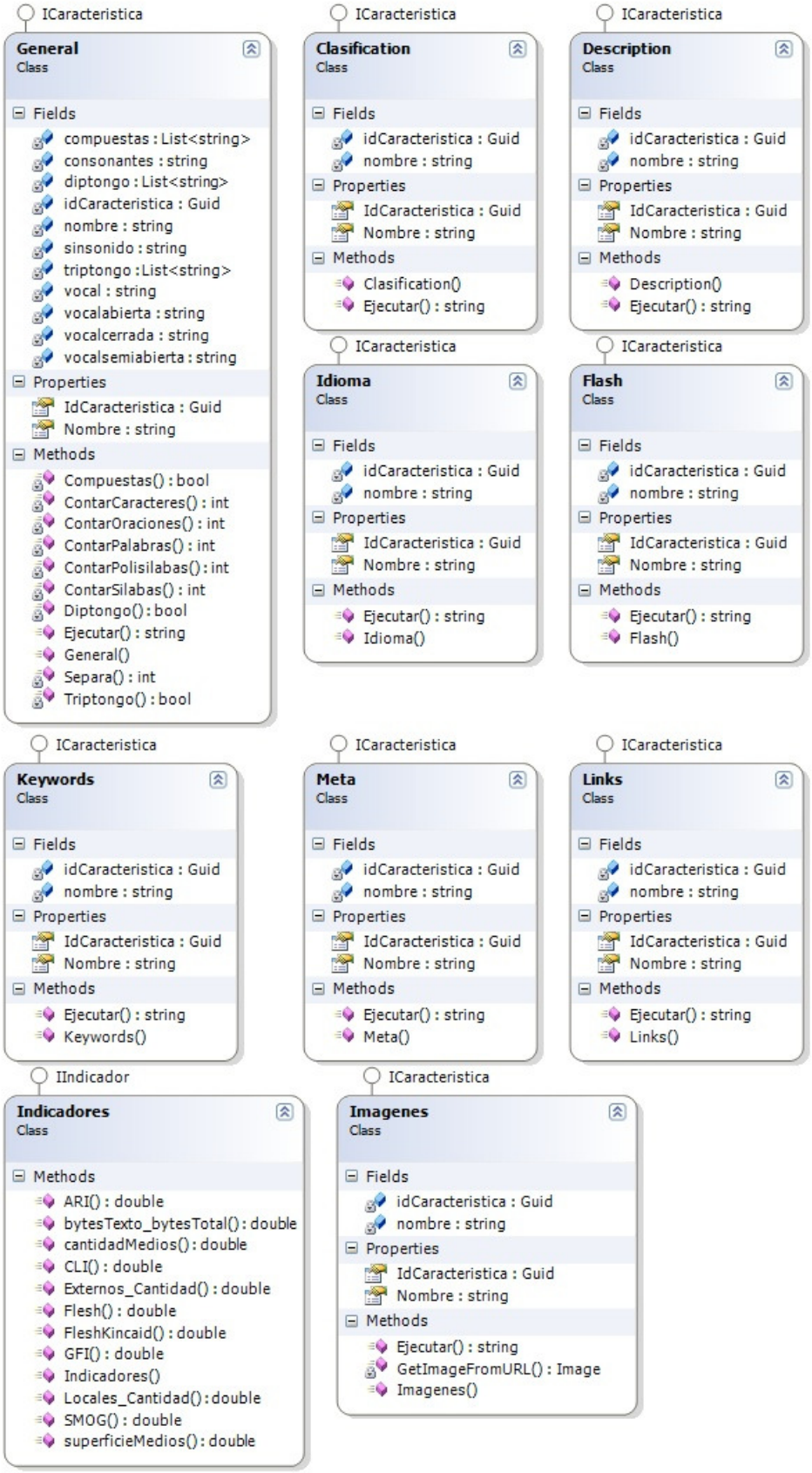

Figura 13: Diagrama de clases del módulo CaracteristicasBasicas

Para que estas funciones sean llamadas deben estar en el archivo de configuración del prototipo. Y éstas serán ejecutadas mediante las facilidades que ofrece el framework de hacer *reflection* sobre la librería.

En la sección anterior se dio una descripción más detallada de cada característica analizada. Esta librería es la básica que necesita el prototipo para funcionar, pero como premisa está la posibilidad de armar cualquier otra librería con la arquitectura de `CaracteristicasBasicas.dll` según se explicó y se aumentará la funcionalidad del analizador acorde a los nuevos desarrollos.

3.6. Algoritmo de separación en sílabas

En este prototipo se utilizaron algunos algoritmos interesantes para la obtención de los distintos índices de legibilidad. Tal es el caso del algoritmo de separación en sílabas para el idioma castellano estudiado por Figueroa en su tesis de licenciatura [Figueroa, 1998].

En ella se describen las reglas del idioma español para su formación, así como su estructura y clasificación. También se plantea un algoritmo para realizar la segmentación de una palabra en sílabas. Se llegó a la conclusión que hay 10 reglas para la separación en sílabas que se describen en la Tabla 2.

En la tesis se agrupan las letras en vocales (V) y consonantes (C) y se reconocen las consonantes compuestas como br, gr, bl, etc. y los diptongos y triptongos de vocales existentes en el idioma castellano. Luego se crea un mapa de las vocales y consonantes que tiene la palabra y se comienza con el algoritmo del Diagrama 1.

Regla	Condición
1	En las sílabas, por lo menos, siempre tiene que haber una vocal. Sin vocal no hay sílaba.
2	Cada elemento del grupo de consonantes inseparables, no puede ser separado al dividir una palabra en sílabas.
3	Cuando una consonante se encuentra entre dos vocales, se une a la segunda vocal.
4	Cuando hay dos consonantes entre dos vocales, cada vocal se une a una consonante. Excepto si cumple con la regla 2
5	Si son tres las consonantes colocadas entre dos vocales, las dos primeras consonantes se asociarán con la primera vocal y la tercera consonante con la segunda vocal. Esta regla no se cumple cuando la segunda y tercera consonante forman parte del grupo de consonantes inseparables.
6	Las palabras que contienen una h precedida o seguida de otra consonante, se dividen separando ambas letras.
7	La unión de dos vocales que no forman diptongo, deben separarse en la segmentación silábica. Pueden quedar solas o unidas a una consonante.
8	La h entre dos vocales, no destruye un diptongo.
9	La acentuación sobre la vocal cerrada de un diptongo provoca su destrucción.
10	La unión de tres vocales forma un triptongo.

Tabla 2: Reglas de separación en sílabas

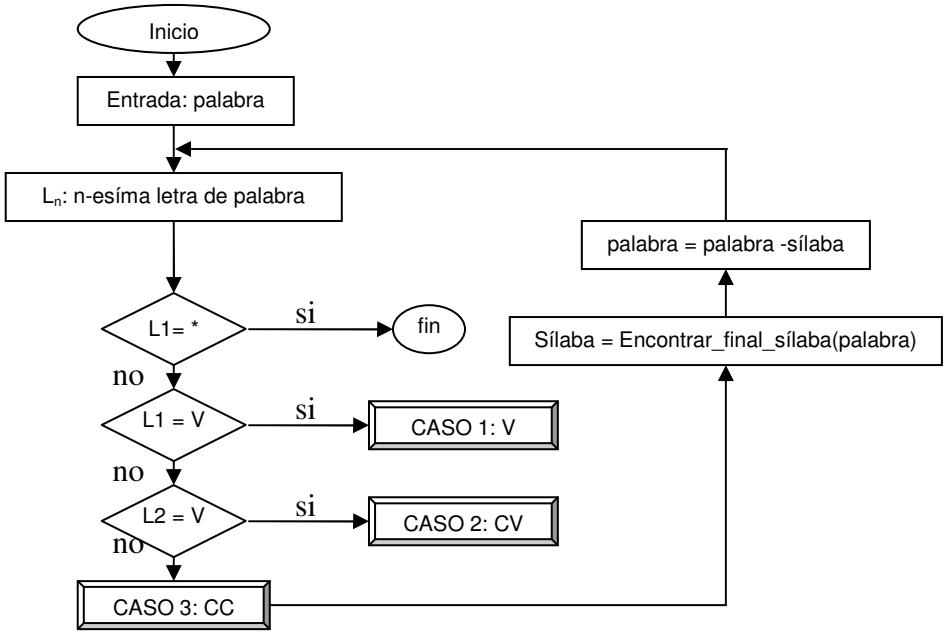

Diagrama 1: Flujo principal del separador de sílabas

Para cada uno de los tres casos que se ven en el diagrama hay un estudio que determina cuantas letras siguen en la palabra de acuerdo al mapa que siga. Las terminaciones según el caso se pueden ver en la Tabla 3.

En esta tesis, estas reglas fueron codificadas en C# para obtener el conteo de separación en sílabas de las palabras de cada página analizada y así aplicar los indicadores de legibilidad que basan su cálculo en el conteo de cantidad de sílabas sobre el texto.

Caso	Combinación	Ejemplo	Número de regla
1	V	a	1
1	VC	á – rbol	4
1	VV	au – tomóvil	7
1	VVC	aun – que	7,4
2	C	y	1
2	CV	la	1
2	CVC	las	1
2	CVCC	cons – tante	5
2	CVV	jau – la	7
2	CVVC	cuan – do	4
2	CVVV	cuau – tla	10
2	CVVVC	cuauh – temoc	10, 6
3	CCV	tra – po	2
3	CCVC	tras – to	2,4
3	CCVCC	trans – porte	2, 5
3	CCVV	trau – ma	2,7
3	CCVVC	claus – trofobia	2,7,5

Tabla 3: Reglas aplicadas para cada caso

Capítulo 4.

Experimentaciones realizadas

4.1. Caso de uso

Como aplicación práctica de este prototipo se presenta un ejemplo de un caso particular. Este ejemplo muestra un caso de búsqueda de tutoriales o cursos de plantillas en C++ por parte de un usuario que requiere un documento de nivel avanzado. Cabe aclarar que la búsqueda fue el resultado de aplicar el prototipo en la Web en general, y no en algún sitio repositorio de cursos.

Para esta ocasión se usó como consulta el texto “sintaxis plantillas c++”. Esta búsqueda, mostrada en la Figura 14, debe dar como resultado una mejor ordenación de las páginas de tutoriales y/o cursos dentro de los resultados retornados por Google. La misma consulta hecha en Google se puede ver en la Figura 15.

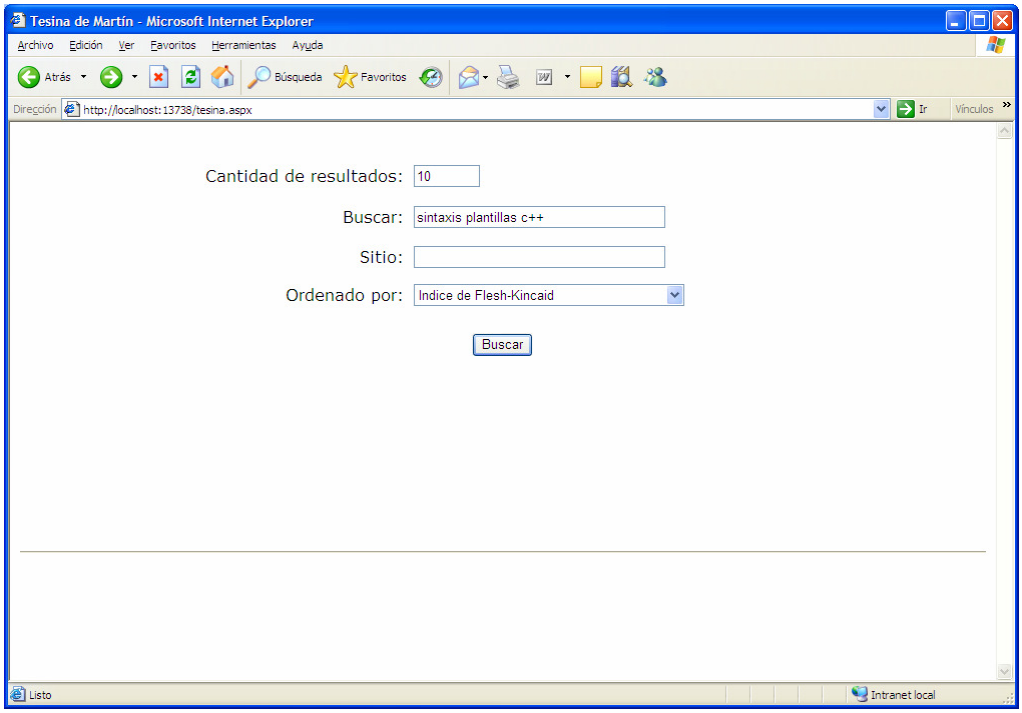

Figura 14: Ejemplo de búsqueda

En la búsqueda se pide que el ordenamiento se realice por el índice de Flesh-Kincaid. Como este índice da la facilidad de lectura de los documentos hay que tener en cuenta que a mayor valor, más complejo será el documento, y en el caso de páginas Web de este tipo, se podría inferir que esto implica que es mayor la calidad del texto y que el curso es más avanzado. Para este cálculo se utiliza solamente el texto de los párrafos significativos, que son los seleccionados del código HTML como el texto contenido entre etiquetas `<P></P>`.

Esto indica que la página a la que el prototipo le de mayor número de *Orden*, en este caso 10 por ser ese el número total de páginas requeridas, será la candidata a ser la mejor página para nuestra búsqueda. Esta página es la que Google devolvió en tercer lugar: “Cool C/C++ Programación en C/C++ - Tutoriales”.

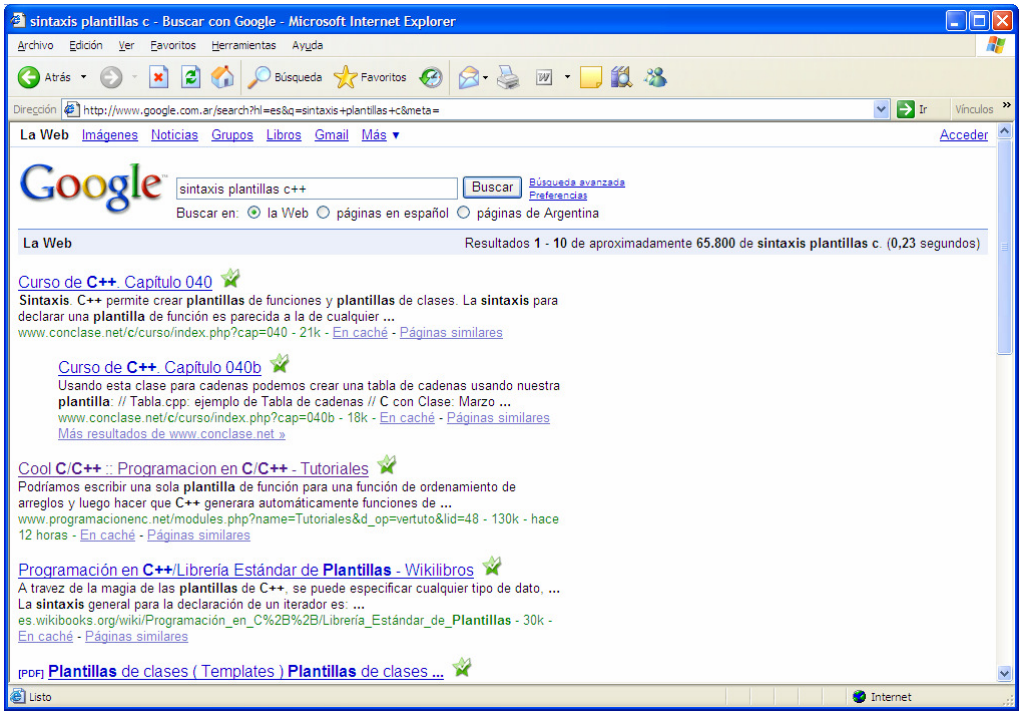

Figura 15: Resultados de la consulta en Google

En la Figura 16 se puede ver el XML expandido en la sección de índices para la página en cuestión y se puede apreciar que el indicador de Flesh Kincaid dio como resultado 9,84, un número alto en su escala. Además se ve que la página tiene 21.434 caracteres como texto principal lo cual implica que contiene bastante información de lo que se está solicitando.

Se observa que tiene pocos links y no tiene elementos embebidos de flash, lo que indicaría que no tiene publicidades o videos, se ve que todas las palabras extraídas como *keywords* se refieren en su mayoría a términos de informática.

Como se puede ver en la Figura 17, donde se listan todas las páginas solo con los atributos recolectados a nivel de página, Google le dio el tercer lugar a la página mencionada anteriormente y el prototipo le asigno un 10, lo que sería un primer lugar para nuestro contexto pues indica que es la página con mayor requerimiento de nivel intelectual.

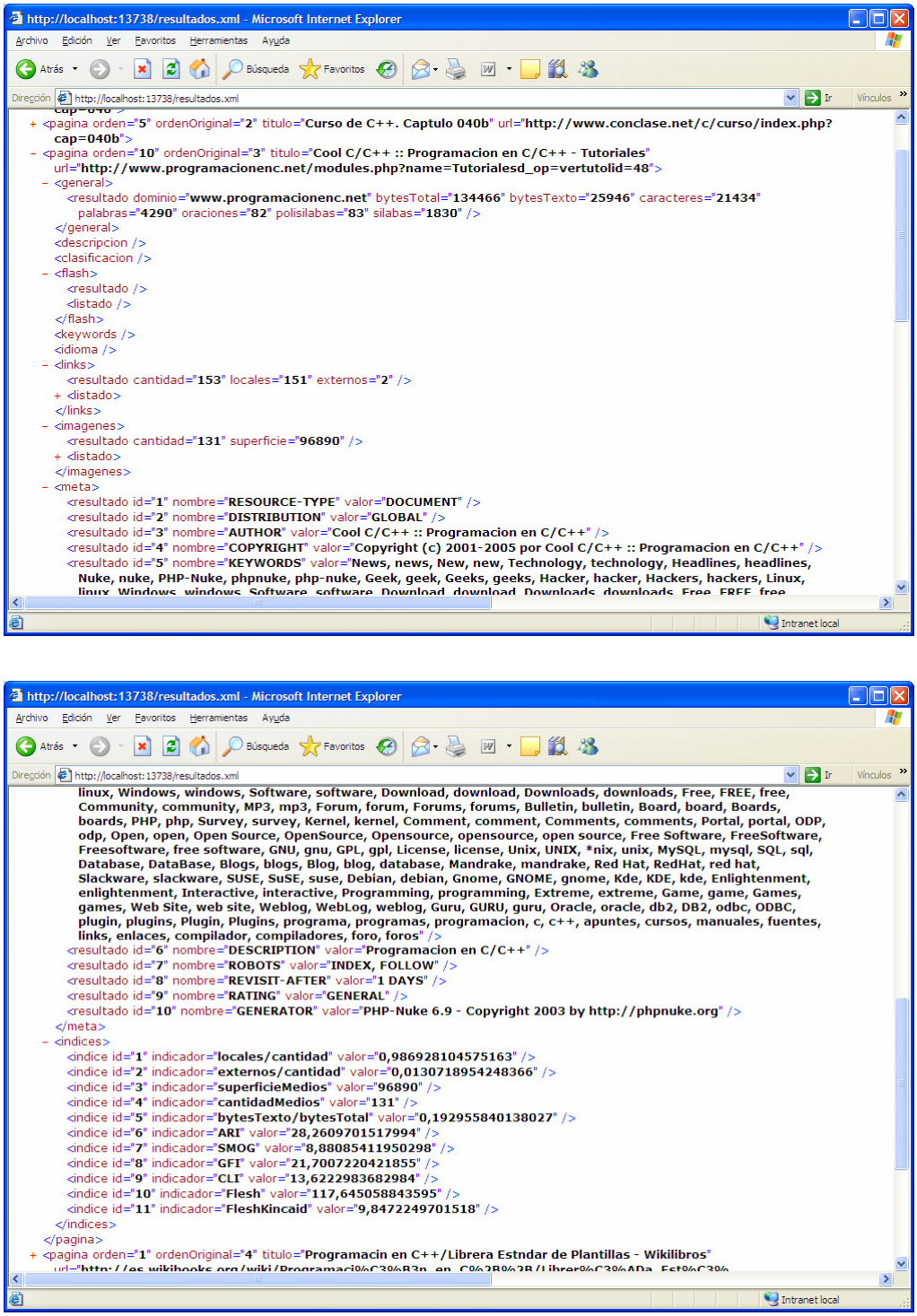

Figura 16: XML expandido en la página de mejor resultado

En la Figura 16 pueden observarse todas las características extraídas de la página seleccionada. La utilidad de este prototipo no reside únicamente en el orden que devuelve las páginas, sino que el usuario tiene a su disposición todas las características extraídas para un análisis posterior.

En la Figura 18 se muestra una captura de pantalla de la página elegida por el prototipo como la más adecuada para los requerimientos del usuario.

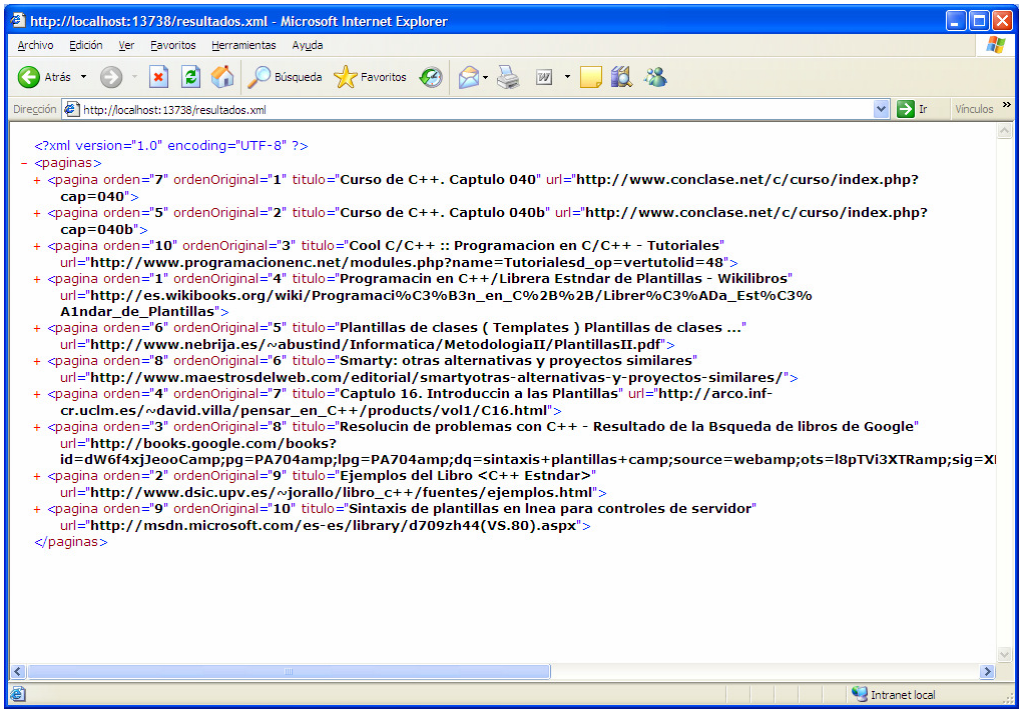

Figura 17: Resultado colapsado a nivel de páginas

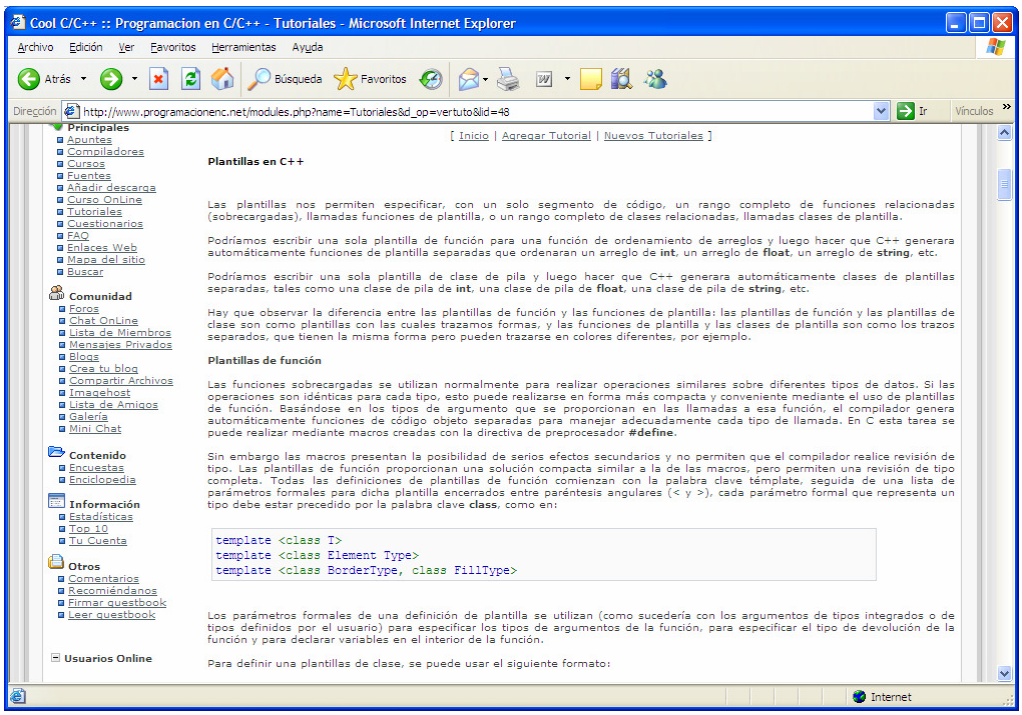

Figura 18: Muestra de la página

4.2. Extracción de los metadatos de las páginas Web

La aplicación del prototipo debe dividirse en dos casos bien diferenciados: la aplicación en la Web en general, y la aplicación dentro de sitios específicos, en los cuales las páginas tienen un estilo homogéneo. Esta diferenciación es muy importante a la hora de evaluar el rendimiento del prototipo propuesto, ya que en la Web las páginas pueden tener tanta cantidad de estilos distintos como número de desarrolladores Web existan. Esto es debido a que el estándar HTML es libre y abierto, y no impone limitaciones en la estructura de las páginas, lo cual por otro lado es la clave de su éxito.

Además, y esta es la característica mas importante que limita la capacidad del prototipo en la Web, hay muchas páginas devueltas por los buscadores que no se relacionan en nada con lo que se está buscando y pueden ser catalogadas erróneamente. Por ejemplo, al buscar "cursos de matemática" en Internet pueden aparecer páginas de propaganda de libros, páginas personales de algún usuario que expresa alguna anécdota, propagandas de institutos, etc. El prototipo puede calificar estas páginas según el texto, la cantidad de imágenes, el nivel intelectual necesario para leerlas, etc., pero obviamente cometerá errores porque no puede distinguir inteligentemente entre ellas, además de que algunos tipos de páginas, por ejemplo las de propaganda, carecen de texto.

En cambio si la búsqueda se hace sobre sitios específicos, por ejemplo repositorios de cursos, se sabe con certeza que el buscador devolverá páginas relacionadas con la consulta, que ahora si podrán ser ordenadas según preferencias de los usuarios.

De todas maneras el prototipo ha demostrado ser eficiente en búsquedas en la Web general. Por ejemplo al buscar: "desarrollo Web manuales" sin especificar ningún sitio en particular, de 20 páginas retornadas, 11 de las mismas fueron ordenadas correctamente por el prototipo. Esto ocurrió porque esas 11 páginas tenían texto para ser extraído y pudieron calcularse correctamente los índices de legibilidad.

Las 9 restantes eran páginas de publicidad o no contenían texto que pudiera ser extraído.

Por lo dicho anteriormente, las pruebas para evaluar la calidad de las extracciones de metadatos que realiza el prototipo se hicieron sobre búsquedas en sitios específicos que contengan cierta homogeneidad.

A continuación se muestran ejemplos de extracciones de los diferentes metadatos que componen las características extraídas por el prototipo para ordenar las páginas (detalladas en la sección 3.3). Esto se hace para comprobar con que exactitud el prototipo propuesto extrae estos metadatos. Las evaluaciones hechas para comprobar la extracción de metadatos se pueden dividir en varios casos.

En algunos casos en los que se pueden evaluar las extracciones realizadas por el prototipo comparándolas con ciertos parámetros externos o por mera observación, se muestran tablas con dichas evaluaciones y comparaciones. En otros casos es imposible comparar, por carecer de ejes de comparación externos (por ejemplo cuando el método de extracción es exclusivo del prototipo y no se conoce otro sistema que haga lo mismo) o porque es imposible establecer la exactitud de la extracción a simple vista. Por último hay ciertos metadatos que siempre son extraídos correctamente, y esto se comprueba con solo evaluar el código HTML de la página analizada, que es justamente lo que hace el prototipo y la razón de la exactitud de estos resultados.

En las siguientes subsecciones se dividen las pruebas de extracción de metadatos según la característica que se extrae. La búsqueda sobre la cuál se evalúa el prototipo consiste en solicitar 15 páginas del sitio W3C sobre el protocolo http como se ve en la Figura 19.

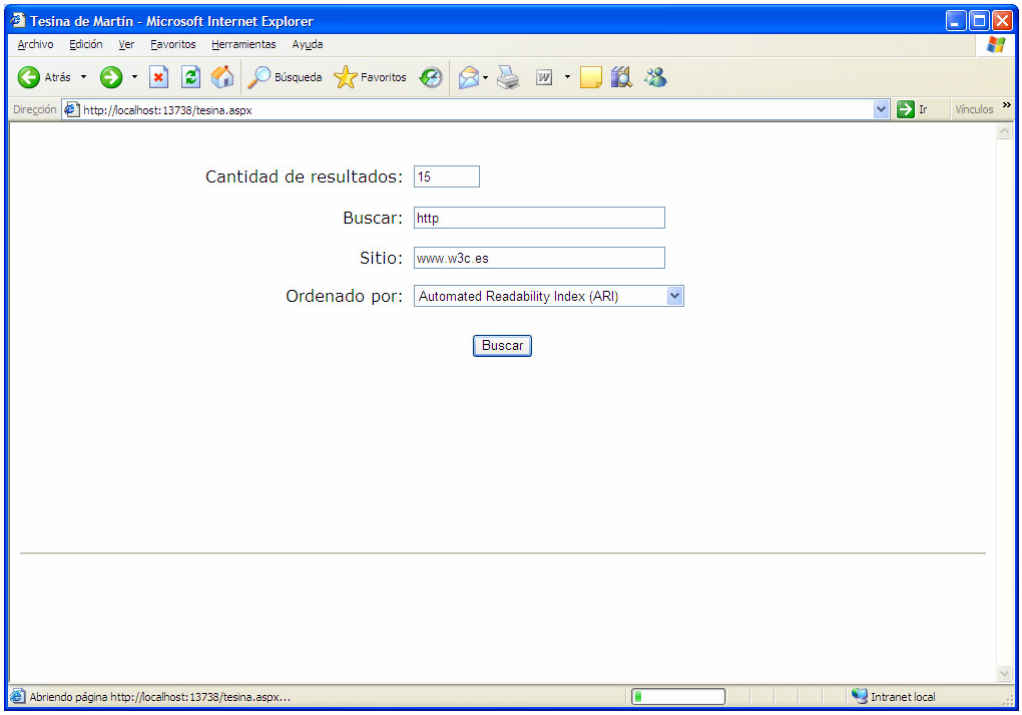

Figura 19: Búsqueda para experimentación

Si bien lo ideal hubiera sido hacer las búsquedas sobre sitios repositorios de cursos específicos, esto no fue posible pues es necesario tener acceso a este tipo de sitios que en general no son libres.

El sitio elegido para realizar las comparaciones en los casos en que esto es posible es Style & Diction[*], una página interactiva que chequea características sobre muestras de texto usando las utilidades *style* y *diction* de Unix, a la cual se le pasa el texto de la página Web analizada extraído manualmente.

Como se podrá ver en las siguientes subsecciones los resultados obtenidos fueron más que satisfactorios.

4.2.1. General

La siguiente tabla muestra diferentes metadatos extraídos para calcular la característica *General* de las 15 páginas devueltas por el prototipo. Los resultados de

[*] http://www.editcentral.com/gwt/com.editcentral.EC/EC.html

extraer estos atributos se comparan con los resultados calculados por la página Style & Diction.

Por observación se pudo comprobar que la extracción del dominio se hizo correctamente en todos los casos. El metadato *BytesTotal* es imposible de chequear a simple vista, y a los metadatos *polisílabas* y *sílabas* fue imposible compararlos con resultados externos al prototipo ya que se desconoce otra utilidad que haga separación en sílabas en castellano que no sea la desarrollada en esta tesina con el algoritmo de Figueroa (el sitio Style & Diction lo hace en ingles).

Los restantes atributos extraídos para calcular esta característica se muestran en las columnas de la Tabla 4. Cada columna indica en primer lugar los resultados obtenidos de las extracciones del prototipo, en segundo lugar los resultados calculados por el sitio externo, y por último se muestra la precisión obtenida por el prototipo. En la última fila de la tabla puede verse el promedio de la precisión de las extracciones del prototipo para las 15 páginas.

Como puede observarse la precisión lograda por el prototipo para los tres metadatos extraídos es muy buena.

Pág.	Dominio	BytesTexto			Caracteres			Palabras			Oraciones		
		Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión
1	http://www.w3c.es/	4324	4358	99%	3511	3475	99%	675	677	100%	36	35	97%
2	http://www.w3c.es/divulgacion/guiasbreves/Accesibilidad	4230	4219	100%	3501	3501	100%	654	650	99%	27	23	83%
3	http://www.w3c.es/Traducciones/es/WAI/intro/accessibility	7518	7515	100%	6265	6176	99%	1132	1131	100%	49	48	98%
4	http://www.w3c.es/Consorcio/	4641	4641	100%	3825	3750	98%	713	719	99%	31	32	97%
5	http://www.w3c.es/divulgacion/guiasbreves/HojasEstilo	2165	2157	100%	1746	1746	100%	372	371	100%	19	19	100%
6	http://www.w3c.es/Divulgacion/GuiasReferencia/CSS21/	719	711	99%	584	577	99%	110	107	97%	6	11	55%
7	http://www.w3c.es/Divulgacion/Guiasbreves/WebSemantica	6922	6896	100%	5603	5603	100%	1106	1104	100%	41	56	73%
8	http://www.w3c.es/Divulgacion/Guiasbreves/XHTML	2014	1996	99%	1634	1634	100%	312	313	100%	15	16	94%
9	http://www.w3c.es/divulgacion/guiasbreves/	773	764	99%	631	631	100%	113	112	99%	7	8	88%
10	http://www.w3c.es/divulgacion/guiasbreves/tecnologiasXML	3650	3615	99%	2922	2922	100%	593	591	100%	28	31	90%
11	http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb	4050	4028	99%	3321	3321	100%	628	629	100%	28	29	97%
12	http://www.w3c.es/divulgacion/guiasbreves/internacionalizacion	5515	5504	100%	4548	4548	100%	878	877	100%	40	37	92%
13	http://www.w3c.es/Traducciones/es/WAI/intro/components	3028	3027	100%	2545	2502	98%	439	438	100%	23	18	72%
14	http://www.w3c.es/Eventos/2008/DiaW3C/	2307	2299	100%	1888	1846	98%	372	369	99%	16	20	80%
15	http://www.w3c.es/Divulgacion/GuiasReferencia/XHTML1/	775	767	99%	636	626	98%	113	110	97%	11	12	92%
Promedio:				99%			99%			99%			87%

Tabla 4: Experimentación de característica General

4.2.2. Descripción

Se extrajo el metadato *description* de las páginas Web correctamente en todas las páginas que contenían datos en la etiqueta HTML `<Meta description>`. Esto se pudo comprobar observando los resultados extraídos por el prototipo y luego el código HTML.

4.2.3. Clasificación

Se extrajo el metadato *classification* correctamente al igual que el anterior en las 15 páginas consultadas. La verificación fue hecha de la misma manera que se hizo la de la subsección anterior.

4.2.4. Flash

En los ejemplos analizados en este caso no había ninguna inserción Flash, y el resultado devuelto por el prototipo fue el correcto. En todas las páginas el atributo *cantidad* dio igual a cero, al igual que el atributo *superficie*.

4.2.5. Keywords

Al igual que en los metadatos extraídos para las características *Descripción* y *Clasificación* en este caso también se comprobó observando el código HTML que el prototipo extrajo todos los *keywords* que figuraban en el código contenidos en la etiqueta `<meta name="keywords">`.

4.2.6. Idioma

El idioma extraído en las 15 páginas buscadas era el mismo: español. El prototipo lo extrajo correctamente en todos los casos.

4.2.7. Links

La extracción de los links en todas las páginas revisadas por el prototipo fue perfecta. Además de extraer la cantidad de links identificó correctamente los enlaces a la misma página y los enlaces a servidores externos, como puede observarse en la Tabla 5. La comprobación de los resultados extraídos se hizo nuevamente comparándolos con lo observado en el código HTML de cada página.

Pág.	Cantidad			Locales			Externos		
	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión
1	105	105	100%	55	55	100%	50	50	100%
2	77	77	100%	19	19	100%	58	58	100%
3	66	66	100%	24	24	100%	42	42	100%
4	86	86	100%	32	32	100%	54	54	100%
5	47	47	100%	21	21	100%	26	26	100%
6	117	117	100%	103	103	100%	14	14	100%
7	64	64	100%	27	27	100%	37	37	100%
8	42	42	100%	18	18	100%	24	24	100%
9	36	36	100%	22	22	100%	14	14	100%
10	48	48	100%	17	17	100%	31	31	100%
11	40	40	100%	17	17	100%	23	23	100%
12	42	42	100%	19	19	100%	23	23	100%
13	38	38	100%	15	15	100%	23	23	100%
14	30	30	100%	13	13	100%	17	17	100%
15	267	267	100%	248	248	100%	19	19	100%
Promedio:			100%			100%			100%

Tabla 5: Experimentación de característica Links

4.2.8. Imágenes

Como puede observarse en la Tabla 6, la extracción de los metadatos de las imágenes, tanto el atributo *cantidad* como el atributo *superficie*, se hizo con una precisión de 100%. La comparación de los resultados devueltos por el prototipo se hizo contra la funcionalidad "información de la página → medios" del navegador Mozilla Firefox 3, que devuelve toda la información de las imágenes de una página.

Pág.	Cantidad			Superficie		
	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión
1	9	9	100%	38286	38286	100%
2	5	5	100%	33436	33436	100%
3	3	3	100%	12889	12889	100%
4	7	7	100%	76148	76148	100%
5	4	4	100%	23260	23260	100%
6	4	4	100%	86836	86836	100%
7	5	5	100%	288340	288340	100%
8	4	4	100%	23260	23260	100%
9	4	4	100%	23260	23260	100%
10	4	4	100%	23260	23260	100%
11	6	6	100%	376013	376013	100%
12	4	4	100%	23260	23260	100%
13	7	7	100%	584441	584441	100%
14	7	7	100%	34012	34012	100%
15	6	6	100%	52276	52276	100%
Promedio:			100%			100%

Tabla 6: Experimentación de característica Imágenes

4.2.9. Meta

Se comprobó por observación del código HTML que el prototipo extrajo correctamente la información meta contenida dentro de la etiqueta `<meta>` de las páginas que contenían dicha información.

4.2.10. Indicadores

La Tabla 7 muestra la comparación de los resultados de los índices ARI y de Coleman Liau obtenidos por el prototipo y por la página Style & Diction.

El motivo de que sólo se compruebe la precisión de estos dos índices radica en el mismo problema que surgió para los metadatos *sílabas* y *polisílabas* de la característica *General*: el algoritmo de separación en sílabas. Estos dos índices no necesitan la cantidad de sílabas del texto para ser calculados, en cambio los índices de Flesch, SMOG y Gunning Fog sí. El sitio Style & Diction utiliza un algoritmo de separación en sílabas en inglés, por lo cuál no se pueden comparar sus resultados con

los del prototipo. Como puede verse en la Tabla 7 los resultados calculados con el prototipo tienen una precisión del 91% y del 87% para los índices ARI y Coleman Liau con respecto a la comparación externa.

Pág.	ARI			CLI		
	Prototipo	Referencia externa	Precisión	Prototipo	Referencia externa	Precisión
1	12,444	12,4	100%	14,821	12,9	85%
2	15,895	18,1	88%	15,718	14,9	95%
3	16,188	16,1	99%	16,785	15,1	89%
4	15,338	14,4	93%	15,785	13,6	84%
5	10,466	10,5	100%	11,830	10,4	86%
6	12,742	8,8	55%	15,454	12,9	80%
7	15,919	12,3	71%	14,028	12,6	89%
8	13,637	12,9	94%	15,033	13,4	88%
9	12,942	12,1	93%	17,072	15,2	88%
10	12,368	11,4	92%	13,209	11,7	87%
11	14,692	14,3	97%	15,334	13,9	90%
12	13,943	14,8	94%	14,696	13,5	91%
13	15,419	17,6	88%	18,330	16,6	90%
14	14,100	14,4	98%	14,080	12	83%
15	10,216	10	98%	17,322	14,4	80%
Promedio:			**91%**			**87%**

Tabla 7: Experimentación de característica Indicadores

4.3. Conclusiones de la experimentación

El caso de uso explicado en la Sección 4.1 demuestra que el prototipo tiene un buen rendimiento en las búsquedas en la Web en general, y no sólo en sitios específicos, aunque por supuesto el rendimiento del sistema mejora considerable cuando las búsquedas se hacen en sitios especializados. El resultado más adecuado a la solicitud del usuario fue devuelto en primer lugar, y todas las características de interés de la página fueron correctamente extraídas y guardadas para su posterior utilización.

Las pruebas de extracción de los metadatos de interés (Sección 4.2) muestran que las extracciones de todo lo que sea código HTML dentro de etiquetas, por ejemplo Descripción, Clasificación, Keywords, etc., tiene una precisión del 100%. Además, pudo comprobarse usando comparaciones con medios externos que la extracción del texto, imágenes, links y cálculos de índices también alcanza un alto nivel de

precisión. Todas estas pruebas demuestran que las clasificaciones y los datos obtenidos por el prototipo son altamente confiables. En los únicos casos en los que no se obtuvieron buenos resultados de ordenación fue en los que por razones ajenas al funcionamiento del prototipo las páginas devueltas por Google no contenían información que pudiera ser extraída. En estos casos el prototipo obtiene hojas para el XML resultante con metadatos prácticamente vacíos.

Para realizar las pruebas que corroboren la precisión del prototipo en la ordenación obtenida se requiere un grupo de personas heterogéneas que cumplan los requisitos de una población estadística imparcial que realice pruebas sobre los resultados obtenidos dentro de un grupo considerable y verificar que lo que se obtiene se corresponde con los resultados esperados.

Para dar una muestra de cómo serían estas pruebas se hizo un ejemplo que se observa en la Tabla 8. En este ejemplo se buscaron 5 páginas y se le consultó a un usuario en que orden las pondría teniendo en cuenta la dificultad de los textos presentes en las mismas. Luego se ingresaron los resultados de la ordenación hecha por el prototipo usando el índice automatizado de legibilidad (ARI).

		ARI		
Pág.	**URL**	Posición	Referencia	Correcto
1	http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o	5	5	Si
2	http://es.wikipedia.org/wiki/Categor%C3%ADa:Patrones_de_dise%C3%B1o	3	3	Si
3	http://es.wikipedia.org/wiki/Abstract_Factory_(patr%C3%B3n_de_dise%C3%B1o)	1	2	No
4	http://es.wikipedia.org/wiki/Antipatr%C3%B3n_de_dise%C3%B1o	4	4	Si
5	http://es.wikipedia.org/wiki/Iterador_(patr%C3%B3n_de_dise%C3%B1o)	2	1	No
…	…	…	…	…

Tabla 8: Ejemplo de pruebas de precisión en la ordenación

Como puede verse, en las 5 páginas buscadas hubo un error de clasificación del prototipo en relación con el criterio del usuario que actuó como sujeto de prueba. Las páginas 1, 2 y 4 fueron puestas en la misma ubicación por el prototipo y el usuario, pero las páginas 3 y 5 fueron invertidas en su orden. Esto indica que hay un error de ordenación, lo que supone dos páginas en distinto lugar del que las hubiera puesto el usuario. Este análisis se debe repetir un número significativo de veces y tomar el promedio de aciertos para obtener la precisión promedio del prototipo.

Es evidente que este tipo de análisis es muy subjetivo a la observación del sujeto que realiza la prueba, por lo cual se hacen necesarios los grupos de prueba con las características antes mencionadas para obtener un resultado significativo. Obviamente cuando se realicen estas pruebas se necesitará un número mayor de resultados para evaluar, en este caso solo se mostraron cinco pues solo se busca ejemplificar el procedimiento.

Capítulo 5.

Conclusiones

En la actualidad la cantidad de información disponible en la Web es extremadamente grande y continuará con un crecimiento igual o mayor en los tiempos venideros. Uno de los mayores problemas que esto ocasiona es una enorme dificultad para administrar grandes cantidades de información. En este trabajo se hizo un aporte a una de las ramas de la administración de la información, la que consiste en la localización y extracción de la misma.

En las ciencias de la computación hay diferentes disciplinas que intentan acercar soluciones a los requerimientos de los usuarios al momento de encontrar información. En el primer capítulo fueron presentadas varias propuestas de solución de diferentes autores y se mencionaron los problemas que algunas de ellas presentan. El prototipo que se introdujo en esta tesina propone una manera alternativa de realizar la extracción y análisis de la información contenida en las páginas.

En este trabajo se presentó una solución que podría ser de utilidad en sitios o aplicaciones Web que recolectan cursos y/o material académico, así como en búsquedas en la Web en general. La solución consiste en extraer características de las páginas recolectadas para luego generar un XML que será utilizado para indexarlas. Esto origina mayor información de contexto para clasificar correctamente la

información. Además, los datos contenidos en el XML pueden ser usados en diferentes aplicaciones, más allá del ordenamiento.

La arquitectura propuesta del prototipo realizado para esta tesina brinda la posibilidad de crecimiento del contenido de los metadatos obtenidos. De esta forma es posible ampliar el prototipo agregando nuevas librerías que extraigan otros datos y/o generen más indicadores con fines más específicos. Por lo tanto, el sistema propuesto es sumamente flexible y adaptable a otras áreas.

En las pruebas de búsquedas realizadas desde la página del prototipo se pudo comprobar que se agrega un valor a los resultados obtenidos directamente desde un buscador estándar, permitiendo que la búsqueda sea más personalizada. Además se forma un XML con datos que pueden ser de interés para organizar de manera más eficiente la clasificación de las páginas.

Finalmente, se realizaron diversas pruebas para comprobar la idoneidad del prototipo para las tareas de extracción de información y clasificación, obteniéndose los resultados esperados, y se sentó una base para pruebas futuras.

5.1. Trabajos futuros

Como ampliaciones para trabajos futuros sobre el desarrollo del prototipo que fueron surgiendo durante este trabajo se puede destacar tener mayor diversidad de buscadores para no restringirse solo a Google. Esta tarea se podría llevar a cabo con el concepto de configuración utilizado para el resto del desarrollo, teniendo en un archivo de configuración el formato de la cadena requerida para hacer la búsqueda y una expresión regular o de XPATH para la extracción de los resultados de la página devuelta.

Una particularidad del prototipo es que no se tuvo en cuenta la velocidad de ejecución de las consultas. Cada consulta de una página individual podría hacerse en diferentes hilos de ejecución para mejorar los tiempos de respuesta. Actualmente se ejecutan las consultas de manera secuencial aumentando los tiempos totales.

Como trabajo futuro puede resultar interesante agregar un módulo de acceso a datos para almacenar los datos de cada visita a la página para generar y evaluar estadísticas sobre el uso y resultados obtenidos. Este sería un buen método para estudiar y agregar nuevas funcionalidades.

Finalmente, en las páginas analizadas no se tuvo en cuenta en ningún caso las que al ingresar ejecutan un javascript y re-direccionan a la verdadera página del sitio. Estos casos deberían ser tenidos en cuenta con la utilización e implementación de métodos propios de los crawlers. Incluso se podría extender más aún para poder acceder a páginas que son el resultado de una ejecución dinámica.

Bibliografía

[Ashish, 1997] Ashish, N., and Knoblock, C. (1997). Wrapper generation for semi-structured Internet sources. *ACM SIGMOD Record,* 26:8-15.

[Blanco Suárez, 2004] Blanco Suárez, S. (2004). *Biblioteca semántica de webquest.* Tesis doctoral, Departamento de Informática, Universidad de Valladolid, España.

[Coleman, 1975] Coleman, M., and Liau, T. L. (1975). A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, 60:283-284.

[Cowie, 1996] Cowie, J., and Lehnert W. (1996). Information extraction. *Communications of the ACM,* 39:80—91.

[Cunningham, 1999] Cunningham, H. (1999). *Information Extraction – a user guide.* Second Edition. Institute of language, speech and Hearing (ILASH) and Department of Computer Science, University of Sheffield, UK.

[Embley, 1999] Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., Lonsdale, D. W., Ng Y. K., and Smith, R. D. (1999). Conceptual-Model-Based data extraction from multiple-record web pages. *Data & Knowledge Engineering,* 31:227-251.

[Farr, 1951] Farr, J. N., Jenkins, J. J., and Paterson, D. G. (1951). Simplification of Flesch Reading Ease Formula. *Journal of Applied Psychology*, 35:333-337.

[Figueroa, 1998] Figueroa K. (1998). *Síntesis de Voz en español, un enfoque silábico.* Tesis de Licenciatura, Facultad de Ingeniería Eléctrica, Universidad Michoacana, Morelia, Michoacán, México.

[Flesch, 1948] Flesch, R. (1948). A new readability yardstick. *Journal of Applied Psychology*, 32:221—233.

[Grumbach, 1999] Grumbach, S., Mecca G. (1999). In search of lost schema. *Proceedings of International Conference on Database Theory (ICDT)*, 314-331.

[Hoffman, 2006] Hoffman, K. *Microsoft visual C# 2005 unleashed.* 1º Edición, SAMS Publishing, Indianapolis, Indiana, EEUU.

[Kincaid, 1975] Kincaid, J. P., Fishburne, R. P. Jr, Rogers, R. L., and Chissom, B. (1975). Derivation of new readability formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy enlisted personnel. *Research Branch Report,* 8-75, Millington, TN: Naval Technical Training, U.S. Naval air Station, Memphis, TN.

[Lozano Tello, 2001] Lozano Tello, A. (2001). Ontologías en la Web semántica. *I Jornadas de Ingeniería Web*, España.

[McLaughlin, 1969] McLaughlin H. G. (1969). SMOG grading a new readability formula. *Journal of Reading*, 12:639-646.

[Mecca, 1999] Mecca, G., Azteni, P. (1999). Cut and paste. Journal of computing and system sciences, special ISSUE on PODS'99.

[Murua, 2004] Murua, I. (2004). La utilidad de las herramientas de anotación. Proyecto RODA. Disponible en http://roda.ibit.org/articulo1.cfm (Consultado en Septiembre de 2008)

[Myllymaki, 2002] Myllymaki, J. (2002). Effective data extraction with Standard XML Technologies. *Computer Networks,* 39:635-644.

[News, 1] Plain Language At Work Newsletter, 23 March 2004, *http://www.impact-information.com/impactinfo/newsletter/plwork08.htm* (Consultado en Septiembre de 2008).

[Ponce, 2006] Ponce, I. R., Zárate J. A., Olivares, J. C. (2006). Web Page Retrieval Using an Ontology that is Populated by Automatic Text Classification. *IEEE Looking Forward*, IEEE Computer Society, 13:31-34.

[Senter, 1967] Senter, R. J., Smith, E. A. (1967). *Automated readability index.* AMRL-TR, 66-22. Wright-Patterson AFB, OH: Aerospace Medical Division.

[Téllez Valero, 2005] Téllez Valero, A. (2005). *Extracción de Información con Algoritmos de Clasificación.* Tesis de Maestría en Ciencias, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Tonantzintla, Puebla, México.

[Tramullas, 1999] Tramullas, J. (1999), Agentes y ontologías para el tratamiento de información: clasificación y recuperación en Internet. *IV Congreso ISKO-España, EOCONSID'99,* 247-252.

[Web, 1] (2004) W3C Recommendation: *XML Information Set (Second Edition).* http://www.w3.org/TR/xml-infoset/ (Consultado en Septiembre de 2008).

[Web, 2] *Extensible Markup Language (XML).* http://www.w3.org/XML/ (Consultado en Septiembre de 2008).

[Web, 3] XML Tutorial. http://www.w3schools.com/xml/default.asp (Consultado en Septiembre de 2008).

[Web, 4] Guía Breve de XHTML. http://www.w3c.es/Divulgacion/Guiasbreves/XHTML (Consultado en Septiembre de 2008).

[Web, 5] (2007) W3C Recommendation: *XQuery 1.0: An XML Query Language*. http://www.w3.org/TR/xquery (Consultado en Septiembre de 2008).

[Web, 6] (1999) W3C Recommendation: *XML Path Language (XPATH) Version 1.0*. http://www.w3.org/TR/XPATH (Consultado en Septiembre de 2008).

[Web, 7] Html Agility Pack Home. http://www.codeplex.com/htmlagilitypack (Consultado en Agosto de 2008).

[Zhou, 2002] Zhou, Y. (2002). *Data-extraction Ontology Generation by example.* Thesis proposal for the Degree Master of Science, Department of Computer Science, Brigham Young University, EE.UU.

Apéndice A.

Archivo de configuración

Este apéndice contiene el archivo de configuración utilizado para hacer funcionar el prototipo implementado. El archivo es un XML que representa el documento del Analizador. Éste contiene los registros de las tablas *Características* e *Indicadores* que se cargan inicialmente cuando arranca el prototipo.

```
<?xml version="1.0" standalone="yes"?>
<DocAnalizador xmlns="http://tempuri.org/AnalizadorDataSet.xsd">
  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>General</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Description</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Clasification</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
```

```
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Flash</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Keywords</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Idioma</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Links</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Imagenes</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Caracteristicas>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Meta</Clase>
    <Metodo>Ejecutar</Metodo>
  </Caracteristicas>

  <Indicadores>
    <Nombre>locales/cantidad</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>Locales_Cantidad</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>externos/cantidad</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>Externos_Cantidad</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>superficieMedios</Nombre>
```

```
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>SuperficieMedios</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>cantidadMedios</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>CantidadMedios</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>bytesTexto/bytesTotal</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>BytesTexto_BytesTotal</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>ARI</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>ARI</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>SMOG</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>SMOG</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>GFI</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>GFI</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>CLI</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>CLI</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>Flesh</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
```

```
    <Metodo>Flesh</Metodo>
  </Indicadores>

  <Indicadores>
    <Nombre>FleshKincaid</Nombre>
    <Libreria>Tesina.CaracteristicasBasicas.dll</Libreria>
    <NameSpace>Tesina.CaracteristicasBasicas</NameSpace>
    <Clase>Indicadores</Clase>
    <Metodo>FleshKincaid</Metodo>
  </Indicadores>
</DocAnalizador>
```