

# Sistema de recopilación de objetos digitales de texto en dominios restringidos

Tesina de Grado

Licenciatura en Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Autor: Santiago Beltramone  
Directoras: Dra. Ana Casali  
Dra. Claudia Deco

24 de septiembre de 2014

## Resumen

Para incrementar el contenido y consecuente uso de Repositorios Institucionales Educativos de Producción Nacional, además de políticas y estrategias de difusión adecuadas, es necesario desarrollar herramientas informáticas para la automatización de la recopilación de objetos digitales educativos que ya están publicados en sitios web institucionales y que pueden ser cargados al repositorio. Esta tarea de recopilación es tediosa y es realizada en forma manual actualmente. En este trabajo se presenta una arquitectura para automatizar esta tarea de recopilación de documentos de texto dentro de un dominio restringido con el objetivo de recuperar documentos plausibles de ser cargados en un repositorio, junto a información relevante como idioma, título, autores y sus correspondientes datos de contacto: emails y filiaciones. En la extracción de información, no solamente se propone procesar el texto plano de los documentos, si no además el código HTML de páginas webs enlazadas en el mismo sitio web donde los documentos son recuperados. Se desarrolló un prototipo de este sistema, donde diferentes dominios de nuestra facultad fueron utilizados como casos de estudio.

# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Motivación . . . . .	3
1.2. Objetivo . . . . .	3
1.3. Trabajos relacionados . . . . .	4
<b>2. Conceptos Básicos y Análisis de Herramientas</b>	<b>9</b>
2.1. Web Crawling . . . . .	9
2.1.1. Políticas y caracterización de Web Crawlers . . . . .	10
2.1.2. Crawlers Open Source . . . . .	11
2.1.3. Consulta a motores de búsqueda externos . . . . .	12
2.2. Extracción, Recuperación y Recopilación de Información . . . . .	13
2.3. Herramientas de extracción de información sobre documentos de texto . . . . .	15
2.4. Técnicas de matching de nombres propios . . . . .	17
2.5. Base de Datos Orientadas a Gráfos . . . . .	19
<b>3. Arquitectura del Sistema</b>	<b>22</b>
3.1. Propuesta de Arquitectura y descripción de componentes . . . . .	23
3.2. Secuencia de Procesamiento e Interacción Entre Componentes	26
<b>4. Desarrollo del prototipo</b>	<b>28</b>
4.1. Inicio y Configuración del Sistema . . . . .	28
4.2. Controlador . . . . .	31
4.3. Interfaz de usuario . . . . .	31
4.4. Base de Datos Orientada a Grafos Embebida y Entidades del Modelo . . . . .	32
4.5. Crawler . . . . .	35
4.6. Wrappers de Extracción de Datos . . . . .	35
4.7. Post-Procesamiento y Entidades Persistidas Durante el Mismo	37
<b>5. Experimentación</b>	<b>40</b>
<b>6. Conclusiones y trabajo a futuro</b>	<b>55</b>
<b>A. Apéndice A: Principales clases Java del sistema</b>	<b>58</b>
A.1. Controlador . . . . .	58
A.2. BDOG Embebida . . . . .	61
A.3. Crawler . . . . .	68
A.4. Post-Procesador . . . . .	75
A.5. Configuración caso de prueba 1 . . . . .	87

A.6. Configuración caso de prueba 2 . . . . . 89

# 1. Introducción

## 1.1. Motivación

El desarrollo de Repositorios Institucionales (RI) de Acceso Abierto (AA) dependientes de las universidades públicas de Argentina, es una prioridad en el marco de las políticas del Ministerio de Ciencia, Tecnología e Innovación y el Consejo Interuniversitario Nacional [1]. Están en curso seis proyectos de investigación y desarrollo sobre RI de AA, donde participan muchos docentes e investigadores de la universidad. En particular, el proyecto “Hacia el desarrollo y utilización de Repositorios de Acceso Abierto para Objetos Digitales Educativos en el contexto de las universidades públicas de la región centro-este de Argentina”<sup>1</sup> se diseñó a partir de los marcos organizacionales de la Universidad de Rosario (UNR), Universidad Nacional del Litoral (UNL) y Universidad Tecnológica Nacional (UTN) regional Santa Fe. Su objetivo se focaliza en diseñar y transferir un primer modelo teórico, metodológico y tecnológico de carácter experimental para Repositorios Institucionales de Acceso Abierto de Objetos Digitales Educativos (ODE). El trabajo de esta Tesina se vincula a una de las líneas de trabajo que se originan a partir de este proyecto. A raíz del diseño de Repositorios Institucionales surge la necesidad de poblarlo con ODEs que representen la producción de la universidad. Además de políticas y estrategias de difusión adecuadas, es de interés automatizar la tarea de recopilación de todos los documentos que ya están publicados en las distintos dominios de la UNR: Facultades, Departamentos, páginas de los docentes, etc., que pueden ser cargados al repositorio de nuestra universidad. Actualmente, esta tarea es realizada manualmente por personal encargado del repositorio.

Además, resultan de suma importancia los datos de contacto del autor de un ODE relevante al repositorio, puesto que es mandatorio que el autor autorice la publicación del documento de su autoría otorgando la licencia correspondiente.

## 1.2. Objetivo

El objetivo de este trabajo es brindar una arquitectura de sistema para la recopilación de documentos de texto dentro de dominios restringidos (orientados a universidades públicas nacionales). Siguiendo esta arquitectura, se desarrolla un prototipo en el dominio de la problemática mencionada

---

<sup>1</sup>PICTO-CIN II Bicentenario. Financiado por la Agencia Nacional de Promoción Científica y Tecnológica y el CIN. Contrapartes: CONICET-UNR-UNL-UTN

en la sección anterior. Dicho prototipo ayuda en la tarea de recopilación de ODEs de texto dentro de un dominio restringido, junto a la automatización de extracción de datos relevantes como título; categoría; autores, idioma, derechos de autor, palabras claves y correspondientes datos de contacto para poder solicitar la publicación del documento. Específicamente, los datos de contacto que se buscan son el email y la filiación del autor del ODE.

Una característica presente en el dominio de aplicación descrito, es que muchas veces, los datos requeridos (autor, mail, filiación) no se encuentran en el ODE, aunque si pueden hallarse en diferentes páginas del mismo sitio web. La arquitectura propuesta, se encarga de explotar esta característica, para mejorar la automatización de extracción de información. Por lo tanto, en el prototipo, algunos datos o metadatos a extraer se buscan en el documento de texto. Otros, además, se buscan en las páginas web vecinas a la página que referencia el documento, dentro de la estructura del sitio web que está siendo examinado.

El sistema recibe como entrada una lista de URLs de sitios web donde se desea realizar la recopilación de ODEs. La salida final del sistema resume los documentos encontrados, junto con la información extraída. Para llevar a cabo sus tareas, el sistema utiliza herramientas como Web Crawlers, Extractores de Información de texto, Base de Datos orientadas a gráficos entre otras. Todos estos conceptos serán ampliados en la sección 2.

### **1.3. Trabajos relacionados**

Algunos sistemas de recopilación de información han sido propuestos, en particular, se destacan los siguientes.

#### **AGATHE [2]**

Es un sistema de arquitectura genérica multiagente para la recopilación de información sobre dominios web restringidos. Implementa un enfoque cooperativo de recopilación de información basado en agentes de software que interactúan y explotan ontologías relacionados a los diferentes dominios web sobre los que operan. Tener en cuenta el contexto le permite un mejor tratamiento de la información contenida en las páginas web recolectadas. Por ejemplo, como tarea inicial las páginas web pueden ser clasificadas de acuerdo con la diferentes clases específicas para el dominio en cuestión, con lo cuál la información relevante puede ser extraída de páginas que pertenecen a la misma clase con mayor precisión.

La arquitectura general de AGATHE está compuesta por tres subsistemas principales que interactúan entre sí:

- Subsistema de búsqueda: es el encargado de realizar las consultas a motores de búsqueda externos de la Web (como Google) a fin de obtener las páginas web para procesar por los otros subsistemas.
- Subsistema de extracción de información (EI): se compone de múltiples “clusters” de extracción (CE), cada una de ellos especializado en el procesamiento de páginas web de un dominio específico (de diferentes contextos, por ejemplo, en el ámbito de la investigación académica o del turismo).
- Subsistema de oficina: se encarga del almacenamiento de la información extraída por el subsistema anterior y proporciona una interfaz de consulta para los usuarios finales.

En su primer versión los agentes encargados de la extracción de información de AGATHE utilizaban reglas simbólicas escritas en forma ad hoc en la plataforma Jess, para los diferentes dominios de aplicación. Por lo cuál, para cada nuevo contexto en el que el sistema de EI requería ser aplicado, nuevas reglas debían ser codificadas. Debido a esto, en las nuevas versiones de AGATHE el subsistema de EI fue reemplazado por una combinación entre reglas simbólicas y técnicas de aprendizaje automatizado que logran adaptarse a diferentes contextos. Estas técnicas fueron desarrolladas en tres pasos de procesamiento: (1) reconocer semánticamente la información relevante, (2) extraer la información y (3) persistir toda la información en forma estructurada para futuros análisis, en orden de poder enriquecer la EI del dominio siendo aplicado.

### **CROSSMARC [3]**

Es un proyecto europeo de sistema multi-dominio y multilingüe basado en agentes para la extracción de información en páginas web. Utiliza técnicas de análisis lingüístico y al igual que AGATHE, algoritmos de aprendizaje automatizado con enfoques adaptativos a diferentes contextos.

La motivación de este trabajo fue la de mejorar los primeros sistemas de EI en sitios de venta de productos online, que fueron propuestos para automatizar la tarea de recolección y extracción de información para la centralización, resumen y comparación de diferentes productos en un solo portal web. Los mismos tenían un desempeño pobre al tratar de reutilizarse en diferentes dominios y/o páginas de diferentes idiomas, puesto que se basaban principalmente en la EI por reconocimiento de etiquetas o estructuras del HTML específicas.

CROSSMARC fue aplicado en dos dominios diferentes: bienes de computación y ofertas de trabajos informáticos: el primero se caracteriza por descripciones breves y semi-estructuradas, ricas de términos técnicos y acrónimos; mientras que el segundo, contiene descripciones de dominios más largas y enriquecidas.

La arquitectura general de CROSSMARC puede ser concebida como un conjunto de agentes comunicados a través de lenguaje XML dedicado utilizando ontologías. Los principales roles en la arquitectura están relacionados con tres tareas principales:

- Proceso de consultas y configuración del sistema a partir de la interacción con el usuario.
- Extracción de información de la Web: varios pasos de procesamiento se coordinan para encontrar, recuperar, analizar y extraer información de Internet.
- Almacenamiento de la información extraída en una base de datos, con el fin de alimentar el sistema con los datos que más tarde se muestran al usuario.

Los agentes de extracción CROSSMARC se pueden dividir en dos grandes categorías, dependiendo de sus tareas específicas:

- Los agentes de Recuperación de Información (IR), que identifican los sitios web en el dominio correspondiente (crawling focalizado) y recopilación de páginas web dentro de estos sitios que son susceptibles de contener la información deseada.
- Los agentes de Extracción de Información: se utiliza uno para cada idioma, estos procesan las páginas web recuperadas. Existen funciones específicas para cada paso del proceso de extracción: a) el Reconocimiento y Clasificación de entidades relativas al dominio de interés dentro de las páginas web, b) la identificación del número de productos y su distribución en las páginas web, c) Extracción de Datos, que es la extracción de las características de los productos. Toda la información recopilada es volcada en un XML de acuerdo a diferentes esquemas. Estos esquemas juegan un papel fundamental tanto en el apoyo a la interpretación de los resultados, como en la comprobación de la coherencia de los resultados.

Cada agente contribuye a una ontología compartida que coordina el análisis a lo largo de todas las fases. La bases de conocimiento correspondientes

a los dos dominios en donde el sistema fue aplicado, fueron desarrolladas y mantenidas como aplicaciones independientes basadas en Protégé2000 API<sup>2</sup>.

### SeerSuite [4]

SeerSuite es un framework para bibliotecas digitales de literatura científica en constante evolución, que cuenta con un motor de búsqueda construido a través del crawling de sitios científicos y académicos de la Web. Adicionalmente incorpora la indexación completa de texto y la indexación de citas en forma autónoma, vinculando las referencias de diferentes artículos facilitando la navegación a través de los mismos. SeerSuite permite el acceso a su repositorio de documentos científicos, con sus correspondientes citas y metadatos automáticamente extraídos y persistidos por el mismo framework. CiteSeerX es una instancia de SeerSuite, con cerca de medio de millón de artículos científicos disponibles en forma online, en tópicos de ciencias de la computación y áreas relacionadas como matemática, estadística y otras. Estos documentos son obtenidos a través del crawling focalizado de sitios web científicos/académicos donde están libremente accesibles.

CiteSeerX está basado en su prototipo anterior CiteSeer, considerado uno de los predecesores de herramientas como Google Scholar<sup>3</sup> y Microsoft Academic Research<sup>4</sup> quienes cuentan con la ventaja de poder enriquecer sus bases de datos procesando sitios web de editoriales con restricciones de acceso aranceladas. CiteSeer fue un prototipo de investigación y, como tal, sufría de grandes limitaciones. SeerSuite fue diseñado para reemplazar CiteSeer, y proveer la mayor parte de su funcionalidad en forma extensible, mejorando aspectos de integridad, robustez y escalabilidad. Esto lo logra mediante la adopción de multi-arquitecturas con orientación a servicios y articulación flexible de módulos altamente desacoplados.

El procesamiento de CiteSeer se puede resumir en cinco pasos: (1) los documentos son recopilados de la web utilizando los crawlers focalizados, (2) los mismos se convierten primero de PDF o Formato PostScript a texto plano con herramientas de aplicación como PDFBox o GhostScript para PostScript, (3) un filtro de relevancia basado en expresiones regulares escritas en forma ad hoc es aplicado para evitar el procesamiento de documentos no académicos, (4) algoritmos de aprendizaje automatizado son utilizados para parsear el encabezado del documento y el resto de las secciones<sup>5</sup> y (5) Los metadatos

---

<sup>2</sup><http://protege.stanford.edu/>

<sup>3</sup><http://scholar.google.com.ar/>

<sup>4</sup><http://academic.research.microsoft.com/>

<sup>5</sup>Una de las herramientas que se utiliza es ParsCit que también se aplica en el presente trabajo y será detallada en la sección 2.3

extraídos son persistidos junto con el documento y sus citas, cuyas referencias a otros artículos recalculadas y actualizadas.

SeerSuite se encuentra disponible bajo licencia Creative Commons, y la mayor parte de sus módulos hacen uso intensivo de herramientas de software libre. Entre estas se destacan Heretrix para el web crawling, Apache Solr para la indexación de texto de páginas web y documentos, Apache Tomcat como servidor de aplicaciones Web, MySQL como motor base de datos relacional y Greenstone Library para la estructuración del repositorio digital de documentos.

Los sistemas recién mencionados son muy buenas referencias de arquitecturas para sistemas de recopilación de información y fueron considerados en el desarrollo de esta propuesta, aunque diferentes razones obligan al diseño de una arquitectura particular para el presente trabajo.

A diferencia de AGATHE y CROSSMARK, en este trabajo las bases de conocimientos para diferentes dominios de aplicación no son requeridas y la extracción de información es realizada en base a objetivos específicos sobre los documentos y páginas web recopiladas, por lo cuál los enfoques adaptativos tampoco son utilizados.

Con respecto a SeerSuite, se trata de un proyecto de gran alcance que excede los requerimientos funcionales previamente mencionados y a la vez introduce limitaciones sobre los formatos de documentos a considerar que hacen inviable su aplicación para el fin necesario en este trabajo. Esto se debe a que solo trabaja con documentos de formato científico que comparten cierto tipo de estructura, filtrando aquellos que no satisfacen determinados estándares. Por lo tanto, gran variedad de tipos de documentos académicos que son de interés en esta tesina, como ser diapositivas, prácticas, apuntes teóricos, resúmenes de clases, no son alcanzados por el tratamiento que SeerSuite realiza con los documentos y si son tenidos en cuenta en la presente propuesta. Otra de las ideas que se incluyen en este trabajo y que no contempla SeerSuite es la extracción de información en páginas web vecinas a la URL que posee el enlace del ODE, lo que muchas veces logra mejorar la cobertura de metadatos en el proceso de extracción de información del mismo.

## 2. Conceptos Básicos y Análisis de Herramientas

### 2.1. Web Crawling

Un Web Crawler<sup>6</sup> (o Araña Web) [5] es un programa que inspecciona las páginas del World Wide Web de forma metódica y automatizada. Uno de sus usos más frecuentes consiste en crear una copia de todas las páginas web visitadas para su procesamiento posterior por un motor de búsqueda que indexa las páginas proporcionando un sistema de búsquedas rápido. Los web crawlers comienzan visitando una lista de URLs, identifican los hiperenlaces en dichas páginas y los añaden a la lista de URLs a visitar de manera recurrente de acuerdo a determinado conjunto de reglas. El cómputo usual de un crawler es a partir de un grupo de direcciones iniciales (URLs) cuyos recursos apuntados son descargados y analizados en busca de enlaces a nuevos recursos, por lo general páginas html, repitiendo este proceso hasta que las condiciones finales sean alcanzadas. Estas condiciones varían de acuerdo a la política de crawling deseada.

Los web crawlers son casi tan viejos como la Web misma. El primero, Wanderer, fue escrito por Matthew Gray en 1993, coincidiendo con la primera versión de NCSA Mosaic [7]. El web crawling ha sido y sigue siendo un gran tópico de investigación presentado en numerosas conferencias sobre la Web. Desde los inicios [8], la web se ha incrementado en 2 o 3 órdenes de magnitud, por lo cual numerosos problemas de escalabilidad que no estaban presentes en el comienzo, fueron apareciendo con el tiempo y dando al área de estudio nuevas problemáticas a resolver.

Hoy en día, existen muchos motores de búsqueda Web, algunos muy famosos como Google o Yahoo, los cuales usan web crawlers que necesitan recorrer porciones significativas de la Web. Si bien los detalles de arquitectura e implementación de estos crawlers modernos se encuentran ocultos por razones comerciales, si es posible conocer las primeras versiones de sus diseños. El motor de Google consiste con un sistema distribuido de cómputo en múltiples procesadores [9] y contaba en un principio (1998) con varias componentes funcionales operando en diferentes procesos. Se detalla brevemente estos componentes, ya que clarifican la organización del cómputo junto con la división de tareas que es requerida en todo web crawler.

En sus inicios el motor de Google contaba con un servidor para leer URLs desde un archivo y distribuirlas a múltiples procesos de crawling, cada uno de los cuales corría en diferentes máquinas y utilizaba operaciones asíncronas

---

<sup>6</sup>también conocidos como robots, spiders, worms, walkers

de entrada/salida para recuperar datos servidores web externos en paralelo. Los crawlers transmitían las páginas web recuperadas a un único proceso de almacenamiento que las comprimía y las persistía en disco. Luego las páginas eran leídas por el proceso de indexación, que se encargaba de extraer del HTML nuevos enlaces y guardarlos en otro archivo. Un proceso de resolución de URLs leía el archivo y convertía las direcciones relativas a absolutas para la realimentación del servidor que procesaba las URL.

### 2.1.1. Políticas y caracterización de Web Crawlers

El comportamiento de un web crawler resulta de la combinación de diferentes políticas [5]:

- Política de selección: sobre la decisión de qué páginas descargar a medida de que las URL son localizadas.
- Política de revisita: sobre la decisión de cuándo visitar una URL para verificación de cambios.
- Política de diplomacia: sobre los intervalos de tiempo entre solicitudes a un mismo servidor para evitar su sobrecarga
- Política de paralelización: sobre la coordinación de procesos de crawling usualmente presentes debido a la naturaleza extensa de procesamiento del problema inherente.

Entre las políticas enumeradas, las variantes con respecto a la política de selección, son las que originan las diferentes caracterizaciones de crawlers comunes, entre las cuáles nos interesa destacar:

- **Web Crawlers en Dominios Restringidos:** aquellos que deciden inspeccionar las URLs que pertenecen al mismo dominio dentro del conjunto de URLs semillas<sup>7</sup> para completar la tarea de recopilación de los recursos provistos en determinado sitio web.
- **Web Crawlers Restringidos en Enlaces:** un crawler que sólo busca determinado tipo de enlaces, por ejemplo páginas HTML, evitando cualquier otro tipo MIME<sup>8</sup> en la descarga de recursos asociados a una

---

<sup>7</sup>Se llaman semillas a las primeras URLs que recibe un Web Crawler como entrada, que forman parte de la primer iteración de recuperación de páginas Web

<sup>8</sup>Multipurpose Internet Mail Extensions o MIME son una serie de convenciones o especificaciones dirigidas al intercambio a través de Internet de todo tipo de archivos de forma transparente para el usuario - [http://es.wikipedia.org/wiki/Multipurpose\\_Internet\\_Mail\\_Extensions](http://es.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions)

URL. Muchas veces esta tarea puede llevarse a cabo simplemente verificando la extensión presente en la URL. Como resultado, durante el proceso numerosos enlaces son intencionalmente ignorados.

- **Web Crawlers Focalizados:** La importancia de recuperación de una página determinada, puede ser expresada como función de la similitud de la misma con las aquellas que son provistas como semillas del proceso. Se llaman crawlers focalizados a las que seleccionan los recursos a descargar en base a la similitud con las semillas.
- **Web Crawlers Académicos Focalizados:** Estos consisten en un caso particular de los anteriores, su objetivo es recopilar artículos académicos de acceso libre. Ejemplo de estos crawlers son CiteSeerX [4], Google Scholar<sup>9</sup> y Microsoft Academic Search<sup>10</sup>. Básicamente, al trabajo del crawler focalizado se agrega la tarea de selección de formatos específicos de texto, como pueden ser PDF, PostScript o Microsoft Word. Además utilizan técnicas de detección de artículos académicos en post-procesamiento, donde pueden emplearse algoritmos de aprendizaje automatizado, expresiones regulares, reglas ad hoc entre muchas otras. Los documentos académicos son obtenidos de sitios web de instituciones de educación y de investigación y la selección de semillas juega un papel preponderante en los resultados. Sin embargo, los artículos completos que pueden encontrarse son una porción minoritaria del total, puesto que a menudo se encuentran con derechos restringidos por ser comercializados.

El crawler utilizado en el prototipo del presente trabajo entra en esta última categoría, donde además se aplican restricciones de dominio en las URL semillas, puesto que los ODEs de interés son los pertenecientes a autores vinculados a la educación universitaria nacional.

### 2.1.2. Crawlers Open Source

Entre los crawlers de acceso abierto, se destacan:

- **Heritrix** [10]: extensible y escalable es el crawler utilizado en la Internet Archive<sup>11</sup>. Su desarrollo comenzó en el 2003 y la intención era desarrollar un crawler con el propósito específico de utilizarse en sitios

---

<sup>9</sup><http://scholar.google.com.ar/>

<sup>10</sup><http://academic.research.microsoft.com/>

<sup>11</sup>Organización sin ánimo de lucro destinada a la preservación de historiales Web y recursos multimedia - <https://archive.org/>

web de archivo y a su vez dar soporte a múltiples casos de crawling focalizado. El software es de código abierto para fomentar la colaboración y el desarrollo conjunto entre las instituciones con necesidades similares. Su arquitectura modularizada facilita la personalización y contribución exterior.

- **Apache Nutch** [11]: altamente escalable y modular. Provee interfaces extensibles de parsing, indexación y filtros de selección por puntajes de gran ayuda para implementaciones personalizadas. Quizás, el módulo de conexión mas utilizado en la comunidad sea Apache Solr<sup>TM</sup>, que brinda un motor de indexación de texto sobre gran variedad de tipos MIME junto con un motor de búsqueda sobre la indexación. Nutch se puede ejecutar en una sola máquina, pero gana mucho de su fuerza en un clúster Hadoop.
- **Crawler4j** [12]: liviano, escalable, rápido, escrito en Java y muy fácil de configurar. Su desarrollo estuvo a cargo de Yasser Ganjisaffar<sup>12</sup>, que actualmente se desempeña como ingeniero del equipo de desarrollo del motor de búsquedas Bing de Microsoft. Como muestra de su potencial se observa que procesó el sitio web entero de Wikipedia en sólo 10 horas.

En el presente trabajo se decidió utilizar el Crawler4j, por la facilidad de configuración en cuanto a puesta en marcha y selección de políticas de crawling, como también por contar con un desempeño notable y suficiente a los fines de un prototipo de sistema de recopilación de información desarrollado. Cabe destacar, que los proyectos de mucho mayor embergadura como Nutch y Heritrix, aprovechan su potencial en sistemas distribuidos dedicados en tareas de crawling permanentes, características no requeridas por el prototipo aquí presentado.

### 2.1.3. Consulta a motores de búsqueda externos

Una de las alternativas evaluadas y descartadas, fue la utilización de consultas a motores de búsqueda externos, como Google, Yahoo, etc. Como se explicó previamente, estos motores ya cuentan con motores de crawling funcionando permanentemente y con una amplia gama de consultas en diferentes tipos de archivos soportados (PDFs, PostScript, etc... ). Las ventajas de consultas a los mismos, radican naturalmente en el ahorro del procesamiento que ésta tarea pueda insumir.

Si bien muchos sistemas de recopilación de información utilizan este enfoque (por ejemplo Agathe [2]) en el presente trabajo no pudo ser aplicado,

---

<sup>12</sup><http://www.ganjisaffar.com/>

ya que es menester contar con la estructura del sitio web que posea el enlace a un ODE de interés. Esto se debe a que es muy poco frecuente encontrar los datos de contacto de autores en un ODE, los cuales analizaremos mas adelante, pueden encontrarse en páginas HTML cercanas a las que vinculan al ODE.

## **2.2. Extracción, Recuperación y Recopilación de Información**

El objetivo principal de los sistemas de Extracción de Información (EI) es localizar información a partir de documentos de texto en lenguaje natural, produciendo como salida del sistema un formato tabular estructurado, en un formato fijo de datos sin ambigüedad, que pueden ser resumidos y presentados de manera uniforme [13].

La representación uniforme de los datos y sus relaciones resulta conveniente para la inspección y comparación de hechos, que pueden ser visualizados con mayor facilidad en una tabla. Además, contando con una representación uniforme y estructurada de un documento, éste puede analizarse con herramientas automáticas tales como técnicas de minería de datos para el descubrimiento de patrones y posterior interpretación de los mismos.

Una de las principales tareas de los sistemas de EI, consiste en encontrar la información relevante del documento de entrada, donde por supuesto, la relevancia se define de acuerdo a la información que se espera encontrar. Muchas veces la información se encuentra distruibuida en la Web, que puede ser entendida como una colección de documentos con grandes porciones de lenguaje natural y distribuida en diferentes servidores y archivos de variados formatos. Por lo cuál, la Web es una gran fuente para el descubrimiento de conocimiento.

Un sistema de EI, puede pensarse como el intento de convertir información desde diferentes documentos de texto a una base de datos. Por lo tanto, la extracción de información exitosa de la Web puede ser vista como el volcado de la Web a una base de datos.

Extracción de información refiere a una tarea diferente de la recuperación de información (RI). Esta última, se enmarca en un campo mucho más antiguo y maduro en el que la principal atención está puesta en la selección de un subconjunto de documentos dentro de un conjunto mucho más grande a través de una consulta. El usuario de un sistema de RI, luego de una consulta debe examinar el/los documentos de la salida para la consiguiente extracción de información.

El contraste entre los objetivos de las tareas de RI con la de EI, puede

entonces resumirse en: *un sistema de RI recupera los documentos relevantes dentro de una colección más grande*, mientras que un *sistema de EI extrae información relevante dentro de uno o mas documentos*. Por lo tanto, ambas técnicas son complementarias y utilizadas en combinación pueden resultar en herramientas poderosas de procesamiento de texto.

Ambas áreas difieren además en las técnicas de cómputo utilizadas. Estas diferencias se deben en parte a los objetivos inherentes de las mismas y en parte a la historia de cada una de las áreas. Gran parte del trabajo que ha emergido en la EI proviene de sistemas basados en reglas, computación lingüística y procesamiento de lenguaje natural, mientras que en el campo de la RI, han influido áreas como teoría de la información, probabilidad y estadística.

Por otra parte, suele llamarse sistemas de *recopilación de información*, a los encargados de realizar la recuperación y la extracción de información sobre colecciones bien definidas [2].

## Métricas de Evaluación

La necesidad de métricas de evaluación en los problemas de EI aparecieron rápidamente en las conferencias sobre la problemática [13]. El punto de partida para el desarrollo de las mismas fueron las ya conocidas métricas en RI de cobertura (recall) y precisión. Aunque las medidas para EI no son las mismas que en RI, los nombres si se han mantenido. En EI, la precisión puede ser entendida como la fracción de datos correctos dentro de todos los recuperados, mientras que la cobertura como la fracción de datos correctamente extraídos con respecto a todos los datos disponibles. Así es que Precisión P y Cobertura C, son definidas como:

$$P = \frac{\#Respuestas\ Correctas}{\#Respuestas\ Producidas}$$
$$C = \frac{\#Respuestas\ Correctas}{\#Total\ Posible\ de\ Respuestas\ Correctas}$$

Los casos bordes sobre estas métricas suceden cuando se tiene denominador nulo tanto en P como en C. Para C, cuando  $\#Total\ Posible\ de\ Respuestas\ Correctas = 0$ , el cómputo resulta indeterminado (NaN). La convención usual al respecto<sup>13</sup> es tomar  $C = 1$  en caso de que  $\#Respuestas\ Producidas = 0$  y  $C = 0$  en caso contrario; premiando o penalizando el hecho de haber producido o no datos espurios ante la ausencia de datos. Por otra

---

<sup>13</sup>[http://en.wikipedia.org/wiki/Talk%3APrecision\\_and\\_recall](http://en.wikipedia.org/wiki/Talk%3APrecision_and_recall)

parte, cuando  $\#Respuestas\ Producidas = 0$  se asigna un valor no numérico a P, considerando que carece de importancia medir la precisión en esta situación.

Al comparar el desempeño de un sistema, las dos métricas necesitan ser consideradas. Como no resulta de gran ayuda la comparación directa de ambos parámetros al mismo tiempo, muchos enfoques para combinar ambas tasas han sido propuestas. Una de esas y quizás la más famosa es la medida F (F-measure), que combina la precisión P y cobertura C, en una sola métrica:

$$F = \frac{(\beta^2 + 1) PR}{\beta^2 P + R}$$

El parámetro  $\beta$  determina cuánto se favorece la cobertura sobre la precisión. En investigaciones sobre EI, frecuentemente se utiliza  $\beta = 1$  ponderando por igual precisión y cobertura. Con la medida F, el desempeño relativo de diferentes sistemas de EI pueden fácilmente ser comparados.

### 2.3. Herramientas de extracción de información sobre documentos de texto

Hasta el día de hoy, no hay muchos trabajos en la extracción automática de metadatos en documentos de texto [14, 15]. Cada herramienta extrae distintos tipos de metadatos, tiene sus propios objetivos, arquitectura y usa distintas técnicas. En [15] se revisan herramientas extractoras de metadatos generales tales como el título, los autores, las palabras claves, el resumen y el idioma. En particular, se compararon las herramientas: KEA<sup>14</sup>, MrDlib<sup>15</sup>, AlchemyAPI<sup>16</sup>, ParsCit<sup>17</sup>. Luego de realizar distintas experimentaciones con las herramientas mencionadas, sobre un corpus de 760 documentos de un repositorio universitario se analizaron los resultados obtenidos respecto a los distintos metadatos que pueden ser extraídos por cada uno ellos: Mr. DLib para título y autores, KEA para palabras claves y Alchemy para Título, Palabras Claves e Idioma. Se observó que los resultados encontrados con KEA y Alchemy respecto a palabras claves son similares en precisión y que los resultados obtenidos con MrDLib y Alchemy para título y autores, también lo son. Además, se contrastaron los resultados de Alchemy con su combinación con ParsCit para el preprocesamiento de documentos, obteniendo con la

---

<sup>14</sup>[www.nzdl.org/Kea/index\\_old.html](http://www.nzdl.org/Kea/index_old.html)

<sup>15</sup>[www.mr-dlib.org](http://www.mr-dlib.org)

<sup>16</sup>[www.alchemyapi.com](http://www.alchemyapi.com)

<sup>17</sup>[wing.comp.nus.edu.sg/parsCit/](http://wing.comp.nus.edu.sg/parsCit/)

combinación Alchemy+ParsCit los mejores resultados. ParsCit permite dar estructura al documento y genera en un documento XML en el cual intenta identificar Título, Autor, Resumen y Palabras Clave. Esta información se concatena en un nuevo archivo, el cual se utiliza para subir al servidor de AlchemyAPI en lugar del archivo original. A partir de los resultados obtenidos en dicho trabajo, se decidió en esta propuesta utilizar Alchemy+ParsCit para la extracción de información y Apache PDFBox<sup>18</sup> para la conversión a texto plano de archivos en formato PDF. A continuación se describen brevemente las herramientas utilizadas.

### **Alchemy API**

*AlchemyAPI* es una plataforma de minería de texto la cual proporciona un conjunto de herramientas que permiten el análisis semántico utilizando técnicas de procesamiento de lenguaje natural. Provee un conjunto de servicios que permiten analizar de forma automática documentos de texto plano o HTML. La herramienta expone varios servicios a partir de su RESTful API, entre los que se encuentran: Extracción de Autor, Entidades, Palabras Claves, Categorización del Contenido e Identificación del Idioma. En su versión gratuita, el servicio presenta una limitación de 1000 consultas diarias y un límite por consulta de 150 kbs.

### **ParsCit**

*ParsCit* es una aplicación de código abierto que realiza dos tareas: el análisis sintáctico de cadenas de referencia, también llamado extracción de citas, y el análisis de la estructura lógica de documentos científicos. Estas tareas las realiza a partir de un archivo de texto plano utilizando procedimientos de aprendizaje automático supervisado que usan campos aleatorios condicionales como mecanismo de aprendizaje. Incluye utilidades para ejecutarse como un servicio web o como una aplicación independiente.

### **PDFBox**

*Apache PDFBox* es una herramienta Java de código abierto para trabajar con documentos PDF. Permite la creación de nuevos documentos PDF, la manipulación de documentos existentes y la capacidad de extraer el contenido de los documentos. Esta herramienta extrae texto de este tipo de archivos. Además, realiza una búsqueda de metadatos (autor, título, organización, palabras claves, etc) que pueden estar embebidos en el archivo binario, cargados

---

<sup>18</sup><http://pdfbox.apache.org/>

en el momento de la construcción del PDF. La herramienta parsea el “Document Catalog” del binario en búsqueda de contenido dentro de la sección Metadata.

## 2.4. Técnicas de matching de nombres propios

Encontrar y combinar nombres de personas y/o entidades es un problema muy frecuente en áreas de IE, IR y Web Mining; tanto en motores de búsqueda, como sistemas de deduplicación y/o enlace de datos. Las variaciones de idioma, de formato, problemas de codificación de caracteres y diferencias entre características de nombres retornan muy dificultosa el enfoque de coincidencia exacta, por lo que no se conoce hasta el momento una técnica predominante en el área que resulte efectiva en diferentes circunstancias [17].

En particular, un problema muy frecuente en IE, es que diferentes extractores de entidades para el mismo documento de texto, pueden originar diferentes cadenas de texto para una misma entidad. Por ejemplo, las cadenas “Dra. Ana Casali”, “A. Casali” y “Casali A” representan a la misma persona, en formatos diferentes. En el área, diversas técnicas de correspondencia aproximadas basadas en la codificación fonética o coincidencia de patrones han sido propuestas. En [17] se realiza una recopilación de las mismas y se postulan diferentes recomendaciones para afrontar el problema basadas en las características del dominio de aplicación. Según estas indicaciones, se optó por utilizar las que se detallan a continuación.

### Distancia Levenshtein o de edición

La distancia Levenshtein [18] entre dos cadenas de texto es definida como la mínima cantidad de operaciones de edición (inserción, borrado o sustitución) para convertir una cadena en la otra. Existen algoritmos de programación dinámica que logran calcular este número en  $O(|s_1| \times |s_2|)$ .

Además, la distancia puede ser convertida en una medida de similitud (entre 0.0 y 1.0) mediante:

$$sim_{ld}(s_1, s_2) = 1,0 - \frac{dist_{ld}(s_1, s_2)}{\max(|s_1|, |s_2|)}$$

donde  $dist_{ld}(s_1, s_2)$  es la distancia de edición calculada por el algoritmo. La distancia Levenshtein es simétrica y siempre se cumple que  $0 \leq dist_{ld}(s_1, s_2) \leq \max(|s_1|, |s_2|)$  y que  $abs(|s_1| - |s_2|) \leq dist_{ld}(s_1, s_2)$ , esto último permite filtrar cadenas de longitudes muy diferentes.

## Distancia Damerau-Levenshtein

Esta distancia es una variación de la de Levenshtein tomando en consideración la transposición de un carácter como otra operación básica de costo 1 [18]. La medida de similitud correspondiente  $sim_{dl}$  puede ser calculada en forma análoga a la de Levenshtein  $sim_{ld}$ .

## Distancia Smith-Waterman

El algoritmo de Smith-Waterman es una reconocida estrategia originalmente propuesta para realizar alineamiento local óptimo de secuencias biológicas (ADN, ARN o proteínas), determinando regiones similares entre 2 secuencias [19]. Está basado en un enfoque similar a la distancia de edición, con el agregado de permitir saltos o espacios de no coincidencia entre las cadenas. Utiliza algoritmos de programación dinámica, de tal forma de garantizar que el alineamiento local encontrado entre dos cadenas sea óptimo con respecto a un determinado sistema de puntajes utilizado para valorar positivamente coincidencias y negativamente las diferencias, permitiendo además, la posibilidad de inserción, borrado (valorado negativamente) y/o espacio en blanco (con valor nulo comúnmente).

Las alternativas básicas para realizar el alineamiento de un par de secuencias son: el alineamiento local y el alineamiento global. Los alineamientos globales pretenden alinear cada símbolo de cada secuencia. Esta estrategia es especialmente útil cuando las secuencias a alinear son altamente similares y aproximadamente del mismo tamaño. En contraste, los alineamientos locales son más útiles cuando las secuencias a alinear poseen grandes diferencias, pero se sospecha que existen regiones de similitud. Al permitir espacios de no coincidencia, este algoritmo resulta muy útil para el matcheo de nombres compuestos o aquellos abreviaciones que poseen una inicial. La complejidad de este algoritmo es  $O(\min(|s_1|, |s_2|) \times |s_1| \times |s_2|)$ . Mientras que la complejidad espacial es  $O(|s_1| \times |s_2|)$

La medida de similitud puede ser calculada como:

$$sim_{swd}(s_1, s_2) = \frac{bs_{swd}(s_1, s_2)}{div_{swd}(s_1, s_2) \times match\_score}$$

donde  $bs_{swd}(s_1, s_2)$  es el puntaje que logra el alineamiento óptimo,  $match\_score$  es el puntaje que se asigna en una coincidencia y  $div_{swd}(s_1, s_2)$  se puede tomar en una de tres formas: (1)  $div_{swd}(s_1, s_2) = \min(|s_1|, |s_2|)$ , (2)  $div_{swd}(s_1, s_2) = \max(|s_1|, |s_2|)$  o (3)  $div_{swd}(s_1, s_2) = 0,5 \times (|s_1| + |s_2|)$

En particular, la medida de similitud del algoritmo Smith-Waterman con alineamientos óptimos locales, fue utilizada para la búsqueda de entidades

(de tipo autor y filiación) en fragmentos de texto y para el deduplicación de entidades de tipo autor. La medida de similitud del algoritmo Damerau-Levenshtein se utilizó en la deduplicación de entidades de tipo filiación, ya que al tratarse de cadenas de mayor longitud y no buscar una alineación parcial mostraban mejores resultados que las del algoritmo Smith-Waterman.

## 2.5. Base de Datos Orientadas a Gráfos

Un sistema de gestión de base de orientada a grafo BDOG [20] consiste en un sistema de gestión de base de datos en línea con métodos para crear, leer, actualizar y eliminar datos (CRUD<sup>19</sup>) que exponen un modelo orientado a grafos, formado por nodos de información (entidades) y aristas representando relaciones entre ellos.

Las BDOG fueron concebidas para ser empleadas con sistemas transaccionales OLTP<sup>20</sup>. En consecuencia, son optimizadas para tener buen desempeño transaccional, y diseñadas en base a la integridad de las transacciones y disponibilidad operacional.

Dos características importantes en BDOG son:

- *Almacenamiento nativo orientado a grafos*: diseñado y optimizado para mejor desempeño y escalabilidad de modelos orientados a grafos. Muchos motores de BDOG utilizan este tipo de almacenamiento, sin embargo, otros serializan los datos para embeberlos en base de datos relacionales y utilizar el almacenamiento provisto por éstas últimas (lo cuál conlleva en procesamiento extra en algunas operaciones).
- *Procesamiento nativo orientada a grafos*: algunas operaciones en BDOG tienen mejor desempeño en un modelo de cómputo libre de índice de adyacencia para nodos, lo cuál indica qué nodos conectados del grafo apuntan “físicamente” entre ellos. Esta característica beneficia significativamente las consultas sobre la BDOG que requieren recorridos por el grafo subyacente. Aunque sin embargo, muchos motores de BDOG modernos optan por relajar esta característica puesto que gran parte de consultas que no requieren recorridos, resultan demasiado costosas en términos de procesamiento y memoria.

---

<sup>19</sup>CRUD es el acrónimo de Crear, Leer, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete)

<sup>20</sup>OLTP es la sigla en inglés de Procesamiento de Transacciones En Línea, un tipo de procesamiento que facilita y administra aplicaciones transaccionales, usualmente para entrada de datos y recuperación y procesamiento de transacciones.

En general y mas allá de las características recién mencionadas, se toma como BDOG a aquellos sistemas en línea que exponen un modelo de datos orientado a grafos a través de operaciones CRUD.

Naturalmente, todas las virtudes de la teoría de grafos son tomadas en las BDOG. Constituyendo, por lo tanto, una herramienta poderosa para consultas vinculadas a grafos como por ejemplo, computar el camino más corto entre dos nodos. Esta es la razón por la cuál se optó por una base de datos orientada a grafos en el presente trabajo, puesto que la estructura de los sitios web a crawlear puede ser fácilmente representada por un grafo a recorrer.

Mas allá del beneficio de un modelo de gran alcance y legibilidad que brinda un grafo, existe una motivación extra para el empleo de BDOG. Esta radica en un conjunto de casos de uso y patrones de datos, en los cuáles el rendimiento mejora de uno o más órdenes de magnitud para consultas sobre conjuntos de datos fuertemente conectados. En estos, el enfoque “por recorridos” es significativamente mas veloz que el enfoque por “secuencia de asociaciones” (join-intensive) utilizado en las bases de datos relacionales estándares. En estas últimas, la performance de consultas de tipo join suele verse deteriorada en la medida de que la DB incrementa su tamaño. Aquí es donde, para datos fuertemente conectados y poco asociados, las BDOG cuentan con ventaja, por tener complejidad relacionada con la porción del grafo subyacente que involucra la consulta, mas allá del tamaño total del mismo.

## Neo4j

En el presente trabajo se utilizó Neo4j<sup>21</sup>, que es una herramienta de software libre de Base de Datos Orientada a Grafos, ACID<sup>22</sup>, OLTP (completamente transaccional que expone CRUD), implementada en Java. Utiliza almacenamiento nativo orientado a grafos, y cuenta con la implementación de algoritmos usuales de recorrido de grafos (camino más corto, búsqueda primero a lo ancho, etc...), lo cuál ahorra tiempo de programación. Puede ser utilizado en forma embebida dentro de cualquier aplicación java o como servicio de DBMS independiente y accesible a través de interfaces REST en una gran variedad de lenguajes, que además cuenta con una interface de usuario Web para gestión y visualización de datos (figura 1).

---

<sup>21</sup><http://www.neo4j.org/>

<sup>22</sup>En bases de datos se denomina ACID (acrónimo de Atomicity, Consistency, Isolation and Durability), a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

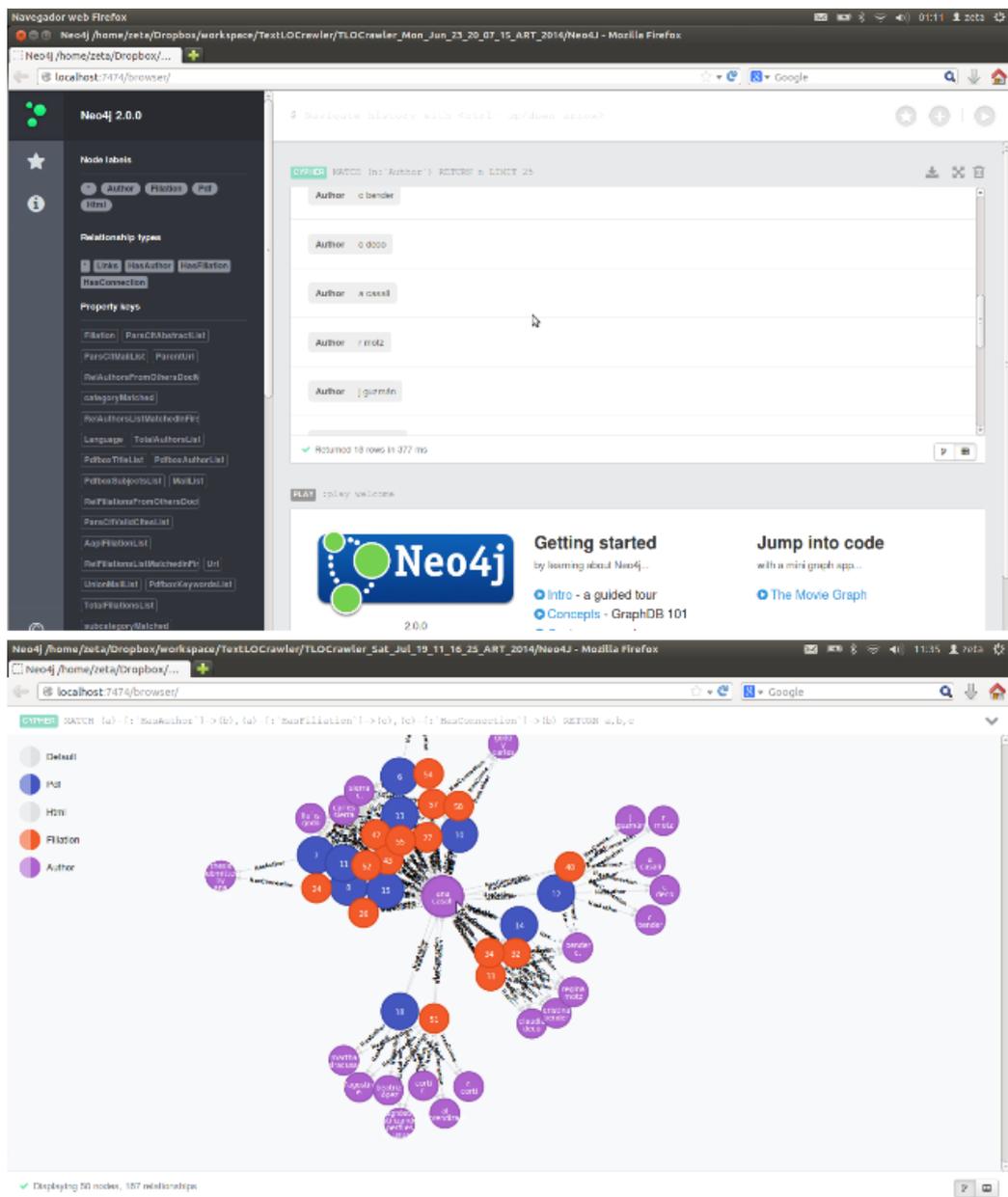


Figura 1: Arriba se observa la interfaz web de Neo4j. Abajo, un grafo con nodos y relaciones originadas por el sistema.

### 3. Arquitectura del Sistema

En esta sección se propone la arquitectura de un sistema de recopilación de información en dominios restringidos. La misma fue pensada de acuerdo a las necesidades de la problemática de motivación descrita anteriormente para la población de repositorios institucionales con ODEs. En base a esto, se repasan las principales necesidades funcionales en la automatización de búsqueda y clasificación de documentos relevantes para publicar en un repositorio:

1. Recolectar los objetos digitales de una lista de dominios Web configurable (en el producto final, se planifica setear los dominios pertenecientes a universidades públicas nacionales).
2. Clasificar los documentos (dentro de categorías a definir, como por ejemplo publicación, material de clase, etc...).
3. Extraer información de autoría, filiación y contacto (nombre del autor, universidad, institución, email de contacto).
4. Por cada objeto relevante encontrado, almacenar el documento en una base de datos junto con la información obtenida, para la posterior visualización de un operador.

#### Precondiciones del sistema

Si bien la arquitectura que se propone se describirá en términos generales, existen 2 condiciones que se hayan presentes en el dominio de la problemática de recopilación de ODEs y que deberían verificarse para poder aplicar la arquitectura aquí propuesta a otro dominio. Se trata de:

1. Relevancia de Documentos: se considerarán relevantes todos los documentos en formato PDF hallados desde las URLs semillas. Esto en particular se cumple en el dominio de interés de ODEs por estar orientado a semillas de URL de sitios web de materias de carreras específicas de universidades o páginas de investigadores.
2. Información distribuida en diferentes páginas de un mismo sitio web: se parte de la observación empírica de que la información de un autor (nombre, filiación, email) de un documento muchas veces no se encuentra en el mismo documento ni en la página web que referencia a ese documento en la Web, pero sí puede encontrarse en otra página del mismo sitio web. Esto es muy común en páginas educativas de materias,

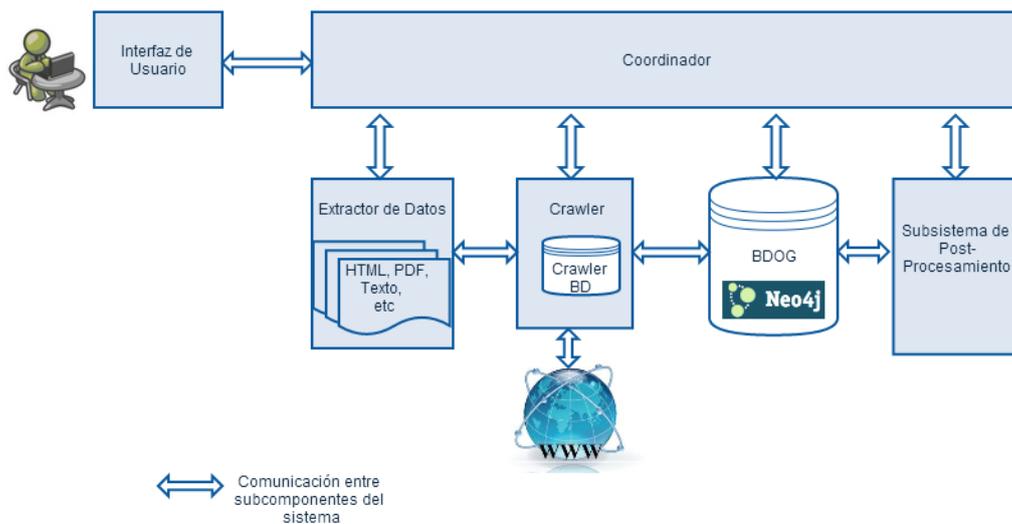


Figura 2: Diagrama de Arquitectura.

donde el material de clase se halla en una sección del determinado sitio, mientras que la información de contacto de los docentes de la cátedra se encuentra otra. Como también en sitios web de investigadores, cuya sección de publicaciones donde se encuentra la mayoría de los links a documentos relevantes, puede no contener la información de contacto, que muchas veces se halla en la página raíz de su sitio web.

### 3.1. Propuesta de Arquitectura y descripción de componentes

Como ya se mencionó, para la elaboración de la arquitectura aquí propuesta, fueron estudiadas otras arquitecturas presentadas en sistemas de recopilación de información (AGATHE, CROSSMARC, CiseSeerX). Las principales diferencias con la que aquí se propone, nacen especialmente en la incorporación de la representación de la estructura de los sitios web correspondientes

a las URLs semillas en una base de datos orientada a grafos. En la Figura 2 se aprecian la componentes principales. A continuación se describe cada uno de ellos:

- **Interfase de Usuario:** es el medio con que el usuario puede comunicarse con el sistema, iniciando y configurando las tareas de recopilación de información y visualización de datos al finalizar. Es la componente con quién interactúa el administrador de contenidos.
- **Coordinador:** es el encargado de coordinar, configurar y comunicar el resto de los componentes. Su propósito, es mantener de manera aislada y modularizable el resto de las componentes, de forma que puedan ser modificadas (por extensión o compresión) con facilidad. Se encarga de gestionar los procesos de configuración e inicialización del crawler, post-procesamiento al finalizar el crawling y visualización de datos por parte del operador.
- **Crawler:** es el componente encargado de realizar las tareas específicas de web crawling (detalladas en la Sección 2.1). A medida que recupera los recursos apuntados por las URLs, los cuáles pueden ser de diferentes formatos (HTML, PDF, CSS, etc...), y en base a la política de crawling (pertenencia al dominio de la URL semilla), notifica al coordinador sobre el recurso hallado. Además, se encarga de solicitar la extracción de información de dicho recurso y luego redirigirlo, junto con la información extraída, a la BDOG para su persistencia.
- **Extractor de Información:** este componente es el encargado de realizar la extracción de información de los diferentes recursos recuperados por el crawler. Se compone de diferentes módulos especializados en diferentes tipos de archivos. Dependiendo del dominio de aplicación del sistema, podría ser necesario implementar esta componente con submódulos que se especialicen según el tópico de información a extraer. Existen sistemas de recopilación de información que operan sobre un mismo tipo de archivo, por ejemplo páginas HTML, y la información a extraer varía en base a diferentes tópicos de interés. Para dominios con diversos tipos de documentos y que se presentan en diferentes formatos, resulta beneficioso utilizar una arquitectura que cuente con un componente que determine los tópicos de interés para que luego delegue a diferentes módulos especializados en la extracción de información de cada tópico en particular. En el presente trabajo, se utiliza un enfoque distinto para cada tipo de archivo, puesto que diferentes archivos se procesan con diferentes herramientas. PDFBox

realiza la extracción de texto de un archivo PDF. Además, esta herramienta realiza una búsqueda de metadatos (autor, título, palabras claves, etc.) que pueden estar embebidos en el archivo binario y son cargados por el autor en el momento de la construcción del PDF. ParsCit realiza el análisis de la estructura de documentos científicos y con ella logra reconocer diferentes secciones de un ODE: título, autores, emails, filiaciones, resumen y citas, entre otras. AlchemyAPI se utiliza para la extracción de palabras claves y reconocimiento de entidades a través de servicios web, los cuáles reciben como entrada HTML y retornan su salida en formato XML o JSON. Específicamente las entidades por las que se consulta son de tipo organización y persona, como potenciales valores de filiación y autor respectivamente.

- **Base de Datos Gráfica:** se encarga de persistir la estructura de cada uno de los sitios Web siendo crawleados (cuyas páginas principales usualmente son las URLs semillas de entrada del sistema). Se genera en la misma, una estructura de tipo árbol, donde los nodos corresponden a URLs recuperada por el crawler y las hojas son o bien URLs que no tienen otras URLs salientes, o bien recursos de cierto tipo de formato que son el objetivo del sistema recopilador. En la Figura 3 se puede apreciar una consola a una BDOG, donde se visualiza la estructura de un sitio web (los nodos negros corresponden a páginas HTML mientras que los azules a archivos PDF). Para cada nodo, almacena en diferentes propiedades del mismo la información extraída por el Extractor de Información y por algunos campos provenientes del crawler (URL padre, URL hijas, dominio, tamaño). Además provee a la componente de Post-Procesamiento, las facilidades de recorridos entre los nodos del árbol.
- **Subsistema de Post-Procesamiento:** una vez terminado el proceso de crawling, el coordinador es notificado, quien a su vez inicia el post-procesamiento. En el mismo, se recorren las hojas recuperadas, y a partir de cada una de ellas se realizan recorridos ascendentes en la jerarquía de la estructura del sitio Web que quedó persistida en la BDOG. Los recorridos empiezan en una hoja y ascienden hasta una distancia determinada a partir de la configuración del sistema. El objetivo del recorrido es encontrar y vincular información extra que no pudo ser hallada en la hoja, y que se cree probable encontrar en un nodo no tan lejano. Por ejemplo, en el prototipo propuesto, se busca un mail de contacto, posibles filiación y posible autores. Además, las entidades correspondientes a autores y filiaciones, son persistidas en no-



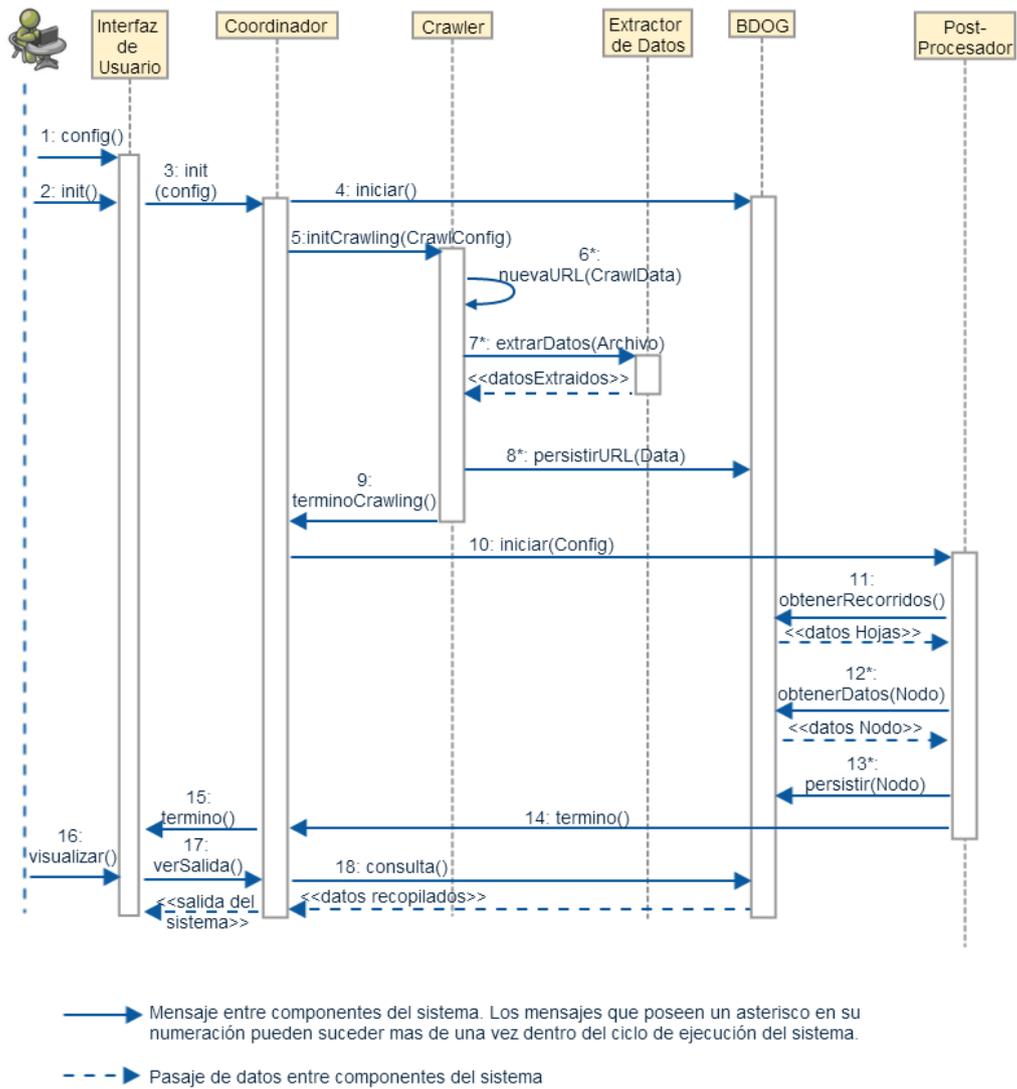


Figura 4: Diagrama de Secuencia e Interacción Entre Componentes.

sente en el recorrido y establece nuevas asociaciones con la información que potencialmente puede vincularse (Mensajes 12 y 13). Al finalizar su trabajo, notifica al Coordinador quién a su vez lo expone a la interfaz (Mensajes 14 y 15).

Por último, el usuario puede solicitar la visualización de datos, lo que deriva en consultas desde el Coordinador a la Base de Datos Gráfica y la visualización de los resultados obtenidos por el sistema a través de la interfaz (Mensajes 16, 17 y 18).

## 4. Desarrollo del prototipo

Se desarrolló un primer prototipo del sistema recopilador propuesto. El mismo fue implementado en Java en el mismo lenguaje del web crawler Crawler4j y el motor de BDOG Neo4j como base de datos embebida del sistema. También se utiliza el cliente web de la BDOG para la visualización de datos. Los wrappers de las herramientas de extracción de información: Apache PDFBox, Alchemy API y ParsCit también fueron implementados en Java. El proyecto completo junto con las librerías dependientes e instrucciones de ejecución e instalación pueden descargarse desde el repositorio público <https://github.com/ZetaSMB/TextLOGatheringSystem>.

Si bien la implementación se desarrolló teniendo en cuenta la arquitectura propuesta, por tratarse de un prototipo, algunos aspectos menores difieren ligeramente de la misma, para facilitar su implementación.

A continuación se examinan las principales clases del sistema y se analiza en detalle las características más relevantes de las mismas.

### 4.1. Inicio y Configuración del Sistema

El sistema se inicia desde la línea de comandos y para su configuración espera encontrar un archivo XML, llamado `Parameters.xml`, como el siguiente:

Parameters.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <general>
3
4   <crawlerParameters>
5     <SEEDS>http://www.fceia.unr.edu.ar/ingsoft/</SEEDS>
6   <ONLY_SUBPATHS>true</ONLY_SUBPATHS>
7   </crawlerParameters>
8
9   <parsCit_Home>
```

```

11 /home/Documents/ParsCit-master/bin
   </parsCit_Home >

13 <neo4j_Home >
   /home/Documents/neo4j-community-2.0.0
15 </neo4j_Home >

17 <alchemyApiKey >
   <!-- COMPLETE-->
19 </alchemyApiKey >

21 <stopWords_authors >
   el,la,las,del,lo,los,del,rosario,mit,tesis,páginas,práctica
   ,et al,tla
23 </stopWords_authors >

25 <stopWords_filiations >
   el,la,las,del,lo,los,del,csp,abril,for
27 </stopWords_filiations >

29 <postCrawlGraphTraversal >
<max_depth_emailist >3</max_depth_emailist >
31 <key_max_depth_potentialauthorlist >3</
   key_max_depth_potentialauthorlist >
<key_confidence_lower_bound_potentialauthorlist_aapi >0.0</
   key_confidence_lower_bound_potentialauthorlist_aapi >
33 <key_similarity_lower_bound_author_matching >0.7</
   key_similarity_lower_bound_author_matching >
<key_similarity_lower_bound_author_removeDups >0.55</
   key_similarity_lower_bound_author_removeDups >
35 <key_max_depth_potentialfiliationlist >3</
   key_max_depth_potentialfiliationlist > <
   key_confidence_lower_bound_potentialfiliationlist_aapi
   >0.0</
   key_confidence_lower_bound_potentialfiliationlist_aapi > <
   key_similarity_lower_bound_filiation_matching >0.65</
   key_similarity_lower_bound_filiation_matching > <
   key_similarity_lower_bound_filiation_removeDups >0.6</
   key_similarity_lower_bound_filiation_removeDups >
   </postCrawlGraphTraversal >

37 <postCrawlLabeling >
39 <label >
   <category >Material Educativo </category >
41 <subcategory >Práctica </subcategory >
   <maxCharactersInDoc >1500 </maxCharactersInDoc >
43 <matchingWords >práctica,prácticas,ejercicio,
   ejercicios,ejercitación,práctico </matchingWords >
</label >

```

```

45 <label>
      <category>Material Educativo</category>
47      <subcategory>Apunte de Clase</subcategory>
      <maxCharactersInDoc>1500</maxCharactersInDoc>
49      <matchingWords>apunte ,apuntes ,notas ,material ,clase ,
          capítulo ,capítulos</matchingWords>
</label>
51 <label>
      <category>Tesis</category>
53      <subcategory>>null</subcategory>
      <maxCharactersInDoc>1500</maxCharactersInDoc>
55      <matchingWords>tesis ,tesina</matchingWords>
</label>
57 <label>
      <category>Publicación</category>
59      <subcategory>>null</subcategory>
      <maxCharactersInDoc>1500</maxCharactersInDoc>
61      <matchingWords>publicación ,publicaciones</
          matchingWords>
</label>
63 </postCrawlLabeling>
</general>

```

Las diferentes secciones del XML, configuran el comportamiento de diferentes componentes del sistema:

- **crawlerParameters**: parámetros utilizados por el **Crawler**. El primero, **SEEDS** es la lista de URLs semillas del crawler, mientras que el segundo **ONLY\_SUBPATHS** indica un valor booleano de una regla de crawling. Esta última cuándo está habilitada, establece como criterio necesario para la visita a una URL recuperada en el crawling, que la misma debe estar formada por alguna de las URLs semillas mas otra cadena o subpath (la URL no es visitada en caso contrario).
- **neo4j\_Home**: directorio de instalación de Neo4j donde se encuentra el ejecutable **neo4j.sh** necesario para levantar la instancia de la BDOG en modo servidor<sup>23</sup>.
- **parsCit\_Home**: directorio de instalación de ParsCit donde se encuentra el ejecutable **citeExtract.pl** necesario para el wrapper que aplica dicha herramienta sobre los documentos de texto<sup>24</sup>.

<sup>23</sup>Mas información sobre la instalación de Neo4j puede encontrarse en <http://docs.neo4j.org/chunked/stable/deployment.html>

<sup>24</sup>Mas información sobre la instalación de ParsCit puede encontrarse en <https://github.com/knmnyn/ParsCit>

- `alchemyApiKey`: clave válida de consulta a servicios web de AlchemyAPI<sup>25</sup>.
- `stopWords_Authors` y `stopWords_Filiaions`: palabras que son excluidas por los extractores de posibles autores o filiaciones respectivamente. Este campo ayuda a mejorar la precisión de los extractores que pueden producir falsos positivos en ciertos tipos de entidades. Usualmente tienden a repetirse por lo que los errores en los resultados de corridas previas pueden ser prevenidos en las siguientes.
- `postCrawlGraphTraversal` y `postCrawlLabeling`: parámetros utilizados por el **Post-Procesador** que serán detallados mas adelante.

## 4.2. Controlador

Este componente se implementó en la clase `PdfCrawController.java` (incluida en el apéndice A.1). Sus principales tareas son:

- Instanciar la clase singleton `Parameters.java` que se encarga de parsear archivo XML de igual nombre de configuración.
- Configurar e instanciar el **Crawler**.
- Iniciar la etapa de post-procesamiento, una vez terminado el proceso de crawling (con su consiguiente recuperación de archivos).
- Terminar el proceso de la BDOG embebida e iniciar el proceso de base de datos en modo servidor<sup>26</sup>, para iniciar la interfaz Web de la misma en un navegador y permitir al usuario la visualización de datos recuperados.

## 4.3. Interfaz de usuario

Como ya se mencionó, el sistema se inicia por línea de comandos y la visualización de salida se realiza mediante un navegador Web accediendo a la interfaz de la BDOG Neo4j. En la figura 5 se observa la página inicial de dicha interfaz, se pueden visualizar los accesos directos a las diferentes propiedades, relaciones y etiquetas de nodos, y una terminal para ejecución de consultas en el lenguaje Cypher (el utilizado por la herramienta).

<sup>25</sup>Una clave con cuota limitada puede solicitarse gratuitamente en <http://www.alchemyapi.com/>

<sup>26</sup>este paso depende de la correcta configuración del parámetro `neo4j.Home`

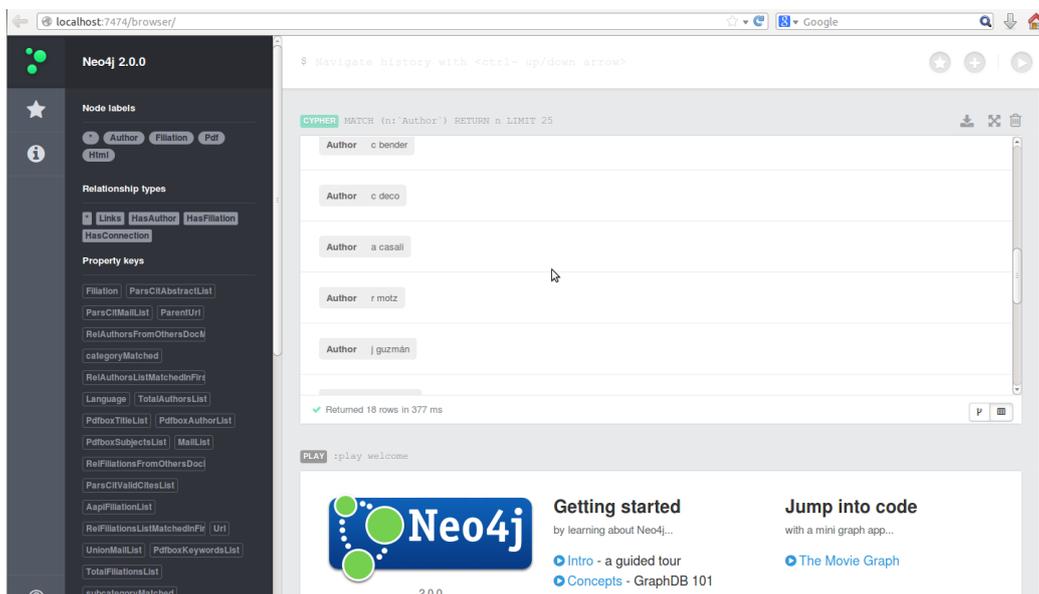


Figura 5:  $\hat{I}$ nterfaz web de consulta Neo4j

#### 4.4. Base de Datos Orientada a Grafos Embebida y Entidades del Modelo

Este componente se implementó mediante las librerías de **Neo4j** y su interfaz principal se halla en la clase `EmbeddedNeo4j.java` (incluida en el apéndice A.2). Esta es la encargada de crear la base de datos; iniciar y terminar los servicios; crear, modificar y eliminar las entidades: nodos, relaciones e índices.

Al crear la BDOG se crean 2 índices, uno para referenciar entidades de tipo *Html* que son utilizados para persistir la información de páginas HTML y otro para referenciar entidades de tipo *Pdf* para igual propósito con archivos PDF. Ambos índices son de gran utilidad para ejecutar consultas y recorridos sobre nodos dentro de la BDOG. Durante el crawling, los nodos creados son etiquetados con alguna de estas 2 categorías.

En nodos de tipo *Html* las propiedades que se persisten (en caso de encontrarse y en formato cadena de texto), son las que provienen del proceso de crawling:

- Key: hash MD5 de la URL
- Url: URL de la página
- Html: código HTML de la página

- PlainText: texto que se obtiene de la página luego de remover el HTML
- Domain: dominio de la URL
- SubDomain: sub-dominio de la URL
- ParentUrl: primer URL desde donde fue encontrada la URL siendo explorada
- Title: título de la página

sumadas a las que provienen de la extracción de datos:

- MailList: lista de mails provenientes del extractor `RegExMailExtractor` aplicados al HTML de la página
- AapiAuthorList: lista de entidades de tipo Autor provenientes del extractor `AlchemyapiWrapperExtractor`
- AapiFiliationList: lista de entidades de tipo Organización provenientes del extractor `AlchemyapiWrapperExtractor`
- AapiKeywordList: lista de palabras claves provenientes del extractor `AlchemyapiWrapperExtractor`
- AapiLanguage: idioma detectado proveniente del extractor `AlchemyapiWrapperExtractor`

Además, para cada URL saliente encontrada dentro del HTML de la página, en caso de que la misma deba ser persistida<sup>27</sup> se establece la relación *Links*, desde el nodo siendo persistido a cada una de estas URLs salientes.

En nodos de tipo *Pdf*, las propiedades que se persisten (también en caso de encontrarse y en formato cadena de texto), son las que provienen del proceso de crawling:

- Key: hash MD5 de la URL
- Size: tamaño en bytes del archivo
- Url: URL de la página
- Html: código HTML de la página

---

<sup>27</sup>cumple con los criterios descritos en la sección Crawler, método `public boolean shouldPersisted(WebURL url)`

- `KeyParentDbPage`: propiedad key de la primer URL desde donde fue encontrada la URL siendo explorada
- `ParentDbPage`: primer URL desde donde fue encontrada la URL siendo explorada

sumadas a las que provienen de la extracción de datos:

- `TextFragment`: texto plano proveniente del extractor `PDFTextAndMetadataExtractorWrapper`
- `PdfboxAuthorList`: lista de entidades de tipo Autor provenientes del extractor `PDFTextAndMetadataExtractorWrapper`
- `PdfboxTitleList`: lista de posibles títulos provenientes del extractor `PDFTextAndMetadataExtractorWrapper`
- `PdfboxRights`: derechos del documento provenientes del extractor `PDFTextAndMetadataExtractorWrapper`
- `PdfboxKeywordsList`: lista de palabras claves provenientes del extractor `PDFTextAndMetadataExtractorWrapper`
- `MailList`: lista de mails provenientes del extractor `RegExMailExtractor` aplicado al texto plano de la primer página del documento
- `ParsCitMailList`: lista de mails provenientes del extractor `ParsCitWrapper`
- `ParsCitAuthorList`: lista de entidades de tipo Autor provenientes del extractor `ParsCitWrapper`
- `ParsCitFiliationsList`: lista de entidades de tipo Filiación provenientes del extractor `ParsCitWrapper`
- `ParsCitTitleList`: lista de posibles títulos provenientes del extractor `ParsCitWrapper`
- `ParsCitValidCitesList`: lista de citas válidas provenientes del extractor `ParsCitWrapper`

No se examina la existencia de URLs salientes dentro del texto de un archivo PDF, por lo cuál, estos nodos solo pueden tener relaciones de tipo *Links* entrantes. Un ejemplo de visualización de nodos y relaciones terminado el proceso de crawling puede verse en la Figura 3.

## 4.5. Crawler

Este componente se implementó mediante las librerías del crawler **Crawler4j** y su interfaz principal se halla en la clase `PdfCrawler.java` (incluida en el apéndice A.3) . El proceso de crawling se configura e inicia con los parámetros `crawlerParameters` (detallados anteriormente).

Por cada nueva URL encontrada, la instancia de esta clase es notificada<sup>28</sup> y se encarga de decidir si los recursos apuntados en este enlace serán recuperados o no. La respuesta es positiva en caso de tratarse de un archivo PDF (analizando la extensión) o tener como dominio base alguno de los configurados en el parámetro **SEEDS** y negativa en caso contrario. Correspondientemente, para una URL que se decidió explorar, la instancia es notificada<sup>29</sup> y provista con los recursos apuntados descargados. En ese momento se produce la interacción del crawler con las clases que realizan el parseo y extracción de datos en diferentes archivos, los cuáles son persistidos en la BDOG.

## 4.6. Wrappers de Extracción de Datos

Estas componentes se hallan en el paquete `textExtraction` y son las encargadas de tareas específicas de extracción y encapsulamiento de herramientas externas.

### `PDFTextAndMetadataExtractorWrapper.java`

Encapsula y provee la interfaz al sistema para la herramienta `PDFBox`, que recibe un archivo binario y extrae el texto plano del mismo. Además, la herramienta examina la entrada *Metadata* del *catálogo de documentos*<sup>30</sup> del PDF cuyas propiedades en caso de encontrarse, son luego almacenadas en la BDOG. Un archivo binario PDF se estructura de objetos definidos en diccionarios. Por ejemplo, cada página del documento está representada por un objeto de tipo página, el cual es un diccionario que incluye referencias a contenido de la página. Estos objetos están conectados entre sí y forman un árbol de páginas, que se declara con una referencia indirecta en el catálogo de documentos. Este es el objeto raíz del cuál nacen las referencias de todos los objetos que componen el archivo. Una de las entradas de este objeto es la llamada *metadata* que contiene los metadatos del documento. Cabe mencionar que estos metadatos, son parte opcional del PDF, razón por la cuál pueden no estar presentes en el mismo.

<sup>28</sup>método `public boolean shouldVisit(WebURL url)`

<sup>29</sup>método `public void visit(Page page)`

<sup>30</sup>Mas información en [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=51502](http://www.iso.org/iso/catalogue_detail.htm?csnumber=51502)

### `RegexMailExtractor.java`

Esta clase implementa simplemente la búsqueda de cadenas que satisfagan una expresión regular que denota un posible email.

### `AlchemyapiWrapperExtractor.java`

Esta clase implementa las llamadas a servicios Web de Alchemy API y el parseo de los resultados obtenidos. Los servicios consultados son:

- API de Extracción de Entidades<sup>31</sup>: recibe una URL, código HTML o cualquier contenido Web. Procesa, normaliza y limpia el contenido (eliminando anuncios, enlaces de navegación y otros contenidos no importantes para la tarea); detecta el idioma principal del documento y extrae las entidades con nombre propio que puede inferir, entre otras cosas. Las entidades que son de interés en este trabajo son las de tipo Organización, que asociamos a potenciales filiaciones y de tipo Persona, que asociamos a potenciales autores<sup>32</sup>. Esta API es utilizada sobre el contenido de las páginas Web crawleadas.
- API de Extracción de Palabras Claves<sup>33</sup>: realiza la extracción de palabras clave desde HTML, texto plano o contenido Web, empleando algoritmos estadísticos de procesamiento de lenguaje natural para analizar los datos. Esta API es utilizada sobre el contenido de las páginas Web crawleadas.
- API de Detección de Idioma<sup>34</sup>: realiza la detección de idioma desde HTML, texto plano o contenido Web. Esta API es utilizada con la primer página del texto plano (extraído con PDFBox) del PDF.

### `ParsCitWrapper.java`

Esta clase implementa las llamadas a la librería *ParsCit* (con el texto plano de un PDF extraído por PDFBox) y el parseo de los resultados obtenidos. *ParsCit* realiza el análisis estructural del PDF, asignando porciones de texto a una de 23 categorías pre-definidas. En particular, las que son de interés en este trabajo son: título, autor, filiación, mail, cita y palabras claves. En la salida del programa se incluye la porción de texto de la categoría, junto

---

<sup>31</sup><http://www.alchemyapi.com/api/entity/urls.html>

<sup>32</sup>La lista completa de entidades capaz de reconocer por AlchemyAPI se encuentra en <http://www.alchemyapi.com/api/entity/types.html>

<sup>33</sup><http://www.alchemyapi.com/api/keyword-extraction/>

<sup>34</sup><http://www.alchemyapi.com/api/language-detection/>

con la confianza asociada (calculada por el algoritmo interno de aprendizaje automatizado). Además, para los campos de tipo mail, el texto de entrada así categorizado puede ser parseado. Esto se debe a que reconoce patrones como por ejemplo: “{rlolivia,rlbtsou}@cityu.edu.hk” retornando por separado ambos mails.

## 4.7. Post-Procesamiento y Entidades Persistidas Durante el Mismo

Las tareas del post-procesamiento se hallan en la clase `DBTraverseHelper.java` (incluida en el apéndice A.4) y tienen lugar cuando el crawling finaliza. En este momento ya se dispone de la información de los extractores que operan únicamente sobre el documento (`PDFTextAndMetadataExtractorWrapper` y `ParsCitWrapper`) y lo que se intenta es vincular la información que se extrae desde el sitio web. Por lo tanto, para cada documento, se aplican otros 2 extractores que llamaremos:

- **EI 2:** se recorren las páginas web ascendiendo en la jerarquía de enlaces que induce la relación *Links*, en búsqueda de entidades de tipo autor o filiación (encontradas por `AlchemyAPI`). Todas las encontradas son luego filtradas, quedando sólo aquellas que aparecen en los primeros caracteres del documento. El objetivo de este extractor es encontrar entidades dentro del documento (reconocidas por `AlchemyAPI` en el sitio web), que pudieron no ser reconocidas por los extractores que operan sólo con el texto del mismo.
- **EI 3:** se busca en cada documento, la existencia de entidades de tipo autor o filiación encontradas en el resto de los documentos. Esto se lleva a cabo por dos razones, la primera es que en muchos casos, por diferentes características de la estructura de un ODE, una entidad presente en más de un documento sólo logra ser reconocida por `PDFTextAndMetadataExtractorWrapper` o `ParsCitWrapper` en uno. La segunda es que al tratarse de documentos de un sitio web con las características ya mencionadas, es probable que una misma entidad de tipo autor o filiación esté presente en más de un ODE.

En ambos extractores la búsqueda de entidades se realiza con las técnicas de *matcheo* aproximado, explicadas en la sección 2.5, donde los umbrales de decisión se hayan en los parámetros de ejecución<sup>35</sup>.

---

<sup>35</sup>Entradas `key_similarity_lower_bound_author_matching` y `key_similarity_lower_bound_filiation_matching` del archivo `Parameters.xml`

Al finalizar el post-procesamiento de un ODE, se calcula la unión de todos los valores conseguidos por los diferentes extractores, filtrando duplicados nuevamente con los algoritmos de razonamiento aproximado.

Se realizan 2 iteraciones sobre todos los nodos *Pdf* (referenciados en el índice). En la primer iteración, por cada nodo *Pdf*, se recorren todos los nodos de tipo *Html* ascendiendo en la jerarquía del árbol formado por la relación *Links* y hasta la profundidad configurada en los parámetros dentro de la categoría `postCrawlGraphTraversal`<sup>36</sup>. En estos recorridos, sobre cada nodo *Html*, se buscan las propiedades `MailList`, `AapiAuthorList` y `AapiFiliationList`. Los valores de las mismas se unen y se persisten en nuevas propiedades del nodo *Pdf* del cuál se inició el recorrido. Estas son `RelMailList`, `RelAuthorsList` y `RelFiliationsList`. En el caso de las últimas 2, es posible incluir sólo los valores de entidades de tipo autor o filiación (encontradas por `AlchemyAPI`) mayores a una confianza previamente configurada<sup>37</sup>. Al terminar un recorrido, se realiza un filtrado de los valores almacenado en las propiedades

`RelAuthorsList` y `RelFiliationsList`, recuperando solamente aquellos que se encontraron en la primera página del documento analizado. El resultado es persistido en nuevas propiedades del *Pdf*:

`RelFiliationsListMatchedInFirstTextFragment` y

`RelAuthorsListMatchedInFirstTextFragment` y contienen los potenciales autores o filiaciones encontradas en enlaces relacionados<sup>38</sup> que fueron encontrados en la primer página del PDF. El contenido de estas propiedades es lo que llamamos el extractor **EI 2**. Luego, las entidades de tipo autor y filiación encontradas por **ParsCit** y por **EI 2** son persistidas como nuevos nodos etiquetados como *Author* y *Filiation*. También se establecen nuevas relaciones: *HasAuthor* y *HasFiliation* desde un nodo *Pdf* a otro de tipo *Author* y *Filiation* respectivamente; y la relación *HasConnection* desde un nodo *Filiation* a otro de tipo *Author* entre todas las nuevas entidades originadas a partir del nodo *Pdf*. Un ejemplo del grafo final originado por el sistema puede verse en la figura 6, donde se puede apreciar la estructura de nodos que se crean durante el post-procesamiento.

En la segunda iteración sobre los nodos *Pdf*, se busca en cada uno de ellos la aparición en la primer página del documento de otros autores y filiaciones encontrados en nodos *Pdf* del mismo sitio web, y son almacenados en las

---

<sup>36</sup> `max_depth_emailist` , `max_depth_potentialauthorlist` y `max_depth_potentialfiliationlist`

<sup>37</sup> `key_confidence_lower_bound_potentialauthorlist` y `key_confidence_lower_bound_potentialauthorlist`, valor 0 en estos parámetros equivale a no aplicar ningún filtro por confianza

<sup>38</sup>Por la relación *Links* y hasta una profundidad configurable

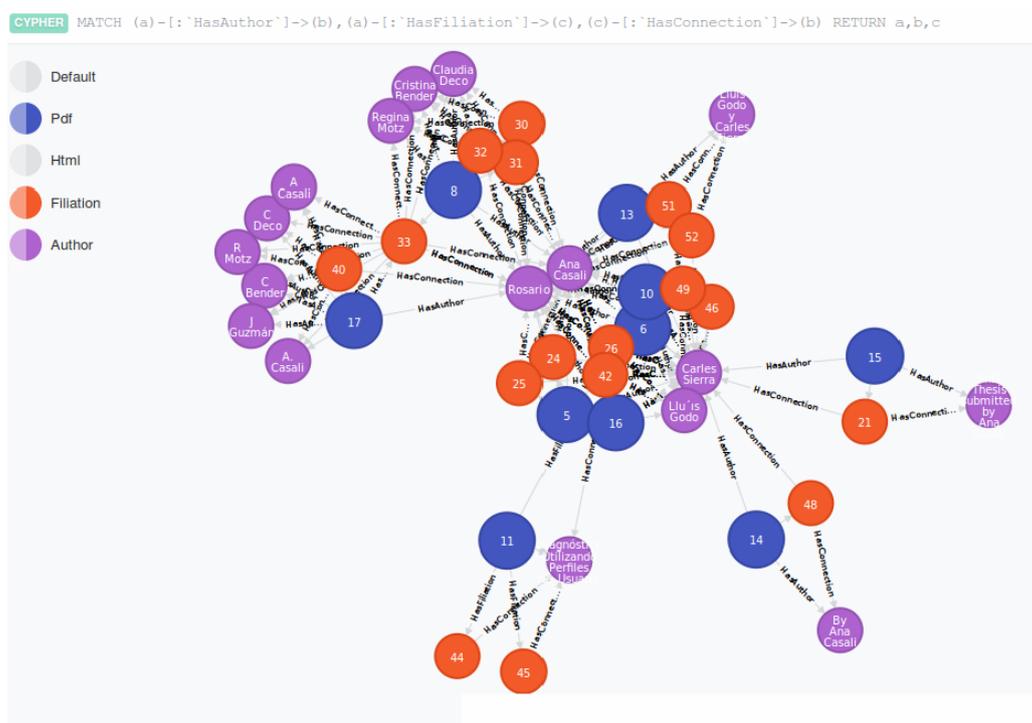


Figura 6: Visualización de BDOG resultante del crawling del sitio Web <http://www.fceia.unr.edu.ar/~acasali/>

propiedades: `RelAuthorsFromOthersDocMatched` y `RelFiliationsFromOthersDocMatched`. El contenido de estas propiedades es lo que llamamos el extractor **EI 3**.

La concatenación de los valores obtenidos por los extractores **ParsCit**, **EI 2** y **EI 3** es persistido en las variables `TotalAuthorsList` y `TotalFiliationsList`. Por último, se calcula la unión de los resultados obtenidos, filtrando los valores duplicados con los algoritmos de Smith-Waterman para las entidades de tipo autor y Levenshtein-Demearu para las de tipo filiación. El resultado se almacena en las propiedades `UnionAuthorsList` y `UnionFiliationsList`.

En el caso de los emails, se calcula la unión de los resultados de las propiedades: `MailList` (mails en primer hoja del documento, encontrados aplicando expresiones regulares al texto), `ParsCitMailList` (mails encontrados por **ParsCit**), `RelMailList` (mails en nodos relacionados); y la misma es persistida en la variable `UnionMailList` (en este caso el filtrado se realiza comparando las cadenas, sin aplicar técnicas aproximadas).

## Etiquetado

Durante la primer iteración sobre los nodos *Pdf* para cada uno de ellos se realiza un etiquetado por matcheo de palabras, que son preconfiguradas en la sección `postCrawlLabeling` de los parámetros. Dentro de esta sección se pueden incluir entradas de tipo `Label` que representan diferentes categorías con las se puedan etiquetar los documentos encontrados por el sistema. Por cada documento, se buscan dentro de los primeros `maxCharactersInDoc` caracteres y en el texto de la página *Html* padre inmediata del *Pdf* (relación *Links*), las palabras incluidas en el campo `matchingWords`. En caso de encontrarse, se persisten en las propiedades `category` y `subcategory` del *Pdf*, los valores del mismo nombre correspondientemente, de la sección `Label`. Por ejemplo, en los casos de estudio donde se ejecutó el prototipo del sistema, un ODE se etiquetaba con la categoría “*Material Educativo*” y subcategoría “*Práctica*”, si se encontraba alguna de las palabras: “práctica”, “ejercicio”, “ejercitación”, etc... que fueron incluidas en los parámetros iniciales de configuración.

## 5. Experimentación

Durante el desarrollo y puesta a punto, el prototipo del sistema se probó en diferentes sitios web correspondientes a páginas de materias de carreras con orientación en informática y páginas personales de docentes-investigadores,

en ambos casos de universidades públicas nacionales. El objetivo fue analizar la viabilidad de la arquitectura propuesta y evaluar los resultados obtenidos mediante distintos extractores con el fin de diseñar el módulo extractor con la combinación que brinde los mejores resultados.

Si bien en el sistema se incluye la extracción de muchos tipos de metadatos para un documento PDF, el esfuerzo y análisis se focalizó en la recuperación de: título, autor/es, filiación/es, idioma, mail de contacto y etiquetado.

Para evaluar el desempeño del mismo, a continuación se analizan los resultados de la ejecución en 2 sitios web específicos, en donde se calculan las métricas de precisión  $P$  y cobertura  $C$ , para los extractores aplicados. En ambas, se aplicaron convenciones de uso común en el área para casos bordes (mencionadas previamente en la sección 2.2):

- Si el extractor no produce respuestas (denominador nulo en  $P$ ): discriminamos este caso con valor  $P = NA$  (no aplica), puesto que no podemos medir precisión en este caso.
- Si no existen respuestas correctas (denominador nulo en  $C$ ) y el extractor devuelve algún resultado incorrecto tomamos  $P = 0$  y  $C = 0$ .
- Si no existen respuestas correctas (denominador nulo en  $C$ ) y el extractor no devuelve ningún resultado: en caso de que sea de interés indicar la ausencia de respuestas correctas, diferenciaremos con valor  $C = V$  (vacío), y tomaremos  $C = 1$  en caso contrario.

De esta forma, se tienen diferenciadas la efectividad de un extractor en los resultados producidos y la cobertura sobre el total de datos a extraer, penalizando en la cobertura la generación de datos espurios. Es válido aclarar que las entrada no numéricas en las tablas no se toma en cuenta en el cálculo del promedio total.

## Caso de Prueba 1

El primer caso de prueba que se analiza es la página personal de un docente-investigador de la UNR: <http://www.fceia.unr.edu.ar/~acasali/>. Los parámetros de configuración completos utilizados para la ejecución se muestran en el apéndice A.5. Diferentes consultas sobre la BDOG resultante para este sitio pueden observarse en la figura 7.

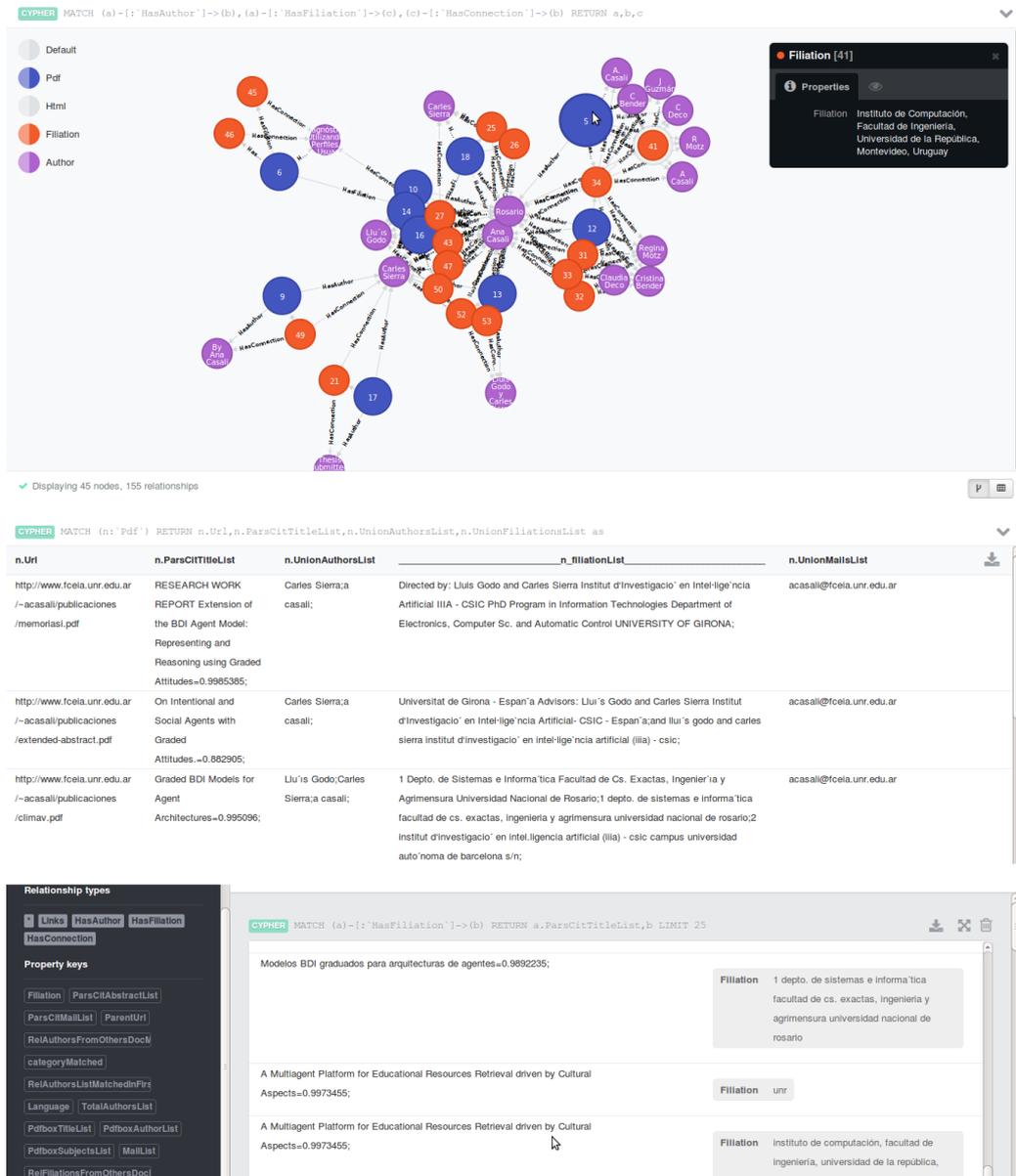


Figura 7: Diferentes consultas sobre la BDOG construida por el sistema para el sitio web del caso 1.

En este sitio, el sistema fue capaz de recuperar todos los documentos en formato PDF que se hallaban en el mismo, sin traer documentos no deseados de otros sitios<sup>39</sup>.

El extractor `PDFTextAndMetadataExtractorWrapper.java` logró recu-

<sup>39</sup>esto último gracias a la configuración inicial con la bandera `ONLY_SUBPATHS` activada

perar el texto plano de todos los documentos procesados. Sin embargo, los resultados para los metadatos recuperados por este extractor tuvieron P y C casi nulas, por lo que no se muestran detallados abajo.

La detección del idioma por el extractor `AlchemyapiWrapperExtractor.java` fue correcta en todos los documentos procesados, razón por la cuál no se incluyen las medidas detalladas de P y C para esta variable (100 % en ambas).

En la Tabla 1, se observan las métricas para el etiquetado, donde la columna Tipo de Documento indica la clasificación real hecha por el usuario y la columna Categoría las medidas P y C sobre el etiquetado propuesto por el sistema a los diferentes tipos de documentos. En el 90 % de los casos los documentos fueron etiquetados con 2 categorías entre las propuesta en los parámetros de configuración: *Publicación* y *Tesis*, de aquí surgen los valores más comunes de P y C de 0,5 y 1,0 respectivamente que arrojan un promedio sobre el total de 0,555 y 1,000.

Etiquetado Post-Crawling				
Documento	Tipo de Documento	Categoría		Subcategoría
		P	C	
Doc1	Tesis doctoral	1,000	1,000	NA
Doc2	Publicación	0,500	1,000	NA
Doc3	Publicación	0,500	1,000	NA
Doc4	Publicación	0,500	1,000	NA
Doc5	Publicación	0,500	1,000	NA
Doc6	Publicación	0,500	1,000	NA
Doc7	Publicación	0,500	1,000	NA
Doc8	Publicación	0,500	1,000	NA
Doc9	Publicación	0,500	1,000	NA
Doc10	Publicación	0,500	1,000	NA
		<b>0,550</b>	<b>1,000</b>	

Tabla 1: Precisión y corbertura para el etiquetado en el sitio web del Caso 1.

En la Tabla 2, se detalla el idioma de los documentos y luego, las medidas P y C para los campos Título, Autores y Filiaciones extraídos con `ParsCitWrapper.java`. Para Autores se tiene un 90 % en P y un 70 % en C. Respecto de la filiaiones, se tiene un 85 % en P, con un 73,33 % en C. En el caso del Título la extracción es óptima, 100 % en ambas medidas.

En la Tabla 3, se muestran las medidas P y C para los campos Autores y Filiaciones extraídos por EI 2 y EI 3 (explicados en la sección 4.7). Acá se observan excelentes medidas de precisión, aunque valores moderados en la

Documento	Idioma	ParsCit					
		Título		Autores		Filiaciones	
		P	C	P	C	P	C
Doc1	Ingles	1,000	1,000	1,000	0,333	1,000	1,000
Doc2	Ingles	1,000	1,000	1,000	0,333	0,500	0,333
Doc3	Ingles	1,000	1,000	1,000	1,000	1,000	1,000
Doc4	Ingles	1,000	1,000	1,000	1,000	1,000	0,500
Doc5	Ingles	1,000	1,000	1,000	1,000	1,000	1,000
Doc6	Español	1,000	1,000	0,000	0,000	0,000	0,000
Doc7	Ingles	1,000	1,000	1,000	1,000	1,000	0,500
Doc8	Ingles	1,000	1,000	1,000	0,333	1,000	1,000
Doc9	Ingles	1,000	1,000	1,000	1,000	1,000	1,000
Doc10	Español	1,000	1,000	1,000	1,000	1,000	1,000
		<b>1,000</b>	<b>1,000</b>	<b>0,900</b>	<b>0,700</b>	<b>0,850</b>	<b>0,733</b>

Tabla 2: Precisión y cobertura en la EI de ParsCit para el sitio web del Caso 1.

cobertura. Desde luego, la intersección entre resultados de los extractores ParsCit, EI 2 y EI 3 es no vacía en la mayoría de los casos. Con EI 2, se logran precisiones de 98,3 % y 100 %, con coberturas de 64 % y 28,3 % para autores y filiaciones respectivamente. Respecto de EI 3, se tienen precisiones del 100 % en ambas entidades con coberturas de 53,6 % y 63,3 % respectivamente.

Observando la Tabla 4, donde se detallan los resultados de P y C para la unión de extractores ParsCit, EI 2 y EI 3; se puede observar que la cobertura para los campos autor y filiación mejora con respecto a las del mejor extractor por sí solo, sin perder precisión. Esto es, con precisiones casi óptimas (del 98 % y 95 %), la cobertura aumentó en un 25,5 % en el caso de autores y un 22 % para las filiaciones recuperadas. Esto nos indica que la incorporación de datos que se encuentran en el sitio web donde fueron recuperados los documentos, logra mejorar la cobertura sobre la extracción de los mismos.

En la Tabla 5, se muestran las propiedades que resultan de la extracción de mails explicadas previamente. Cabe destacar que el análisis de P y C de los mails hallados en el sitio web presentes en la propiedad *RelMailList*, analizando cada caso (por tratarse de un sitio web conocido) se decidió si cada mail hallado correspondía a alguno de los autores o no. Observando los valores numéricos de las columnas de C para MailList y ParsCitMailList, se tiene que la mitad de los ODEs poseen al menos un mail de contacto. Al igual que en el caso de los campos Autores y Filiación de la Tabla 3, al calcular la unión de los resultados se logra mejorar precisión y cobertura, llegando

Documento	EI 2				EI 3			
	Autores		Filiaciones		Autores		Filiaciones	
	P	C	P	C	P	C	P	C
Doc1	1,000	0,666	NA	0,000	1,000	0,666	1,000	1,000
Doc2	1,000	0,666	1,000	0,500	1,000	0,666	1,000	0,500
Doc3	1,000	0,500	1,000	0,333	1,000	1,000	1,000	0,333
Doc4	1,000	0,400	1,000	0,500	1,000	0,200	1,000	0,500
Doc5	1,000	0,666	1,000	0,500	1,000	0,333	1,000	1,000
Doc6	0,833	0,833	1,000	0,500	1,000	0,166	1,000	0,333
Doc7	1,000	0,666	1,000	0,500	1,000	0,333	1,000	0,333
Doc8	1,000	0,666	NA	0,000	1,000	0,666	1,000	0,333
Doc9	1,000	0,666	NA	0,000	1,000	0,666	1,000	1,000
Doc10	1,000	0,666	NA	0,000	1,000	0,666	1,000	1,000
	<b>0,983</b>	<b>0,640</b>	<b>1,000</b>	<b>0,283</b>	<b>1,000</b>	<b>0,536</b>	<b>1,000</b>	<b>0,633</b>

Tabla 3: Precisión y cobertura de los extractores EI 2 y EI 3, para el sitio web del Caso 1.

Unión de extractores ParsCit, EI 2 y EI 3				
Documento	Autores		Filiación	
	P	C	P	C
Doc1	1,000	1,000	1,000	1,000
Doc2	1,000	1,000	1,000	1,000
Doc3	1,000	1,000	1,000	1,000
Doc4	1,000	1,000	1,000	1,000
Doc5	1,000	1,000	1,000	1,000
Doc6	0,833	0,833	0,500	0,500
Doc7	1,000	1,000	1,000	1,000
Doc8	1,000	0,667	1,000	1,000
Doc9	1,000	1,000	1,000	1,000
Doc10	1,000	1,000	1,000	1,000
	<b>0,983</b>	<b>0,950</b>	<b>0,950</b>	<b>0,950</b>

Tabla 4: Precisión y cobertura para la unión de los extractores ParsCit EI 2 y EI 3, en el sitio web del Caso 1.

al óptimo en P. Si bien la cobertura no es óptima, se logra recopilar el mail del 60% de los autores, teniendo además de cada uno de los documentos al menos un mail de contacto.

---

Documento	MailList		ParsCitMailList		RelMailList		UnionMailList	
	P	C	P	C	P	C	P	C
Doc1	NA	V	NA	V	1,000	0,333	1,000	0,333
Doc2	NA	V	NA	V	1,000	0,333	1,000	0,333
Doc3	1,000	0,666	1,000	1,000	1,000	0,333	1,000	1,000
Doc4	1,000	0,500	1,000	1,000	1,000	0,250	1,000	1,000
Doc5	NA	V	1,000	1,000	1,000	0,250	1,000	1,000
Doc6	1,000	0,333	1,000	1,000	1,000	0,333	1,000	1,000
Doc7	1,000	0,166	0,000	0,000	1,000	0,166	1,000	0,333
Doc8	NA	V	NA	V	1,000	0,333	1,000	0,333
Doc9	NA	V	NA	V	1,000	0,333	1,000	0,333
Doc10	NA	V	NA	V	1,000	0,333	1,000	0,333
	<b>1,000</b>	<b>0,416</b>	<b>0,800</b>	<b>0,800</b>	<b>1,000</b>	<b>0,300</b>	<b>1,000</b>	<b>0,600</b>

---

Tabla 5: Precisión y cobertura para los extractores de mails, para el sitio web del Caso 1.

## Caso de Prueba 2

El segundo caso de prueba que se analiza es el sitio web de la materia Ingeniería de Software, en la carrera Lic. en Cs. de la Computación, UNR: <http://www.fceia.unr.edu.ar/ingsoft/>. Los parámetros de configuración completos utilizados para la ejecución se muestran en el apéndice A.6. Diferentes consultas sobre la BDOG resultante para este sitio pueden observarse en la figura 8.



Figura 8: Diferentes consultas sobre la BDOG construida por el sistema para el sitio web del Caso 2.

En este sitio, el sistema fue capaz de recuperar todos los documentos en formato PDF que se hallaban en el mismo. Entre estos, dos documentos se excluyen del análisis, el primero por tratarse del CV de un docente y el segundo por tratarse de un PDF de imágenes, en el que no se pudo realizar la extracción de texto.

En este caso también se obtienen los textos planos de todos los documentos (con excepción del PDF de imágenes) mediante `PDFTextAndMetadataExtractorWrapper.java`. Por otra parte, los metadatos recuperados por este extractor son muy pocos, por lo que no se muestran detallados en ninguna tabla.

La detección del idioma por el extractor `AlchemyApiWrapperExtractor.java` nuevamente es correcta en el 100 % de los documentos procesados.

En la Tabla 6, se pueden ver las métricas para el etiquetado, donde la columna Tipo de Documento indica la clasificación real del usuario. A diferencia del sitio web del caso 1, aquí observamos mayor variedad de tipos de documentos (prácticas, PPTs, publicaciones). Se logran precisiones del 82,4 % y 73,5 % para la categoría y subcategoría, con coberturas óptimas en ambos casos.

En la Tabla 7, se detalla el idioma de los documentos y luego, las medidas P y C para los campos Título, Autores y Filiaciones extraídos con `ParsCitWrapper.java`. En el 64,7 % se recupera el título de forma correcta. Respecto de los autores, se tiene un 64,1 % en P y 55,9 % en C. Respecto de las filiaciones se logra un 83,3 % en P y 76,5 % en C. En comparación con el caso 1, en este extractor se observan medidas con menor eficacia y cobertura. Esto puede adjudicarse a dos razones: la primera es que la mayoría de los documentos son en español, y la segunda es la ya mencionada variedad en los tipos de documento (documentos menos estructurados).

En la Tabla 8, se muestran las medidas P y C para los campos Autores y Filiaciones extraídos por EI 2 y EI 3. Con EI 2, se logran precisiones de 68,5 % y 85,7 %, con coberturas de 88,2 % y 50 % para autores y filiaciones respectivamente. Respecto de EI 3, se tienen precisiones del 100 % en ambas entidades con coberturas de 64,7 % y 70,6 % respectivamente.

Observando la Tabla 9, donde se detallan los resultados de P y C para la unión de extractores ParsCit, EI 2 y EI 3. Para autores, la precisión es del 61,5 %, levemente menor a las precisiones de ParsCit y de EI 2, aunque mejorando la cobertura que llega al 94,1 %. En el caso de las filiaciones, se encuentran valores similares de P y C, 84,6 % y 76,5 % respectivamente, que las del mejor extractor individual (ParsCit para este campo). En este caso también se observa que la incorporación de datos que se encuentran en el sitio web donde fueron recuperados los documentos, logran mejorar la cobertura

Etiquetado Post-Crawling					
Documento	Tipo de Documento	Categoría		Subcategoría	
		P	C	P	C
Doc1	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc2	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc3	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc4	Material de Clase/Apunte	0,500	1,000	1,000	1,000
Doc5	Material de Clase/Apunte	0,500	1,000	1,000	1,000
Doc6	Material de Clase/Publicación	0,500	1,000	1,000	1,000
Doc7	Material de Clase/Publicación	0,500	1,000	1,000	1,000
Doc8	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc9	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc10	Material de Clase/Práctica	1,000	1,000	1,000	1,000
Doc11	Material de Clase/Apunte	0,500	1,000	0,500	1,000
Doc12	Material de Clase/Práctica	1,000	1,000	1,000	1,000
Doc13	Material de Clase/PPT	0,500	1,000	0,500	1,000
Doc14	Material de Clase/Práctica	1,000	1,000	1,000	1,000
Doc15	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc16	Material de Clase/Apunte	1,000	1,000	0,500	1,000
Doc17	Material de Clase/Práctica	1,000	1,000	1,000	1,000
		<b>0,824</b>	<b>1,000</b>	<b>0,735</b>	<b>1,000</b>

Tabla 6: Precisión y cobertura para el etiquetado en el sitio web del Caso 2.

ParsCit							
URL	Idioma	Título		Autores		Filiaciones	
		P	C	P	C	P	C
Doc1	Español	1,000	1,000	1,000	1,000	1,000	1,000
Doc2	Español	1,000	1,000	1,000	1,000	NA	1,000
Doc3	Español	0,000	0,000	0,000	0,000	1,000	1,000
Doc4	Español	0,000	0,000	1,000	1,000	1,000	1,000
Doc5	Español	1,000	1,000	1,000	1,000	0,500	1,000
Doc6	Ingles	0,000	0,000	0,333	0,500	0,500	1,000
Doc7	Ingles	1,000	1,000	1,000	1,000	1,000	1,000
Doc8	Español	1,000	1,000	0,000	0,000	1,000	1,000
Doc9	Español	0,000	0,000	0,000	0,000	1,000	1,000
Doc10	Español	1,000	1,000	NA	0,000	NA	0,000
Doc11	Español	1,000	1,000	1,000	1,000	1,000	1,000
Doc12	Español	0,000	0,000	NA	0,000	NA	0,000
Doc13	Español	1,000	1,000	NA	1,000	NA	1,000
Doc14	Español	0,000	0,000	0,000	0,000	0,000	0,000
Doc15	Español	1,000	1,000	1,000	1,000	1,000	1,000
Doc16	Español	1,000	1,000	1,000	1,000	1,000	1,000
Doc17	Español	1,000	1,000	NA	0,000	NA	0,000
		<b>0,647</b>	<b>0,647</b>	<b>0,641</b>	<b>0,559</b>	<b>0,833</b>	<b>0,765</b>

Tabla 7: Precisión y cobertura en la extracción por ParsCit en el sitio web del Caso 2.

Documento	EI 2				EI 3			
	Autores		Filiaciones		Autores		Filiaciones	
	P	C	P	C	P	C	P	C
Doc1	NA	0,000	NA	0,000	NA	0,000	NA	0,000
Doc2	1,000	1,000	NA	1,000	NA	0,000	NA	1,000
Doc3	0,500	1,000	NA	0,000	1,000	1,000	1,000	1,000
Doc4	1,000	1,000	NA	0,000	1,000	1,000	1,000	1,000
Doc5	0,500	1,000	NA	0,000	1,000	1,000	1,000	1,000
Doc6	0,800	1,000	1,000	1,000	NA	0,000	NA	1,000
Doc7	1,000	1,000	1,000	1,000	NA	0,000	NA	1,000
Doc8	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Doc9	0,333	1,000	1,000	1,000	1,000	0,000	1,000	1,000
Doc10	0,500	1,000	NA	0,000	1,000	0,000	NA	0,000
Doc11	1,000	1,000	NA	0,000	1,000	1,000	1,000	1,000
Doc12	0,500	1,000	NA	0,000	1,000	1,000	NA	0,000
Doc13	0,000	0,000	NA	1,000	NA	1,000	NA	1,000
Doc14	1,000	1,000	NA	0,000	1,000	1,000	NA	0,000
Doc15	0,500	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Doc16	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Doc17	0,333	1,000	0,000	0,000	1,000	1,000	NA	0,000
	<b>0,685</b>	<b>0,882</b>	<b>0,857</b>	<b>0,470</b>	<b>1,000</b>	<b>0,647</b>	<b>1,000</b>	<b>0,706</b>

Tabla 8: Precisión y cobertura por los extractores EI 2 y EI 3 en el sitio web del Caso 2.

sobre la extracción de los mismos sin perjudicar la precisión.

Unión de extractores ParsCit, EI 2 y EI 3				
Documento	Autores		Filiación	
	P	C	P	C
Doc1	1,000	1,000	1,000	1,000
Doc2	1,000	1,000	NA	1,000
Doc3	0,250	1,000	1,000	1,000
Doc4	1,000	1,000	1,000	1,000
Doc5	0,500	1,000	1,000	1,000
Doc6	0,666	1,000	1,000	1,000
Doc7	1,000	1,000	1,000	1,000
Doc8	0,500	1,000	1,000	1,000
Doc9	0,200	1,000	1,000	1,000
Doc10	0,500	1,000	NA	0,000
Doc11	1,000	1,000	1,000	1,000
Doc12	0,500	1,000	NA	0,000
Doc13	0,000	0,000	NA	1,000
Doc14	0,500	1,000	0,000	0,000
Doc15	0,500	1,000	1,000	1,000
Doc16	1,000	1,000	1,000	1,000
Doc17	0,333	1,000	0,000	0,000
	<b>0,615</b>	<b>0,941</b>	<b>0,846</b>	<b>0,765</b>

Tabla 9: Precisión y cobertura para la unión de los extractores ParsCit EI 2 y EI 3, en el sitio web del Caso 2.

En la Tabla 10, se muestran las propiedades que resultan de la extracción de mails explicadas previamente. Aquí también se consideró la clasificación del usuario para los valores de la propiedad *RelMailList*. Observando las columnas de cobertura de MailList y ParsCitMailList, se observa que sólo un documento incluía al menos un mail de contacto/autor. Por lo que en la mayoría de los casos, los únicos mails que se pueden asociar al documento son los encontrados en el sitio web (columna RelMailList), teniendo en la unión de los mismos, precisión del 75,5% y cobertura del 82,4%.

Documento	MailList		ParsCitMailList		RelMailList		UnionMailList	
	P	C	P	C	P	C	P	C
Doc1	NA	V	NA	V	0,000	0,000	0,000	0,000
Doc2	NA	V	NA	V	0,000	0,000	0,000	0,000
Doc3	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc4	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc5	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc6	1,000	1,000	NA	0,000	0,000	0,000	0,333	1,000
Doc7	NA	V	NA	V	0,000	0,000	0,000	0,000
Doc8	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc9	NA	V	0,000	0,000	1,000	1,000	0,500	1,000
Doc10	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc11	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc12	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc13	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc14	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc15	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc16	NA	V	NA	V	1,000	1,000	1,000	1,000
Doc17	NA	V	NA	V	1,000	1,000	1,000	1,000
	<b>1,000</b>	<b>1,000</b>	<b>0,000</b>	<b>0,000</b>	<b>0,765</b>	<b>0,765</b>	<b>0,755</b>	<b>0,824</b>

Tabla 10: Precisión y cobertura para los extractores de mails, en el sitio web del Caso 2.

## Conclusiones de la experimentación

En ambos casos, el sistema realizó correctamente el crawling del sitio web recuperando todos los documentos PDF de cada sitio. El extractor `PDFTextAndMetadataExtractorWrapper.java` logró extraer el texto plano de todos los ODEs recuperados (con excepción de uno, por tratarse de un PDF construido con imágenes). Sin embargo, los metadatos recuperados por este extractor son prácticamente nulos, lo que nos indica que muy pocos autores deciden incluir metadatos dentro en los campos de metadatos especiales del PDF (como se detalló previamente, esto es dentro de la header "metadata" del Document Catalog) a la hora de construir el mismo.

El desempeño del extractor `ParsCitWrapper.java` resulta significativamente mejor (del orden del 30 % en P y C) en los campos Título y Autores del sitio del Caso 1 con respecto al sitio del Caso 2. Esto puede adjudicarse a dos razones: la primera es que la mayoría de los documentos en el Caso 2 se encuentran en español, y la segunda es que en los mismos se halla mayor variedad de tipos de documento.

La cobertura del extractor **EI 2** es mayor en el Caso 2 que en el Caso 1, en un 24 % para el campo Autores y en un 22 % para el campo Filiaciones. Esto nos indica que en el sitio web del Caso 2 un mayor número de entidades son reconocidas en enlaces relacionados que luego son encontradas en el texto de algún ODE.

Teniendo en cuenta la extracción en los campos autores, filiación y mail en la unión de extractores, se tiene que para los tres campos se logra mejorar notablemente la cobertura respecto a la de los extractores que sólo tienen en cuenta el texto del ODE (ParsCit, PDFBox, expresiones regulares en texto). Esto se logra sin perder precisión, mediante el procesamiento de las páginas web vecinas dentro del sitio web donde se encontró el ODE.

En la Tabla 11, se muestran los resultados de P y C promediados tomando todos los documentos de ambos casos, para el mejor extractor en cada campo.

ParsCit		Union ParsCit, EI 2 y EI 3				Union de Extractores		Alchemy API		Etiquetado			
Título		Autores		Filiaciones		Mail		Idioma		Categoría		Subcategoría	
P	C	P	C	P	C	P	C	P	C	P	C	P	C
0,823	0,823	0,799	0,945	0,898	0,857	0,877	0,712	1,000	1,000	0,687	1,000	0,735	1,000

Tabla 11: Resultados de precisión y cobertura promedios para los mejores extractores de cada campo.

## 6. Conclusiones y trabajo a futuro

En el trabajo se ha propuesto una arquitectura para automatizar la tarea de recopilación de documentos de texto dentro de un dominio web restringido con el objetivo de detectar objetos digitales educativos plausibles de ser cargados en un repositorio institucional. Se ha implementado un prototipo que permitió evaluar la viabilidad de la arquitectura propuesta y experimentar sobre diferentes combinaciones de herramientas extractoras para obtener los mejores resultados en los campos de interés. A partir del análisis de los resultados obtenidos, se diseñaron extractores que combinan las herramientas de extracción tales como PDFBox, ParsCit, Alchemy, técnicas de matcheo aproximado y expresiones regulares, planteado como un Post-procesamiento de la información utilizando la Base de Datos orientada a Grafos generada durante el crawling del sitio web. En la experimentación, sobre todos los documentos de los sitios analizados: se recuperaron en forma correcta la totalidad del idioma, el 82,7% de los títulos, el 94,5% de autores, el 85,7% de las filiaciones y el 71,2% de los emails de contacto de autores. Las principales diferencias de esta propuesta con respecto a otros sistemas recopiladores son: la incorporación de la representación de la estructura de los sitios web correspondientes a las URLs semillas en una base de datos orientadas a grafos lo que permite recorrer y extraer información de nodos familiares enlazados, la diversidad de tipo de documentos a recopilar y la extracción de información tanto en idioma inglés como en español. Además, los resultados de la experimentación indican que la incorporación del procesamiento de información en nodos familiares, logran mejorar notablemente la cobertura sobre la extracción de metadatos, respecto a la de los extractores que solo tienen en cuenta el texto del ODE.

Se espera que esta herramienta sea de utilidad a los administradores de repositorios institucionales ya que automatiza una parte importante de la tarea de recopilar documentos.

Como trabajo a futuro se tienen las siguientes propuestas:

- Mejorar el etiquetado e incluir clasificación de relevancia sobre los documentos de modo de evitar el procesamiento en documentos no deseados (como por ejemplo programas de materias o CV de docentes).
- Explotar la información sobre las relaciones que se generan entre autores, filiaciones y documentos
- Extender soporte sobre más formatos de texto (PS, Microsoft Word y otros)

## Referencias

- [1] C. Deco, C. Bender, A. Casali, J. Sare “Sistemas de Información Inteligentes en el Dominio de Educación: Extracción y Calidad de Metadatos de Objetos de Aprendizaje”, Facultade de Cs. Exáctas y Agrimensura, UNR, Rosario, Aregntina.
- [2] B. Espinasse, S. Fournier, S.Albitar, Combining Agents and Wrapper Induction for Information Gathering on Restricted Web Domains, Research Challenges in Information Science, Nice, France, 2010.
- [3] Pazienza, M.I., A. Stellato, y M. Vindigni, Combining Ontological Knowledge and Wrapper Induction Techniques into an e-Retail System, in Workshop on ATEM03 held with ECMLPKDD 2003, Cavtat. 2003.
- [4] Li Huajing, I. Councill, L. Bolelli, Zhou Ding, Song Yang, Wang-chien Lee An , Sivasubramaniam C. Lee Giles, “CiteSeer X- A Scalable Autonomous Scientific Digital Library”,Department of Computer Science and Engineering, The School of Information Sciences and Technology, Pennsylvania State University, 2007.
- [5] C. Castillo, PhD Thesis: Web Crawling, Dept. of Computer Science - University of Chile November, 2004.
- [6] A. Heydon, M. Najork, “Mercator: A scalable, extensible Web crawler”, Compaq Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA.
- [7] M. Gray ,“Internet Growth and Statistics: Credits and Backgroun”, [www.mit.edu/people/mkgray/net/background.html](http://www.mit.edu/people/mkgray/net/background.html).
- [8] Eichmann, D.,“The RBSE Spider – Balancing Effective Search Against Web Loa” , 1994.
- [9] S. Brin, L. Page “The Anatomy of a Large-Scale Hypertex- tual Web Search Engine” In Proceedings of the Seventh International World Wide Web Conference, pp. 107–117, 1998.
- [10] G. Mohr, M. Stack, I. Ranitovic, D. Avery and M. Kimpton, “An Introduction to Heritrix An open source archival quality web crawler”, International Web Archiving Workshop, 2004.
- [11] <https://nutch.apache.org/>
- [12] <https://code.google.com/p/crawler4j/>

- [13] Eikvil L. “Information Extraction from WWW”, Norwegian Computer Center, Oslo, 1999.
- [14] T. Pire, B. Espinase, A. Casali y C. Deco, “Extracción automática de metadatos de objetos de aprendizaje: un estudio comparativo”, 2011.
- [15] A. Casali, C. Deco, A. Romano, G. Tomé, “An assistant for loading Learning Object Metadata: An ontology based approach. Interdisciplinary Journal of E-Learning and Learning Objects, IJELLO, Volume 9, pages 77-87. Informing Science + IT Education, 2013.
- [16] A. Casali, C. Deco, C. Bender, S. Fontanarrosa, C. Sabater, “Asistente para el Depósito de Objetos en Repositorios con Extracción Automática de Metadatos. XV Simposio Internacional de Tecnologías de la Información y las Comunicaciones en la Educación (SINTICE 2013)”, pp 133-136. Madrid, España, 2013.
- [17] P. Christen, “A Comparison of Personal Name Matching: Techniques and Practical Issues”, Joint Computer Science Technical Report Series, Department of Computer Science, Faculty of Engineering and Information Technology, The Australian National University, Australia, 2006.
- [18] G. Navarro, “A guided tour to approximate string matching, ACM Computing Surveys”, 33(1):31–88, 2001.
- [19] A. E. Monge y C. P. Elkan, “The field matching problem: Algorithm and applications”, In Proceedings of ACM SIGKDD, pages 267–270, Portland, 1996.
- [20] I. Robinson, J. Webber, y E. Eifrem, “Graph Databases”, Published by O’Reilly Media, Inc., First Edition, June 2013.

# A. Apéndice A: Principales clases Java del sistema

## A.1. Controlador

Archivo 1: PdfCrawController.java

```
1 package textL0Crawler;
2
3 import dbBroker.DBTraverseHelper;
4 import dbBroker.EmbeddedNeo4j;
5 import edu.uci.ics.crawler4j.crawler.CrawlConfig;
6 import edu.uci.ics.crawler4j.crawler.CrawlController;
7 import edu.uci.ics.crawler4j.fetcher.PageFetcher;
8 import edu.uci.ics.crawler4j.robotstxt.RobotstxtConfig;
9 import edu.uci.ics.crawler4j.robotstxt.RobotstxtServer;
10
11 import java.awt.Desktop;
12 import java.net.URI;
13
14 import textExtraction.Utilities;
15
16 public class PdfCrawController {
17     public static void main(String[] args) throws Exception {
18
19         Parameters parameters = Parameters.getInstance();//global system config
20
21         if (parameters.parsCitHome().isEmpty()) {
22             System.out.println("Error: You must provide the path to ParsCit installation to
23                 be able to execute the script associates with ParsCit!");
24             return;
25         }
26
27         if ( parameters.neo4jHome().isEmpty()) {
28             System.out.println("Error: You must provide the path to Neo4j installation to be
29                 able to visualize the system output.");
30             return;
31         }
32
33         if ( parameters.alchemyApiKey().isEmpty()) {
34             System.out.println("Error: You must provide the Alchemy API activation key.");
35             return;
36         }
37
38         java.util.Date date= new java.util.Date();
39         String dateSufix= date.toString().replace(' ','_').replace(':', '_');
40         String rootFolder = System.getProperty("user.dir")+ "/TLOCrawler_" + dateSufix;
```

```

40     CrawlConfig config = new CrawlConfig();
42     config.setCrawlStorageFolder(rootFolder);
44     /*
46      * numberOfCrawlers shows the number of concurrent threads that should
48      * be initiated for crawling.
50      */
52     int numberOfCrawlers = 2;
54     /*
56      * Be polite: Make sure that we don't send more than 1 request per
58      * second (1000 milliseconds between requests).
60      */
62     config.setPolitenessDelay(1000);
64     /*
66      * You can set the maximum crawl depth here. The default value is -1 for
68      * unlimited depth
70      */
72     config.setMaxDepthOfCrawling(-1);
74     /*
76      * You can set the maximum number of pages to crawl. The default value
78      * is -1 for unlimited number of pages
80      */
82     config.setMaxPagesToFetch(10000);
84     /*
86      * This config parameter can be used to set your crawl to be resumable
88      * (meaning that you can resume the crawl from a previously
90      * interrupted/crashed crawl). Note: if you enable resumable feature and
92      * want to start a fresh crawl, you need to delete the contents of
94      * rootFolder manually.
96      */
98     config.setResumableCrawling(false);
100    config.setConnectionTimeout(30000);
102    config.setIncludeHttpsPages(true);
104    config.setFollowRedirects(true);
106    config.setCrawlStorageFolder(rootFolder);
108    /*
110     * Since images are binary content, we need to set this parameter to
112     * true to make sure they are included in the crawl.
114     */
116    config.setIncludeBinaryContentInCrawling(true);
118    String[] crawlDomains = parameters.crawlerSeeds();//extract seeds to crawl

```

```

88     PageFetcher pageFetcher = new PageFetcher(config);
89     RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
90     RobotstxtServer robotstxtServer = new RobotstxtServer(robotstxtConfig,
91         pageFetcher);
92     CrawlController controller = new CrawlController(config, pageFetcher,
93         robotstxtServer);
94     for (String domain : crawlDomains) {
95         controller.addSeed(domain);
96     }
97
98     PdfCrawler.configure(crawlDomains, rootFolder, parameters.crawlOnlySubpaths
99         ()); //set the domains and the folder to persist pdfs files
100
101     //this call is blocking
102     controller.start(PdfCrawler.class, numberOfCrawlers);
103 //End crawling job
104
105     DBTraverseHelper.getInstance().traversePdfFilesInDB(); //post-crawling
106         processing
107     EmbeddedNeo4j.getInstance().shutDown(); //shutdown embedded database
108
109 Process proc;
110 try { //set config properties and open web interface of neo4j on server mode
111     String newConfig = Utilities.defaultNeo4jConfig.replace("<DBPATHTOREPLACE>",
112         rootFolder+"/Neo4J");
113     String newPathConfigFile = parameters.neo4jHome() + "/conf/neo4j-server-" +
114         dateSuffix + ".properties";
115
116     Utilities.saveStringToFile(newConfig, newPathConfigFile);
117
118     String pathConfig = parameters.neo4jHome() + "/conf/neo4j-wrapper.conf";
119     String oldWrapperConfig = Utilities.getFileContents(pathConfig);
120     String strToFind = "wrapper.java.additional=-Dorg.neo4j.server.properties=";
121     int indStart = oldWrapperConfig.indexOf(strToFind);
122     int indEnd = oldWrapperConfig.indexOf("\n", indStart);
123     indStart+=strToFind.length();
124
125     String strToReplace = oldWrapperConfig.substring(indStart, indEnd);
126     oldWrapperConfig = oldWrapperConfig.replaceFirst(strToReplace, newPathConfigFile
127         +"\n");
128
129     Utilities.saveStringToFile(oldWrapperConfig, pathConfig);
130
131     //stop the server
132     String[] strCmd = new String[]{parameters.neo4jHome() + "/bin/neo4j", "stop"};
133     System.out.println(strCmd);
134     proc = Runtime.getRuntime().exec(strCmd);
135     proc.waitFor();
136 //start the server

```

```

130     strCmd = new String[]{parameters.neo4jHome() + "/bin/neo4j","start"};
131     System.out.println(strCmd);
132     proc = Runtime.getRuntime().exec(strCmd);
133     proc.waitFor();
134
135     proc.destroy();
136     proc = null;
137
138     String neo4jWebClientUrl = "http://localhost:7474";
139
140     if (Desktop.isDesktopSupported()) {
141         // Windows
142         Desktop.getDesktop().browse(new URI(neo4jWebClientUrl));
143     } else {
144         // Ubuntu
145         Runtime runtime = Runtime.getRuntime();
146         runtime.exec("/usr/bin/firefox -new-window " + neo4jWebClientUrl);
147     }
148 } catch (Exception e) {
149     e.printStackTrace();
150 }
151
152 }

```

## A.2. BDOG Embebida

Archivo 2: EmbeddedNeo4j.java

```

package dbBroker;
2
import java.io.File;
4 import java.io.IOException;
import java.lang.reflect.Field;
6 import java.util.Map;
7
8 import org.apache.log4j.Level;
import org.neo4j.graphdb.*;
10 import org.neo4j.graphdb.factory.GraphDatabaseFactory;
import org.neo4j.graphdb.index.Index;
12 import org.neo4j.graphdb.index.UniqueFactory;
import org.neo4j.kernel.impl.util.FileUtils;
14
import textL0Crawler.CrawlUtilities;
16
18 @SuppressWarnings("deprecation")

```

```

20 public class EmbeddedNeo4j {
21
22     private static volatile EmbeddedNeo4j _instance = null;
23
24     private static String DB_PATH = null;
25     public static UniqueFactory<Node> factory;
26
27     public static GraphDatabaseService graphDb;
28
29     public static org.apache.log4j.Logger logger;
30
31     public static Index<Node> allDbPagesIndex;//index for speed the search and traverse
        over all nodes created
32     private static final String NAME_PAGES_INDEX = "pages";
33
34     public static Index<Node> pdfIndex;//index for speed the search and traverse over
        all pdf nodes created
35     private static final String NAME_PDF_INDEX = "pdf";
36
37     public static final String FIELDNAME_KEY = "Key";
38
39     public static final String FILIATION_NAME = "Filiation";
40     public static final String AUTHOR_NAME = "Author";
41
42     public static EmbeddedNeo4j getInstance() {
43         if(_instance == null)
44             return getInstance(System.getProperty("user.dir"));
45         else
46             return _instance;
47     }
48
49     public static EmbeddedNeo4j getInstance(String rootDBPath) {
50         if(_instance == null){
51             _instance = new EmbeddedNeo4j();
52             DB_PATH = rootDBPath + "/Neo4J";
53             File file = new File(DB_PATH);
54             file.mkdirs();
55             clearDb();
56             createDb();
57             registerShutdownHook( graphDb );
58         }
59         return _instance;
60     }
61
62     public static enum RelTypes implements RelationshipType {Links,HasFiliation,
        HasAuthor,HasConnection} //,Linked} deprecated
63
64     @Deprecated
65     public static void Save(DbPage nodeToSave){//deprecated

```

```

66 Transaction tx = graphDb.beginTx();
67     try
68     {
69
70         Node nodo = factory.getOrCreate( FIELDNAME_KEY, nodeToSave.getKey() );
71         nodo.setProperty("Url", nodeToSave.getUrl());
72         nodo.setProperty("Html", nodeToSave.getHtml());
73         nodo.setProperty("Domain", nodeToSave.getDomain());
74         nodo.setProperty("SubDomain", nodeToSave.getSubDomain());
75         nodo.setProperty("ParentUrl", nodeToSave.getParentUrl());
76         nodo.setProperty("Title", nodeToSave.getTitle());
77
78         for ( DbPage innerPage : nodeToSave.getLinkedPages() ){
79
80             //no linkear a la misma pagina
81             if (innerPage.getKey()== nodeToSave.getKey()) continue;
82
83             Node nodoLinked = factory.getOrCreate( FIELDNAME_KEY, innerPage.getKey() );
84             nodoLinked.setProperty("Url", innerPage.getUrl());
85             nodoLinked.setProperty("Domain", innerPage.getDomain());
86             nodoLinked.setProperty("SubDomain", innerPage.getSubDomain());
87             nodoLinked.setProperty("ParentUrl", innerPage.getParentUrl());
88
89             nodo.createRelationshipTo(nodoLinked, RelTypes.Links);
90         }
91
92         tx.success();
93     }
94     catch(Exception ex){
95         throw ex;
96     }
97     finally
98     {
99         tx.finish();
100     }
101 }
102
103 public void saveObject(IDbNeo4jObject nodeToSave) throws Exception{
104
105     Transaction tx = graphDb.beginTx();
106     try
107     {
108         //get or create the unique node for the key (implemented by the interface)
109         Node nodo = factory.getOrCreate( FIELDNAME_KEY, nodeToSave.getKey() );
110
111         logger.log(Level.INFO, "Saving □nodo, □Key: □" + nodeToSave.getKey() + " □nodo □ID
112             : □" + nodo.getId());

```

```

114 //recover all the public fields from the pojo
Field[] fields = nodeToSave.getClass().getDeclaredFields();
116 for(Field f : fields){
    if (f.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue; //skip
        implementation
118
    //Current Property Value
120 Object propertyValue = f.get(nodeToSave);
    if (propertyValue == null) continue;
122
    //if the field is not a collection set it in the node
124 if (!java.lang.Iterable.class.isAssignableFrom(propertyValue.getClass())) {
126     //if it is a dbNeoObject, save it (like a page or file)
    if (IDbNeo4jObject.class.isAssignableFrom(propertyValue.getClass())) {
128     Node propertyNode = factory.getOrCreate( FIELDNAME_KEY, ((IDbNeo4jObject)
        propertyValue).getKey() );
        logger.log(Level.INFO, "Saving Node Key: " + ((IDbNeo4jObject)
            propertyValue).getKey() + " node ID: " + propertyNode.getId());
130     for(Field innerPropF : propertyValue.getClass().getDeclaredFields()){
        if (innerPropF.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue;
132         Object innerPropertyValue = innerPropF.get(propertyValue);
            //Skip if the inner object implements iterators
134         if (innerPropertyValue == null) continue;
            if (java.lang.Iterable.class.isAssignableFrom(innerPropertyValue.
                getClass())) continue;
136         logger.log(Level.INFO, "Traverse intern property of node " + nodo.
            getId() + " => Field: " + innerPropF.getName() + " Value: " +
                innerPropertyValue);
            propertyNode.setProperty(innerPropF.getName(), innerPropertyValue)
                ;
138     }
        /* In the model, only the DbPage has a ParentDbPage:IDbNeo4jObject
            property, so setting this relationship is redundant cause the RelType.
            Links
140     */
        //nodo.createRelationshipTo(propertyNode, RelTypes.Linked);
142     }
        else
144     {
        //just a property
146     nodo.setProperty(f.getName(), propertyValue);
        }
148     continue; //move next for the next field
    }
150
    //if the object implements an iterator loop it (i.e. outgoing links)
152 @SuppressWarnings("rawtypes")

```

```

154 java.lang.Iterable coleccion = (Iterable) f.get(nodeToSave);
156     //Loop the colection
156     @SuppressWarnings("rawtypes")
158     java.util.Iterator iter = coleccion.iterator();
158     while (iter.hasNext()) {
160         Object innerObject = iter.next();
162         //see if implements the interfaz for save in the DB
162         if (!IDbNeo4jObject.class.isAssignableFrom(innerObject.getClass())) continue;
164         Node innerNode = factory.getOrCreate( FIELDNAME_KEY, ((IDbNeo4jObject)
            innerObject).getKey() );
166         logger.log(Level.INFO, "Saving_nodo,_Key:_ " + ((IDbNeo4jObject)innerObject).
            getKey() + "_nodo_ID:_ " + innerNode.getId());
168         for(Field innerF : innerObject.getClass().getDeclaredFields()){
170             if (innerF.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue;
172             Object innerPropertyValue = innerF.get(innerObject);
174             //Skip if the inner object implements iterators
174             if (innerPropertyValue == null) continue;
176             if (java.lang.Iterable.class.isAssignableFrom(innerPropertyValue.
                getClass())) continue;
178             logger.log(Level.INFO, "Traver_intern_properties_intern_node:_ " + nodo.
                getId() + "_=>_Field:_ " + innerF.getName() + "_Value:_ " +
                innerPropertyValue);
180             innerNode.setProperty(innerF.getName(), innerPropertyValue);
182             }
184             //finally, create the relationship
184             logger.log(Level.INFO, "Node_ " + nodo.getId() + "_links_ " + innerNode.
                getId() );
186             nodo.createRelationshipTo(innerNode, RelTypes.Links);
188         }
190     }
192     //If it is a pdf file, we add it to the pdfIndex
192     if (nodeToSave.getKey().endsWith(".pdf")) //TODO: improve and abstract this
        logic
192     if ( pdfIndex.get(FIELDNAME_KEY, nodeToSave.getKey()) != null )
        pdfIndex.add( nodo, FIELDNAME_KEY, nodeToSave.getKey() );
194

```

```

196         tx.success();
198     } catch (Exception e) {
199         logger.log(Level.INFO, "Neo4J_unsuccessful_transaction.\nError:␣" );
200         logger.log(Level.INFO, e);
201         e.printStackTrace();
202         throw e;
203     }
204     finally
205     {
206         tx.finish();
207     }
208 }
209
210 private static void createDb()
211 {
212
213     graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
214     registerShutdownHook( graphDb );
215
216     try ( Transaction tx = graphDb.beginTx() )
217     {
218         pdfIndex=graphDb.index().forNodes(NAME_PDF_INDEX);
219         allDbPagesIndex=graphDb.index().forNodes(NAME_PAGES_INDEX);
220         //http://docs.neo4j.org/chunked/stable/tutorials-java-embedded-unique-
221         //nodes.html#tutorials-java-embedded-unique-get-or-create
222         //https://github.com/neo4j/neo4j/issues/73
223         factory = new UniqueFactory.UniqueNodeFactory( allDbPagesIndex )
224     {
225         @Override
226         protected void initialize( Node created, Map<String, Object>
227             properties )
228         {
229             /*
230              * Implement this method to initialize the Node or Relationship created
231              * for being stored in the index.
232              * This method will be invoked exactly once per created unique entity.
233              * The created entity might be discarded if another thread creates an
234              * entity concurrently.
235              * This method will however only be invoked in the transaction that
236              * succeeds in creating the node.
237              */
238             String key = (String) properties.get( FIELDNAME_KEY);
239             if (key!=null) {
240                 created.setProperty( FIELDNAME_KEY, properties.get( FIELDNAME_KEY ) );
241                 String extension = CrawlUtilities.getExtensionFromPathName((
242                     String)properties.get( FIELDNAME_KEY ));

```

```

238         if (extension.endsWith(".pdf")) //TODO: improve and abstract
                this logic
                created.addLabel( DynamicLabel.label( "Pdf" ) );
240         else {
                created.addLabel( DynamicLabel.label( "Html" ) );
242         }
        }
244
        String author = (String) properties.get( AUTHOR_NAME);
246         if (author != null) {
                created.setProperty( AUTHOR_NAME, properties.get( AUTHOR_NAME ) );
248                 created.addLabel( DynamicLabel.label( "Author" ) );
        }
250
        String filiation = (String) properties.get( FILIATION_NAME);
252         if (filiation != null) {
                created.setProperty( FILIATION_NAME, properties.get( FILIATION_NAME ) )
                ;
254                 created.addLabel( DynamicLabel.label( "Filiation" ) );
        }
256     }
};
258
        tx.success();
260    }
}
262
264
}
266
private static void clearDb()
268 {
        try
270        {
                FileUtils.deleteRecursively( new File( DB_PATH ) );
272        }
        catch ( IOException e )
274        {
                throw new RuntimeException( e );
276        }
}
278
void removeData()
280 {
        Transaction tx = graphDb.beginTx();
282        try
        {
284                // START SNIPPET: removingData

```

```

286         // let's remove the data
        //firstNode.getSingleRelationship( RelTypes.KNOWS, Direction.OUTGOING ).
            delete();
288         //firstNode.delete();
        //secondNode.delete();
        // END SNIPPET: removingData

290         tx.success();
292     }
    finally
294     {
        tx.finish();
296     }
}

298
public void shutDown()
300 {
    System.out.println();
302     System.out.println( "Shutting down database..." );
    graphDb.shutdown();
304 }

306 private static void registerShutdownHook( final GraphDatabaseService graphDb )
    {
308         // Registers a shutdown hook for the Neo4j instance so that it
        // shuts down nicely when the VM exits (even if you "Ctrl-C" the
310         // running application).
        Runtime.getRuntime().addShutdownHook( new Thread()
312         {
            @Override
314             public void run()
                {
316                 graphDb.shutdown();
                }
            } );
318     } );
}

320
public void setLogger(org.apache.log4j.Logger oLogger) {
322     logger = oLogger;
}

324 }

```

### A.3. Crawler

### Archivo 3: PdfCrawler.java

```
1 package textLOCrawler;
import java.io.File;
3 import java.util.ArrayList;
import java.util.List;
5 import java.util.regex.Pattern;

7 import org.apache.log4j.Level;

9 import textExtraction.AlchemyapiWrapperExtractor;
import textExtraction.HTMLTextExtractor;
11 import textExtraction.PDFTextAndMetadataExtractorWrapper;
import textExtraction.ParsCitWrapper;
13 import textExtraction.RegExMailExtractor;
import textExtraction.Utilities;
15 import dbBroker.DbPage;
import dbBroker.DbPdf;
17 import dbBroker.EmbeddedNeo4j;
import edu.uci.ics.crawler4j.crawler.Page;
19 import edu.uci.ics.crawler4j.crawler.WebCrawler;
import edu.uci.ics.crawler4j.parser.BinaryParseData;
21 import edu.uci.ics.crawler4j.parser.HtmlParseData;
import edu.uci.ics.crawler4j.url.WebURL;
23 import edu.uci.ics.crawler4j.util.IO;

25
public class PdfCrawler extends WebCrawler {
27
private static final Pattern pdfPatterns = Pattern.compile(".*(\\.pdf)$");
29 private final static Pattern FILTERS = Pattern.compile(".*(\\.css|js|bmp|gif|jpe?g"
+ "|png|tiff?|mid|mp2|mp3|mp4"
31 + "|wav|avi|mov|mpeg|ram|m4v"
+ "|rm|smil|wmv|swf|wma|zip|rar|gz)$");
33
private static File storageFolder;
35 private static String[] crawlDomains;
private static String[] crawlBaseDomains;
37
public static void configure(String[] domains, String storageFolderName, boolean
onlySubpaths) {
39 storageFolder = new File(storageFolderName);
if (!storageFolder.exists()) {
41 storageFolder.mkdirs();
}
43
EmbeddedNeo4j db = EmbeddedNeo4j.getInstance(storageFolderName);
45 db.setLogger(PdfCrawler.logger);
```

```

47     logger.log(Level.INFO, "Seeds To Crawl");
48     for(String seed : domains)
49         logger.log(Level.INFO, "Seed: " + seed);
50     logger.log(Level.INFO, "Only Subpaths: " + onlySubpaths);
51
52     PdfCrawler.crawlDomains = domains;
53
54 PdfCrawler.crawlBaseDomains = new String[domains.length];
55 for(int i=0; i<domains.length ;i++) {
56     if (onlySubpaths)
57         PdfCrawler.crawlBaseDomains[i] = PdfCrawler.crawlDomains[i];
58     else
59         PdfCrawler.crawlBaseDomains[i] = CrawlUtilities.getDomainWithoutPath(PdfCrawler.
60             crawlDomains[i]);
61 }
62
63 @Override
64 public boolean shouldVisit(WebURL url) {
65     String href = url.getURL().toLowerCase();
66
67     if (FILTERS.matcher(href).matches()) { //not interested file to visit
68         logger.log(Level.INFO, "shouldVisit false: FILTERS" + href);
69         return false;
70     }
71
72     if (pdfPatterns.matcher(href).matches()) { // is a pdf file
73         logger.log(Level.INFO, "shouldVisit true: pdfPattern" + href);
74         return true;
75     }
76
77     for (String baseDomain : crawlBaseDomains) {
78         if (href.indexOf(baseDomain )>=0 ){
79             logger.log(Level.INFO, "shouldVisit true: crawlBaseDomains" + href);
80             return true;
81         }
82     }
83
84     logger.log(Level.INFO, "shouldVisit false: crawlBaseDomains" + href);
85     return false;
86 }
87
88 private boolean urlShouldBePersisted(String url) {
89
90     if (FILTERS.matcher(url).matches()) { //not interested file to persist
91         return false;
92     }
93

```

```

95     if (pdfPatterns.matcher(url).matches()) // is a pdf file
        return true;

97     for (String baseDomain : crawlBaseDomains) {
        if (url.indexOf(baseDomain )>=0 )
99         return true;
    }

101

    return false;
103 }

105 @Override
public void visit(Page page) {
107     WebURL url = page.getWebURL();

109     logger.trace("Url:␣" + url);
    logger.log(Level.INFO, "Visited␣" + url);

111

113     if ((page.getParseData() instanceof HtmlParseData)){//HTML file

        HtmlParseData htmlParseData = (HtmlParseData) page.getParseData();
115         String text = htmlParseData.getText();
        String html = htmlParseData.getHtml();
117         List<WebURL> links = new ArrayList<WebURL>();
        for(WebURL wbUrl : htmlParseData.getOutgoingUrls())//filter outgoing links
119             if (urlShouldBePersisted(wbUrl.getURL().toLowerCase())
                && !links.contains(wbUrl))
121                 links.add(wbUrl);

123         logger.log(Level.INFO,"From␣url␣" + url + "␣outgoing␣links␣(" + links.size() +
            ")␣are:␣" + links);

125         DbPage pageVisited = new DbPage();
        pageVisited.setKey(CrawlUtilities.createHashForString(url.getURL()));
127         pageVisited.setUrl(url.getURL());
        pageVisited.setHtml(html);
129         pageVisited.setPlainText(text);
        pageVisited.setDomain(url.getDomain());
131         pageVisited.setSubDomain(url.getSubDomain());
        pageVisited.setParentUrl(url.getParentUrl());
133         pageVisited.setTitle(htmlParseData.getTitle());

135         String stringCSV;
        //RegexMail extraction
137         List<String>mails = RegexMailExtractor.listMailsExtractedFromString(text);
        stringCSV = CrawlUtilities.encodeListAsCSVString(mails);
139         pageVisited.setMailList(stringCSV);

141         //HTML .bib ref extraction

```

```

143 HTMLTextExtractor textFromHtmlextractor = new HTMLTextExtractor();
    textFromHtmlextractor.processHtmlString(html,url.toString());
    stringCSV = CrawlUtilities.encodeListAsCSVString(textFromHtmlextractor.
145 getListOfBibsRef());
pageVisited.setBibRefList(stringCSV);

147 //ALchemy API extraction
AlchemyapiWrapperExtractor alchemyWrapper = new AlchemyapiWrapperExtractor();
149 try {
alchemyWrapper.processHtmlString(html,url.toString());
151 stringCSV = CrawlUtilities.encodePairListAsCSVString(alchemyWrapper.
    getEntitiesPersonConfidenceMap());
    pageVisited.setAapiAuthorList(stringCSV);

153 stringCSV = CrawlUtilities.encodePairListAsCSVString(alchemyWrapper.
    getFiliationsConfidenceMap());
155 pageVisited.setAapiFiliationList(stringCSV);

157 stringCSV = CrawlUtilities.encodePairListAsCSVString(alchemyWrapper.
    getKeywordsConfidenceMap());
    pageVisited.setAapiKeywordList(stringCSV);

159 pageVisited.setAapiLanguage(alchemyWrapper.getLanguage());

161 } catch (Exception e) {
163 e.printStackTrace();
165 }

167 for(WebURL innerUrl: links){
    DbPage pageInner = new DbPage();
169 pageInner.setKey(CrawlUtilities.createHashForString(innerUrl.getURL()));
    pageInner.setUrl(innerUrl.getURL());
171 pageInner.setDomain(innerUrl.getDomain());
    pageInner.setSubDomain(innerUrl.getSubDomain());
173 pageInner.setParentUrl(innerUrl.getParentUrl());
    pageVisited.getLinkedPages().add(pageInner);
175 }

177 try {
179 EmbeddedNeo4j.getInstance().saveObject(pageVisited);
} catch (Exception e) {
181 e.printStackTrace();
    logger.log(Level.INFO, "Error al guardar el nodo: " + e);
183 }
185 }

// PDF file

```

```

187     if ((page.getParseData() instanceof BinaryParseData)) {
189         if (!pdfPatterns.matcher(url.getURL()).matches()) return;

191     // Not interested in very small pdfs
192     if (page.getContentData().length < 10 * 1024) return;
193
194     // get a unique name for storing this pdfs
195     String hashedName = CrawlUtilities.createHashForString(url.getURL());

196
197     // store pdfs
198     //TODO: check if the file exists
199     String pdfPath = storageFolder.getAbsolutePath() + "/" + hashedName;
200     String textFromPdfPath = storageFolder.getAbsolutePath() + "/" + Utilities.
        removeExtension(hashedName) + ".txt";

201
202     IO.writeBytesToFile(page.getContentData(), pdfPath);

203
204     DbPage pageParent = new DbPage();
205     pageParent.setKey(CrawlUtilities.createHashForString(url.getParentUrl()));

206
207     DbPdf pdf = new DbPdf();
208     pdf.setKey(hashedName);
209     pdf.setSize(page.getContentData().length);
210     pdf.setUrl(url.getURL());
211     pdf.setParentDbPage(pageParent);
212     pdf.setKeyParentDbPage(pageParent.getKey());

213
214     //TODO: should be better dispatch this job in a different thread
215     PDFTextAndMetadataExtractorWrapper wrapperPdfToText= new
        PDFTextAndMetadataExtractorWrapper();
216     try {
217         wrapperPdfToText.processPdfFile(new File(pdfPath), -1, -1);
218         wrapperPdfToText.saveToFile(textFromPdfPath);
219         wrapperPdfToText.prettyPrintResult();
220         pdf.setPdfboxAuthorList(CrawlUtilities.encodeListAsCSVString(
                wrapperPdfToText.getAuthorsList()));
221         pdf.setPdfboxTitleList(CrawlUtilities.encodeListAsCSVString(
                wrapperPdfToText.getTitlesList()));
222         pdf.setPdfboxRights(wrapperPdfToText.getRights()==null?"":
                wrapperPdfToText.getRights());
223         pdf.setPdfboxSubjectsList(CrawlUtilities.encodeListAsCSVString(
                wrapperPdfToText.getSubjectsList()));
224         pdf.setPdfboxKeywordsList(CrawlUtilities.encodeListAsCSVString(
                wrapperPdfToText.getKeywordsList()));
225         pdf.setTextFragment(wrapperPdfToText.getFirstTextSegment().substring(0,
                Math.min(Parameters.getInstance().maxCharactersFragment(),
                wrapperPdfToText.getFirstTextSegment().length())));

```

```

227         if (wrapperPdfToText.getLanguage() == null) {
228             //Alchery API extraction
229             AlchemyapiWrapperExtractor alchemyWrapper = new
                AlchemyapiWrapperExtractor();
230             try {
231                 pdf.setLanguage(alchemyWrapper.getLanguageOfText(wrapperPdfToText.
                    getFirstTextSegment()));
232             } catch (Exception e) {
233                 e.printStackTrace();
234             }
235             } else {
236                 pdf.setLanguage(wrapperPdfToText.getLanguage());
237             }
238
239             //RegexMail extraction
                List<String>mails = RegExMailExtractor.listMailsExtractedFromString(
                    wrapperPdfToText.getFirstTextSegment());
240             pdf.setMailList(CrawlUtilities.encodeListAsCSVString(mails));
241
242     } catch (Exception e) {
243         System.err.println( "Error: extracting text for document:" + pdfPath );
244         e.printStackTrace();
245         return;
246     }
247
248 //TODO: add this limit to parameters files
249 if (page.getContentData().length < 150 *1024* 1024) { //ParsCit fails with very
    large files
250     ParsCitWrapper parsCitWrapper = new ParsCitWrapper();
251     parsCitWrapper.parseFile(new File(textFromPdfPath), storageFolder);
252     parsCitWrapper.saveToFile(Utilities.removeExtension(textFromPdfPath) + ".
        parsCitOut" );
253     parsCitWrapper.prettyPrintResult();
254     pdf.setParsCitAbstractList(CrawlUtilities.encodePairListAsCSVString(
        parsCitWrapper.getPotentialAbstract()));
255     pdf.setParsCitAuthorList(CrawlUtilities.encodePairListAsCSVString(
        parsCitWrapper.getPotentialAuthorsList()));
256     pdf.setParsCitFiliationsList(CrawlUtilities.encodePairListAsCSVString(
        parsCitWrapper.getFiliationsList()));
257     pdf.setParsCitMailList(CrawlUtilities.encodeListAsCSVString(parsCitWrapper.
        getMailList()));
258     pdf.setParsCitTitleList(CrawlUtilities.encodePairListAsCSVString(
        parsCitWrapper.getPotentialTitle()));
259     pdf.setParsCitValidCitesList(CrawlUtilities.encodeListAsCSVString(
        parsCitWrapper.getValidCitesList()));
260 }
261
262 logger.log(Level.INFO, "pdf_persisted_url:" + url.getURL() + "_parent:" + url.
    getParentUrl());

```

```

265     try {
266         EmbeddedNeo4j.getInstance().saveObject(pdf);
267     } catch (Exception e) {
268         e.printStackTrace();
269         logger.log(Level.INFO, "Error_saving_node:_" + e);
270         logger.log(Level.INFO, "Error_saving_node:_" + e.getMessage());
271         logger.log(Level.INFO, "Error_saving_node:_" + e.getStackTrace());
272     }
273
274     logger.trace("saved_node:_" + url);
275 }
276 }
277 }

```

## A.4. Post-Procesador

Archivo 4: DBTraverseHelper.java

```

1  package dbBroker;
2
3  import jaligner.DamerauLevenshteinDistanceCalculator;
4  import jaligner.JAlignerHelper;
5
6  import java.util.ArrayList;
7  import java.util.Iterator;
8  import java.util.List;
9  import java.util.Map.Entry;
10 import java.util.regex.Matcher;
11
12 import org.apache.log4j.Level;
13 import org.neo4j.graphdb.Direction;
14 import org.neo4j.graphdb.Node;
15 import org.neo4j.graphdb.ReturnableEvaluator;
16 import org.neo4j.graphdb.StopEvaluator;
17 import org.neo4j.graphdb.Transaction;
18 import org.neo4j.graphdb.Traverser;
19 import org.neo4j.graphdb.index.Index;
20 import org.neo4j.graphdb.index.IndexHits;
21
22 import textLOCrawler.CrawlUtilities;
23 import textLOCrawler.Parameters;
24 import textLOCrawler.TLOLabel;
25 import dbBroker.EmbeddedNeo4j.RelTypes;
26
27

```

```

29 public class DBTraverseHelper {
31     private static volatile DBTraverseHelper _instance = null;
33     private static Index<Node> authorsIndex;
34     private static final String AUTHORS_INDEX = "authorsIndex";
35     private static Index<Node> filiationIndex;
36     private static final String FILIATION_INDEX = "filiationIndex";
37
38     public static DBTraverseHelper getInstance() {
39         if(_instance == null) {
40             _instance = new DBTraverseHelper();
41         }
42         return _instance;
43     }
44
45     public void traversePdfFilesInDB() {
46         Transaction tx = EmbeddedNeo4j.graphDb.beginTx();
47         try
48         {
49             Parameters parameters = Parameters.getInstance();
50
51             authorsIndex=EmbeddedNeo4j.graphDb.index().forNodes(AUTHORS_INDEX);
52             filiationIndex=EmbeddedNeo4j.graphDb.index().forNodes(FILIATION_INDEX);
53
54             //Graph traversal parameters
55             int mailDepth = parameters.maxDepthForKey(Parameters.KEY_MAX_DEPTH_EMAILLIST
56                 );
57             int authorDepth = parameters.maxDepthForKey(Parameters.
58                 KEY_MAX_DEPTH_POTENTIALAUTHORLIST);
59             int filiationDepth = parameters.maxDepthForKey(Parameters.
60                 KEY_MAX_DEPTH_POTENTIALFILIATIONLIST);
61             int maxOfAllDepth = Math.max(mailDepth,Math.max(authorDepth,filiationDepth)
62                 );
63             float authorConfidenceLowerBound = parameters.confidenceLowerBoundForKey(
64                 Parameters.KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST);
65             float filiationConfidenceLowerBound = parameters.confidenceLowerBoundForKey(
66                 Parameters.KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIATIONLIST);
67
68             float authorSimilLowerBound = parameters.confidenceLowerBoundForKey(
69                 Parameters.KEY_SIMILARITY_LOWER_BOUND_AUTHOR_MATCHING);
70             float filiationSimilLowerBound = parameters.confidenceLowerBoundForKey(
71                 Parameters.KEY_SIMILARITY_LOWER_BOUND_FILIATION_MATCHING);
72
73             //first travel
74             for ( Node pdfNode : EmbeddedNeo4j.pdfIndex.query( EmbeddedNeo4j.
75                 FIELDNAME_KEY, "*" ) )
76                 {

```

```

69     /*Graph traversal
70     *
71     */
72     Traverser traverse = pdfNode.traverse(Traverser.Order.BREADTH_FIRST,
73         StopEvaluator.END_OF_GRAPH,
74         ReturnableEvaluator.ALL_BUT_START_NODE,
75         EmbeddedNeo4j.RelTypes.Links,
76         Direction.BOTH);
77     String RelMailList = ""; //list of mails separated by ';' and "(depth)" of
78         the relationship, i.e. (2) means 2 degrees of distance
79     String RelMailList_justPairs = ""; //list of mails separated by ';' (used to
80         duplicated values)
81     String UnionMailList = CrawlUtilities.filterDuplicatedValuesSimpleList(
82         pdfNode.getProperty( "ParsCitMailList", "").toString(), pdfNode.
83         getProperty( "MailList", "").toString());
84     String RelAuthorsList = ""; //list of "entity person=confidence value"
85         separated by ';' and "(depth)" of the relationship, i.e. (2) means 2
86         degrees of distance
87         String RelAuthorsList_justPairs = ""; //list of "entity person=
88             confidence value" separated by ';' (used to duplicated values)
89         String RelFiliationsList = ""; //list of "entity organization=confidence
90             value" separated by ';' and "(depth)" of the relationship, i.e. (2)
91             means 2 degrees of distance
92         String RelFiliationsList_justPairs = ""; //list of "entity person=
93             confidence value" separated by ';' (used to duplicated values)
94         //Start traverse
95     for(Node page:traverse) {
96         int currentDepth = traverse.currentPosition().depth();
97         if (currentDepth > maxOfAllDepth )
98             break; //BREADTH_FIRST => other nodes from now on are deeper
99
100         if ( ((String)page.getProperty( "Url", "")).endsWith(".pdf")) //
101             TODO: improve and abstract this logic
102             continue; //we don't examine pdf relatives nodes
103
104     if (currentDepth <= mailDepth) { //add mails
105         String nodeMailList = page.getProperty("MailList", "").toString();
106         if ( nodeMailList != null && nodeMailList.length() > 0 ) {
107             String newList = nodeMailList.substring(0, Math.max(nodeMailList.length
108                 () - 1, 0));
109             String newValues = CrawlUtilities.filterDuplicatedValuesSimpleList(
110                 newList, RelMailList_justPairs);
111             if ( newValues != null
112                 && newValues.length() > 0 ) {
113                 RelMailList_justPairs += (RelMailList_justPairs.length() > 0 ? ";" : "") +
114                     newValues;
115                 RelMailList += newValues + "□(" + (new Integer(currentDepth))
116                     .toString() + ");";
117             }
118         }
119     }

```

```

String newValuesOfUnion = CrawlUtilities.
    filterDuplicatedValuesSimpleList(RelMailList_justPairs, UnionMailList)
    ;
103  if ( newValuesOfUnion!=null
      && newValuesOfUnion.length()>0) {
105      UnionMailList += (UnionMailList.length()>0?";":"" ) + newValuesOfUnion;
      }
107  }
    }
109
111  if (currentDepth <= authorDepth) { //add potential authors
String nodeAuthorList = page.getProperty("AapiAuthorList", "").toString();
113  if ( nodeAuthorList!=null && nodeAuthorList.length()>0 ) {
String newValues = CrawlUtilities.filterStringPairList(nodeAuthorList.
    substring(0,Math.max(nodeAuthorList.length()-1,0)),
    authorConfidenceLowerBound, RelAuthorsList_justPairs);
115  if ( newValues!= null
      && newValues.length()>0 ) {
RelAuthorsList_justPairs += (RelAuthorsList_justPairs.length()>0 ? ";" : ""
117  ") + newValues;
RelAuthorsList += newValues + "□(" + (new Integer(currentDepth)).
    toString() + ");";
    }
119  }
    }
121
123  if (currentDepth <= filiationDepth) { //add potential filiations
String nodeFiliationList = page.getProperty("AapiFiliationList", "").
    toString();
125  if ( nodeFiliationList!=null && nodeFiliationList.length()>0 ) {
String newValues = CrawlUtilities.filterStringPairList(
    nodeFiliationList.substring(0,Math.max(nodeFiliationList.length()
    -1,0)), filiationConfidenceLowerBound, RelFiliationsList_justPairs);
127  if ( newValues!= null
      && newValues.length()>0 ) {
RelFiliationsList_justPairs += (RelFiliationsList_justPairs.length()
    >0 ? ";" : "") + newValues;
129  RelFiliationsList += newValues + "□(" + (new Integer(currentDepth)).
    toString() + ");";
    }
131  }
    }
133  }
    }
135  if (RelMailList.length()>0)
pdfNode.setProperty("RelMailList", RelMailList);
137  if (UnionMailList.length()>0)
pdfNode.setProperty("UnionMailsList", UnionMailList);
139  if (RelAuthorsList.length()>0)
pdfNode.setProperty("RelAuthorsList", RelAuthorsList);

```

```

141         if (RelFiliationsList.length()>0)
            pdfNode.setProperty("RelFiliationsList", RelFiliationsList);
143
144         /*
145          * Look if the related entities appear on the first fragment of text
146          * */
String textFragment = pdfNode.getProperty("TextFragment","").toString();
147
148         List<Entry<String,String>> listEntitiesAuthorMatched = JAlignerHelper.
            filterStringsInText(textFragment, CrawlUtilities.getKeysFromPairList(
                RelAuthorsList_justPairs),authorSimillLowerBound);
149         if (listEntitiesAuthorMatched.size()>0) {
            String strMatched = CrawlUtilities.encodePairListAsCSVString(
                listEntitiesAuthorMatched);
151         pdfNode.setProperty("RelAuthorsListMatchedInFirstTextFragmentWithConfidence",
            strMatched);
            String list =CrawlUtilities.encodeListAsCSVString(CrawlUtilities.
                getKeysFromPairList(strMatched));
153         pdfNode.setProperty("RelAuthorsListMatchedInFirstTextFragment", list);
        }
155
List<Entry<String,String>> listEntitiesFiliationMatched = JAlignerHelper.
    filterStringsInText(textFragment, CrawlUtilities.getKeysFromPairList(
        RelFiliationsList_justPairs),filiationSimillLowerBound);
157         if (listEntitiesFiliationMatched.size()>0) {
            String strMatched = CrawlUtilities.encodePairListAsCSVString(
                listEntitiesFiliationMatched);
159         pdfNode.setProperty("RelFiliationsListMatchedInFirstTextFragmentWithConfidence",
            strMatched);
            String list =CrawlUtilities.encodeListAsCSVString(CrawlUtilities.
                getKeysFromPairList(strMatched));
161         pdfNode.setProperty("RelFiliationsListMatchedInFirstTextFragment", list);
        }
163
this.createAuthorsAndFiliationNodes(pdfNode);//create authors and filiations
    nodes
165
this.performLabelingJob(pdfNode);
167     }
169
    EmbeddedNeo4j.logger.log(Level.INFO, "\n*****Second旅行pdfNode*****");
    //second travel
171     for ( Node pdfNode : EmbeddedNeo4j.pdfIndex.query( EmbeddedNeo4j.
        FIELDNAME_KEY, "*" ) )
        {
173         this.findAuthorsFromOtherDocuments(pdfNode);
        this.findFiliationsFromOtherDocuments(pdfNode);
175

```

```

177 EmbeddedNeo4j.logger.log(Level.INFO,
178     "\n\n\n--{\n\nPdf_file_with_URL:_" + pdfNode.getProperty( "Url", "" )
179 + "\n\nKey:_" + pdfNode.getProperty( EmbeddedNeo4j.FIELDNAME_KEY, "" )
180 + "\n\n" + pdfNode.getProperty( "TextFragment", "" ).toString()
181 + "\n\nLanguage_" + pdfNode.getProperty( "Language", "" )
182     + "\n\n---LABELING_FIELDS "
183 + "\n\nCategory:_" + pdfNode.getProperty( "category", "" )
184 + "\n\nSubcategory:_" + pdfNode.getProperty( "subCategory", "" )
185
186 + "\n\n---MAIL "
187 + "\n\nPDF_first_text_fragment_Mail_list:_" + pdfNode.getProperty( "MailList", "" )
188     )
189 + "\n\nRel_Mail_list:_" + pdfNode.getProperty( "RelMailList", "" )
190     + "\n\nParsCit_Mail_list:_" + pdfNode.getProperty( "ParsCitMailList", "" )
191     + "\n\nUnion_Mail_list:_" + pdfNode.getProperty( "UnionMailsList", "" )
192
193 + "\n\n---RIGHTS "
194     + "\n\nPdfbox_Rights:_" + pdfNode.getProperty( "PdfboxRights", "" )
195
196 + "\n\n---TITLE "
197     + "\n\nPdfbox_Title:_" + pdfNode.getProperty( "PdfboxTitleList", "" )
198     + "\n\nParsCit_Title:_" + pdfNode.getProperty( "ParsCitTitleList", "" )
199
200 + "\n\n---AUTHOR_FIELDS "
201     + "\n\nPdfbox_Author_list:_" + pdfNode.getProperty( "PdfboxAuthorList", "" )
202     + "\n\nRelatives_nodes'_Author_list:_" + pdfNode.getProperty( "
203     RelAuthorsList", "" )
204     + "\n\nParsCit_Author_list:_" + pdfNode.getProperty( "ParsCitAuthorList", "" )
205     + "\n\nAuthor_from_AAPI_Matched_in_First_Text_Fragment_list:_" + pdfNode.
206     getProperty( "RelAuthorsListMatchedInFirstTextFragmentWithConfidence", "" )
207     + "\n\nAuthor_From_Other_Docs_Matched_in_First_Text_Fragment_list:_" +
208     pdfNode.getProperty( "RelAuthorsFromOthersDocMatchedWithConfidence", "" )
209     + "\n\nTotal_Author_list:_" + pdfNode.getProperty( "TotalAuthorsList", "" )
210     + "\n\nUnion_Filtering_Dups_Author_list:_" + pdfNode.getProperty( "
211     UnionAuthorsList", "" )
212
213 + "\n\n---FILIACTION_FIELDS "
214     + "\n\nParsCit_Filiation_list:_" + pdfNode.getProperty( "
215     ParsCitFiliationsList", "" )
216     + "\n\nRelatives_nodes'_Filiation_list:_" + pdfNode.getProperty( "
217     RelFiliationsList", "" )
218     + "\n\nFiliation_from_AAPI_Matched_in_First_Text_Fragment_list:_" + pdfNode
219     .getProperty( "RelFiliationsListMatchedInFirstTextFragmentWithConfidence"
220     , "" )
221     + "\n\nFiliation_From_Other_Docs_Matched_in_First_Text_Fragment_list:_" +
222     pdfNode.getProperty( "RelFiliationsFromOthersDocMatchedWithConfidence", "" )
223     )

```

```

215         + "\n\nTotal_Filiation_list:" + pdfNode.getProperty("TotalFiliationsList",
216             "")
217         + "\n\nUnion_Filtering_Dups_Filiation_list:" + pdfNode.getProperty("
218             UnionFiliationsList", "")
219
220     + "\n\n--}\n\n\n\n\n\n");
221     }
222
223     tx.success();
224     }
225     catch(Exception ex){
226         ex.printStackTrace();
227         ex.printStackTrace();
228         ex.printStackTrace();
229         throw ex;
230     }
231     finally
232     {
233         tx.finish();
234     }
235 }
236 /*
237 * Look at the parsCit 'filiations', 'authors', the '
238 * RelFiliationsListMatchedInFirstTextFragment' and '
239 * RelAuthorsListMatchedInFirstTextFragment'
240 * properties, and make new nodes filiations/authors (if needed) setting the
241 * bidirectional rel 'conecction' them and
242 * outgoing relationships 'hasFiliation' and 'hasAuthor' from the pdfNode.
243 * */
244 private void createAuthorsAndFiliationNodes(Node pdfNode) {
245     try {
246         List<Node>authorNodes = new ArrayList<Node>();
247         List<Node>filiationNodes = new ArrayList<Node>();
248
249         String listValuesStr = pdfNode.getProperty("ParsCitAuthorList", "").toString();
250         List <String> values = CrawlUtilities.getKeysFromPairList(listValuesStr.
251             substring(0,Math.max(listValuesStr.length()-1,0)));
252         authorNodes.addAll(this.getOrCreateEntityNodes(values, authorsIndex, EmbeddedNeo4j
253             .AUTHOR_NAME, pdfNode, RelTypes.HasAuthor));
254
255         listValuesStr = pdfNode.getProperty("RelAuthorsListMatchedInFirstTextFragment",
256             "").toString();
257         values = CrawlUtilities.decodeListFromCSVString(listValuesStr);
258         authorNodes.addAll(this.getOrCreateEntityNodes(values, authorsIndex, EmbeddedNeo4j
259             .AUTHOR_NAME, pdfNode, RelTypes.HasAuthor));
260
261         String filiationAuthorList = pdfNode.getProperty("ParsCitFiliationsList", "").
262             toString();

```

```

253 values = CrawlUtilities.getKeysFromPairList(filiationAuthorList.substring(0, Math
    .max(filiationAuthorList.length()-1,0)));
filiationNodes.addAll(this.getOrCreateEntityNodes(values, filiationIndex,
    EmbeddedNeo4j.FILIATION_NAME, pdfNode, RelTypes.HasFiliation));
255
listValuesStr = pdfNode.getProperty("RelFiliationsListMatchedInFirstTextFragment
    ", "").toString();
257 values = CrawlUtilities.decodeListFromCSVString(listValuesStr);
filiationNodes.addAll(this.getOrCreateEntityNodes(values, filiationIndex,
    EmbeddedNeo4j.FILIATION_NAME, pdfNode, RelTypes.HasFiliation));
259 /*
    * make relationships
    * */
261 Iterator<Node>authorIter = authorNodes.iterator();
263 while(authorIter.hasNext()) {
    Node author = authorIter.next();
265 Iterator<Node>filIter = filiationNodes.iterator();
    while(filIter.hasNext()) {
267 Node fil = filIter.next();
        fil.createRelationshipTo(author, RelTypes.HasConnection);
269     }
    }
271 } catch (Exception e) {
    //todo: handle it!
273 }
    }
275
private List<Node> getOrCreateEntityNodes(List <String> values, Index<Node>
    indexLookup, String propertyName, Node pdfNode, RelTypes relationshipName ) {
277 List<Node>nodes = new ArrayList<Node>();
    if (values.size()<=0) return nodes;
279 Iterator<String> iter = values.iterator();
while(iter.hasNext()){
281 String text = iter.next().toLowerCase().trim();//avoid different nodes with same
    string but different case
283 if (text.length()<=0) continue;
    IndexHits<Node>matches = indexLookup.query( propertyName, text);
    Node created;
285 if (matches.size() > 0) {
        created = matches.getSingle();
287 nodes.add(created);
    } else {
289 created = EmbeddedNeo4j.factory.getOrCreate( propertyName, text);
        created.setProperty(propertyName, text);
291 nodes.add(created);
        indexLookup.add( created, propertyName, text);//add to the respective index
293     }
    pdfNode.createRelationshipTo(created, relationshipName);
295 }
}

```

```

297     return nodes;
    }

299     private void findAuthorsFromOtherDocuments(Node pdfNode) {
List<String> listToMatch = new ArrayList<String>();

301     String listValuesStr = pdfNode.getProperty("ParsCitAuthorList","").toString();
303     List <String> existingValues = CrawlUtilities.getKeysFromPairList(listValuesStr.
        substring(0,Math.max(listValuesStr.length()-1,0)));
listValuesStr = pdfNode.getProperty("RelAuthorsListMatchedInFirstTextFragment","").
    toString();
305     CrawlUtilities.unionToFirstListIgnoringCase(existingValues,CrawlUtilities.
        decodeListFromCSVString(listValuesStr));

307     for ( Node node : authorsIndex.query( EmbeddedNeo4j.AUTHOR_NAME, "*" ) ){
        String authorName = node.getProperty(EmbeddedNeo4j.AUTHOR_NAME,"").toString();
309         if (!CrawlUtilities.listStringContainsStringIgnoringCase(existingValues,
            authorName))
            listToMatch.add(authorName);
311     }

313     String textFragment = pdfNode.getProperty("TextFragment","").toString();
List<Entry<String,String>> listMatched = JAlignerHelper.filterStringsInText(
        textFragment, listToMatch,
315     Parameters.getInstance().confidenceLowerBoundForKey(Parameters.
        KEY_SIMILARITY_LOWER_BOUND_AUTHOR_MATCHING));
String strMatched,strMatchedWithConfidence;
317     List<String> entitiesAuthorsMatched;
if (listMatched.size()>0) {
319     strMatchedWithConfidence = CrawlUtilities.encodePairListAsCSVString(listMatched);
entitiesAuthorsMatched = CrawlUtilities.getKeysFromPairList(
        strMatchedWithConfidence);
321     strMatched = CrawlUtilities.encodeListAsCSVString(entitiesAuthorsMatched);
    }
323     else {
        strMatched = "";
325     strMatchedWithConfidence = "";
        entitiesAuthorsMatched = null;
327     }
pdfNode.setProperty("RelAuthorsFromOthersDocMatched", strMatched);
329     pdfNode.setProperty("RelAuthorsFromOthersDocMatchedWithConfidence",
        strMatchedWithConfidence);

331     //Set Union property
CrawlUtilities.unionToFirstListIgnoringCase(existingValues,entitiesAuthorsMatched);
333     String unionStr = CrawlUtilities.encodeListAsCSVString(existingValues);
pdfNode.setProperty("TotalAuthorsList", unionStr);

335     //Filter duplicated and calculate the final union

```

```

337 float treeHold = Parameters.getInstance().confidenceLowerBoundForKey(Parameters.
    KEY_SIMILARITY_LOWER_BOUND_AUTHOR_REMOVEDUPS);
List<String> unionFilteringDups = new ArrayList<String>();
339 Iterator<String> iter = existingValues.iterator();
    while(iter.hasNext()) {
341 String newValue = iter.next().trim();
Iterator<String> itIntern = unionFilteringDups.iterator();
343 List<String> unionFilteringDups_CopyList = new ArrayList<String>();
unionFilteringDups_CopyList.addAll(unionFilteringDups);
345 boolean didReplaceValue = false;
    while(itIntern.hasNext()) {
347 String oldValue = itIntern.next();
    if (JAlignerHelper.similarityBetweenStrings(newValue,oldValue)>=treeHold) {
349 String shortest,largest;
    if (newValue.length() > oldValue.length()) {
351 shortest = oldValue;
    largest = newValue;
353 } else {
    largest = oldValue;
355 shortest = newValue;
    }
357 List<String>toFindMatch = new ArrayList<String>();
toFindMatch.add(shortest.toLowerCase());
359 if (CrawlUtilities.findMatch(largest.toLowerCase(),toFindMatch).size()>0) {
    //
    // shortest included in largest, so we add the shortest (may the case of
    // garbage surround a name or title indicator like 'Phd')
    if (largest.equalsIgnoreCase(oldValue)) {
361 unionFilteringDups_CopyList.remove(largest);
    if (!unionFilteringDups_CopyList.contains(shortest))
363 unionFilteringDups_CopyList.add(shortest);
    }
365 } else {
    //otherwise, we include the largest
    if (shortest.equalsIgnoreCase(oldValue)) {
367 unionFilteringDups_CopyList.remove(shortest);
    if (!unionFilteringDups_CopyList.contains(largest))
369 unionFilteringDups_CopyList.add(largest);
    }
371 }
    didReplaceValue = true;
373 }
    }
375 unionFilteringDups = unionFilteringDups_CopyList;
if (!didReplaceValue)
377 unionFilteringDups.add(newValue);
    }
379 pdfNode.setProperty("UnionAuthorsList", CrawlUtilities.encodeListAsCSVString(
    unionFilteringDups));
    }
381

```

```

383     private void findFiliationsFromOtherDocuments(Node pdfNode) {
List<String> listToMatch = new ArrayList<String>();

385     String listValuesStr = pdfNode.getProperty("ParsCitFiliationsList","").toString();
List <String> existingValues = CrawlUtilities.getKeysFromPairList(listValuesStr.
    substring(0,Math.max(listValuesStr.length()-1,0)));
387     listValuesStr = pdfNode.getProperty("RelFiliationsListMatchedInFirstTextFragment",
    "").toString();
CrawlUtilities.unionToFirstListIgnoringCase(existingValues,CrawlUtilities.
    decodeListFromCSVString(listValuesStr));
389     for ( Node node : filiationIndex.query( EmbeddedNeo4j.FILIATION_NAME, "*" ) ){
    String name = node.getProperty(EmbeddedNeo4j.FILIATION_NAME,"").toString();
391     if (!CrawlUtilities.listStringContainsStringIgnoringCase(existingValues, name))
        //filter duplicated in the union, not before
        listToMatch.add(name);
393     }
    String textFragment = pdfNode.getProperty("TextFragment","").toString();
395

    List<Entry<String,String>> listMatched = JAlignerHelper.filterStringsInText(
        textFragment, listToMatch,Parameters.getInstance().confidenceLowerBoundForKey(
            Parameters.KEY_SIMILARITY_LOWER_BOUND_FILIATION_MATCHING));
397     String strMatched,strMatchedWithConfidence;
List<String> entitiesMatched;
399     if (listMatched.size()>0) {
        strMatchedWithConfidence = CrawlUtilities.encodePairListAsCSVString(listMatched);
401     entitiesMatched = CrawlUtilities.getKeysFromPairList(strMatchedWithConfidence);
        strMatched = CrawlUtilities.encodeListAsCSVString(entitiesMatched);
403     }
    else {
405     strMatched = "";
        strMatchedWithConfidence = "";
407     entitiesMatched = null;
    }
409     pdfNode.setProperty("RelFiliationsFromOthersDocMatched", strMatched);
    pdfNode.setProperty("RelFiliationsFromOthersDocMatchedWithConfidence",
        strMatchedWithConfidence);
411

    //Set Union property
413     CrawlUtilities.unionToFirstListIgnoringCase(existingValues,entitiesMatched);
    String unionStr = CrawlUtilities.encodeListAsCSVString(existingValues);
415     pdfNode.setProperty("TotalFiliationsList", unionStr);

    //Filter duplicated and calculate the final union
417     float treeHold = Parameters.getInstance().confidenceLowerBoundForKey(Parameters.
        KEY_SIMILARITY_LOWER_BOUND_FILIATION_REMOVEDUPS);
List<String> unionFilteringDups = new ArrayList<String>();
419     Iterator<String> iter = existingValues.iterator();
    DamerauLevenshteinDistanceCalculator dld;
421     while(iter.hasNext()) {

```

```

423     String newValue = iter.next().trim();
Iterator<String> itIntern = unionFilteringDups.iterator();
425 List<String> unionFilteringDups_CopyList = new ArrayList<String>();
unionFilteringDups_CopyList.addAll(unionFilteringDups);
427 boolean didReplaceValue = false;
    while(itIntern.hasNext()) {
429         String oldValue = itIntern.next();
        d1D = new DamerauLevenshteinDistanceCalculator(newValue,oldValue);
431         if (d1D.getOSASimilarityMetric() >= treeHold) {
            String shortest,largest;
433             if (newValue.length() > oldValue.length()) {
                shortest = oldValue;
435                 largest = newValue;
            } else {
437                 largest = oldValue;
                shortest = newValue;
439             }
            if (shortest.equalsIgnoreCase(oldValue)) {
                unionFilteringDups_CopyList.remove(shortest);
441                 if (!unionFilteringDups_CopyList.contains(largest))
                    unionFilteringDups_CopyList.add(largest);
443             }
            didReplaceValue = true;
445         }
    }
unionFilteringDups = unionFilteringDups_CopyList;
449 if (!didReplaceValue)
    unionFilteringDups.add(newValue);
451 }
pdfNode.setProperty("UnionFiliationsList", CrawlUtilities.encodeListAsCSVString(
    unionFilteringDups));
453 }

455 /* Labeling: here we just look at the html parent PDFFile's text
*
457 */
private void performLabelingJob(Node pdfNode) {
459     Iterator<TLQLabel> iter = Parameters.getInstance().labelingList().iterator();
        //Labeling parameters
        String category = "";
461         String subCategory = "";
        for ( Node parentNode : EmbeddedNeo4j.allDbPagesIndex.query( EmbeddedNeo4j.
            FIELDNAME_KEY, pdfNode.getProperty("KeyParentDbPage",""))) { //this query
            returns at most one entry
463             String previousSearchCategory = (String) parentNode.getProperty("
                categoryMatched", "");
            String previousSearchSubCategory = (String) parentNode.getProperty("
                subcategoryMatched", "");
465             if (previousSearchCategory!=null && previousSearchCategory.length()>0) {

```

```

category = previousSearchCategory.equalsIgnoreCase("none")?"":
    previousSearchCategory;
467 subCategory = previousSearchSubCategory.equalsIgnoreCase("none")?"":
    previousSearchSubCategory;
} else {
469 String strToScrap1 = (String) parentNode.getProperty("PlainText","");
String strToScrap2 = (String) pdfNode.getProperty("TextFragment","");
471 while(iter.hasNext()) {
    TLOLabel label = iter.next();
473 Matcher m = label.getRegExpPattern().matcher(strToScrap1);
    boolean findSuccess = false;
475 if (m.find())
        findSuccess = true;
477 else {
        m = label.getRegExpPattern().matcher(strToScrap2);
479 if (m.find())
            findSuccess = true;
481 }
    if (findSuccess) {
483 String newValue = CrawlUtilities.filterDuplicatedValuesSimpleList(
        label.category,category);
        if ( newValue!=null
485 && newValue.length()>0) {
            category+=newValue + ";";
487 }
        subCategory+=(label.subcategory!=null && label.subcategory.length()>0)
            ?label.subcategory + "":"";
489     }
    }
491 //persist the match to avoid search again on eventually brother node
    parentNode.setProperty("categoryMatched", category.length()>0?category:"
        none");
493 parentNode.setProperty("subcategoryMatched", subCategory.length()>0?
        subCategory:"none");
    }
495 //finally set the category and sub-category fields
    pdfNode.setProperty("category", (category.length()>0)?category:"");
497 pdfNode.setProperty("subCategory", (subCategory.length()>0)?subCategory:"");
    }
499 }
}

```

## A.5. Configuración caso de prueba 1

Archivo 5: Paramameters.xml

```

2 <?xml version="1.0" encoding="UTF-8"?>
  <general>
4     <crawlerParameters>
      <SEEDS>http://www.fceia.unr.edu.ar/~acasali/</SEEDS>
6     <ONLY_SUBPATHS>true</ONLY_SUBPATHS>
      </crawlerParameters>
8
      <parsCit_Home>
10     /home/Documents/ParsCit-master/bin
      </parsCit_Home>
12
      <neo4j_Home>
14     /home/Documents/neo4j-community-2.0.0
      </neo4j_Home>
16
      <alchemyApiKey>
18     <!-- COMPLETE -->
      </alchemyApiKey>
20
      <postCrawlGraphTraversal>
22     <MAX_DEPTH_EMAILLIST>5</MAX_DEPTH_EMAILLIST>
      <KEY_MAX_DEPTH_POTENTIALAUTHORLIST>3</KEY_MAX_DEPTH_POTENTIALAUTHORLIST>
24     <KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST>0.0</
        KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST>
      <MAX_POTENTIALFILIIATIONLIST>3</MAX_POTENTIALFILIIATIONLIST>
26     <KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIIATIONLIST>0.0</
        KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIIATIONLIST>
      </postCrawlGraphTraversal>
28
      <stopWords_authors>
30     el,la,las,del,lo,los,del,rosario,práctica,et al,tla,vocabulario
      </stopWords_authors>
32
      <stopWords_filiations>
34     el,la,las,del,lo,los,del,csp,abril
      </stopWords_filiations>
36
      <postCrawlLabeling>
38     <label>
        <category>Material Educativo</category>
40     <subcategory>Práctica</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
42     <matchingWords>práctica,prácticas,ejercicio,ejercicios,ejercitación,práctico</
        matchingWords>
        <shouldApplyParsCitRule>>false</shouldApplyParsCitRule>
44     </label>
      <label>
46     <category>Material Educativo</category>

```

```

48     <subcategory>Apunte de Clase</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
        <matchingWords>apunte , apuntes , notas , material , clase , capítulo , capítulos</
            matchingWords >
50     <shouldApplyParsCitRule>false</shouldApplyParsCitRule>
</label>
52 <label>
        <category>Tesis</category>
54     <subcategory>null</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
56     <matchingWords>tesis , tesina</matchingWords>
        <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
58 </label>
<label>
60     <category>Publicación</category>
        <subcategory>null</subcategory>
62     <maxCharactersInDoc>1500</maxCharactersInDoc>
        <matchingWords>publicación , publicaciones</matchingWords>
64     <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
</label>
66 </postCrawlLabeling>
</general>

```

## A.6. Configuración caso de prueba 2

Archivo 6: Paramameters.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
<general>
3
    <crawlerParameters>
5     <SEEDS>http://www.fceia.unr.edu.ar/ingsoft/</SEEDS>
<ONLY_SUBPATHS>true</ONLY_SUBPATHS>
7     </crawlerParameters>
9
    <parsCit_Home>
        /home/Documents/ParsCit-master/bin
11    </parsCit_Home>
13
    <neo4j_Home>
        /home/Documents/neo4j-community-2.0.0
15    </neo4j_Home>
17
    <alchemyApiKey>
        <!-- COMPLETE -->
19    </alchemyApiKey>

```

```

21 <postCrawlGraphTraversal >
22 <max_depth_emaillist >5</max_depth_emaillist >
23 <key_max_depth_potentialauthorlist >3</key_max_depth_potentialauthorlist >
24 <key_confidence_lower_bound_potentialauthorlist_aapi >0.0</
    key_confidence_lower_bound_potentialauthorlist_aapi >
25 <key_similarity_lower_bound_author_matching >0.7</
    key_similarity_lower_bound_author_matching >
26 <key_similarity_lower_bound_author_removeDups >0.55</
    key_similarity_lower_bound_author_removeDups >
27
28 <max_potentialfiliationlist >3</max_potentialfiliationlist >
29 <key_confidence_lower_bound_potentialfiliationlist_aapi >0.0</
    key_confidence_lower_bound_potentialfiliationlist_aapi >
30 <key_similarity_lower_bound_filiation_matching >0.65</
    key_similarity_lower_bound_filiation_matching >
31 <key_similarity_lower_bound_filiation_removeDups >0.6</
    key_similarity_lower_bound_filiation_removeDups >
32
33 </postCrawlGraphTraversal >
34
35 <stopWords_authors >
36 el , la , las , del , lo , los , ese , esa , del , rosario , mit , tesis , t sis , unidad , p ginas , pr ctica ,
    practica , pr cticas , et al , clements , vocabulario , for
37 </stopWords_authors >
38
39 <stopWords_filiations >
40 el , la , las , del , lo , los , ese , esa , del , csp , abril , mit , tesis , t sis , unidad , p ginas , pr ctica ,
    practica , pr cticas , et al , for , vocabulario
41 </stopWords_filiations >
42
43 <postCrawlLabeling >
44 <label >
45 <category >Material Educativo</category >
46 <subcategory >Pr ctica</subcategory >
47 <maxCharactersInDoc >1500</maxCharactersInDoc >
48 <matchingWords >pr ctica , pr cticas , ejercicio , ejercicios , ejercitaci n , pr ctico</
    matchingWords >
49 <shouldApplyParsCitRule >false</shouldApplyParsCitRule >
50 </label >
51 <label >
52 <category >Material Educativo</category >
53 <subcategory >Apunte de Clase</subcategory >
54 <maxCharactersInDoc >1500</maxCharactersInDoc >
55 <matchingWords >apunte , apuntes , notas , material , clase , cap tulo , cap tulos</
    matchingWords >
56 <shouldApplyParsCitRule >false</shouldApplyParsCitRule >
57 </label >
58 <label >

```

```
59     <category>Tesis</category>
60     <subcategory>null</subcategory>
61     <maxCharactersInDoc>1500</maxCharactersInDoc>
62     <matchingWords>tesis , tesina</matchingWords>
63     <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
64 </label>
65 <label>
66     <category>Publicación</category>
67     <subcategory>null</subcategory>
68     <maxCharactersInDoc>1500</maxCharactersInDoc>
69     <matchingWords>publicación , publicaciones</matchingWords>
70     <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
71 </label>
72 </postCrawlLabeling>
73 </general>
```