Universidad Zaragoza
1542

Trabajo Fin de Grado

# GeoCrawler: sistema de crawler web enfocado al descubrimiento de información geográfica


Autor

**Jorge Cáncer Gil**

Director

**Francisco Javier López Pellicer**

Escuela de Ingeniería y Arquitectura


2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Jorge Cáncer Gil,

con nº de DNI 25204658Q en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) GeoCrawler: sistema de crawler web enfocado al descubrimiento de información geográfica,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 5 de mayo de 2016

Fdo: Jorge Cáncer Gil

# GeoCrawler: sistema de crawler web enfocado al descubrimiento de información geográfica

## RESUMEN

La búsqueda de información en la web es uno de los aspectos clave en la actualidad. Gran cantidad de herramientas buscan información en la web con motivos muy diversos. Entre esos motivos esta la realización de motores de búsqueda generales o enfocados, búsqueda de imágenes, comparadores de información, etc.

Para alguno de ellos se utilizan *crawlers*. Los *crawlers* son robots o arañas que se van propagando por páginas web a través de los enlaces de estas. Recopilan información, así como enlaces (links) de entrada y enlaces de salida.

El caso de la información geográfica es distinto, no es factible la recopilación de datos geográficos publicados por administraciones públicas por medio de *crawlers* genéricos.

En este trabajo se ha realizado un *crawler* enfocado a la búsqueda y recolección de información geográfica publicada bajo alguno de los estándares OGC

Para ello se han realizado modificaciones al *crawler* de código abierto Nutch. Estas modificaciones actúan en partes concretas del flujo de trabajo del *crawler* para modificar o añadir funcionalidad del mismo.

En concreto se ha modificado el sistema de *scoring* para adaptarlo a un sistema de búsqueda enfocado basado en un algoritmo de búsqueda llamado Shark-Search.

Se ha hecho uso de un tesauro con términos geográficos para ayudar al *crawler* a estimar la importancia de una página web.

Los documentos recuperados se guardan en el sistema de ficheros de la máquina en la cual se está ejecutando.

Una vez se acabó el *crawler* se realizaron pruebas para recolectar información. Durante esas pruebas se detectaron problemas que fueron solucionados. También se realizaron ajustes para mejorar el rendimiento del *crawler*.

Tras las modificaciones se obtuvo un resultado muy prometedor llegando a recuperar gran cantidad de documentos con los que se pueden realizar tareas de análisis de datos o *big data.*

# Contenido

# 1. Introducción

Descubrir y recolectar servicios de información geográfica, colecciones de datos y recursos geográficos en la web mediante los buscadores comerciales tradicionales es muy complicado. La mejor estrategia es desarrollar una *crawler* web enfocado al descubrimiento y recuperación de información geográfica.

## *1.1. Objetivo*

El principal objetivo de este trabajo de fin de grado o TFG es la conversión de un *crawler o araña web* genérico a un *crawler enfocado* al descubrimiento de información geográfica en la web. En este trabajo se ha profundizado en la recuperación de documentos correspondientes a las especificaciones de OGC (Open Geospatial Consortium). OGC proporciona estándares de documentos de descripción de diferentes tipos de servicios de red definidos por OGC (WPS, WCTS, WMS, WMTS, WCS y CSW).

Estos servicios son utilizados para dar acceso a gran parte de la información geográfica producida por entidades públicas y de investigación de todo el mundo, como organismos públicos de las comunidades autónomas y gobiernos centrales. Todas estas variantes serán explicadas posteriormente en esta memoria. También se realiza la recuperación de servicios de descarga Atom. Gracias al *crawler* se podrá recuperar gran cantidad de documentos que describen servicios geográficos. Estos documentos recogidos pueden ser útiles para tareas de *big data* y análisis de datos.

## *1.2. Resolución del problema*

Para conseguir llegar al objetivo expuesto anteriormente se va a partir de Apache Nutch. Apache Nutch es un *crawler* genérico actualmente desarrollado por Apache. Se han desarrollado *plugins* de Nutch para convertirlo en un *crawler* enfocado con el propósito antes indicado.

Para ello, se han utilizado técnicas algorítmicas para extraer los términos relevantes de una página web con el objetivo de detectar si puede o no ser relevante para el descubrimiento de información geográfica, además de diferentes heurísticas de búsqueda basadas en los usos y costumbres de publicación de información geográfica en la web. Estos *plugins* actúan sobre los llamados **puntos de extensión**.

Los puntos de extensión actúan sobre partes del proceso de *crawling* como pueden ser las fases de *fetch, parse, scoring* e *indexing* (estas fases serán explicadas posteriormente en la memoria). Es en estos puntos de extensión donde se ha desarrollado el trabajo necesario para convertir Nutch en un crawler que se dedique a buscar y guardar ficheros de descripción de servicios de información geoespacial.

Nutch por defecto está preparado para indexar la información recogida en el motor de búsqueda Apache Solr, el cual está basado en Lucene. No obstante, Nutch permite la adición de *plugins* que modifican su comportamiento. Nutch basa su arquitectura en Hadoop, esto le permite trabajar con un gran volumen de datos, como el que se genera en el proceso de *crawling*, utilizando operaciones de *MapReduce*.

## *1.3. Estructura de la memoria*

En esta memoria se explica qué son los *crawlers*, para qué se utilizan y por quién. También se mencionan sus componentes y los elementos que rodean a los *crawlers*.

Por otra parte, se ahonda en qué es Nutch, cómo funciona, las herramientas que utiliza y su arquitectura. Posteriormente se incluye el trabajo realizado para preparar Nutch para la tarea que se ha explicado anteriormente, así como las herramientas y tecnologías utilizadas para la realización de este proyecto. Se presentan los resultados a los que se ha llegado con el trabajo, la calidad de estos y la rapidez con la que son obtenidos.

También se habla de posibles mejoras y ampliaciones que pueden darse en este trabajo con el objetivo de aumentar la capacidad y el alcance del *crawler.* Se pueden consultar anexos con información sobre el tipo de ficheros que se pretenden recuperar y para qué se utilizan. También existe un manual de usuario en el cual se explica paso por paso cómo utilizar GeoCrawler.

# 2. Crawlers

Un crawler es una herramienta que recorre e inspecciona páginas web. Lo hace, generalmente, de manera automática y de acuerdo a un método que fija su comportamiento. También son conocidas como arañas web o simplemente arañas en las zonas hispanoparlantes.

Tienen múltiples usos, pero el más usado es el de recorrer todas las páginas web que pueda recogiendo información y datos del contenido de cada una de ellas para posteriormente indexar toda esa información recogida en un motor de búsqueda. Esta sería la definición de un crawler genérico dedicado a recoger la mayor cantidad de información posible. Sin ir más lejos, este sería el método que usa el famoso buscador Google.

## *2.1. Crawlers genéricos*

Son *crawlers* que no tienen un objetivo temático fijo, sino que intentan abarcar la mayor cantidad de información posible. Estos crawlers se utilizan principalmente para darle vida los buscadores web.

Su funcionamiento básico se basa en extraer y seguir la mayor cantidad de URLs posibles extrayendo información de cada una de las páginas visitadas para posteriormente indexarla. Esto permite realizar búsquedas por términos. Recogen y siguen todos los links de salida de una web para llegar al mayor número posible de ellas. Debido a esto, estos *crawlers* tienden a extenderse de manera circular o de mancha de aceite ya que no siguen ningún criterio especifico que modifique su comportamiento.

En la Figura 1: Esquema *crawler* genérico se puede observar una representación esquemática de cómo sería una expansión de un *crawler* genérico desde una semilla. Como se puede observar la expansión se produce de manera circular al recoger y procesar todos los enlaces de salida que tiene una página web.

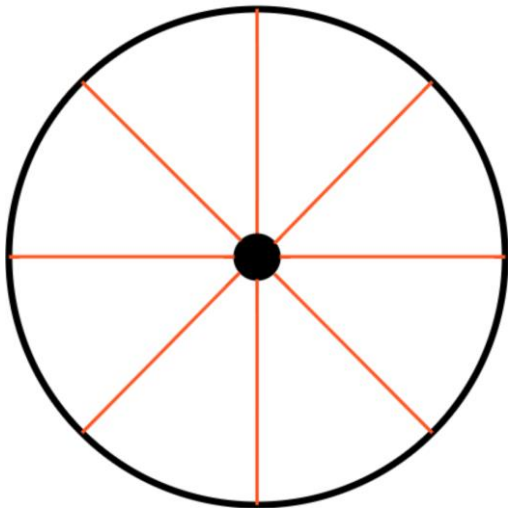


**Figura 1: Esquema *crawler* genérico**

## *2.2. Crawlers enfocados*

El término *crawler* enfocado es muy amplio y puede abarcar a un elevado número de tipos de *crawlers*. Existen muchas maneras para convertir un *crawler* en enfocado, esto

depende en la medida en la que se quiera alterar el comportamiento de un *crawler* genérico [1], [2].

Un *crawler* enfocado está preparado para no seguir el esquema de uno genérico. Es decir, no va a procesar todos los enlaces de salida de una web, sino que va a asignarles un *score* para priorizar qué enlaces va a expandir en la siguiente iteración. Con esto, el flujo de trabajo del *crawler* prioriza unos enlaces antes que otros. De esta manera, el *crawler* es más eficiente en la búsqueda de elementos concretos que uno genérico.

Los crawlers enfocados tienen muchos usos. Se pueden utilizar para indexar información de un tema específico buscando y priorizando enlaces que estén relacionados con este tema. También se pueden utilizar para extraer un tipo de información o documentos de la web. Este es el caso que se ha utilizado en este trabajo.

En este trabajo, nos vamos a centrar en la búsqueda y extracción de ficheros que definan información geográfica. En la Figura 2: Esquema *crawler* enfocado se puede observar una representación esquemática de un *crawler* enfocado. Estos se extienden de manera irregular expandiendo y procesando enlaces que son más relevantes según se haya preparado al *crawler*.

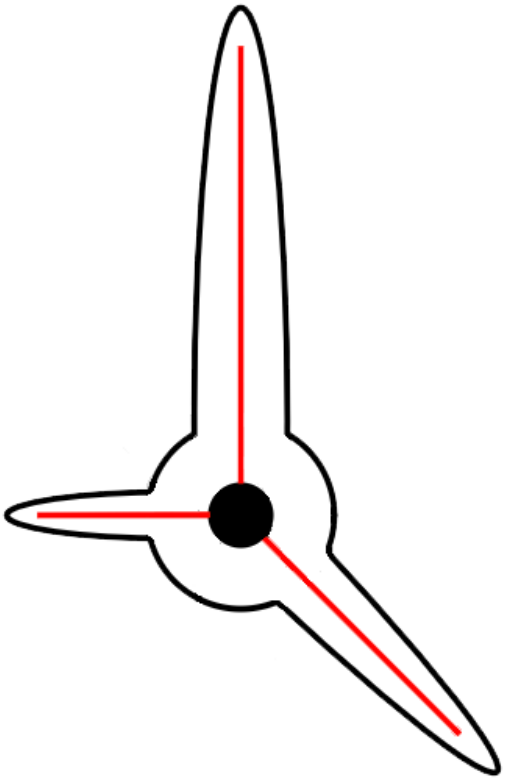


**Figura 2: Esquema *crawler* enfocado**

## 2.3. Funcionamiento de un crawler

Un *crawler web* es un *bot* que busca información en la web, normalmente con el propósito de indexarla. Los motores de búsqueda utilizan *crawlers* para mantener actualizados su contenido y sus índices [3].

Los *crawlers* consumen muchos recursos de los sistemas que visitan e incluso a veces visitan sitios donde no tienen aprobación. Pueden provocar problemas de carga cuando intentan acceder a mucha cantidad de información muy rápidamente. Existen mecanismos para que los sitios que deseen no ser visitados por *crawlers* puedan no ser accedidos por estos.

El funcionamiento de los *crawlers* tiene, como norma general, un comportamiento iterativo. Es decir, que un crawler repite un conjunto de operaciones el número deseado de veces para desarrollar su trabajo. Cada *crawler* puede tener sus peculiaridades en su funcionamiento. No obstante, en este apartado, se va a explicar la situación más típica con las operaciones más comunes.

En la primera ejecución de un *crawler* se realiza una operación denominada **inject**. Esta operación inserta las URLs semilla. Es a partir de estas páginas web desde donde el *crawler* va a partir. La calidad de estas semillas influye mucho en la rapidez con la que se van a empezar a encontrar resultados significativos. Esta operación, mencionada anteriormente, no está dentro del bucle principal ya que solo debe ejecutarse una vez en la vida de un *crawler*.

Una vez entrado en el bucle la primera operación realizada es el ***fetch***. Durante esta fase el crawler selecciona un número determinado de URLs que estén sin visitar. El número de URLs que se seleccionan depende de la configuración que se le haya asignado al *crawler*. Este parámetro permite configurar la rapidez con la que se ejecutaran cada una de las iteraciones. Una vez seleccionados los enlaces que se van a seguir se procede a descargar la información correspondiente a cada uno. Esta información se preserva para la siguiente fase.

En esta segunda operación el bucle, llamada ***parse***, se procede a analizar la información descargada en la anterior operación. Se extraen metadatos y se procesa el contenido de la página web para el fin que se le haya dado al *crawler*. Durante esta fase también se extraen nuevos enlaces presentes en las páginas que se están procesando, los cuales se añaden a la lista de enlaces para ser tenidos en cuenta en la próxima iteración.

A continuación, se procede a guardar toda la información recolectada en algún tipo de almacén. A este proceso se le conoce por ***index***.

En el caso de *crawlers* enfocados, existe una operación trasversal que actúa durante el resto de las operaciones. Esta operación se conoce como ***scoring***. Durante esta operación, se puede modificar el *score* dado a una URL, dependiendo de diversos factores, para conseguir su prioridad en el flujo de trabajo del *crawler*.

En la Figura 3: Esquema arquitectura de un *crawler*se puede observar el esquema del flujo habitual de un *crawler* incluyendo todas las secciones mencionadas en este apartado.

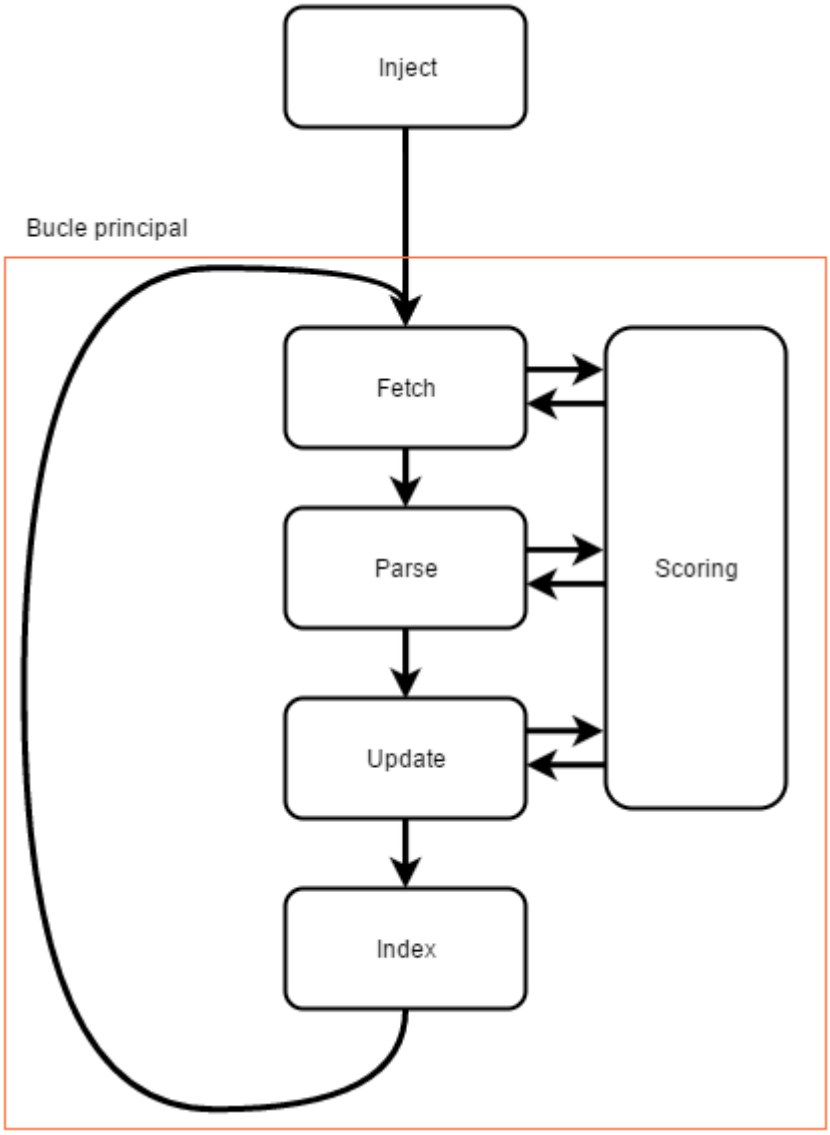



**Figura 3: Esquema arquitectura de un *crawler***

## *2.4. Educación*

Cuando se pone en marcha un *crawler* se puede elegir ser respetuoso con los *hosts* que se visitan o no serlo. Normalmente los *crawlers* están configurados para no hacer muchas peticiones seguidas a un sitio para no mermar su capacidad de respuesta para otras peticiones. Se suele esperar unos segundos entre petición y petición para ser respetuoso. Por supuesto esto depende de la configuración del *crawler*. Por lo tanto, se puede ser tan respetuoso como el que diseña el *crawler* quiera.

Existen mecanismos para limitar el consumo de ancho de banda por *host*. Así como controlar el número de peticiones concurrentes y número de peticiones por hora. Existe otra medida para intentar regular a los *crawlers*. Por defecto los *crawlers* deben mirar el fichero `robots.txt`. En este fichero se pueden establecer limitaciones a los *crawlers* [4].

Con el fichero `robots.txt` se puede limitar el acceso a toda o a una parte de una web. También se pueden establecer límites a algunos *crawlers,* pero a otros no [5]–[7]. A continuación, se pueden observar algunos ejemplos de robots.txt:

En este se indica que cualquier robot puede acceder a cualquier parte de la web.

```
User-agent: *
Disallow:
```

En el siguiente ejemplo todos los robots tendrían el acceso restringido a todas las secciones de la web.

```
User-agent: *
Disallow: /
```

El siguiente corresponde a la página web de la Casa Real de España. Se ha decidido que los *crawlers* no indexen algunas partes de la web.

```
User-agent: *
Disallow:
Disallow: /_*/
Disallow: /ES/FamiliaReal/Urdangarin/
Disallow: /CA/FamiliaReal/Urdangarin/
Disallow: /EU/FamiliaReal/Urdangarin/
Disallow: /GL/FamiliaReal/Urdangarin/
Disallow: /VA/FamiliaReal/Urdangarin/
Disallow: /EN/FamiliaReal/Urdangarin/
Sitemap: http://www.casareal.es/sitemap.xml
```

Por último, podemos ver un ejemplo de una web donde se restringe el acceso de los robots excepto uno concreto. Al *crawler* de *Spatineo* se le permite acceder a los recursos WMS, pero al resto de robots no.

```
User-agent: spatineo
Allow: /wms/request.php
Disallow: /

User-agent: *
Disallow: /
```

## 2.5. Crawlers famosos

Grandes empresas hacen uso de *crawlers* para ofrecer productos y servicios. En la mayoría de casos estos son usados para alimentar un motor de búsqueda. No obstante, existen empresas que hacen uso de los *crawlers* para otros fines como recolección de información para diversos usos.

Estos son los *crawlers* dedicados a motores de búsqueda más famosos [3]:

- Googlebot – Es el *crawler* que da vida al buscador más usado del mundo, Google.
- Baiduspider - El equivalente chino al *crawler* anterior. Tiene el mismo fin, pero para alimentar el buscador de Baidu. Baidu es el buscador más usado en China.
- Bingbot – El equivalente para el motor de búsqueda de Microsoft Bing.

- Yandexbot – Usado por Yandex. Yandex es el buscador más usado en Rusia y algunos países de la antigua Unión Soviética.
- Yahoo! Slurp – Era el antiguo *crawler* de Yahoo. Fue fusionado con el de Microsoft y se pasó a usar Bingbot.

Existen *crawlers* que se dedican a buscar otros elementos específicos, por ejemplo:

Spatineo – Se dedica a la búsqueda de información geográfica.

Kayak – Es una web de comparación de precios de vuelos. Este tipo de webs utilizan *crawlers* para nutrirse de contenido.

# 3. Nutch

Nutch es un *crawler* genérico implementado en **Java** de **código abierto**. Al ser de código abierto permite modificaciones en su estructura central de manera que se puede cambiar de cualquier manera siendo posible cualquier variación. Nutch fue creado por Doug Cutting[1] y Mike Cafarella como un crawler genérico. Desde su creación la herramienta fue ganando popularidad hasta que en 2005 ingresó en el programa Apache Incubator[2], a través del cual consiguió convertirse en un subproyecto de Lucene [8]. Desde abril de 2010 Nutch está considerado como un proyecto de alto nivel e independiente de *Apache Software Foundation*.

A parte de poder modificar el código del núcleo de Nutch, existen los denominados **puntos de extensión**. Los puntos de extensión son puntos del flujo de ejecución de Nutch que pueden ser ampliados mediante la implementación de *plugins*. En la figura inferior se puede observar un diagrama de la arquitectura de Nutch.

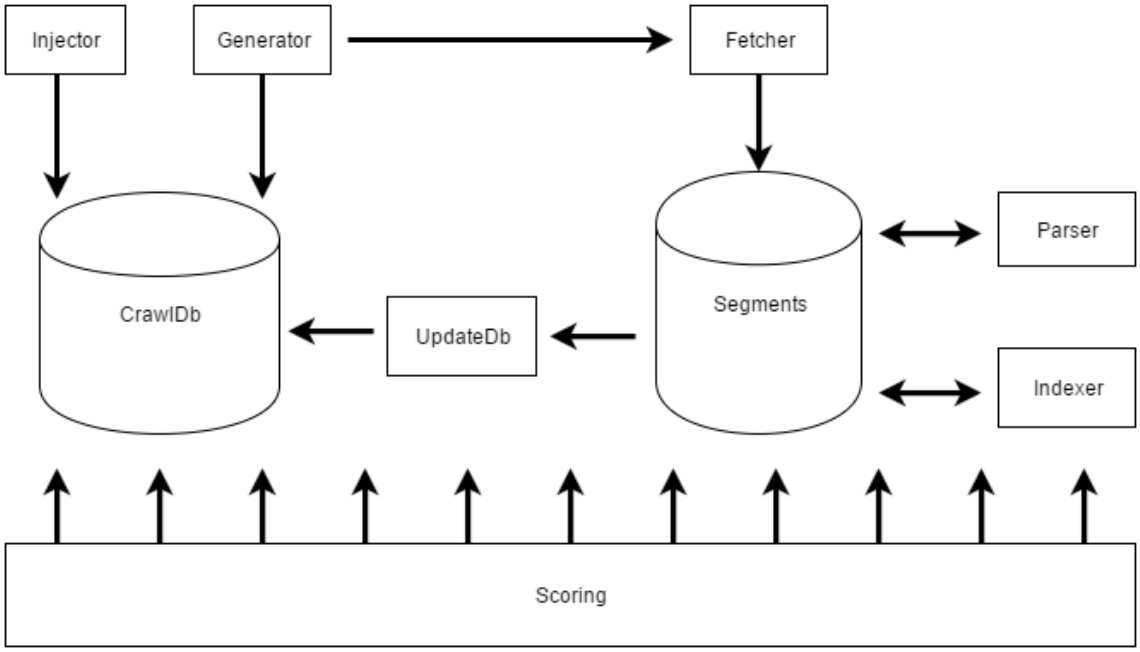



**Figura 4: Esquema arquitectura Nutch**

## *3.1. Estructura*

La estructura de ficheros de Nutch está compuesta por carpetas con distinta finalidad. A continuación, se va a explicar la finalidad de ellas:

- `bin` – Contienen dos scripts para Shell. Estos son:
  - `Nutch.sh` – Este fichero permite ejecutar cualquier sección de Nutch. Consultar Anexo A – Operaciones de Nutch para más información.

[1] Desarrollador también de Lucene y Hadoop

[2] Programa de candidatos para convertirse un parte de la fundación Apache.

    - `Crawl.sh` – Permite la ejecución entera del flujo completo de Nutch. Se deben introducir una serie de parámetros para la configuración básica de la ejecución.
- `build` – Carpeta generada automáticamente al compilar Nutch con **Ant**[3]. Contiene todos los ficheros necesarios para la ejecución de Nutch. Para más información se puede consultar la sección Compilación de Nutch.
- `conf` – Contiene todos los ficheros de configuración de Nutch. Con estos ficheros se puede alterar el comportamiento de Nutch como se desee. Se pueden editar los plugins a usar, el tipo de URLs aceptadas, etc. Se pueden consultar detalles de los ficheros de configuración de Nutch en la sección Configuración de Nutch.
- `docker` – Ficheros necesarios para desplegar Nutch en Docker.
- `ivy` – Configuración de la compilación de Nutch. Se usan **Ivy** [4]como gestor de dependencias y **Ant** para compilar. Para más información consultar la sección Compilación de Nutch.
- `src` – Contiene todos los ficheros fuente de Nutch. Desde los ficheros originales del núcleo de Nutch hasta los plugins que vienen implementados o se implementan.
- `urls` – Dentro de este directorio se encuentra un único documento llamado seeds.txt. En este documento se encuentra la lista de URLs semilla.

## *3.2. Principales componentes*

La representación del **almacén** que utiliza Nutch para su ejecución e infraestructura se llama **CrawlDB**. En este almacén se guarda información relativa a cada URL como pueden ser la dirección web, el nombre, el anchor, el estado, el score, metadatos, etc. Existe una operación de Nutch que permite la lectura en un fichero txt del almacén anteriormente mencionado. Esta operación se llama ***readdb.***

A continuación, se van a presentar ver tres ejemplos reales extraídos de una de las ejecuciones de **GeoCrawler**. El primero de ellos corresponde a una petición a un servicio que devuelve el código 404 de error de HTTP. Esto corresponde a un recurso no encontrado. Se puede observar en el campo ***Status*** el valor db_gone con valor numérico 3 indicando el error al intentar acceder a este recurso.

Los valores del *status* en Nutch pueden tomar los siguientes valores:

- db_unfetched – Representa un enlace que se ha almacenado en la lista, pero no ha sido procesado.
- db_fetched – Representa un enlace que ya ha sido procesado.
- db_gone – Con este estado se marcan aquellos enlaces que llevan a recursos no encontrados o que ya no existen.
- db_redir_temp – Indica que la página redirige temporalmente a otra.
- db_redir_perm – Indica que la página redirige permanentemente a otra.
- db_notmodified – Indica que una página ha sido recuperada correctamente pero no se han encontrado modificaciones.

[3] http://ant.apache.org/

[4] http://ant.apache.org/ivy/

- db_duplicate – Indica que el enlace ya existía.

```
http://www.opengis.net/spec/WCS_service-
model_scaling+interpolation/1.0/conf/scaling+interpolation Version: 7
Status: 3 (db_gone)
Fetch time: Tue Jun 21 20:07:53 CEST 2016
Modified time: Thu Jan 01 01:00:00 CET 1970
Retries since fetch: 0
Retry interval: 3888000 seconds (45 days)
Score: 140000.0
Signature: null
Metadata:
  anchor_context=
_pst_=notfound(14),
lastModified=0: http://www.opengis.net/spec/WCS_service-
model_scaling+interpolation/1.0/conf/scaling+interpolation
_rs_=424
anchor=
nutch.protocol.code=404
```

**Ejemplo 1**

El segundo ejemplo representa un caso de éxito. Se encuentra un recurso de tipo WMS, se analiza y se indexa. Como se puede ver, el **tipo de contenido** que hay en esa URL es de tipo application/xml. Este tipo de documento es en lo que se centra en buscar GeoCrawler. También aparece el ***anchor***[5] y algunos otros elementos que serán explicados más adelante en esta memoria.

```
http://www.opengis.uab.es/cgi-
bin/IdoneitatPI/MiraMon5_0.cgi?request=GetCapabilities&service=WMS Version: 7
Status: 2 (db_fetched)
Fetch time: Mon Jun 06 19:46:59 CEST 2016
Modified time: Thu Jan 01 01:00:00 CET 1970
Retries since fetch: 0
Retry interval: 2592000 seconds (30 days)
Score: 1.221E7
Signature: 5e59a5d17e7b9c6aa45711779e6c942b
Metadata:
  anchor_context=ática de las especies leñosas http://www.opengis.uab.es/cgi-
_pst_=success(1), lastModified=0
_rs_=963
Content-Type=application/xml
anchor=http://www.opengis.uab.es/cgi-bin/IdoneitatPI/MiraMon5_0.cgi
nutch.protocol.code=200
```

**Ejemplo 2**

Para terminar, en el tercer ejemplo, que se puede ver abajo, se puede apreciar un caso en el que se ha intentado recuperar un documento que ya existía.

[5] Anchor: Nombre con el que aparecen los enlaces en el navegador.

http://www.opengis.uab.es/cgi-bin/Educacio/MiraMon.cgi?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.3.0
Version: 7
Status: 7 (db_duplicate)
Fetch time: Mon Jun 06 20:06:00 CEST 2016
Modified time: Thu Jan 01 01:00:00 CET 1970
Retries since fetch: 0
Retry interval: 2592000 seconds (30 days)
Score: 3333000.0
Signature: 59d80766354f3ab52ab4cf8186b2a539
Metadata:
  anchor_context=ut Cartogràfic de Catalunya ( Capabilities ) WMS Base topogr
_pst_=success(1), lastModified=0
_rs_=573
Content-Type=application/xml
anchor=Capabilities
nutch.protocol.code=200

**Ejemplo 3**

## *3.3. Flujo principal de Nutch*

El flujo de trabajo principal de Nutch sigue el esquema de un crawler general explicado con anterioridad en esta memoria. No obstante, tienen sus particularidades debido a que Nutch posee un mayor número de operaciones disponibles. Nutch también trabaja por iteraciones, es decir, repite un conjunto de operaciones un numero especificado de veces.

### 3.3.1. Inject

Previo al inicio de la fase iterativa Nutch **inyecta** las URL semilla. Estas son las URLs de las cuales va a partir el *crawler.* Estas direcciones se recogen del fichero seeds.txt, el cual está localizado en la carpeta URLs. La clase que implementa esta operación es `org.apache.nutch.crawl.Injector`.

### 3.3.2. Generate

Una vez dentro del bucle, la primera operación es ***generate***. En esta operación se seleccionan los documentos para continuar con el proceso. Nutch permite especificar en la configuración cuantos documentos van a ser seleccionados para ser procesados. Así, Nutch seleccionará los N enlaces con más score para su procesamiento. Esta operación genera un ***segment***, este es el elemento de almacenamiento temporal que se utiliza para las distintas fases del flujo de trabajo.

En un *segment* se guarda:

- El contenido de las páginas en crudo (texto sin procesar con las etiquetas de html).
- Contenido después de pasar por la fase de *parse*.
- Metadatos descubiertos o extraidos.
- Enlaces de salida de la web (*outlinks).*
- Texto plano para la indexación.

### 3.3.3. Fetch

Después de haber generado un *segment* se procede a realizar la operación de ***fetch***. Durante esta operación se recupera el contenido de las páginas web seleccionadas para ser procesadas. Extrae tanto el contenido como los metadatos de la misma.

### 3.3.4. Parse

El proceso de *parse* es uno de los más importantes en el proceso de *crawling* para conseguir los resultados deseados. El proceso de *parse* no guarda una gran relación con el procedimiento homologa referida a los compiladores de lenguajes. En los dos casos se trabaja con conjuntos de caracteres con el objetivo de interpretarlos. No obstante, en el caso que nos ocupa el proceso se relaciona más con la extracción del texto perteneciente a la página que se está procesando.

Dependiendo del tipo de contenido que tiene una web, se debe usar un *parser* u otro. Cada tipo de contenido necesita que se tengan en cuenta sus peculiaridades con el objetivo de extraer la mayor cantidad de información y de manera correcta. Un ejemplo sería una página que contiene un documento XML[6]. Para realizar el *parse* de ese tipo de documentos se utilizan analizadores específicos basados en la arquitectura DOM[7]. Para más información consultar el anexo Anexo F – DOM.

### 3.3.5. UpdateDb

Con esta operación se inserta toda la información recogida en la iteración en curso para guardarla en la unidad de persistencia que usa Nutch durante la ejecución. No tienen nada que ver con dónde se van a guardar los datos recolectados, solo es una unidad de persistencia para almacenar metadatos sobre todos los enlaces y su estado actual.

### 3.3.6. Index

Durante esta operación se indexa el contenido que se ha recolectado en el resto de la ejecución del *crawler*. Por defecto Nutch está configurado para almacenar en Solr. No obstante, este comportamiento puede ser modificado.

## *3.4. Map-Reduce*

Nutch hace uso de Hadoop para implementar operaciones de Map-Reduce. El Map-Reduce es un modelo de programación utilizado por primera vez en Google para dar soporte al procesamiento de grandes volúmenes de datos [9]. Hadoop ofrece una implementación *Open* Source. Hadoop fue desarrollado en sus orígenes por Yahoo para posteriormente pasar al proyecto Apache. El Map-Reduce aborda problemas que se puedan disgregar en dos operaciones, la *map* y la *reduce*.

## *3.5. Puntos de extensión*

Nutch proporciona un conjunto de interfaces que representan los llamados puntos de extensión. A través de estos puntos se puede modificar el comportamiento de Nutch como se desee.

---

[6] e*Xtensible Markup Language*

[7] Document Object Model

Estos puntos de extensión se desarrollan implementando la interfaz java correspondiente. Estos puntos actúan en determinados momentos del flujo de trabajo de Nutch.

Los puntos de extensión son:

- `IndexWriter`: Escribe los datos resultantes de la ejecución del proceso de crawling a un almacén de índices especifico. Por ejemplo, Solr[8], ElasticSearch, un fichero CVS, etc.
- `IndexingFilter`: Permite añadir metadatos a los campos indexados. Estos filtros se ejecutan secuencialmente en la tarea de indexación (en la fase Reduce).
- `Parser`: Permite modificar el procesamiento de los documentos recuperados en la fase de *fetch* para extraer datos para ser indexados.
- `HtmlParseFilter`: Permite añadir metadatos adicionales a los parseos de HTML.
- `Protocol`: Permite usar distintos protocolos para fetchear documentos (ftp, http…).
- `URLFilter`: Limita las URLs que Nutch intenta fetchear. Nutch ya contiene un filtro por expresión regular que proporciona control sobre que URLs crawlea Nutch. No obstante, si se desea se puede realizar una implementación propia.
- `URLNormalizer`: Permite convertir URLs a una forma normal y realizarle modificaciones.
- `ScoringFilter`: Manipula las variables en el CrawlDatum[9] y en índices de búsqueda resultantes. Puede darse la existencia de varios filtros encadenados para realizar varios ajustes.
- `SegmentMergeFilter`: Se usa para filtrar los segmentos durante su mezclado. Permite filtrar no solo por la URL, sino por los metadatos adquiridos durante la fase de *parse*.

## 3.6. Compilación de Nutch

Nutch utiliza *Ant* para las tareas de compilación. Sirve para automatizar los procesos de compilación y test. *Ant* requiere un fichero llamado build.xml donde se define el proceso de generación y de las dependencias [10]. Adicionalmente, se utiliza la herramienta *Ivy* como gestor de dependencias para Nutch [11].

Para más información de cómo se utilizan estas herramientas consultar el Anexo B – Manual de GeoCrawler.

## 3.7. Configuración de Nutch

A parte de las modificaciones en forma de plugins que pueden ser realizadas para Nutch, es posible la configuración de gran cantidad de paramentos que también alteran su comportamiento.

La configuración más importante de Nutch viene definida en los siguientes ficheros:

- `nutch-default.xml` → Este fichero contiene los parámetros de configuración por defecto de Nutch. El formato de estos parámetros se explica en la sección 3.7.1.

[8] Solr es un motor de búsqueda basado en Lucene.

[9] El CrawlDatum contiene toda la información de cada URL.

- `nutch-site.xml` → En este fichero se pueden añadir nuevos parámetros o editar los que vienen por defecto en Nutch.
- `parse-plugins.xml` → Sirve para definir qué *parser* actúa dependiendo del tipo de contenido de la página (application/xml, text/html, …) .
- `regex-urlfilter.txt` → Se pueden definir expresiones regulares para filtrar las URLs. Si la URL coincide con una de las expresiones regulares en este fichero, no se procesa.

### 3.7.1. Parámetros de configuración

Los parámetros de configuración se definen en un fichero XML y tienen la siguiente forma:

```
<property>
  <name>plugin.folders</name>
  <value>/home/jorge/Desktop/TFG/GeoCrawler-101Crawlers-/build/plugins</value>
</property>
```

Como se puede ver, la definición de una propiedad viene dada por un nombre y un valor, también es posible añadir una descripción como se puede ver en el ejemplo de abajo.

```
<property>
  <name>ogc.path</name>
  <value>out/</value>
  <description>Path to store the results</description>
</property>
```

### 3.7.2. Parámetros propios

Como se ha comentado, en el fichero `nutch-site.xml` se pueden definir propiedades para ser usadas en los *plugins.* Para leerlo desde el código en Java la clase donde se está utilizando esta propiedad debe implementar la interfaz `Configurable`.

Por ejemplo, para la siguiente regla:

```
<property>
  <name>ogc.outlink.anchor.context</name>
  <value>30</value>
  <description>Boundary</description>
</property>
```

Desde el código se leería así:

```
boundary = conf.getInt("ogc.outlink.anchor.context", 30);
```

El primer paramento del método `getInt` es el nombre de la propiedad y el segundo es el valor por defecto si no se ha podido leer.

# 4. GeoCrawler

GeoCrawler es el nombre del *crawler* que ha sido desarrollado a lo largo de este trabajo. GeoCrawler es un *crawler* enfocado al descubrimiento de información geográfica por una parte y su posterior análisis y almacenamiento. El objetivo de este *crawler* enfocado es el de encontrar archivos de descripción de servicios geoespaciales. Estos servicios ofrecen una gran cantidad de información.

En concreto, en este trabajo se ha tratado la búsqueda de documentos de descripción de servicios definidos por OGC[10]. Estos documentos son WPS, WCTS, WMS, WMTS, WFS, WCS y CSW.

## *4.1. Documentos OGC*

La OGC es una organización internacional sin ánimo de lucro que se encarga de la realización de estándares abiertos de calidad para la comunidad geoespacial. Estos estándares son realizados a través de un proceso de consenso y están abiertos gratuitamente para cualquier persona que quiera mejorar el mundo de los datos geoespaciales.

Los estándares de OGC son usados en una gran cantidad de dominios incluidos medio ambiente, defensa, salud, agricultura, meteorología, desarrollo sostenible y muchos más. Estos estándares son documentos técnicos que detallan interfaces o codificaciones.

Los desarrolladores de software usan este tipo de documentos para construir interfaces abiertas para sus productos y servicios. Para complementar esto se puede consultar información más completa en Anexo C – Tipos de recursos OGC.

## *4.2. Modificaciones*

Anteriormente en esta memoria se ha explicado la posibilidad de incluir *plugins* en el flujo habitual de Nutch. Se han incluido varias modificaciones para conseguir el comportamiento deseado del crawler. Los plugins que van a ser usados en la ejecución del *crawler* se pueden especificar en el fichero de configuración `nutch-site.xml` modificando la regla `plugin.includes`. El estado de la regla con todos los plugins añadidos se puede ver en el Ejemplo 4.

[10] Open Geo Spatial - http://www.opengeospatial.org

```
<property><name>plugin.includes</name>
     <value>
          protocol-http|urlfilter-regex|parse-
          (tika|html|ogc)|index-(basic|anchor|length)|indexer-
          ogc|scoring-geo|urlnormalizer-(pass|regex|basic)
     </value>
     <description>
          Regular expression naming plugin directory names to
          include. Any plugin not matching this expression is
          excluded. In any case you need at least include the
          nutch-extensionpoints plugin. By default Nutch includes
          crawling just HTML and plain text via HTTP, and basic
          indexing and search plugins. In order to use HTTPS
          please enable protocol-httpclient, but be aware of
          possible intermittent problems with the underlying
          commons-httpclient library. Set parsefilter-naivebayes
          for classification based focused crawler.
     </description>
</property>
```

**Ejemplo 4**

En un apartado anterior se han explicado los distintos puntos de extensión a través de los cuales Nutch puede ser modificado. A continuación, se va a explicar qué puntos de extensión se han usado, por qué y cómo. Para la realización de este trabajo se han modificado los siguientes puntos de extensión:

- IndexWriter
- ParseFilter
- IndexingFilter
- Scoring Filter

### 4.2.1. OgcParseFilter

Vamos a empezar con el *plugin* que explota el punto de extensión HtmlParseFilter. Esta extensión se ha utilizado por dos motivos muy importantes pero que no están relacionados entre sí.

Este punto de extensión actual de filtro durante la fase de *parse* por lo tanto es el punto ideal para analizar el contenido del documento que se esta procesando.

Debido a que la sección del flujo de trabajo de Nutch en la que actúa es clave para poder aprovechar y procesar en tiempo de ejecución los datos, se decidió utilizarla para detectar si el documento que se está procesando es un fichero OGC y, si lo es, extraer qué tipo de documento es y la versión del mismo.

La otra tarea que es realizada aquí es la de la extracción del *anchor* de los enlaces de salida que están contenidos en el documento que está siendo procesado, así como el contexto del mismo. Nosotros entendemos por contexto del *anchor* a los caracteres más cercanos a este. Esta información será utilizada posteriormente en el cálculo del *score.* Como se ha comentado previamente, estos plugins se desarrollan implementando un punto de extensión, el cual viene representado por una interfaz de Java.

En este caso la interfaz que se extiende es `HtmlParseFilter`.

```java
/**
 * Extension point for DOM-based HTML parsers. Permits one to add
 * additional metadata to HTML parses. All plugins found which
 * implement this extension point are run sequentially on the parse.
 */
public interface HtmlParseFilter extends Pluggable, Configurable {
  /** The name of the extension point. */
  final static String X_POINT_ID = HtmlParseFilter.class.getName();

  /**
   * Adds metadata or otherwise modifies a parse of HTML content,
   * given the DOM tree of a page.
   */
  ParseResult filter(Content content, ParseResult parseResult,
      HTMLMetaTags metaTags, DocumentFragment doc);

}
```

**Ejemplo 5**

Como se puede ver en el Ejemplo 5, esta interfaz tiene un método llamado *filter*. Dentro de este método se realiza la implementación que se desee. Para ello proporciona unos parámetros para ello.

Los más relevantes son el *Content* y el *ParseResult*. A través del *Content* se puede acceder al contenido en texto plano de la página. Con *ParseResult* contiene datos relevantes sobre la página que está siendo procesado como la URL. Este objeto también contiene los metadatos, a través de los cuales vamos a conseguir la persistencia de lo que extraigamos a lo largo de todo el proceso de *crawling*. Es decir, este objeto también actúa como la salida de *filter.*

### *Detector de OGC*

Como se ha comentado antes la primera de las tareas que se realizan en este plugin es la detección del tipo de documento OGC. Esta detección solo se realiza si el tipo de contenido de la página es XML. Para ellos se extrae el nodo raíz del fichero XML para analizarlo en busca de datos que nos permitan establecer de qué tipo se tratan. A continuación, se pueden ver varios ejemplos de cabeceras para los distintos formatos de ficheros OGC.

Este es un ejemplo para WMS, en su versión 1.3.0.

```
<WMS_Capabilities
xmlns:esri_wms="http://www.esri.com/wms"
xmlns:inspire_vs="http://inspire.ec.europa.eu/schemas/inspire_vs/1.0"
xmlns:gml="http://schemas.opengis.net/gml"
xmlns:inspire_common="http://inspire.ec.europa.eu/schemas/common/1.0"
xmlns="http://www.opengis.net/wms"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.3.0"
updateSequence="2619"
xsi:schemaLocation="http://inspire.ec.europa.eu/schemas/inspire_vs/1.0
http://inspire.ec.europa.eu/schemas/inspire_vs/1.0/inspire_vs.xsd">
```

**Ejemplo 6**

Ejemplo de CSW con versión 2.0.2.

```
<csw:Capabilities
xmlns:csw=http://www.opengis.net/cat/csw/2.0.2
xmlns:gml=http://www.opengis.net/gml
xmlns:gmd=http://www.isotc211.org/2005/gmd
xmlns:ows=http://www.opengis.net/ows
xmlns:ogc=http://www.opengis.net/ogc
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance”
xmlns:inspire_ds="http://inspire.ec.europa.eu/schemas/inspire_ds/1.0"
xmlns:inspire_com=http://inspire.ec.europa.eu/schemas/common/1.0
version="2.0.2"
xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
http://schemas.opengis.net/csw/2.0.2/CSW-discovery.xsd">
```

**Ejemplo 7**

Ejemplo de WFS con versión 1.1.0.

```
<wfs:WFS_Capabilities
xmlns:wfs=http://www.opengis.net/wfs
xmlns:ogc=http://www.opengis.net/ogc
xmlns:gml=http://www.opengis.net/gml
xmlns:ows=http://www.opengis.net/ows
xmlns:xsi=http://www.w3.org/2001/XMLSchema-istance
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:ComarcasAgrarias=http://wms.magrama.es/sig/WFS/ComarcasAgrarias/
wfs.aspx version="1.1.0"
xsi:schemaLocation="http://www.opengis.net/gml
http://schemas.opengis.net/gml/3.1.1/base/gml.xsd
http://www.opengis.net/ogc
http://schemas.opengis.net/filter/1.1.0/filter.xsd
http://www.opengis.net/ows
http://schemas.opengis.net/ows/1.0.0/owsAll.xsd
http://www.opengis.net/wfs
http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
```

**Ejemplo 8**

Ejemplo WCS con versión 2.0.1.

```
<wcs:Capabilities
xmlns:wcs=http://www.opengis.net/wcs/2.0
xmlns:ows=http://www.opengis.net/ows/2.0
xmlns:ogc=http://www.opengis.net/ogc
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:gml=http://www.opengis.net/gml/3.2
xmlns:gmlcov=http://www.opengis.net/gmlcov/1.0
xmlns:swe=http://www.opengis.net/swe/2.0
xsi:schemaLocation="http://www.opengis.net/wcs/2.0
http://schemas.opengis.net/wcs/2.0/wcsAll.xsd "
version="2.0.1">
```

**Ejemplo 9**

Ejemplo de WPS con versión 0.4.0.

```
<wps:Capabilities
xmlns:wps=http://www.opengeospatial.net/wps
xmlns:ows=http://www.opengeospatial.net/ows
xmlns:xlink=http://www.w3.org/1999/xlink
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="0.4.0"
xsi:schemaLocation="http://www.opengeospatial.net/wps..\wpsGetCapabili
ties.xsd">
```

**Ejemplo 10**

Como estas existen otras cabeceras distintas dependiendo del tipo de documento. Analizándolas se puede saber de qué tipo se tratan.

### *Anchor y contexto*

Otra de las tareas principales que se realizan en este *plugin* es la extracción tanto del *anchor* como de su contexto. Como se ha comentado al inicio de este apartado, el contexto es el conjunto de caracteres (contabilizando solo texto plano, es decir, sin etiquetas de XML o HTML) alrededor del *anchor.* El número de caracteres que se tienen en cuenta para el contexto se puede configurar en el fichero de configuración nutchsite.xml mediante la regla `ogc.outlink.anchor.context`.

```
  <property>
    <name>ogc.outlink.anchor.context</name>
    <value>30</value>
    <description>Boundary</description>
  </property>
```

**Ejemplo 11**

En el Ejemplo 11 se puede ver la configuración para que se seleccionen 30 caracteres hacia delante y 30 hacia atrás para formar el contexto. En la Figura 5: Ejemplo de *anchor* y contexto se puede ver un ejemplo donde la parte en rojo sería el *anchor* del enlace. Por otro lado, la parte amarilla podría corresponder a el contexto de ese mismo *anchor*.


**Figura 5: Ejemplo de *anchor* y contexto**

Estos elementos se guardan en los metadatos que acompañaran a este documento por todo el flujo de trabajo de Nutch. Será utilizado posteriormente en el cálculo del *score.*

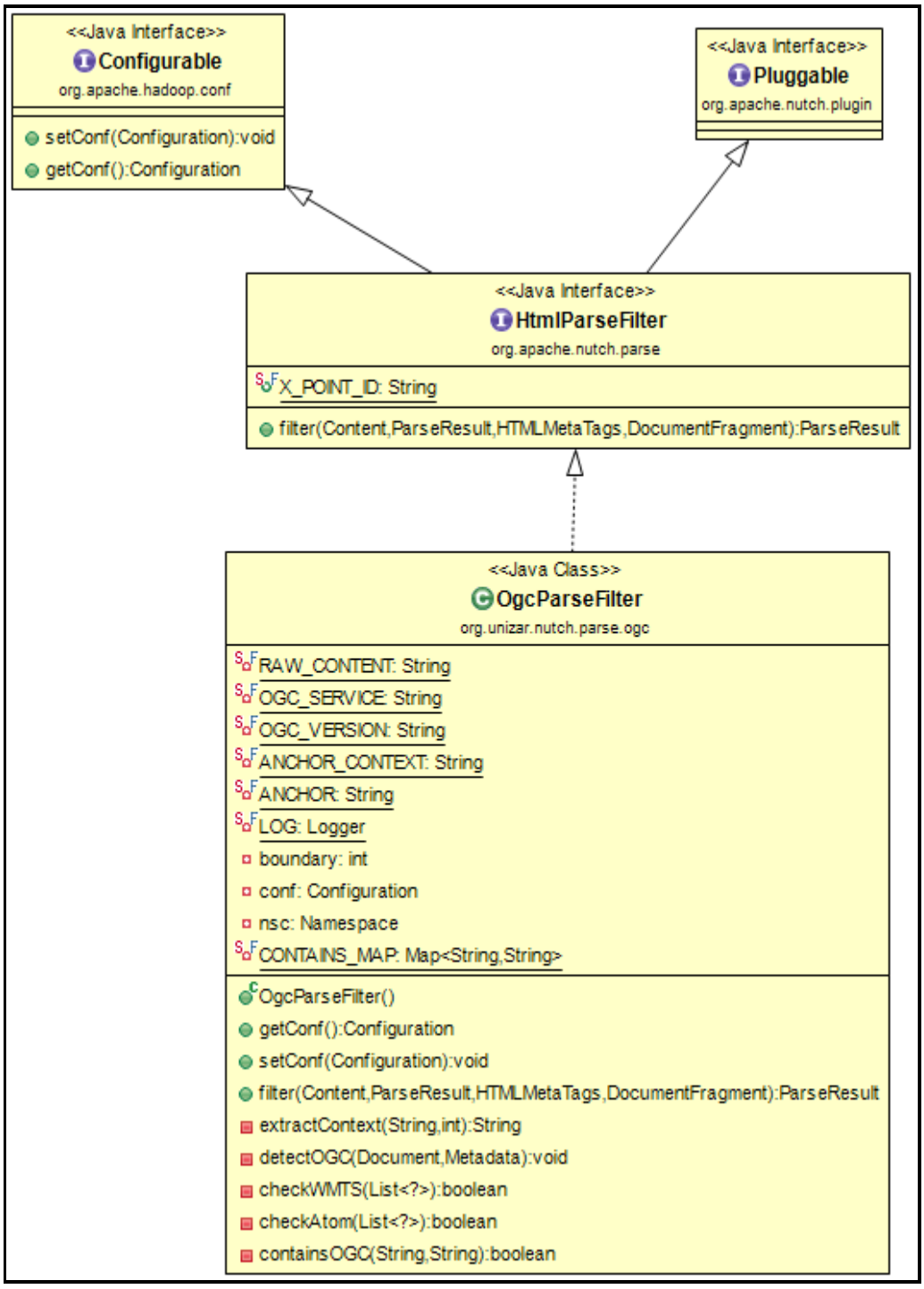



**Figura 6: Diagrama de clases OgcParseFilter**

El código con las modificaciones comentadas puede consultarse en Anexo D – Código de interés. Concretamente en la clase `OgcParseFilter`.

### 4.2.2. OgcIndexingFilter

Este segundo *plugin* explota el punto de extensión *IndexingFilter*. Este punto de extensión es fundamental para indicar a Nutch si queremos que el documento que se está filtrando tiene que ser indexado o no.

Para ellos se analizan los metadatos para buscar si en el *plugin* anterior se ha marcado el documento como un documento OGC. Si este campo está en los metadatos el documento se indexará.

En este caso la interfaz que se extiende es `IndexingFilter`. A continuación, se puede ver el código de la interfaz.

```
/**
 * Extension point for indexing. Permits one to add metadata to the
 * indexed fields. All plugins found which implement this extension
 * point are run sequentially on the parse.
 */
public interface IndexingFilter extends Pluggable, Configurable {
  /** The name of the extension point. */
  final static String X_POINT_ID = IndexingFilter.class.getName();

  /**
   * Adds fields or otherwise modifies the document that will be
   * indexed for a parse. Unwanted documents can be removed from
   * indexing by returning a null value.
   *
   * @param doc   document instance for collecting fields
   * @param parse parse data instance
   * @param url   page url
   * @param datum crawl datum for the page (fetch datum from segment
   * containing fetch status and fetch time)
   * @param inlinks page inlinks
   * @return modified (or a new) document instance, or null (meaning
   * the document should be discarded)
   * @throws IndexingException
   */
  NutchDocument filter(NutchDocument doc, Parse parse, Text url,
      CrawlDatum datum, Inlinks inlinks) throws IndexingException;

}
```

**Ejemplo 12**

Los metadatos pueden ser accedidos a través del objeto *Parse*. Conos se ha comentado antes se analizan esos elementos para determinar si se debe indexar o no. En caso de que se deba indexar, el método debe devolver el objeto *NutchDocument* con las modificaciones que le hayamos realizado, en nuestro caso trasvasamos los metadatos desde el objeto *Parse* al objeto *NutchDocument.*

A través de este objeto los metadatos serán propagados por el resto del flujo de trabajo. En caso de que no queramos que se indexe, debemos devolver *null.* Esto será suficiente para que el documento pase a no tenerse en cuenta en la fase de indexado.

Es importante recordar que estos filtros que se han comentado son llamados por cada documento a procesar, siguiendo una arquitectura de Map-Reduce.

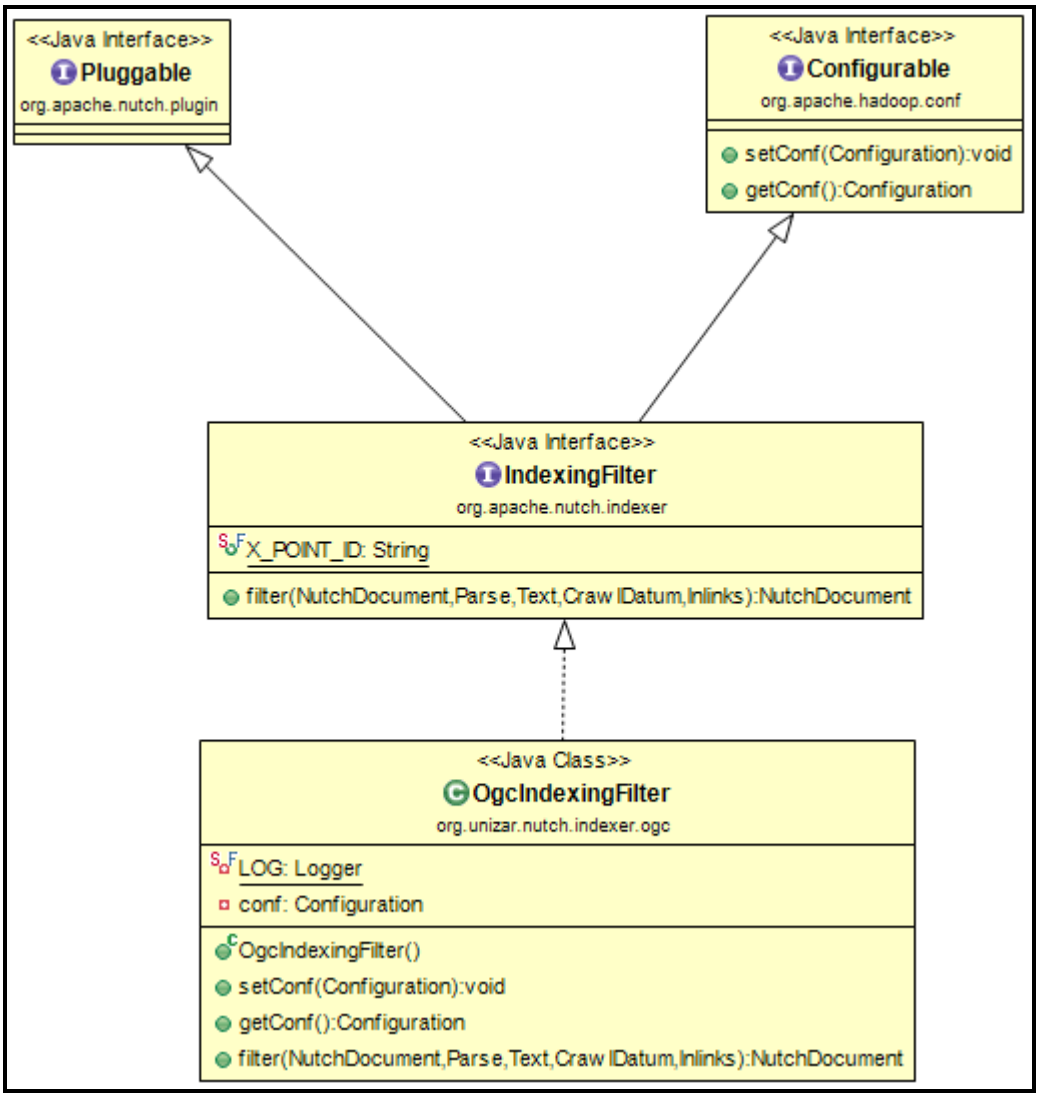



**Figura 7: Diagrama de clases de OgcIndexingFilter**

El código con las modificaciones comentadas puede consultarse en Anexo D – Código de interés. Concretamente en la clase `OgcIndexingFilter`.

### 4.2.3. SharkScoringFilter

Este *plugin* explota el punto de extensión *ScoringFilter*. Este punto de extensión es fundamental para la tarea de conseguir que un *crawler* sea de verdad enfocado. Actúa en la mayoría de las tareas de Nutch para modificar el score del documento está siendo procesado. Dependiendo de la tarea en la que se encuentra contamos con una información u otra para alterar el *score* de un documento. Dependiendo del *score* de un documento, este será seleccionado o no en la fase de *fetch* para su procesamiento. La interfaz que se extiende la implementación que se ha realizado para este trabajo es `ScoringFilter`.

```
/**
 * A contract defining behavior of scoring plugins.
 *
 * A scoring filter will manipulate scoring variables in CrawlDatum
 * and in resulting search indexes. Filters can be chained in a
 * specific order, to provide multi-stage scoring adjustments.
 *
 * @author Andrzej Bialecki
 */
public interface ScoringFilter extends Configurable, Pluggable {
  /** The name of the extension point. */
  final static String X_POINT_ID = ScoringFilter.class.getName();

  public void injectedScore(Text url, CrawlDatum datum)
      throws ScoringFilterException;

  public void initialScore(Text url, CrawlDatum datum)
      throws ScoringFilterException;

  public float generatorSortValue(Text url, CrawlDatum datum, float
initSort) throws ScoringFilterException;

  public void passScoreBeforeParsing(Text url, CrawlDatum datum,
Content content) throws ScoringFilterException;

  public void passScoreAfterParsing(Text url, Content content, Parse
parse) throws ScoringFilterException;

  public CrawlDatum distributeScoreToOutlinks(Text fromUrl,
      ParseData parseData, Collection<Entry<Text, CrawlDatum>>
targets, CrawlDatum adjust, int allCount) throws
ScoringFilterException;

  public void updateDbScore(Text url, CrawlDatum old, CrawlDatum
datum, List<CrawlDatum> inlinked) throws ScoringFilterException;

    public float indexerScore(Text url, NutchDocument doc, CrawlDatum
dbDatum, CrawlDatum fetchDatum, Parse parse, Inlinks inlinks, float
initScore) throws ScoringFilterException;
}
```

**Ejemplo 13**

Como se puede ver en el Ejemplo 13, esta interfaz tiene un número más elevado de métodos respecto a las comentadas anteriormente en la implementación de otros *plugins*. Dentro de cada uno de ellos, se realiza la implementación que se desee para el punto concreto en el que actúan. A continuación, se va a explicar en qué fase del flujo de trabajo de Nutch actúa cada uno.

*InjectedScore*

Con este método se puede establecer un *score* inicial para las páginas nuevas que son inyectadas. Las páginas inyectadas pueden no tener links de entrada o *inlinks*. Por esta razón es recomendable darle un valor que no sea cero para darles un crédito inicial a las páginas inyectadas.

En este trabajo, las páginas inyectadas son las semillas con las que se inicia el proceso de *crawling*.

Este método es llamado durante la fase de inyección.

*InitialScore*

Asigna un score inicial a las páginas que son descubiertas por primera vez. Las páginas que son descubiertas por primera vez durante el flujo de trabajo de Nutch tienen, al menos, un enlace de entrada o *inlink*.

Este enlace actúa como un padre y tiene un *score* asignado previamente el cual contribuye para alterar el *score* de esta página. Por lo tanto, se suele asignar un valor inicial de cero para no alterar el que será asignado posteriormente debido a sus páginas "padre".

Este método es llamado durante varias fases, dependiendo de cuando es descubierto el nuevo enlace. Normalmente es en la fase de *parse* o *fetch*.

*GeneratorSortValue*

Este método devuelve un valor de ordenación con el objetivo de seleccionar los N documentos con el score más alto para ser procesados.

Este método es llamado durante la fase de generación, los selecciona para que en la fase de *fetch* sean procesados.

*PassScoreBeforeParsing*

Este método coge toda la información relevante acerca del *score*. Esta información se extrae del *datum.* El *datum* es el elemento con el que se consigue la persistencia hasta el momento del *parse*. En él está contenida toda la información acerca de un URL.

La idea de este método es traspasar toda la información desde el *datum* a los metadatos del documento. Esto es necesario para pasar el valor del score al mecanismo que lo distribuye a sus enlaces de saluda o *outlinks*.

*PassScoreAfterParsing*

Se usa para guardar parte de la información resultante del proceso de *parsing* para la distribución del *score* entre las páginas hijas. Es necesario la presencia de este método para asegurarnos de tener los datos en los próximos pasos del proceso.

Es llamado durante el proceso de *parse*, en la finalización del mismo.

*DistributeScoreToOutlinks*

Este método es el más importante de todos. A través de este método se recoge toda la información guardada a lo largo de los anteriores métodos y con ella se distribuye según se desee a los enlaces de salida o *outlinks*.

Para este trabajo se ha implementado un algoritmo de búsqueda en la web que se adaptaba a las necesidades de este *crawler.* El algoritmo escogido ha sido el *Shark-Search*.

Este algoritmo será presentado y explicado en un apartado posterior.

Para conseguir esto se proporcionan los siguientes parámetros:

- **fromUrl:** Enlace de la página que está siendo procesada.
- **parseData:** En este objeto se guarda información relevante sobre el *score* en los metadatos. Instancia de *ParseData.*

- **targets:** Son pares de objetos Url y *CrawlDatum*. Modificaremos el *CrawlDatum* de los enlaces hijos.
- **adjust:** Es una instancia de *CrawlDatum* que puede ser usada para pasar ajustes en los valores al *CrawlDatum* original de este documento. En este trabajo no se hace uso de él.
- **allCount:** Numero de *outlinks* o enlaces hijos recolectados en el documento actual.

Este método es llamado también en la parte final del proceso de *parsing*.

### *UpdateDbScore*

Este método calcula un nuevo *score* y lo almacena en un nuevo *CrawlDatum* durante la actualización de la *CrawlDb*. Para ello utiliza los valores del *CrawlDatum* original además de los valores del *score* aportados por sus enlaces padre o *inlinks*.

Este método es llamado en la fase de actualización de la *CrawlDb*.

### *IndexerScore*

Este método se usa para aplicar cambios al *score* enfocados a la indexación en Lucene.

Para este trabajo no se ha utilizado este método debido a que no son necesarios los *scores* en la fase de indexación.

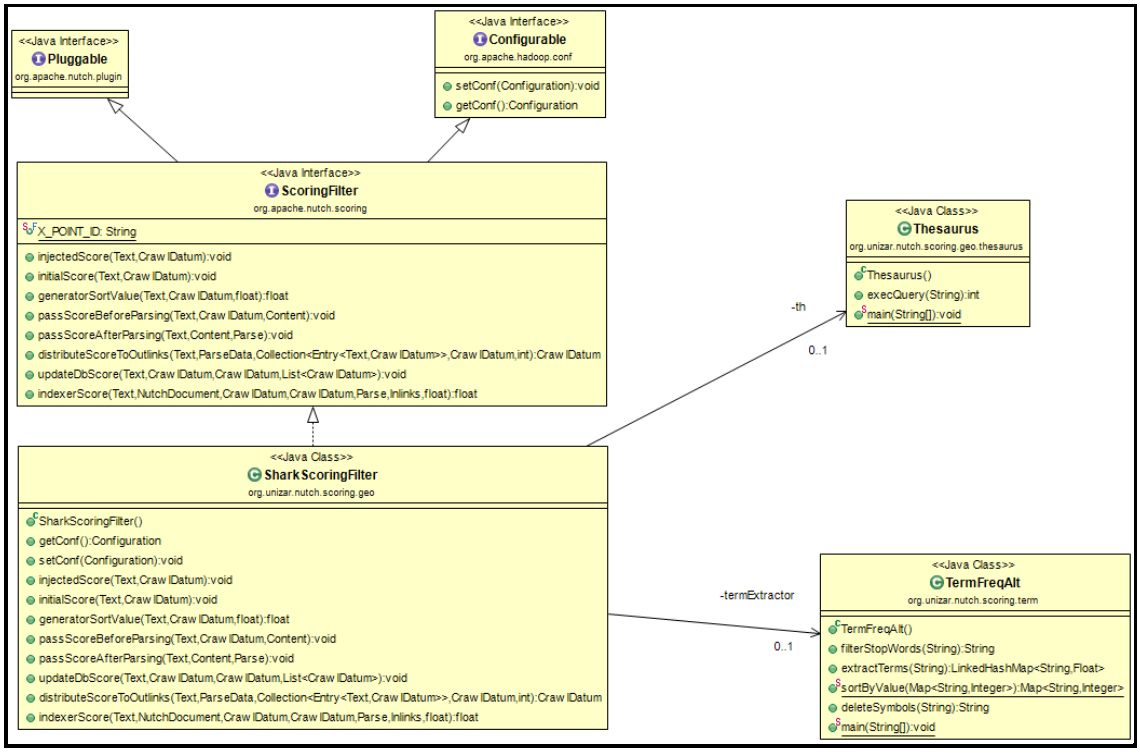



**Figura 8: Diagrama de clases de SharkScoringFilter**

El código con las modificaciones comentadas puede consultarse en Anexo D – Código de interés. Concretamente en la clase `SharkScoringFilter`.

### 4.2.4. OgcIndexWriter

Este ha sido el último *plugin* que se ha necesitado implementar para darle al *crawler* el comportamiento deseado. Este explota el punto *IndexWriter*. Este punto de extensión es el último paso para que el crawler guarde lo que ha ido recolectando.

En este trabajo se almacenan los ficheros en su formato original (XML) en el disco duro. Se crea un fichero por cada documento recuperado y se almacena.

En este caso la interfaz que se extiende es `IndexWriter`. A continuación, se puede ver el código de la interfaz.

```
public interface IndexWriter extends Pluggable, Configurable {
  /** The name of the extension point. */
  final static String X_POINT_ID = IndexWriter.class.getName();

  public void open(JobConf job, String name) throws IOException;

  public void write(NutchDocument doc) throws IOException;

  public void delete(String key) throws IOException;

  public void update(NutchDocument doc) throws IOException;

  public void commit() throws IOException;

  public void close() throws IOException;

  /**
   * Returns a String describing the IndexWriter instance and the
   * specific parameters it can take
   */
  public String describe();

}
```

**Ejemplo 14**

El punto de extensión que ha sido utilizado en esta sección está más enfocado a indexaciones en bases de datos y otros elementos de persistencia de datos más complejos que el guardado en el sistema de ficheros.

Por esa razón este punto de extensión ofrece varios métodos para conseguir una persistencia más sofisticada.

Los métodos más relevantes proporcionados por esta interfaz son:

- open
- write
- delete
- update
- commit
- close

Para este trabajo no se necesitaba una gestión tan sofisticada. Por lo tanto, se ha utilizado únicamente el método *write.*

Para la creación y escritura de los documentos recolectados por el *crawler* se han utilizado las clases File y `BufferedWriter`.

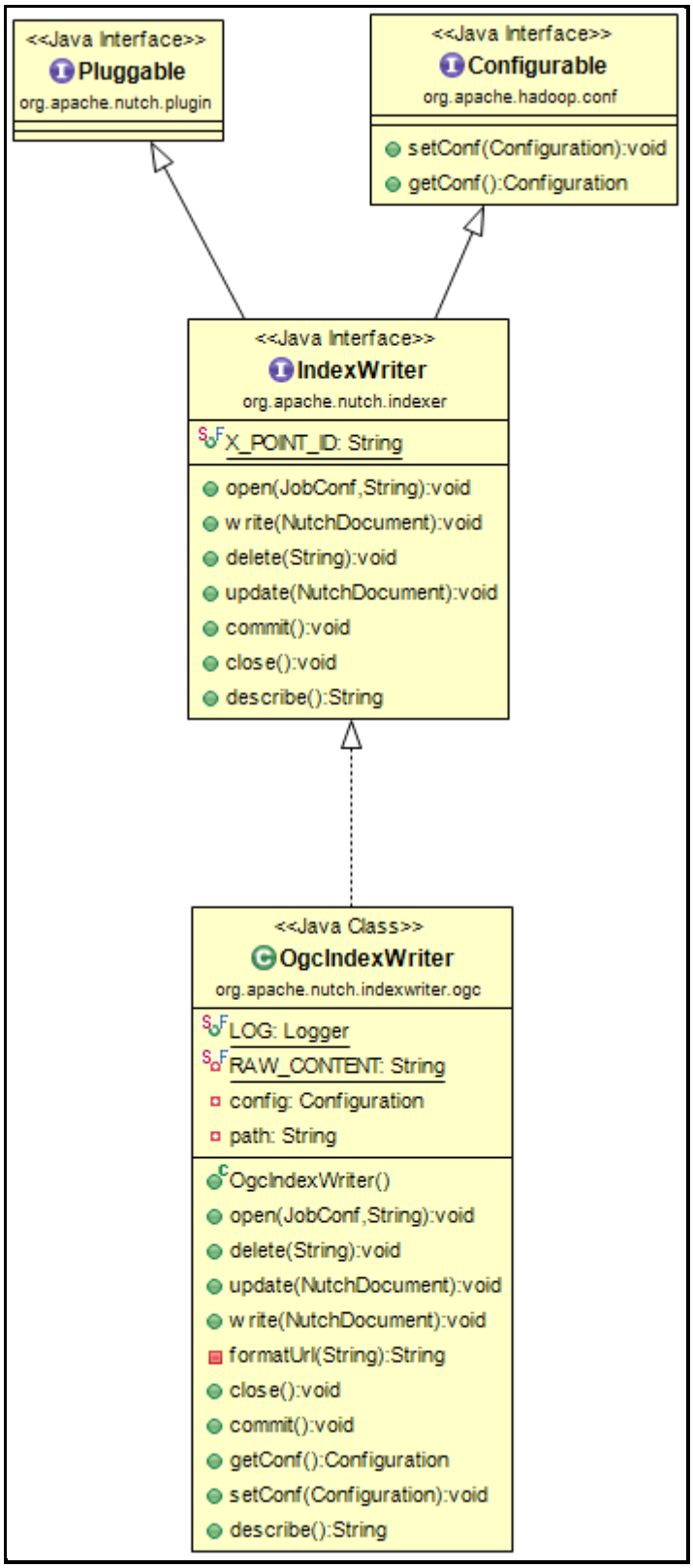



**Figura 9: Diagrama de clases de OgcIndexWriter**

El código con las modificaciones comentadas puede consultarse en Anexo D – Código de interés. Concretamente en la clase `OgcIndexWriter`.

## 4.3. Algoritmo búsqueda

Se barajaron varios algoritmos de búsqueda en la web para ser aplicados en este trabajo. Todos ellos son útiles para su integración en el proceso de expansión de un *crawler* enfocado.

A continuación, se van a comentar las diferentes opciones que se barajaron[12]:

- Fish-Search Algorithm
- Shark-Search Algorithm

### *Fish-Search Algorithm*

Este algoritmo fue creado para búsquedas dinámicas en la web. Puede ser explicado a través de la siguiente metáfora:

> *Si los agentes que buscan información en la web son vistos como una escuela de peces en el mar. Cuando se encuentra la comida (información relevante), los peces (agentes de búsqueda) se reproducen y siguen buscando comida. Si no hay comida o el agua está contaminada (malas condiciones en el ancho de banda, por ejemplo) mueren.*

En este algoritmo se toman una URL semilla y una *query* de búsqueda. Con esto va construyendo una lista con prioridad de las siguientes URLs a ser exploradas.

En cada paso el primer nodo es sacado de la lista y procesado. Se extrae el texto disponible y es analizado evaluando si es relevante o no (valores 0 ó 1). Basado en ese *score* una heurística decide si continuar la exploración en esa dirección o no.

### *Shark-Search Algoritm*

Este algoritmo es una evolución del explicado anteriormente. Se optó por implementar este debido a que discrimina menos las páginas no relevantes y puede llevar al descubrimiento de más información.

La primera mejora respecto al *Fish-Search* es el uso de un valor continuo en la evaluación de la relevancia de un documento, sustituyendo el uso de valores binarios *relevante* o *irrelevante.* Ahora se asignará un valor continuo entre 0 (nada relevante) y 1 (relevancia conceptual perfecta).

Otra de las mejoras que tiene este algoritmo es la herencia del *score*. Ahora el *score* de los ancestros de un nodo afectará siguiendo la cadena de descendientes. Los nodos usan el *score* heredado para calcular el suyo propio. El valor del *score* heredado se multiplica por un factor de decaimiento o $\delta$.

Este algoritmo también tiene en cuenta el *anchor* y el contexto del mismo a la hora de calcular el *score* de un nodo. Se utilizan dos constantes $\beta$ y $\gamma$ para calibrar el peso que se le da a cada elemento del *score.*

$\beta$ se utiliza para calibrar a que componente del *score* de vecindad se le da más peso. El *score* de vecindad sale del cálculo de la relevancia del *anchor* y de su contexto.

Por lo tanto:

$$S_{vec} = \beta * \text{anchorScore} + (1 - \beta) * anchorContextScore$$

La otra constante se utiliza para un procedimiento similar. La diferencia es que $\gamma$ se utiliza para elegir el peso entre el *score* heredado y el *score* de vecindad.

Así:

$$S_{pot} = \gamma * \text{inheritedScore} + (1 - \gamma) * neighbourhoodScore$$

A continuación, se puede ver el algoritmo de *Shark-Search*[12]:

1. **Calcular** el score heredado por un nodo hijo(inherited_score) de la siguiente manera:
   - **Si** relevancia*(nodo_actual) > 0* (el nodo actual es relevante)

     **Entonces** *inherited_score = $\delta$ * relevancia(nodo_actual)*
     donde $\delta$ es el factor de decaimiento (predefinido).
   - **Sino** inherited_*score = $\delta$ * inherited_score(nodo_actual)*
2. **Dado** anchor_*text* como el contenido del anchor que apunta al nodo_hijo y *anchor_text_context* el contenido del contexto del anchor (hasta unos límites definidos)
3. **Calcular** la relecancia del texto del anchor como

   *anchor_score = relevancia(anchor_text)*
4. **Calcular** la relecancia del context del anchor de la siguiente manera:
   - **Si** *anchor_score > 0*,
     **Entonces** *anchor_context_score = 1*
     **Sino** *anchor_context_score = relevancia(anchor_text_context)*
5. **Calcular** el score general del anchor, deinido como score_vecindad, de la siguiente manera:
   score_vecindad = *$\beta$ * anchor_score + (1-$\beta$) * anchor_context_score*
   donde $\beta$ es una constante predefinida
6. **Calcular** el score potencial (potential_score) del hijo de la siguiente manera:
   *potential_score(nodo_hijo) = $\gamma$ * inherited_score(nodo_hijo) + (1-$\gamma$) *** score_vecindad *(nodo_hijo)*
   donde $\gamma$ es una constante predefinida.

Otros algoritmos que no se han considerado para usar, pero se han investigado [1]:

- InfoSpider
- Learning Anchor Algorithm
- History Path Algorithm

El código con el algoritmo comentad puede consultarse en Anexo D – Código de interés. Concretamente en la clase `SharkScoringFilter`.

## 4.4. Tesauro

En el algoritmo *Shark-Search* una componente principal es la estimación de relevancia de una página, así como del *anchor* y su contexto.

Una de las tareas más importantes fue diseñar una manera de conseguir averiguar la relevancia. No se buscaba una solución que utilizara una *query* dado que no se busca

por elementos temáticos concretos. En este trabajo se buscan documentos que describirán información geográfica de muchos tipos.

Entre estos tipos estaban:

- Naturaleza
- Ríos
- Medio Ambiente
- Carreteras
- Zonas de pesca

El problema era que los documentos podían ser de muchos temas diferentes. Después de pensar varias soluciones, se decidió hacer uso de un tesauro[11].

### *Idea básica*

La idea principal era extraer de un documento las palabras más relevantes. Estas palabras tendrían un ranking de relevancia dentro del documento. Además de ese ranking, las palabras eran añadidas a una consulta SPARQL contra un modelo RDF que contenía el tesauro. dependiendo del número de resultados la palabra tiene más o menos valor geográfico.

### *Extracción de términos más relevantes*

Para esta tarea no se utilizó ninguna herramienta externa o ya realizada. Se implementó un extractor de términos relevantes propio.

La extracción cuenta con varias fases:

- Eliminación de caracteres innecesarios (. ,: +"'-{}[]?¿€$%& …)
- Paso a minúsculas de todo el texto
- Eliminación de espacios múiltiples
- Filtrado de *stopwords*[12]
- Se cuentan todas las palabras únicas del texto
- Se ordenan de mayor numero de apariciones a menos
- Cálculo del score de la siguiente manera:

$$score_{term} = \frac{nAp_{term}}{nPal} * nRep$$

Donde:

$nAp_{term}$ → Número de apariciones del término

$nPal$ → Número de palabras únicas en el texto

$nRep$ → Número total de palabras, aunque estén repetidas

### *GEMET*

GEMET o **GE**neral **M**ultilingual **E**nvironmental **T**hesaurus es un tesauro multilenguaje desarrollado por la Agencia Europea de Medio Ambiente. Es una compilación de varios

[11] Tesauro: Lista de palabras o términos utilizados para representar conceptos.

[12] Lista de palabra que deben ser ignoradas en estos cálculos (de, el, la, y…)

vocabularios multilenguaje. Tiene el objetivo de definir una terminología general para el camp del medio ambiente. La versión actual contiene 22 idiomas y contiene más de 6000 descriptores. Hace uso del modelo SKOS[13] para representar la estructura y contenido de esquemas conceptuales para implementar el tesauro. SKOS está basado en RDF.

*RDF*

El RDF o Resource Description Framework es una especificación del W3C[14]. Se utiliza como método general para descripción conceptual. Es parte clave también del LOD[15]. El tesauro de GEMET está formado por varios ficheros RDF, juntando estos ficheros se consigue el modelo completo para poder ser utilizado por nuestro *crawler*. Estos ficheros son:

- gemet-skoscore.rdf: En este fichero se definen los conceptos que posee el tesauro y las relaciones entre los estos. Ver Figura 10: Extracto del fichero gemet-skoscore.rdf.
- gemet-backbone.rdf: En este fichero se definen temas y los conceptos que pertenecen a ellos. También se definen grupos y supergrupos de conceptos. Ver Figura 11: Extracto del fichero gemet-backbone.rdf.
- gemet-definitions-es.rdf[16]: Este documento contiene las definiciones de los conceptos escritas en el idioma correspondiente. Ver Figura 12: Extracto del ficherogemeto-definitions-es.rdf.
- gemet-groups-es.rdf: Este documento contiene las definiciones de los grupos, supergrupos y temas definidos en el fichero gemet-backbone.rdf. Ver Figura 13: Extracto del fichero gemet-groups-es.rdf.

[13] Simple Knowledge Organization System

[14] World Wide Web Consortium

[15] Linked Open Data

[16] El *es* referencia al idioma español, existen para los 22 idiomas soportados.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xml:base="http://www.eionet.europa.eu/gemet/">

<skos:ConceptScheme rdf:about="gemetThesaurus">
  <rdfs:label>The GEMET Thesaurus</rdfs:label>
</skos:ConceptScheme>

<skos:Concept rdf:about="concept/9299">
  <skos:inScheme rdf:resource="gemetThesaurus"/>
  <skos:closeMatch rdf:resource="http://data.uba.de/umt/_00028854" />
  <skos:broader rdf:resource="concept/4257" />
</skos:Concept>

<skos:Concept rdf:about="concept/4027">
  <skos:inScheme rdf:resource="gemetThesaurus"/>
  <skos:broader rdf:resource="concept/7622" />
</skos:Concept>
. . .
```

Figura 10: Extracto del fichero gemet-skoscore.rdf

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
       xmlns:skos="http://www.w3.org/2004/02/skos/core#"
       xmlns:gemet="http://www.eionet.europa.eu/gemet/2004/06/gemet-schema.rdf#"
  xml:base="http://www.eionet.europa.eu/gemet/">

<skos:Collection rdf:about="supergroup/">
   <rdfs:label>GEMET Supergroups</rdfs:label>
   <skos:prefLabel>Supergroups</skos:prefLabel>
   <skos:inScheme rdf:resource="http://www.eionet.europa.eu/gemet/gemetThesaurus"/>
   <skos:member rdf:resource="supergroup/5306"/>
   <skos:member rdf:resource="supergroup/4044"/>
   <skos:member rdf:resource="supergroup/5499"/>
   <skos:member rdf:resource="supergroup/2894"/>
</skos:Collection>

<skos:Collection rdf:about="group/">
   <rdfs:label>GEMET Groups</rdfs:label>
   <skos:prefLabel>Groups</skos:prefLabel>
   <skos:inScheme rdf:resource="gemetThesaurus"/>
   <skos:member rdf:resource="group/10111"/>
   <skos:member rdf:resource="group/10112"/>
   <skos:member rdf:resource="group/10114"/>
   <skos:member rdf:resource="group/10117"/>
   <skos:member rdf:resource="group/10118"/>
. . .
```

Figura 11: Extracto del fichero gemet-backbone.rdf

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:skos="http://www.w3.org/2004/02/skos/core#"
        xmlns:gemet="http://www.eionet.europa.eu/gemet/2004/06/gemet-schema.rdf#"
        xml:lang="es">
<rdf:Description rdf:about="http://www.eionet.europa.eu/gemet/concept/7">
  <skos:prefLabel>zona industrial abandonada</skos:prefLabel>
</rdf:Description>
<rdf:Description rdf:about="http://www.eionet.europa.eu/gemet/concept/8">
  <skos:prefLabel>vehículos abandonados</skos:prefLabel>
</rdf:Description>
<rdf:Description rdf:about="http://www.eionet.europa.eu/gemet/concept/11">
  <skos:prefLabel>factor abiótico</skos:prefLabel>
</rdf:Description>
. . .
```

Figura 12: Extracto del ficherogemeto-definitions-es.rdf

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:lang="es"
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
        xmlns:skos="http://www.w3.org/2004/02/skos/core#"
        xmlns="http://www.eionet.europa.eu/gemet/2004/06/gemet-schema.rdf#">
<SuperGroup rdf:about="http://www.eionet.europa.eu/gemet/supergroup/4044">
  <rdfs:label>ACTIVIDADES Y PRODUCTOS HUMANOS, EFECTOS SOBRE EL MEDIO
  AMBIENTE</rdfs:label>
</SuperGroup>
<SuperGroup rdf:about="http://www.eionet.europa.eu/gemet/supergroup/2894">
  <rdfs:label>ASPECTOS SOCIALES, POLITICA AMBIENTAL, MEDIDAS</rdfs:label>
</SuperGroup>
<SuperGroup rdf:about="http://www.eionet.europa.eu/gemet/supergroup/5499">
  <rdfs:label>MEDIO NATURAL, MEDIO ANTRÓPICO</rdfs:label>
</SuperGroup>
<Group rdf:about="http://www.eionet.europa.eu/gemet/group/96">
  <subGroupOf rdf:resource="http://www.eionet.europa.eu/gemet/supergroup/2894" />
  <rdfs:label>ADMINISTRACIÓN, GESTIÓN, POLÍTICA, INSTITUCIONES,
  PLANIFICACIÓN</rdfs:label>
</Group>
. . .
```

Figura 13: Extracto del fichero gemet-groups-es.rdf

*JENA*

Para el tratamiento del modelo RDF se ha utilizado Apache Jena. Tanto para cargarlo como para las consultas SPARQL.

En la Figura 14: Consulta SPARQL se puede ver la consulta SPARQL que se realiza al tesauro.

```
SELECT distinct ?x WHERE {
?y rdf:type skos:Concept . ?y skos:prefLabel ?x .
FILTER regex(?x, '" + word + "', 'i') .
{ ?y gemet:theme " + themeGeo + "}
UNION
{ ?y gemet:theme " + themeAir + "}
UNION
{ ?y gemet:theme " + themeUrban + "}
}";
```

Figura 14: Consulta SPARQL

*Otros métodos*

El anterior modelo no es el único que se ha utilizado en la práctica con el *crawler*. Otro método que se ha implementado es la estimación del *score* por palabras clave que suelen llevar a recursos OGC. A esto último se le suma una estimación de importancia analizando la URL de la página. Si la URL contiene palabras consideradas de gran importancia el *score* de esa página aumenta considerablemente.

## 4.5. Modificaciones extra

A parte de los *plugins* comentados anteriormente se han añadido otras modificaciones que se han realizado al margen de los puntos de extensión. Cuando se estaban realizando las pruebas para comprobar el funcionamiento del *crawler* bastantes documentos OGC de no eran recogidos correctamente. Esto se debía a varios problemas. Estos eran:

- El fichero robots.txt no nos permitía acceder a algunos ficheros OGC. Ver Figura 15: Fichero robots.txt. Para solucionar este problema se modificó el código original de Nutch para conseguir que el *crawler* ignores los robots.txt.

```
User-agent: spatineo
Allow: /wms/request.php
Disallow: /

User-agent: *
Disallow: /
```

Figura 15: Fichero robots.txt

- Otro problema fue que Nutch por defecto coge hasta 65536 bytes de contenido en una página web. Si esta era de mayor longitud solo cogía los primeros 65536 bytes. Este parámetro se puede modificar en la configuración, a través de la regla http.content.limit. Estos parámetros se configuran en el fichero `nutch-site.xml`.
- El URLFilter de Nutch no acepta, por defecto, URLs que tengan alguno de estos caracteres [?*!@=]. Se eliminó esta regla de la lista de expresiones regulares, la cual está en el fichero `regex-urlfilter.txt`.

*Acelerando la fase de fetch*

Como se ha comentado en la sección de educación. Hay que mantener un cierto espacio temporal entre peticiones para no colapsar al *host* que está siendo accedido. No obstante, se han realizado modificaciones en la configuración base de Nutch para realizar la fase de *fetch* con mayor velocidad. Para ello, se han realizado las siguientes modificaciones:

- Se ha reducido el *delay* entre peticiones sucesivas al mismo servidor.
- Se ha aumentado el número de hilos que el *fetcher* puede usar.
- Se ha aumentado la capacidad de las colas de los hilos.

# 5. Resultados

Se han realizado pruebas con varias configuraciones, con y sin tesauro. El mayor éxito en un corto plazo de tiempo se consigue sin hacer uso del tesauro ya que las consultas al tesauro requieren un tiempo. Para ejecuciones largas, en las que se va a dejar al *crawler* actuar durante semanas es más interesante el uso del tesauro.

Para las pruebas que se han realizado durante la realización de este trabajo se ha utilizado el método alternativo que es más rápido. Los resultados obtenidos son prometedores, se han conseguido extraer satisfactoriamente la inmensa mayoría de los recursos objetivo.

## *5.1. Ejemplo de ejecución*

Para esta ejecución se usó como URL semilla la web del Consejo Superior Geográfico-Infraestructura de Datos Espaciales de España (http://www.idee.es). Se inició GeoCrawler con un número determinado de iteraciones a realizar, en este caso 30. La duración de la ejecución de las 30 iteraciones fue de 53 minutos recuperando en ese tiempo 2040 documentos de descripción de servicios de alguno de los estándares de OGC.

En la figura inferior se puede ver una captura de la carpeta de salida de GeoCrawler con la cantidad de elementos recuperados.

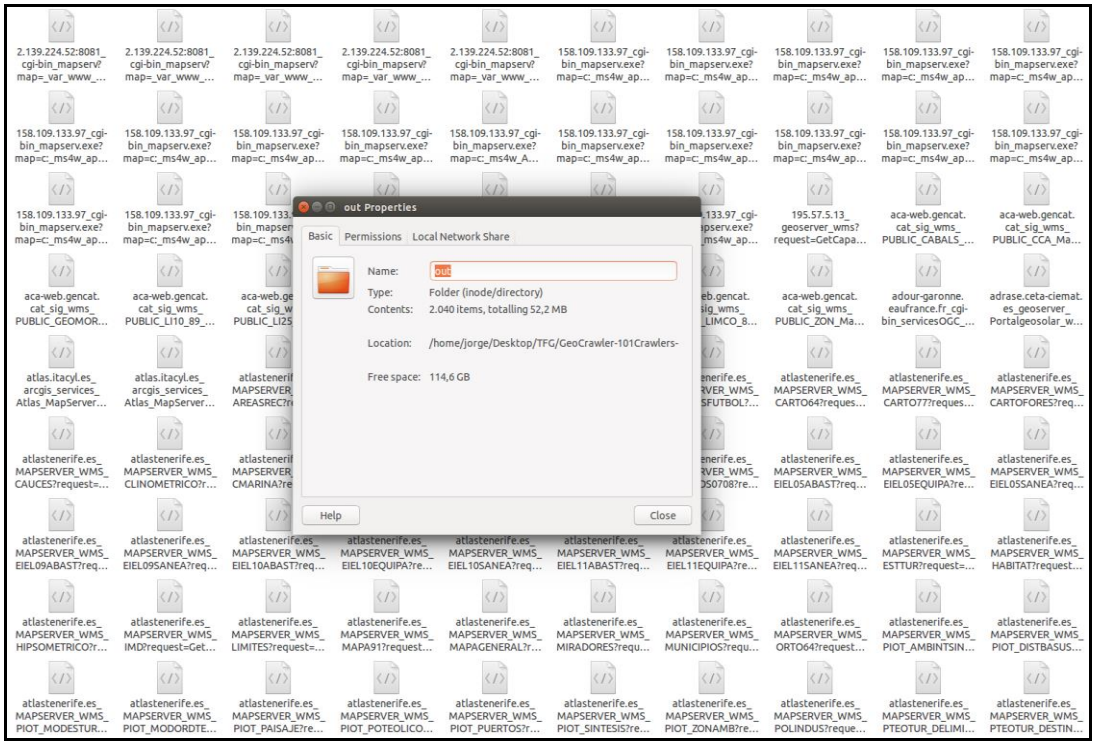



**Figura 16: Carpeta con los ficheros recopilados**

## *5.2. Problemas encontrados*

A la hora de recuperar elementos se reparó en que todos los ficheros OGC que eran encontrados y adquiridos en la fase de parse no eran indexados. En las primeras pruebas se encontraros dos motivos, uno solucionable y otro no.

El primero de ellos era las limitaciones de algunas webs respecto a los *crawlers*. Muchos tenían limitado su acceso por medio del fichero robots.txt. Como se ha comentado anteriormente en esta memoria, esto se puede solucionar ignorando dicho fichero.

El segundo problema no era solucionable. Corresponde al hecho de que algunos recursos ya no eran accesibles por diversos motivos. Estos motivos eran errores HTTP (ver Figura 17, Figura 19) o errores en el DNS (ver Figura 18).

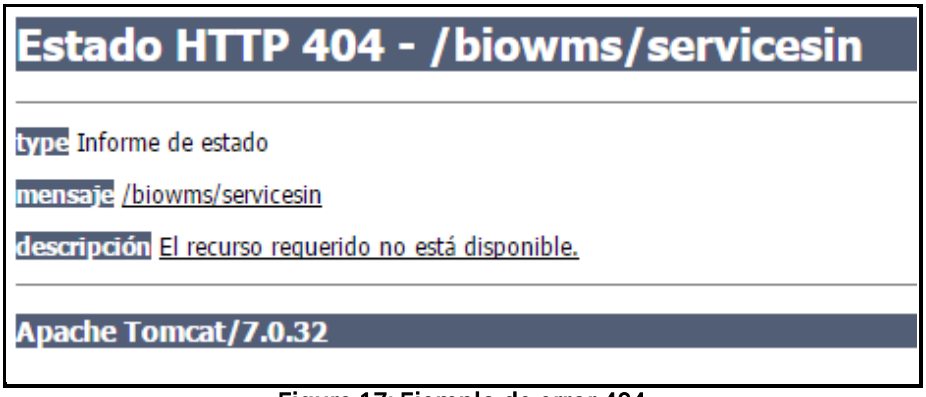



Figura 17: Ejemplo de error 404


Figura 18: Ejemplo de sitio inaccesible

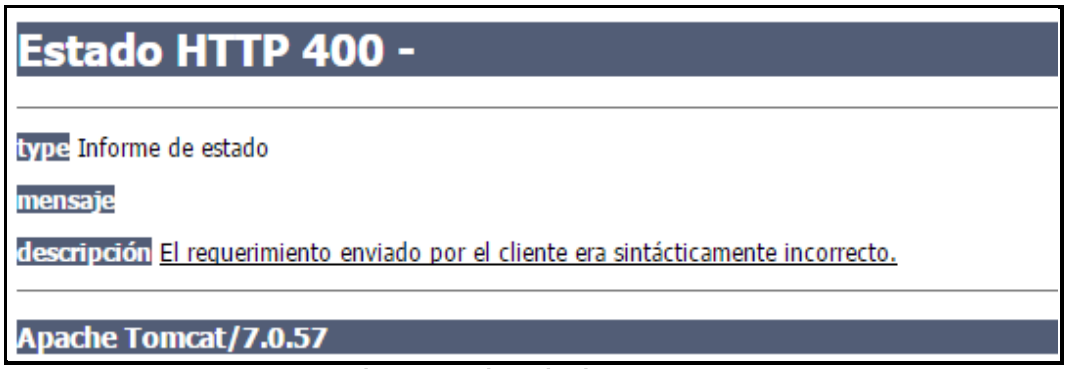



Figura 19: Ejemplo de error 400

# 6. Conclusiones y futuro

Realizar un *crawler enfocado* es una tarea laboriosa que requiere tiempo y dedicación. Se podría equiparar a una labor artesanal. La realización de un *crawler* de estas características tiene un gran interés para empresas.

Una de las cualidades más importante de este proyecto es que se trata de un trabajo realizado enteramente y a partir de software libre.

Han sido utilizadas alrededor de 300 horas para completar este trabajo. Se puede ver la información detallada en el Anexo E – Cronograma.

## *6.1. Conclusión personal*

Gracias a este trabajo he aprendido a gestionar mejor mi tiempo, una tarea que es clave para el futuro laboral. He aprendido a utilizar herramientas que anteriormente no conocía, de lo cual me siento muy satisfecho.

Este proyecto me ha permitido profundizar en disciplinas que ya había visto pero que no había usado mucho tiempo en ampliar mis conocimientos sobre ellas. He conseguido profundizar en mis conocimientos sobre la recuperación de información, sobre todo. Además, el conocimiento relacionado con los *crawlers* es de una gran importancia hoy en día con la cantidad de información que es publicada en la web.

He aprendido a realizar un trabajo basado en iteraciones cortas de dos semanas. Este tipo de organización no la había puesto en práctica hasta ahora. Este tipo de metodología te ayuda a llevar una visión más global sobre el proyecto en el que se está trabajando.

## *6.2. Conclusiones sobre el trabajo*

Como resultado de este trabajo se ha obtenido un cr*awler* enfocado en la búsqueda de documentos de descripción de servicios geográficos pertenecientes a los estándares definidos por la organización OGC.

Gracias a él se han recuperado miles de documentos pertenecientes a la geografía española. Por lo tanto, podemos decir que ha sido un éxito.

Para la realización de este trabajo han sido utilizadas muchas técnicas pertenecientes a varias disciplinas dentro la computación, como es la recuperación de información y la algoritmia. Se aplican una gran cantidad de conocimientos adquiridos a lo largo de mi formación.

A raíz de las pruebas realizadas con el *crawler* se ha llegado a unas conclusiones sobre los elementos que componían el mismo. Estas son:

- El uso del tesauro es lento, por esta razón, es mejor limitarla a usos más limitados y no en todas las páginas que es como fue diseñado este sistema.
- El uso de elementos alternativos para comprobar la importancia de una página web ha sido muy útil a la hora de acelerar GeoCrawler.
- Shark-Search ha resultado ser un algoritmo muy útil a la hora de realizar este trabajo gracias a su predisposición a ser incluido en un *crawler.*

## 6.3. Mejoras futuras

GeoCrawler no es una herramienta perfecta, caben mejoras que puedan hacerla una herramienta más completa y útil. A continuación, se enumeran las posibles mejoras:

- Búsqueda y recuperación de nuevas fuentes de información geográfica que no solo sean ficheros definidos por OGC.
- Mejora del comportamiento de expansión del *crawler*.
- Mejorar el sistema de *score*.
- Mejorar algoritmo de búsqueda.
- Inferir nuevas URLs a partir de algunas ya conocidas.

# 7. Bibliografía


[1] I. G. Dorado, "Focused Crawling: algorithm survey and new approaches with a manual analysis," 2008.

[2] B. Novak, "A SURVEY OF FOCUSED WEB CRAWLING ALGORITHMS."

[3] Wikipedia, "Web Crawler."

[4] Google, "Google Guides robots.txt."

[5] M. Koster, "Internet draft: A Method for Web Robots Control," 1996.

[6] "Robots.txt About."

[7] S. Conner, "An Extended Standard for Robot Exclusion."

[8] Wikipedia, "Apache Nutch."

[9] Wikipedia, "MapReduce."

[10] A. Foundation, "Apache Ant User Manual."

[11] Apache Foundation, "Apache Ivy Documentation."

[12] M. J. Michael Hersovici, D. P. Yoelle S. Maarek, and S. U. Menachem Shtalhaim, "THE SHARK-SEARCH ALGORITHM AN APPLICATION: TAILORED WEB SITE MAPPING."

[13] P. Schut, "Open Geospatial Consortium Inc . Corrigendum for OpenGIS Implementation Standard," *Processing*, 2009.

[14] Universidad Politecnica de Madrid, "Servicio de transformación de coordenadas (WCTS)."

[15] Wikipedia, "Web Map Service."

[16] Wikipedia, "Web Feature Service."

[17] Wikipedia, "Web Map Tile Service."

[18] Wikipedia, "Web Coverage Service."

[19] Wikipedia, "Catalog Service for the Web."

[20] Librosweb, "DOM."

# Anexo A – Operaciones de Nutch

Como se ha comentado en esta memoria Nutch se compone de secciones separadas que se van ejecutando secuencialmente. A parte de las operaciones más importante que han sido mencionadas anteriormente, Nutch posee otras operaciones muy diversas que permiten realizar otras opciones. A continuación, se van a mencionar las más importantes:

- **readdb:** Permite realizar operaciones con la CrawlDb como leerla o exportarla.
- **mergedb:** Permite mezclar CrawlDbs.
- **readlinkdb:** Permite realizar operaciones con la LinkDb como leerla o exportarla.
- **inverlinks:** Crea una LinkDb o inserta nuevos links a partir de *segments* parseados.
- **mergelinkdb:** Permite mezclar LinkDbs.
- **dedup:** Elimina entradas duplicadas en la CrawlDb.
- **webgraph:** Crea un grafo web a partir de los segmentos existentes.

# Anexo B – Manual de GeoCrawler

## *Compilación*

GeoCrawler utiliza Apache Ant para la compilación. Para compilar GeoCrawler basta con estar en la carpeta raíz del *crawler* y ejecutar el comando `ant`.

La configuración de la compilación está ya realizada a través de los archivos build.xml que hay repartido por las diferentes subcarpetas de la estructura de ficheros del *crawler*.

## *Ejecución*

Como se ha comentado anteriormente en esta memoria, Nutch base proporciona dos ficheros scripts para Shell. Estos son:

- `Nutch.sh` – Este fichero permite ejecutar cualquier sección de Nutch.
- `Crawl.sh` – Permite la ejecución entera del flujo completo de Nutch. Se deben introducir una serie de parámetros para la configuración básica de la ejecución.

Para ejecutar GeoCrawler se debe ejecutar el siguiente comando:

```
runtime/local/bin/crawl -i -D hadoop.log.dir=logs -D
hadoop.log.file=hadoop.log urls/seeds.txt GeoCrawler 30
```

`-i` → Flag para activar indexación
`-D hadoop.log.dir=logs` → Se establece el directorio donde se guardarán los logs.
`-D hadoop.log.file=hadoop.log` → Se establece el nombre de los logs.
`urls/seeds.txt` → Fichero que contiene las URLs semilla.
`GeoCrawler` → Nombre del crawler.
`30` → Número de iteraciones deseadas.

Debe existir en la carpeta raíz una carpeta llamada `out` para que se depositen los elementos que son extraídos por GeoCrawler.

# Anexo C – Tipos de recursos OGC

## WPS

El estándar WPS [13] (Web Processing Service) proporciona reglas para estandarizar entradas y salidas (peticiones y respuestas) para invocar servicios de procesamiento geoespacial (por ejemplo, polígonos superpuestos) como un servicio web.

## WCTS

WCTS [14] (Web Coordinate Transformation Server) proporciona transformaciones de datos entre sistemas de referencia para las coordenadas (CRS) y es útil cuando se integran datos de distintas fuentes. La función principal de éste servicio es transformar datos vectoriales o mallas de un CRS a otro CRS.

El uso de datos almacenados en distintos CRS requiere que sean transformados a un único CRS.

## WMS

WMS [15] (Web Map Service) produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica. Este estándar internacional define un "mapa" como una representación de la información geográfica en forma de un archivo de imagen digital conveniente para la exhibición en una pantalla de ordenador.

Un mapa no consiste en los propios datos. Los mapas producidos por WMS se generan normalmente en un formato de imagen como PNG, GIF o JPEG, y opcionalmente como gráficos vectoriales en formato SVG (Scalable Vector Graphics) o WebCGM (Web Computer Graphics Metafile).

## WFS

WFS [16] (Web Feature Service) es un servicio estándar, que ofrece una interfaz de comunicación que permite interactuar con los mapas servidos por el estándar WMS, como por ejemplo, editar la imagen que nos ofrece el servicio WMS o analizar la imagen siguiendo criterios geográficos.

Para realizar estas operaciones se utiliza el lenguaje GML que deriva del XML, que es el estándar a través del que se transmiten las órdenes WFS.

WFS permite hacer consultas y recuperación de elementos geográficos. Por el contrario, WFS-T (Web Feature Service Transactional) permite además la creación, eliminación y actualización de estos elementos geográficos del mapa.

## WMTS

WMTS [17] (Web Map Tile Service) es un protocolo para ofrecer mapas teselados georreferenciados de manera prerenderizada a través de internet.

## WCS

WCS [18] (Web Coverage Service) proporciona una interfaz que permite realizar peticiones de cobertura geográfica a través de la web utilizando llamadas independientes de la plataforma.

Las coberturas son objetos o imágenes en un área geográfica mientras que la interfaz WMS o portales de mapas online como Google Maps devuelven sólo una imagen, que los usuarios no pueden editar o analizar espacialmente.

El servicio de cobertura web básico permite realizar consultas y obtener coberturas.

### CSW

CSW [19] (Catalog Service for the Web) es un estándar para la realización de catálogo de datos geoespaciales en XML en internet. Estos catálogos pueden describir datos geoespaciales (KLM), servicios geoespaciales (WMS) y otros recursos relacionados.

Define una interfaz común para el descubrimiento, búsqueda y consulta de metadatos relacionados a datos, servicios y recursos de tipo geográfico.

# Anexo D – Código de interés

## *OgcIndexingFilter.java*

```
/**
 * Extract fields from Parse metadata and add them to the final Nutch
 * document to be indexed in the database.
 *
 * @author Jorge Cancer
 */

public class OgcIndexingFilter implements IndexingFilter {

	private static final Logger LOG =
LoggerFactory.getLogger(OgcIndexingFilter.class);

	private Configuration conf;

	@Override
	public void setConf(Configuration conf) {
		this.conf = conf;
	}

	@Override
	public Configuration getConf() {
		return this.conf;
	}

	@Override
	public NutchDocument filter(NutchDocument doc, Parse parse, Text
url, CrawlDatum datum, Inlinks inlinks)
			throws IndexingException {
		ParseData dataP = parse.getData();
		Metadata meta = dataP.getParseMeta();
		boolean index = false;

		for (String key : meta.names()) {
			if(key.equals("ogc_service"))
				index = true;
			String value = meta.get(key);
			LOG.info("Adding " + url + " to NutchDocument");
			doc.add(key, value);
		}
		/* Return the document if it is an ogc service, otherwise
return null */
		return index ? doc : null;

	}

}
```

## OgcParseFilter.java

```
/**
 * Parse the content from XML in order to check if it corresponds to
 * an OGC service. In addition, makes full raw content of XML (and
 * HTML)documents available.
 *
 * @author Jorge Cancer
 */
public class OgcParseFilter implements HtmlParseFilter {

	private final static String RAW_CONTENT = "raw_content";

	private final static String OGC_SERVICE = "ogc_service";

	private final static String OGC_VERSION = "ogc_version";

	private final static String ANCHOR_CONTEXT = "anchor_context";

	private final static String ANCHOR = "anchor";

	private static final Logger LOG =
LoggerFactory.getLogger(OgcParseFilter.class);

	private int boundary;

	private Configuration conf;

	private Namespace nsc;
	private static final Map<String, String> CONTAINS_MAP = new
HashMap<>();
	static {
		CONTAINS_MAP.put("wms", "wms");
		CONTAINS_MAP.put("wms_ms", "wms");
		CONTAINS_MAP.put("csw", "csw");
		CONTAINS_MAP.put("wfs", "wfs");
		CONTAINS_MAP.put("wcs", "wcs");
		CONTAINS_MAP.put("wcts", "wcts");
		CONTAINS_MAP.put("wps", "wps");
		CONTAINS_MAP.put("wmt_ms", "wms");
	}

	@Override
	public Configuration getConf() {
		return conf;
	}

	@Override
	public void setConf(Configuration conf) {
		this.conf = conf;
		boundary = conf.getInt("ogc.outlink.anchor.context", 30);
	}

	@Override
	public ParseResult filter(Content content, ParseResult
parseResult, HTMLMetaTags metaTags, DocumentFragment doc) {

		String url = content.getUrl();

		LOG.info("Getting ogc content for " + url);

		Metadata metadata =
parseResult.get(url).getData().getParseMeta();
```

```
            ParseData parseData = parseResult.get(url).getData();

            byte[] raw = content.getContent();

            // Text with HTML tags
              String h = new String(raw);

            // Text without HTML tags
            String he = parseResult.get(url).getText();

            Outlink[] outLinks = parseData.getOutlinks();

            if (outLinks != null) {
                  // For each link read anchor and and around text with
a boundary
                  // definded in conf
                  for (Outlink outlink : outLinks) {
                        String anchor = outlink.getAnchor();
                        if (!Strings.isNullOrEmpty(anchor)) {
                              int index = he.indexOf(anchor); //
Anchor's index (Search around)
                              MapWritable metadataOutlink = new
MapWritable();
                              String context = extractContext(he,
index);
                              metadataOutlink.put(new
Text(ANCHOR_CONTEXT), new Text(context));
                              metadataOutlink.put(new Text(ANCHOR), new
Text(anchor));
                              outlink.setMetadata(metadataOutlink);
                        } else {
                              MapWritable metadataOutlink = new
MapWritable();
                              metadataOutlink.put(new
Text(ANCHOR_CONTEXT), new Text(""));
                              metadataOutlink.put(new Text(ANCHOR), new
Text(""));
                              outlink.setMetadata(metadataOutlink);
                        }
                  }
                  // Save changes in parse data
                  parseData.setOutlinks(outLinks);
            }

            // If the content is xml check if it's an ogc service
            String contentType = content.getContentType();
            if (contentType.equals("application/xml") ||
contentType.equals("text/xml")) {

                  metadata.add(RAW_CONTENT, new String(raw));

                  Document dom = XMLUtils.parseXml(new
ByteArrayInputStream(raw));

                  nsc = dom.getRootElement().getNamespace();

                  try {
                        detectOGC(dom, metadata);
```

```
} catch (JaxenException e) {
                        return new ParseStatus(ParseStatus.FAILED, "XML
Parser Jaxen error : " + e.getMessage())

      .getEmptyParseResult(content.getUrl(), getConf());
                  }
            }
            parseResult.get(url).getData().setParseMeta(metadata);
            return parseResult;
      }

      private String extractContext(String he, int index) {
            String res = "";
            try {
                  res = he.substring(index - boundary, index +
boundary);
            } catch (IndexOutOfBoundsException e) {
                  if (index - boundary < 0 && index + boundary >
he.length()) {
                        res = he.substring(0, he.length());
                  } else if (index + boundary > he.length()) {
                        // index + boundary is larger than string
length
                        res = he.substring(index - boundary,
he.length());
                  } else if (index - boundary < 0) {
                        res = he.substring(0, he.length());
                  }
            }
            return res;
      }

      private void detectOGC(Document dom, Metadata metadata) throws
JaxenException {
            JDOMXPath xp = new JDOMXPath("//*");
            xp.addNamespace(nsc.getPrefix(), nsc.getURI());
            List<?> ls = xp.selectNodes(dom);
            Element root = (Element) ls.get(0);
            String version = root.getAttributeValue("version");
            if (version != null) {
                  metadata.add(OGC_VERSION, version);
            }

            for (Object element : ls) {
                  String text = ((Element) element).getName();
                  for (Map.Entry<String, String> e :
CONTAINS_MAP.entrySet()) {
                        if (containsOGC(text, e.getKey())) {
                              metadata.add(OGC_SERVICE, e.getValue());
                              return;
                        }
                  }
            }

            if (checkAtom(ls)) {
                  metadata.add(OGC_VERSION, "1.0");
                  metadata.add(OGC_SERVICE, "atom");
            } else if (checkWMTS(ls)) {
                  metadata.add(OGC_SERVICE, "wmts");
                  //System.out.println("Contains -> " + "wmts");
```

```
            } else {
                  LOG.info("OGC service not detected");
            }
      }

      private boolean checkWMTS(List<?> ls) {
            Element root = (Element) ls.get(0);
            return containsOGC(root.getNamespace().getURI(), "WMTS");
      }

      private boolean checkAtom(List<?> ls) {
            Element root = (Element) ls.get(0);
            return
root.getNamespace().getURI().equals("http://www.w3.org/2005/Atom");
      }

      private boolean containsOGC(String f, String service) {
            return
org.apache.commons.lang3.StringUtils.containsIgnoreCase(f, service);
      }

}
```

## *SharkScoringFilter.java*

```
/**
 * This plugin implements the shark-search algorithm
 *
 * @author Jorge Cancer
 */
public class SharkScoringFilter implements ScoringFilter {

	private final static Logger LOG =
LoggerFactory.getLogger(SharkScoringFilter.class);

	private final static String ANCHOR_CONTEXT = "anchor_context";
	private static final Text TEXT_ANCHOR_CONTEXT = new
Text(ANCHOR_CONTEXT);

	private final static String ANCHOR = "anchor";
	private static final Text TEXT_ANCHOR = new Text(ANCHOR);

	private final static String TEXT = "text";

	private float delta;

	private float beta;

	private float gamma;

	private Configuration conf;
	private float scoreInjected;
	private Thesaurus th;
	private TermFreqAlt termExtractor;

	private String[] superAnchorTerms = { "services", "servicios",
"servicios web", "web services", "ogc",
			"capabilities" };

	private String[] superURLTerms = { "wms", "wmts", "wms-c",
"csw", "wfs", "atom", "wps-transformacion", "wcts",
			"wcs", "wps", "ogc" };

	public Configuration getConf() {
		return conf;
	}

	public void setConf(Configuration conf) {
		this.conf = conf;
		delta = conf.getFloat("score.geo.delta", 0.5f);
		beta = conf.getFloat("score.geo.beta", 0.5f);
		gamma = conf.getFloat("score.geo.gamma", 0.5f);
		th = new Thesaurus();
		scoreInjected = 0.25f;
		termExtractor = new TermFreqAlt();
	}
```

```
/**
       * Set an initial score for newly injected pages. Note: newly
injected pages
       * may have no inlinks, so filter implementations may wish to
set this score
       * to a non-zero value, to give newly injected pages some
initial credit.
       *
       * @param url
       *            url of the page
       * @param datum
       *            new datum. Filters will modify it in-place.
       * @throws ScoringFilterException
       */
      public void injectedScore(Text url, CrawlDatum datum) throws
ScoringFilterException {
            datum.setScore(1000.0f);
      }

      /**
       * Set an initial score for newly discovered pages. Note: newly
discovered
       * pages have at least one inlink with its score contribution,
so filter
       * implementations may choose to set initial score to zero
(unknown value),
       * and then the inlink score contribution will set the "real"
value of the
       * new page.
       *
       * @param url
       *            url of the page
       * @param datum
       *            new datum. Filters will modify it in-place.
       * @throws ScoringFilterException
       */
      public void initialScore(Text url, CrawlDatum datum) throws
ScoringFilterException {
            datum.setScore(0.0f);
      }

      /**
       * This method prepares a sort value for the purpose of sorting
and
       * selecting top N scoring pages during fetchlist generation.
       *
       * @param url
       *            url of the page
       * @param datum
       *            page's datum, should not be modified
       * @param initSort
       *            initial sort value, or a value from previous
filters in chain
       */
      public float generatorSortValue(Text url, CrawlDatum datum,
float initSort) throws ScoringFilterException {
            return datum.getScore() * initSort;
      }
```

```
	/**
	 *
	 */
	public void passScoreBeforeParsing(Text url, CrawlDatum datum,
Content content) {
		content.getMetadata().set(Nutch.SCORE_KEY, "" +
datum.getScore());
	}

	/**
	 * Copy the value from Content metadata under Fetcher.SCORE_KEY
to
	 * parseData.
	 */
	public void passScoreAfterParsing(Text url, Content content,
Parse parse) {
		parse.getData().getContentMeta().set(TEXT,
parse.getText());
		parse.getData().getContentMeta().set(Nutch.SCORE_KEY,
content.getMetadata().get(Nutch.SCORE_KEY));
	}

	/** Increase the score by a sum of inlinked scores. */
	public void updateDbScore(Text url, CrawlDatum old, CrawlDatum
datum, List<CrawlDatum> inlinked)
			throws ScoringFilterException {
		float adjust = 0.0f;
		for (CrawlDatum linked : inlinked) {
			adjust += linked.getScore();
		}
		if (old == null)
			old = datum;
		datum.setScore(old.getScore() + adjust);
	}

	/**
	 * DistributeScore
	 */
	public CrawlDatum distributeScoreToOutlinks(Text fromUrl,
ParseData parseData,
			Collection<Entry<Text, CrawlDatum>> targets,
CrawlDatum adjust, int allCount)
				throws ScoringFilterException {
		LOG.info("URL: " + fromUrl + "    Num. Dest.: " +
targets.size());
		/*
		 * Get the inherited score
		 */
		@SuppressWarnings("unused")
		float inheritedScore = scoreInjected; // Default value
		String scoreString =
parseData.getContentMeta().get(Nutch.SCORE_KEY);
		if (scoreString != null) {
			try {
				inheritedScore = Float.parseFloat(scoreString);
			} catch (Exception e) {
				LOG.error("Error: ", e);
			}
		}
```

```
            /*
             * Computing the inherited score of child node
             */
            ogcAlgorithm(targets);
            //sharkSearch(targets, parseData, inheritedScore);
            //sharkSearchThesaurus(targets, parseData, inheritedScore);
            return null;
      }

      private void sharkSearch(Collection<Entry<Text, CrawlDatum>>
targets, ParseData parseData, float inheritedScore) {
            String content = parseData.getContentMeta().get(TEXT);
            float dataRel = relevanceOGC(0.0f, content);
            float scoreChild = delta * dataRel > 0 ? dataRel :
inheritedScore;
            for (Entry<Text, CrawlDatum> entry : targets) {
                  float potentialScore =
computePotentialScore(scoreChild, entry.getValue().getMetaData());
                  entry.getValue().setScore(potentialScore);
            }

      }

      private void sharkSearchThesaurus(Collection<Entry<Text,
CrawlDatum>> targets, ParseData parseData, float inheritedScore) {
            String content = parseData.getContentMeta().get(TEXT);
            float dataRel = relevance(content);
            float scoreChild = delta * dataRel > 0 ? dataRel :
inheritedScore;
            for (Entry<Text, CrawlDatum> entry : targets) {
                  float potentialScore =
computePotentialScoreThesaurus(scoreChild,
entry.getValue().getMetaData());
                  entry.getValue().setScore(potentialScore);
            }

      }

      private void ogcAlgorithm(Collection<Entry<Text, CrawlDatum>>
targets) {
            float dataRel = 0.0f;
            for (Entry<Text, CrawlDatum> entry : targets) {
                  float potentialScore = dataRel;
                  Map metadata = entry.getValue().getMetaData();
                  String anchor = metadata.get(TEXT_ANCHOR).toString();
                  potentialScore = relevanceOGC(potentialScore,
anchor);
                  potentialScore = boost(potentialScore,
entry.getKey());
                  entry.getValue().setScore(potentialScore);
            }
      }

      private float boost(float potentialScore, Text fromUrl) {
            int boost = 0;
            for (String superTerm : superURLTerms) {
                  String url = fromUrl.toString().toLowerCase();
                  if (url.contains(superTerm)) {
                        boost++;
                  }
```

```
                if (url.contains("capabilities")) {
                        boost += 10;
                }
            }
            return potentialScore + (10000 * boost);
      }

      private float computePotentialScore(float scoreChild, Map
metadata) {
            String anchor = metadata.get(TEXT_ANCHOR).toString();
            String context =
metadata.get(TEXT_ANCHOR_CONTEXT).toString();
            float anchorScore = relevanceOGC(scoreChild, anchor);
            float anchorContextScore = anchorScore > 0 ? 1f :
relevanceOGC(scoreChild, context);
            float neighbourhoodScore = (beta * anchorScore) + ((1 -
beta) * anchorContextScore);
            // Saving the score
            return (gamma * scoreChild) + ((1 - gamma) *
neighbourhoodScore);
      }

      private float computePotentialScoreThesaurus(float scoreChild,
Map metadata) {
            String anchor = metadata.get(TEXT_ANCHOR).toString();
            String context =
metadata.get(TEXT_ANCHOR_CONTEXT).toString();
            float anchorScore = relevanceText(anchor);
            float anchorContextScore = anchorScore > 0 ? 1f :
relevanceText(context);
            float neighbourhoodScore = (beta * anchorScore) + ((1 -
beta) * anchorContextScore);
            // Saving the score
            return (gamma * scoreChild) + ((1 - gamma) *
neighbourhoodScore);
      }

      private float relevanceText(String text) {
            String[] words = text.split(" ");
            float rel = 0.0f;
            for (String word : words) {
                word = filter(word);
                if (word.length() >= 2 && word.length() < 100) {
                    int pow = th.execQuery(word);
                    rel += pow;
                }
            }
            return rel;
      }

      private float relevanceOGC(float potentialScore, String content)
{
            int boost = 0;
            for (String superTerm : superAnchorTerms) {
                String contentCase = content.toLowerCase();
                if (contentCase.contains(superTerm)) {
                    boost++;
                }
            }
            return potentialScore + (1000 * boost);
      }
```

```
	private String filter(String word) {
		word = termExtractor.deleteSymbols(word);
		word = termExtractor.filterStopWords(word);
		return word;
	}


	private float relevance(String content) {
		LinkedHashMap<String, Float> terms =
termExtractor.extractTerms(content);
		float rel = 0.0f;
		if (terms != null) {
			int indx = 0;
			for (Entry<String, Float> entry : terms.entrySet()) {
				// TODO N top relevant terms
				if (indx >= 10) {
					break;
				}
				if (entry.getKey().length() < 100) {
					int pow = th.execQuery(entry.getKey());
					rel += entry.getValue() * pow;
				}
				indx++;
				// System.out.println(indx);
			}
		}
		return rel;
	}

	@Override
	public float indexerScore(Text url, NutchDocument doc,
CrawlDatum dbDatum, CrawlDatum fetchDatum, Parse parse,
			Inlinks inlinks, float initScore) throws
ScoringFilterException {
		return dbDatum.getScore();
	}

}
```

## OgcIndexWriter.java

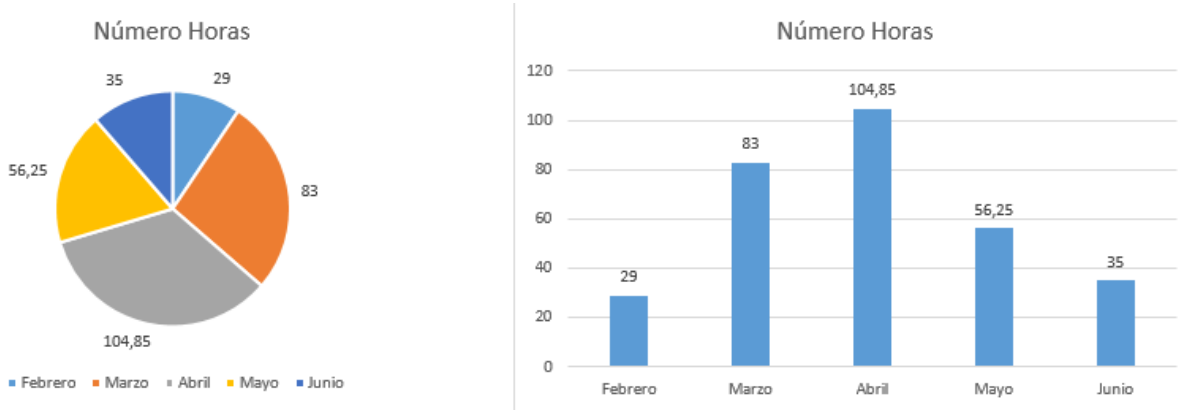
```
/**
 * OgcIndexWriter. This pluggable indexer writes the OGC documents
 * in the file system.
 */
public class OgcIndexWriter implements IndexWriter {
      public static final Logger LOG =
LoggerFactory.getLogger(OgcIndexWriter.class);
      private final static String RAW_CONTENT = "raw_content";
      private Configuration config;
      private String path = "out/";

      public void open(JobConf job, String name) throws IOException {
      }

      @Override
      public void delete(String key) throws IOException {
      }

      @Override
      public void update(NutchDocument doc) throws IOException {
      }

      @Override
      public void write(NutchDocument doc) throws IOException {
            System.out.println("Writing " +
doc.getField("url").toString());
            NutchField raw = doc.getField(RAW_CONTENT);

            try {

                  String content =
doc.getField(RAW_CONTENT).toString();
                  content = content.substring(1, content.length()-1);

                  String url = doc.getField("url").toString();
                  url = formatUrl(url);

                  File file = new File(path + url);

                  /* If file doesn't exists, then create it */
                  if (!file.exists()) {
                        file.createNewFile();
                  }

                  FileWriter fw = new
FileWriter(file.getAbsoluteFile());
                  BufferedWriter bw = new BufferedWriter(fw);
                  bw.write(content);
                  bw.close();

                  System.out.println("Done");

            } catch (IOException e) {
                  e.printStackTrace();
            }
      }
```

```
private String formatUrl(String url) {
	String out = new String(url);
	out = out.substring(8,out.length()-1);
	out = out.replace("/", "_");
	return out;
}

public void close() throws IOException {

}

@Override
public void commit() throws IOException {

}

@Override
public Configuration getConf() {
	return config;
}

@Override
public void setConf(Configuration conf) {
	config = conf;
	path = conf.get("ogc.path");
}

public String describe() {
	StringBuffer sb = new StringBuffer("OgcIndexWriter");
	return sb.toString();
}
```

# Anexo E – Cronograma

Hasta el momento en el que se está escribiendo esto se han utilizado 306,1 horas para completar este proyecto. El trabajo comenzó en febrero. Durante ese primer mes el trabajo se centró en documentarse acerca de los trabajos existentes y sobre Nutch. La mayor cantidad de trabajo se realizó en abril. En la siguiente figura se puede revisar la distribución de horas por mes:

**Figura 20: Gráfica mes a mes**




Como se puede ver en la figura inferior, si se compara la gráfica de tiempo por cada mes con respecto a la gráfica de *commits* en GitHub se puede observar una correspondencia lógica.

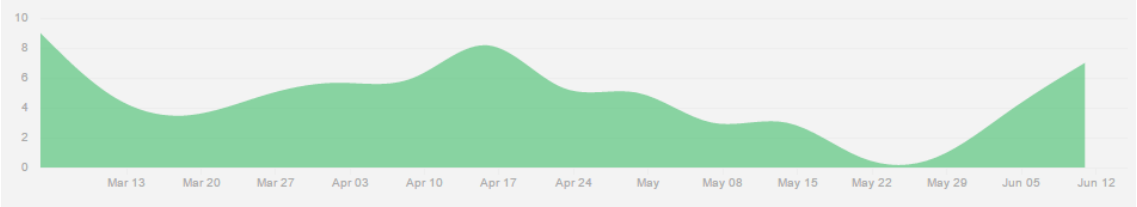



**Figura 21: Gráfica commits GitHub**

A continuación, se pueden ver el desglose por mes de los esfuerzos:

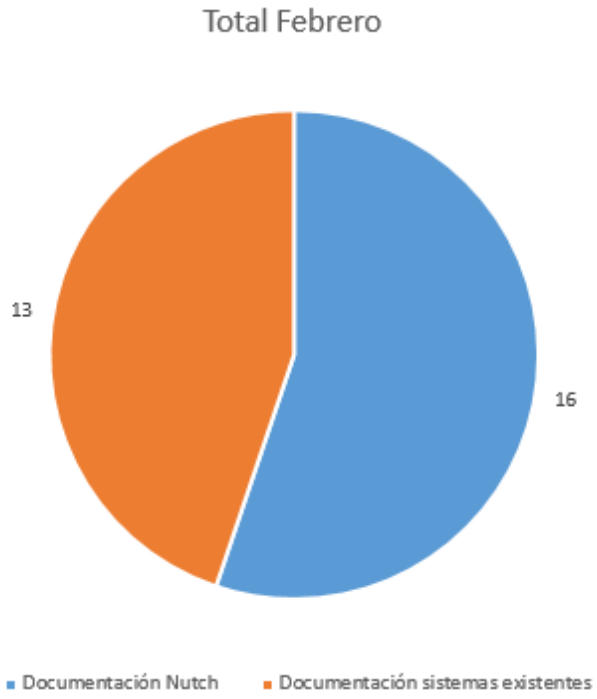



Figura 22: Desglose febrero

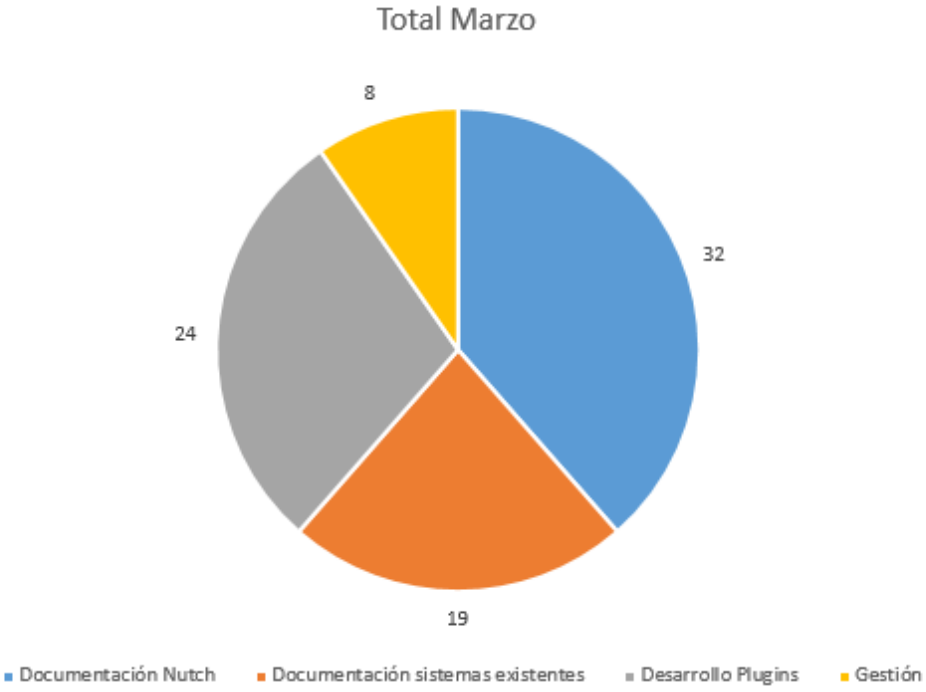



Figura 23: Desglose marzo

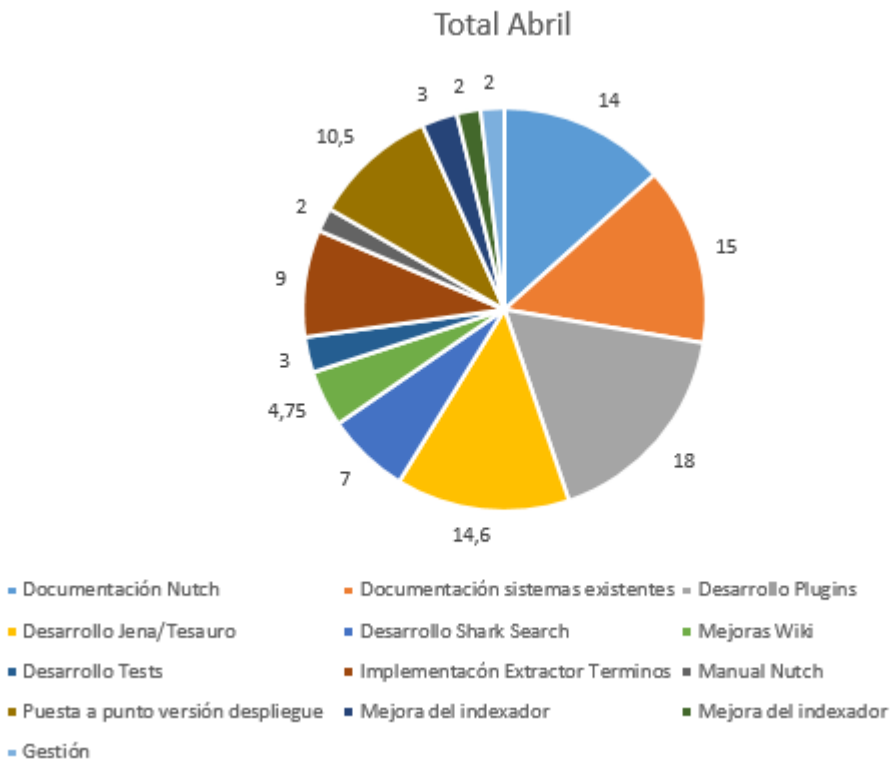



Figura 24: Desglose abril

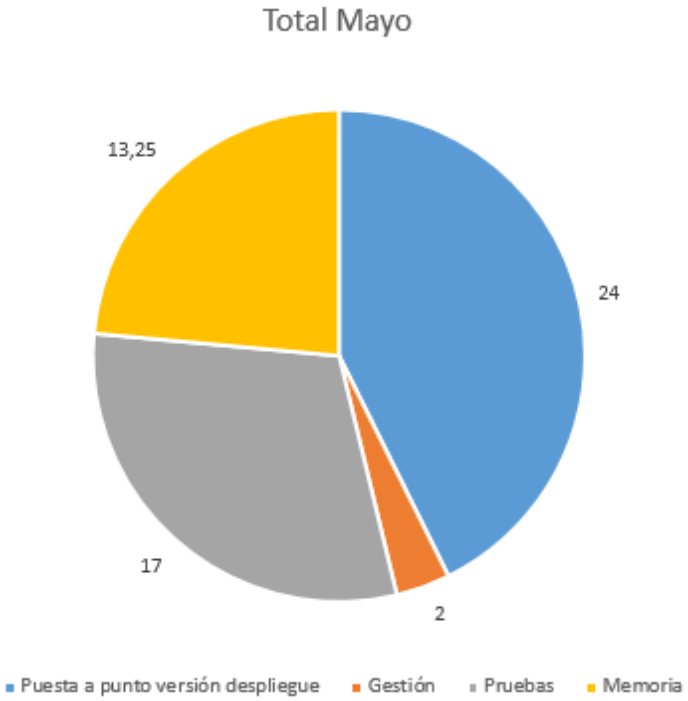



Figura 25: Desglose mayo

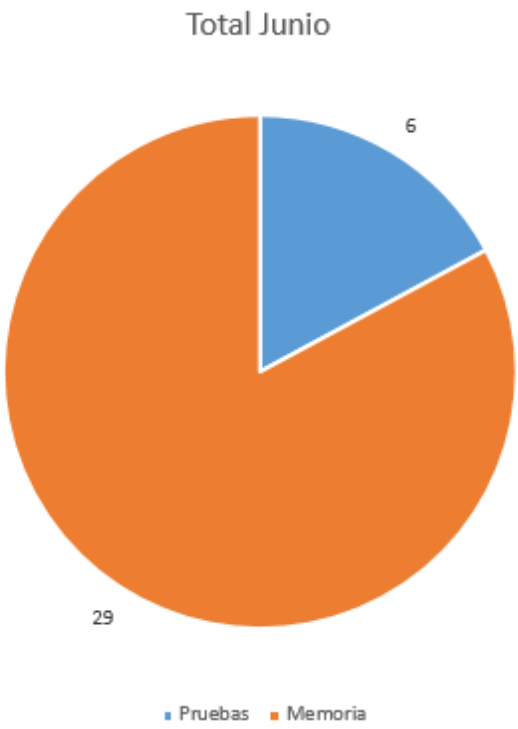



Figura 26: Desglose junio

# Anexo F – DOM

Cuando se creó el lenguaje XML también surgió la necesidad de procesar y manipular el contenido de los archivos XML mediante los lenguajes de programación tradicionales. XML es un lenguaje sencillo de escribir, pero complejo para procesar y manipular de forma eficiente. Por este motivo, surgieron algunas técnicas entre las que se encuentra DOM [9]. DOM es un conjunto de utilidades específicamente diseñadas para manipular documentos XML. Por extensión, DOM también se puede utilizar para manipular documentos XHTML y HTML. Se puede utilizar DOM para manipular las páginas XHTML de forma rápida y eficiente.

Antes de poder utilizar sus funciones, DOM transforma internamente el archivo XML original en una estructura más fácil de manejar formada por una jerarquía de nodos. De esta forma, DOM transforma el código XML en una serie de nodos interconectados en forma de árbol. El árbol generado no sólo representa los contenidos del archivo original, también representa sus relaciones.

Aunque en ocasiones DOM se asocia con la programación web y con JavaScript, la API de DOM es independiente de cualquier lenguaje de programación. De hecho, DOM está disponible en la mayoría de lenguajes de programación comúnmente empleados. En la Ilustración 1 de puede ver un esquema de una jerarquía generada por un documento XHTML al ser transformado en DOM [20].

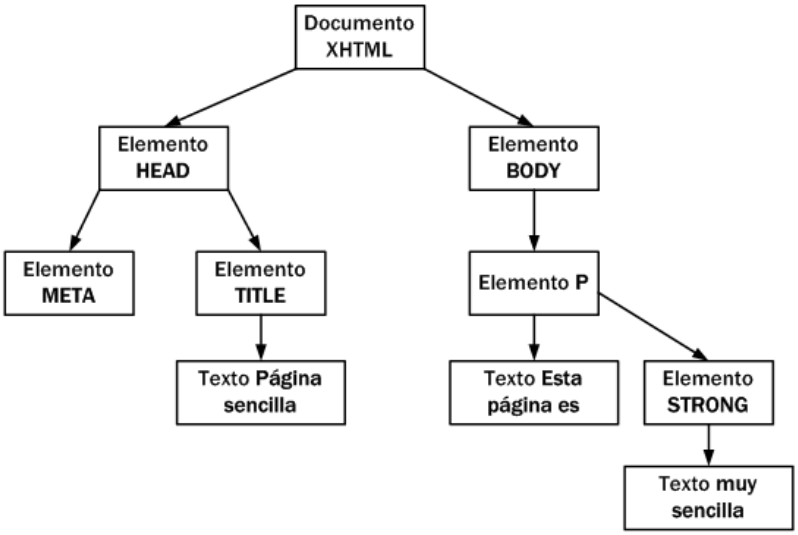



Ilustración 1

Para el desarrollo de este trabajo este tipo de documentos son los más importantes. Nutch provee un *parser* para este tipo de contenidos. Viene implementado en *Tika Parser*. *Tika Parser* es otra herramienta de *Apache* que permite realizar el proceso de *parse* para muchos tipos de documentos entre ellos XML, HTML, PDF… .