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

27 de julio de 2014

Resumen

Para poblar Repositorios Institucionales Educativos de Producción Nacional, además de políticas y estrategias de difusión adecuadas, es necesario desarrollar herramientas informáticas para recopilar todos los 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 actualmente realizada manualmente. 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: mails y filiaciones. En la extracción de información, no solamente se procesa 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

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 docentes e investigadores de casi la totalidad 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"[1] 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 y Rosario. 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 lineas de trabajo que se originan a partir de este proyecto. A raiz del diseño de Repositorios Institucionales surge la posibilidad 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, resulta 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 publicas nacionales). Siguiendo esta arquitectura, se desarrolla un prototipo en el dominio de la problemática mencionada

[1]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.

Un característica presente en el dominio de aplicación descripto, 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 gráfos 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, ricos 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) Reconocimiento y Clasificación de entidades, es decir el reconocimiento de las entidades y conceptos relativos 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[2].

SeerSuite [4]

SeerSuite es un framework para bibliotecas digitales de literatura científica en constante evolución, que cuenta con un motor de búsqueda construído a través del crawling de sitios científicos y académicos de la Web. Adicionalmente incorpora la indexación completa de texto e indexación de citas en forma autónoma, vinculando las referencias de diferentes artículos facilitando la navegación a través de los mismos. SeerSuite permite el acceso al extenso 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[3] y Microsoft Academic Research[4] 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[5] y (5) Los metadatos

[2]http://protege.stanford.edu/

[3]http://scholar.google.com.ar/

[4]http://academic.research.microsoft.com/

[5]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 excelentes 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 realizadas en base a objetivos específicos sobre los documentos y páginas web recopiladas, por lo cuál los enfoques adaptativos tampoco son utilizados. Además, para la estructuración de información sobre entidades relacionadas se utiliza una base de datos orientada a grafos, en lugar de los enfoques con ontologías presentes en los mencionados trabajos.

Con respecto a SeerSuite, este 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[6] (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 procesado 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 procesos hasta que las condiciones finales sean alcanzadas. Estas condiciones varian 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 cuál 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 cuáles 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 5 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 cuáles corría en diferentes máquinas y utilizaba operaciones asíncronas

[6] 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 conviertía 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 desición de cuándo revisitar una URL para verificación de cambios.
- Política de diplomacia: sobre los intevalos 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[7] 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[8] en la descarga de recursos asociados a una

[7]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

[8]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

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

- **Web Crawlers Focalizados:** La importancia de recuperación de una página determinada, puede ser expresada como función de la similaridad 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[9] y Microsof Academic Search[10]. 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 postprocesamiento, donde pueden emplearse algoritmos de aprendizaje automatizado, expresiones regulares, reglas adhoc 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[11]. Su desarrolo comenzó en el 2003 y la intención era desarrollar un crawler con el propósito específico de utilizarse en sitios

[9] http://scholar.google.com.ar/

[10] http://academic.research.microsoft.com/

[11] es un sitio web y una organización sin ánimo de lucro destinada a la preservación de historiales Web y recursos multimedia. Wikipedia

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™, 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[12], que actualmente se desempeña como ingeniero del equipo de desarrollo del motor de búsquedas Bing de Microsoft. 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 en este trabajo. 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 en el presente trabajo, 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,

[12]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 como 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 desde donde se esperan encontrar hechos se encuentra 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 de en sus objetivos, 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 lingüistica y procesamiento de lenguaje natural, mientras que en el campo de la RI, han influido áreas como teoría de la información, probabilidad y estadísitca.

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:

$$P = \frac{\#Respuestas\ \ Correctas}{\#Respuestas\ \ Producidas}$$

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

Ambas métricas, cobertura y precisión caen siempre dentro del intervalo [0,1] y su óptimo es 1.

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[13] es tomar $C = 1$ en caso de que $\#Respuestas$

[13] http://en.wikipedia.org/wiki/Talk%3APrecision_and_recall

$Producidas = 0$ 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 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 β 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 analizan herramientas extractoras de metadatos generales tales como el título, los autores, las palabras claves, el resumen y el idioma. En particular, se analizaron las herramientas: KEA[14], MrDlib[15],AlchemyAPI[16], ParsCit[17]. 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

[14] www.nzdl.org/Kea/index_old.html
[15] www.mr-dlib.org
[16] www.alchemyapi.com
[17] wing.comp.nus.edu.sg/parsCit/

lo son. Además, se contrastaron los resultados de Alchemy con su combinación con ParsCit para el preprocesamiento de documentos, obteniendo con la 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[18] 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.

[18] http://pdfbox.apache.org/

Además, realiza una búsqueda de metadatos (autor, título, organización, palabras claves, etc) que pueden estar embebidos en el archivo binario, cargados 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 si bien se concluye en que experimentalmente no existe una técnica predominante, 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 substitución) para convertir una cadena en la otra. Existen algoritmos de programación dinámica que logran calcular este número en $O\left(|s_1| \times |s_2|\right)$.

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

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

donde $dist_{ld}\left(s_1, s_2\right)$ es la distancia de edición calculada por el algoritmo. La distancia Levenshtein es simétrica y siempre se cumple que $0 \leq dist_{ld}\left(s_1, s_2\right) \leq$

$max\left(|s_1|, |s_2|\right)$ y que $abs\left(|s_1| - |s_2|\right) \leq dist_{ld}\left(s_1, s_2\right)$, 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_{dld} 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 $O(min\left(|s_1|, |s_2|\right) \times |s_1| \times |s_2|)$. 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) $div_{swd}(s_1, s_2) = min\left(|s_1|, |s_2|\right)$, (2) $div_{swd}(s_1, s_2) = max\left(|s_1|, |s_2|\right)$ o (3) $div_{swd}(s_1, s_2) = 0{,}5 \times \left(|s_1| + |s_2|\right)$

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[19]) 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[20]. 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 caracterísica puesto que gran parte de

[19]CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete)

[20]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.

consultas que no requieren recorridos, resultan demasiado costosas en términos de procesamiento y memoria.

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, todos 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 lo cuáles el rendimiento mejora de uno o más órdenes de magnitud para consultas sobre conjuntos de datos fuértemente 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 complejida 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[21], que es una herramienta de software libre de Base de Datos Orientada a Grafos, ACID[22], OLTP (completamente transaccional que expone CRUD), implementada en Java. Utiliza almacenamiento nativo orientada 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).

[21] http://www.neo4j.org/

[22] 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.

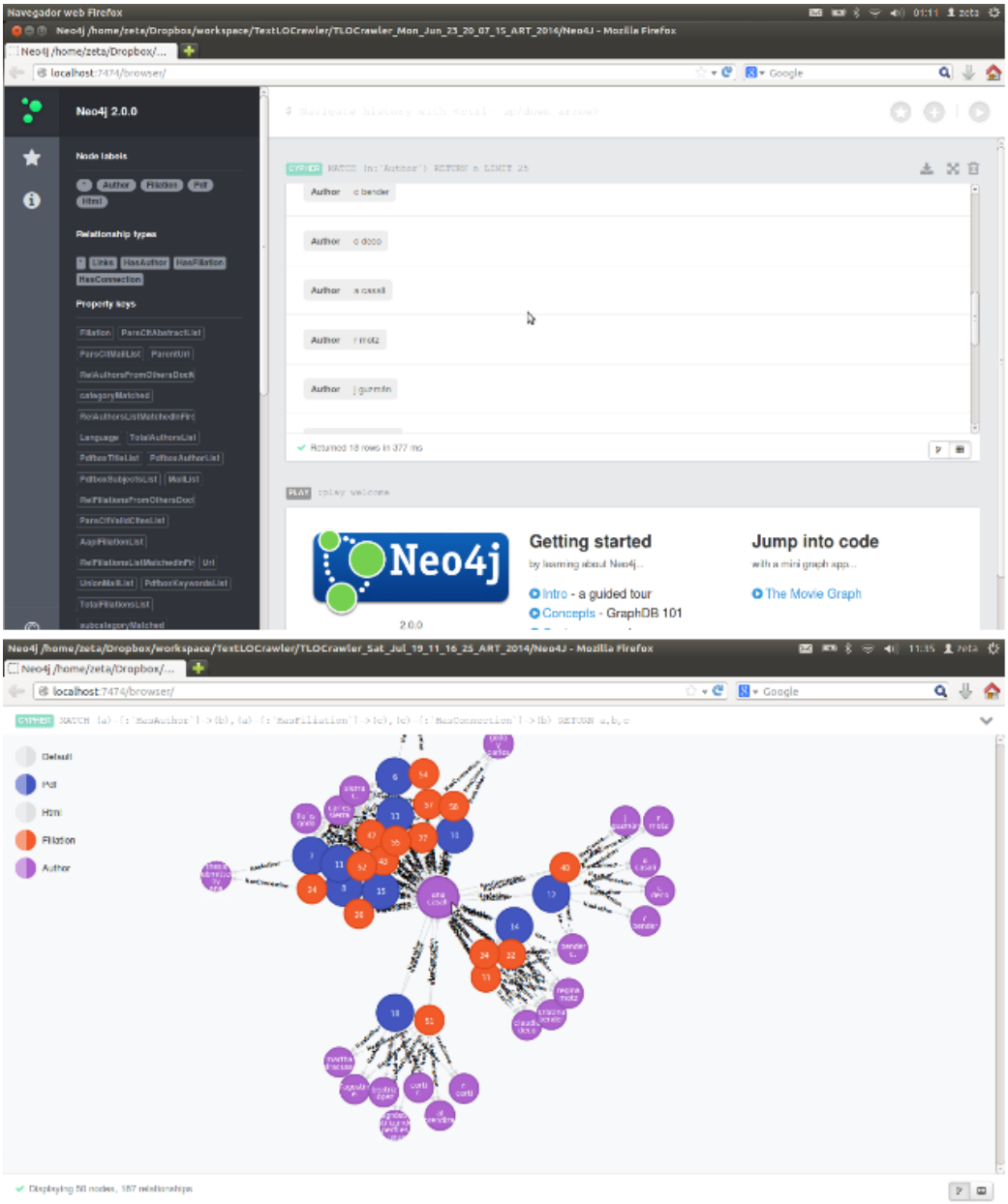


Figura 1: Arriba se observa la interfaz web de Neo4j. Abajo, un grafo con nodos y relaciones originadas por el sistema.

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 la 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 si puede encontrarse en otra página del mismo sitio web. Esto es muy común en páginas educativas de materias,

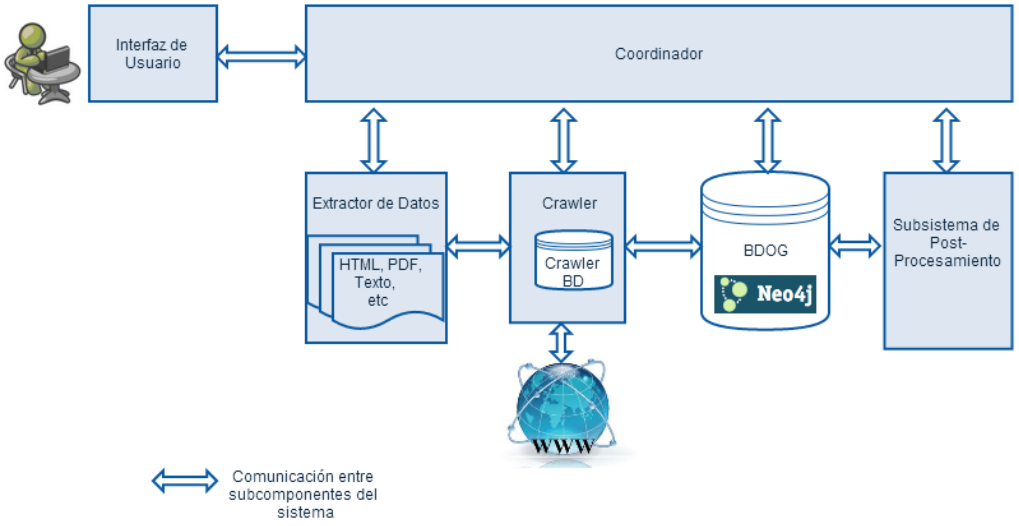


Figura 2: Diagrama de Arquitectura.

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 realizael 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 consula 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-

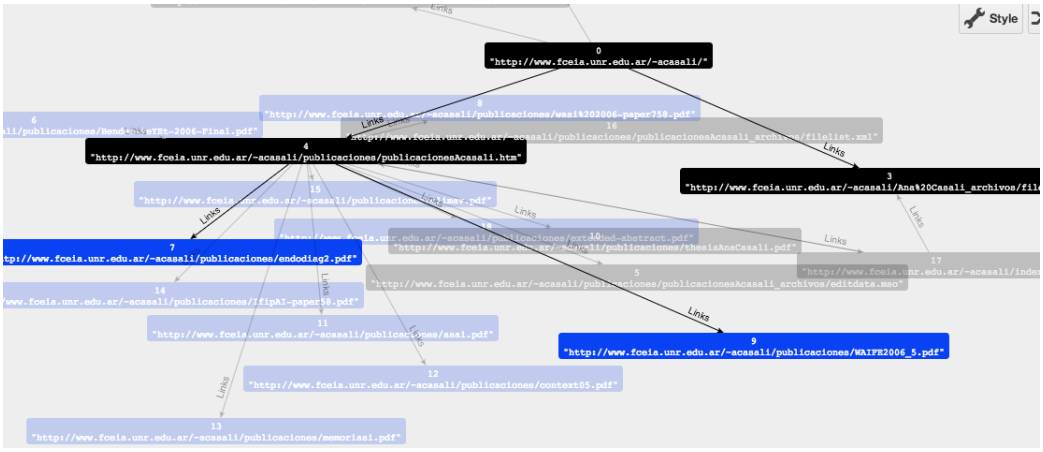


Figura 3: Visualización de Base de Datos Gráfica resultante del crawling del sitio Web http://www.fceia.unr.edu.ar/˜acasali/

dos con conexiones (relaciones de la BDOG) a los PDFs donde fueron encontrados. Dicha información resulta de utilidad para el análisis de conexiones entre las diferentes entidades, de las cuáles pueden inferirse reglas que mejoren la extracción de datos de los ODEs del sitio. Estas ideas se analizan con mas detalle posteriormente.

3.2. Secuencia de Procesamiento e Interacción Entre Componentes

En la Figura 4 se muestra la secuencia de procesamiento y comunicación entre componentes del sistema. El operador es el encargado de configurar e iniciar el el procesamiento a través de la interfaz del sistema (Mensajes 1 y 2). Ésta lo comunica al Coordinador (Mensaje 3), quién se encarga de las diferentes etapas de procesamiento, interactuando y configurando el resto de las componentes. Primero, inicia la instancia de base de datos embebida (Mensaje 4) y luego configura y notifica al crawler el inicio de un nuevo trabajo de crawling (Mensaje 5). Por cada nuevo archivo encontrado proveniente de una URL, el Crawler es quién interactúa con el Extractor de Datos para luego recopilar los datos extraídos con los provenientes del Crawling y persistir la información en la BDOG (Mensajes 6, 7 y 8). Al terminar el trabajo de crawling, el Coordinador es notificado y se comunica con el Subsistema de Post-Procesamiento (Mensajes 9 y 10).

Éste último, solicita información a la BDOG de recorridos sobre la estructura del/los sitios Web semillas, y los atraviesa según la configuración deseada (Mensaje 11). También solicita la información de cada nodo pre-

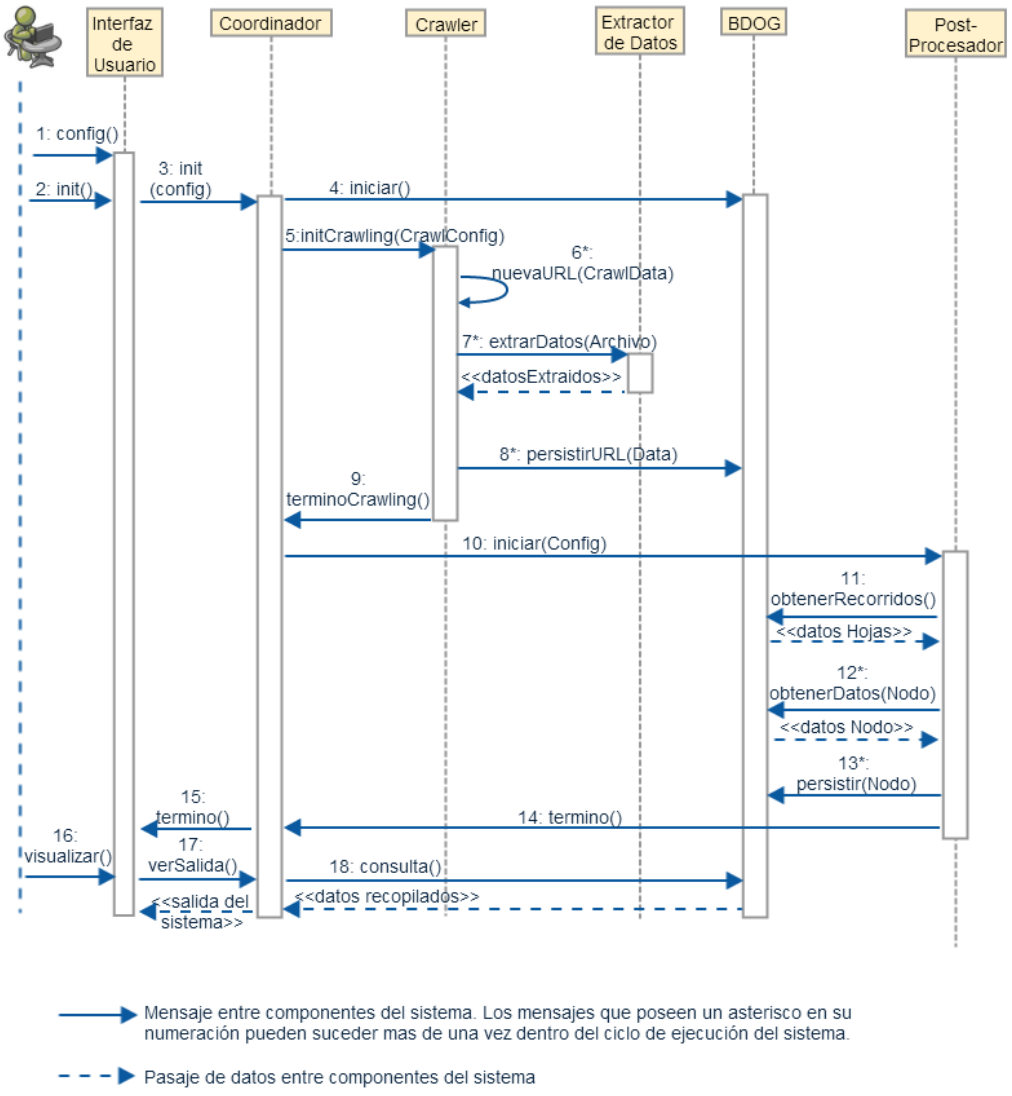


Figura 4: Diagrama de Secuencia e Interacción Entre Componentes.

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 dependendientes 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 mas 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
    <crawlerParameters>
5    <SEEDS>http://www.fceia.unr.edu.ar/ingsoft/</SEEDS>
   <ONLY_SUBPATHS>true</ONLY_SUBPATHS>
7  </crawlerParameters>

9  <parsCit_Home>
```

```
  /home/zeta/Documentos/ParsCit-master/bin
  </parsCit_Home>

  <neo4j_Home>
  /home/zeta/Documentos/neo4j-community-2.0.0
  </neo4j_Home>

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

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

  <postCrawlGraphTraversal>
 <max_depth_emaillist>3</max_depth_emaillist>
 <key_max_depth_potentialauthorlist>3</
    key_max_depth_potentialauthorlist>
<key_confidence_lower_bound_potentialauthorlist_aapi>0.0</
   key_confidence_lower_bound_potentialauthorlist_aapi>
 <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>
 <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>

  <postCrawlLabeling>
    <label>
        <category>Material Educativo</category>
        <subcategory>Práctica</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
        <matchingWords>práctica,prácticas,ejercicio,
           ejercicios,ejercitación,práctico</matchingWords>
 </label>
 <label>
        <category>Material Educativo</category>
        <subcategory>Apunte de Clase</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
```

```
45          <matchingWords>apunte,apuntes,notas,material,clase,
                capítulo,capítulos</matchingWords>
    </label>
47  <label>
            <category>Tesis</category>
49          <subcategory>null</subcategory>
            <maxCharactersInDoc>1500</maxCharactersInDoc>
51          <matchingWords>tesis,tesina</matchingWords>
    </label>
53  <label>
            <category>Publicación</category>
55          <subcategory>null</subcategory>
            <maxCharactersInDoc>1500</maxCharactersInDoc>
57          <matchingWords>publicación,publicaciones</
                matchingWords>
    </label>
59   </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).

- `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[23].

- `stopWords_Authors` y `stopWords_Filiaions`: palabras que son excluídas 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.

- `neo4j_Home`: directorio de instalación de Neo4j donde se encuentra el ejecutable `neo4j.sh` necesario para levantar la instancia de la DBOG

[23]Mas información sobre la instalación de ParsCit puede encontrarse en https://github.com/knmnyn/ParsCit

en modo servidor[24].

- `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` (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[25], 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).

[24]Mas información sobre la instalación de Neo4j puede encontrarse en http://docs.neo4j.org/chunked/stable/deployment.html

[25]este paso depende de la correcta configuración del parámetro `neo4j_Home`

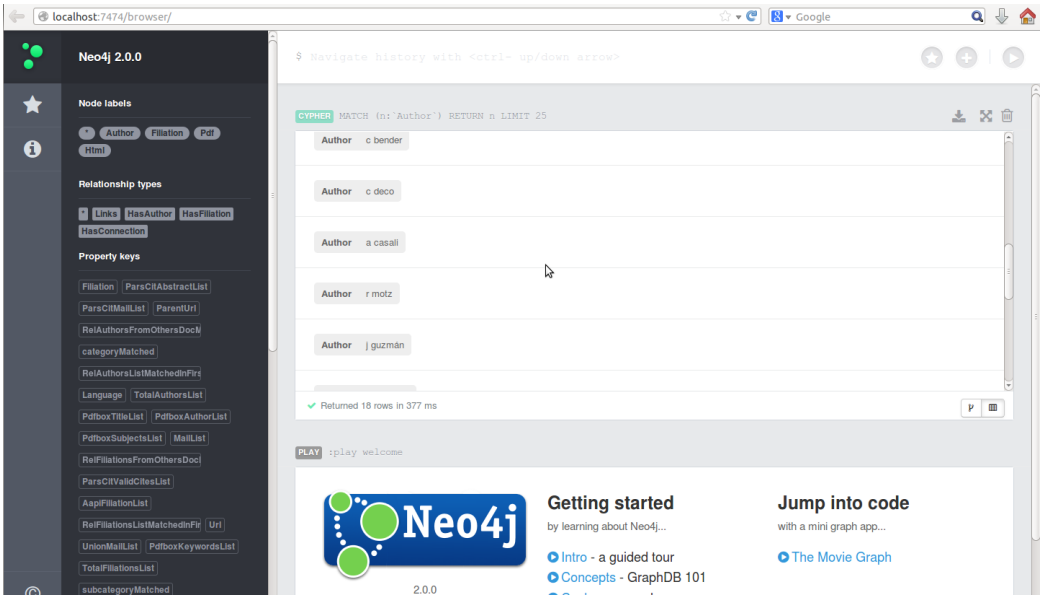


Figura 5: Ínterfaz web de consulta Neo4j

4.4. Base de Datos Orientada a Grafos Embebida y Entidades Persistidas

Este componente se implementó mediante las librerías de **Neo4j** y su interfaz principal se halla en la clase `EmbeddedNeo4j.java` (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[26] 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
- KeyParentDbPage: propiedad key de la primer URL desde donde fue encontrada la URL siendo explorada

[26] cumple con los criterios descritos en la sección Crawler, método `public boolean shouldPersisted(WebURL url)`

- 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` (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[27] 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[28] 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 DBOG.

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*[29] 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.

[27] método `public boolean shouldVisit(WebURL url)`

[28] método `public void visit(Page page)`

[29] Mas información sobre en 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[30]: 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[31]. Esta API es utilizada sobre el contenido de las páginas Web crawleadas.

- API de Extracción de Palabas Claves[32]: 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[33]: 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

[30] http://www.alchemyapi.com/api/entity/urls.html

[31] La lista completa de entidades capaz de reconocer por AlchemyAPI se encuentra en http://www.alchemyapi.com/api/entity/types.html

[32] http://www.alchemyapi.com/api/keyword-extraction/

[33] http://www.alchemyapi.com/api/language-detection/

con la confidencia 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` (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 carácteres 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 muchas casos, por diferentes características de la estructura de un ODE, una entidad presente en mas de un documento solo 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 mas 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[34].

[34] `key_similarity_lower_bound_author_matching` y `key_similarity_lower_bound_filiation_matching`

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`[35]. 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 confidencia previamente configurada[36]. 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[37] 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

[35] `max_depth_emaillist` , `max_depth_potentialauthorlist` y `max_depth_potentialfiliationlist`

[36] `key_confidence_lower_bound_potentialauthorlist` y `key_confidence_lower_bound_potentialauthorlist`, valor 0 en estos parámetros equivale a no aplicar ningún filtro por confidencia

[37] Por la relación *Links* y hasta una profundidad configurable

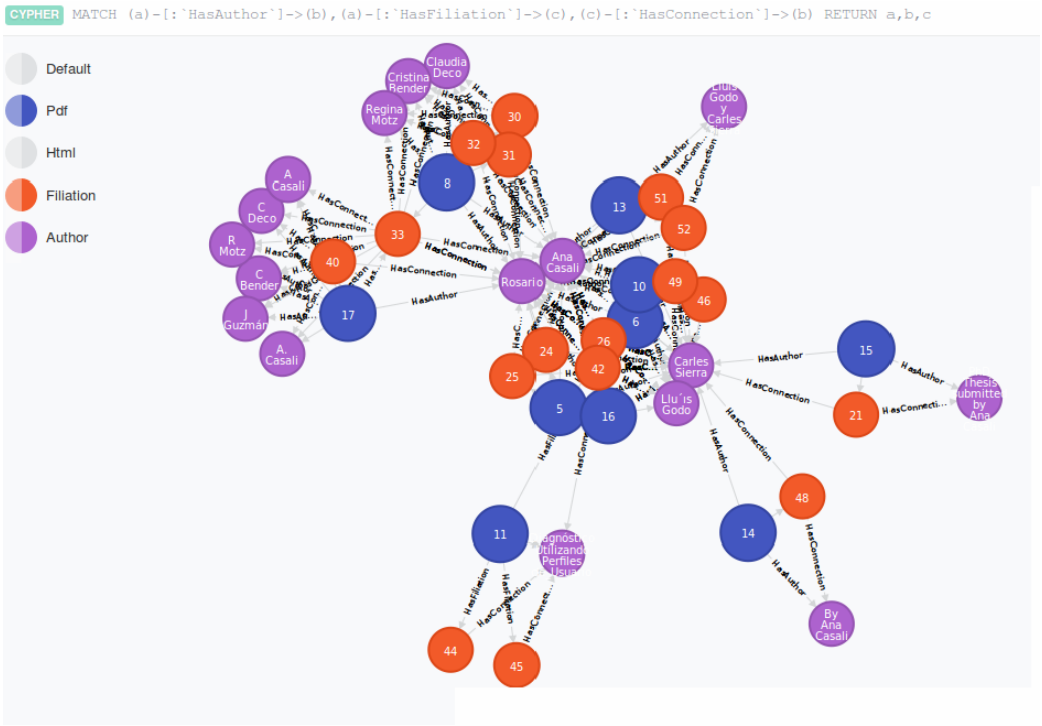


Figura 6: Visualización de BDOG resultante del crawling del sitio Web http://www.fceia.unr.edu.ar/~acasali/

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 las 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 en 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* y 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 Edutcativo*" y subcategoría "*Práctica*", si se encontraba alguna de las palabras:"práctica", "ejercicio", "ejercitación", etc... que fueron incluídas 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): discrimimanos 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 ninún 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 DBOG resultante para este sitio pueden observarse en la figura 7.

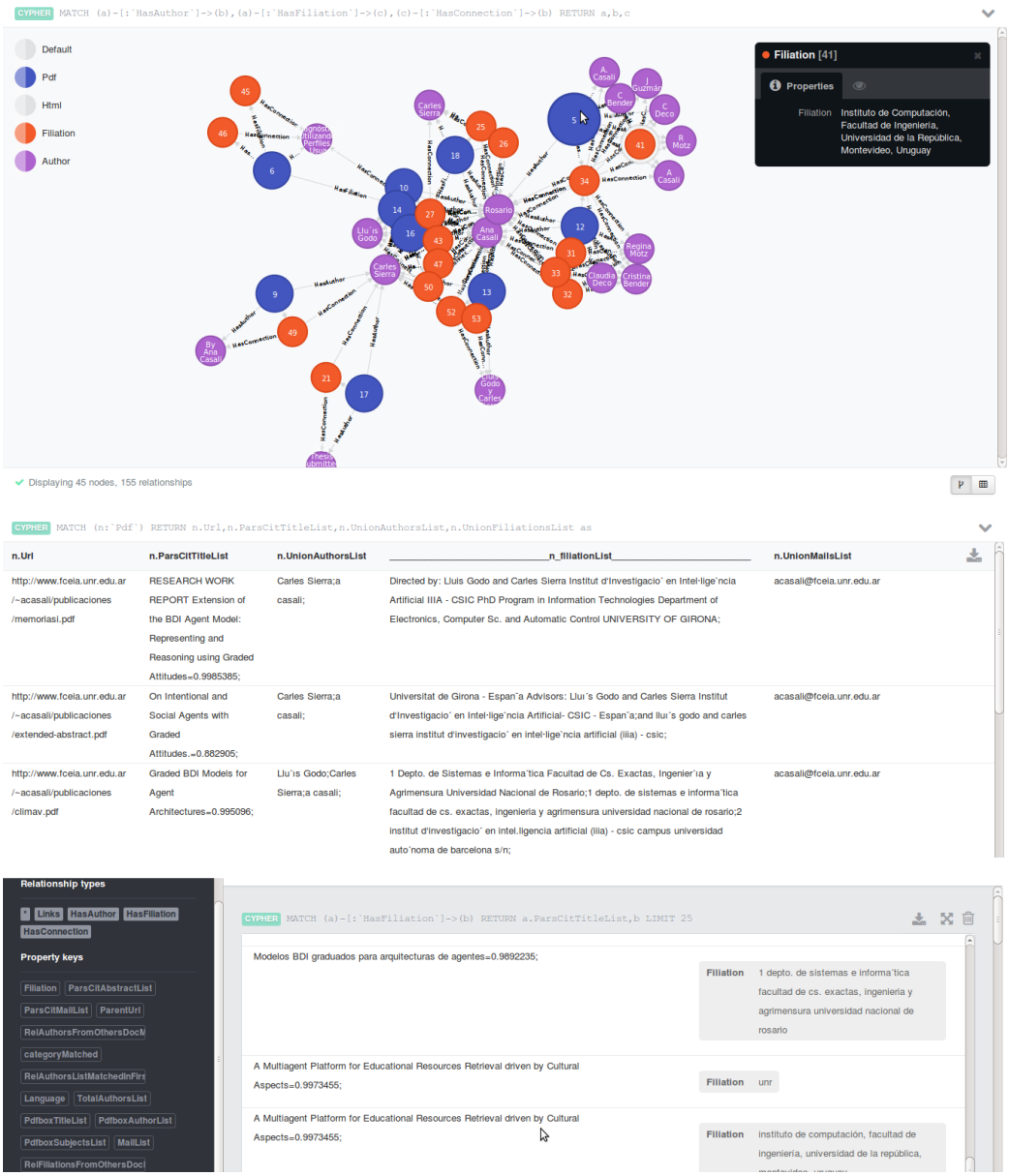


Figura 7: Diferentes consultas sobre la DBOG construida por el sistema para el sitio web del caso 1.

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[38].

El extractor `PDFTextAndMetadataExtractorWrapper.java` logró recu-

[38]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 extracor `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 *Tésis*, 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
		0,550	**1,000**	

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 filiaciones, 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 cobertura. Desde luego, la intersección entre resultados de los extractores

		ParsCit					
Documento	Idioma	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
		1,000	**1,000**	**0,900**	**0,700**	**0,850**	**0,733**

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

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 al óptimo en P. Si bien la cobertura no es óptima, se logra recopilar el mail

	EI 2				EI 3			
Documento	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
	0,983	**0,640**	**1,000**	**0,283**	**1,000**	**0,536**	**1,000**	**0,633**

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
	0,983	**0,950**	**0,950**	**0,950**

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

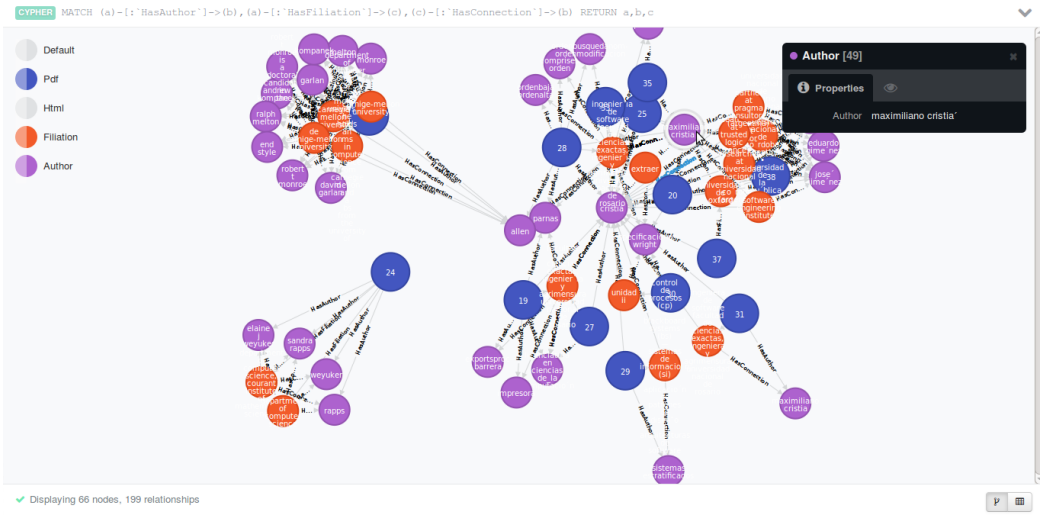
del 60 % de los autores, teniendo además de cada uno de los documentos al menos un mail de contacto.

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
	1,000	**0,416**	**0,800**	**0,800**	**1,000**	**0,300**	**1,000**	**0,600**

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 DBOG resultante para este sitio pueden observarse en la figura 8.



CYPHER MATCH (n:`Pdf`) RETURN n.Url,n.ParsCitTitleList,n.UnionFiliationsList,n.UnionAuthorsList,n.UnionMailsList

http://www.fceia.unr.edu.ar /ingsoft/weyunker00.pdf	Data Flow Analysis Techniques for Test Data Selection=0.999583;	Department of Computer Science, Courant Institute of Mathematical Sciences,;	Rapps;Weyuker;	ingsoft@fceia.unr.edu.ar;mcristia@fceia.unr.edu.ar.
http://www.fceia.unr.edu.ar /ingsoft/estilos-cat.pdf	Blackboard Systems=0.503849;	Ingenier´ıa de Software Facultad de Ciencias Exactas, Ingenier´ıa y Agrimensura Universidad Nacional de Rosario;	maximiliano cristia;	ingsoft@fceia.unr.edu.ar;mcristia@fceia.unr.edu.ar.
http://www.fceia.unr.edu.ar /ingsoft/patrones-doc.pdf	Estandar para documentar el uso de patrones de dise~no en un dise~no de software=0.5587095;	Ingeniera de Software Facultad de Ciencias Exactas, Ingeniera y Agrimensura Universidad Nacional de Rosario;	Maximiliano Cristia;	ingsoft@fceia.unr.edu.ar;mcristia@fceia.unr.edu.ar.
http://www.fceia.unr.edu.ar /ingsoft/testing-estruct-a.pdf	Introduccio´n al Testing de Estructural=0.982987;	Ingenier´ıa de Software Facultad de Ciencias Exactas, Ingenier´ıa y Agrimensura Universidad Nacional de Rosario;	maximiliano cristia;	ingsoft@fceia.unr.edu.ar;mcristia@fceia.unr.edu.ar.

Figura 8: Diferentes consultas sobre la DBOG 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 exluyen 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.

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
		0,824	**1,000**	**0,735**	**1,000**

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

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).

		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
		0,647	**0,647**	**0,641**	**0,559**	**0,833**	**0,765**

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

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.

	EI 2				EI 3			
Documento	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
	0,685	**0,882**	**0,857**	**0,470**	**1,000**	**0,647**	**1,000**	**0,706**

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

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 sobre la extracción de los mismos sin perjudicar la precisión.

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

	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
	0,615	**0,941**	**0,846**	**0,765**

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

son los encontrados en el sitio web (columna RelMailList), teniendo en la union 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
	1,000	**1,000**	**0,000**	**0,000**	**0,765**	**0,765**	**0,755**	**0,824**

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 construído 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 entre 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,técnicasas 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. En la experimentación, sobre todos los documentos de ambos 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, y 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 tareas:

- 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 mas formatos de texto (PS,Microsof Word,etc...)

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 ECMLlPKDD 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.

[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

```
package textLOCrawler;

import dbBroker.DBTraverseHelper;
import dbBroker.EmbeddedNeo4j;
import edu.uci.ics.crawler4j.crawler.CrawlConfig;
import edu.uci.ics.crawler4j.crawler.CrawlController;
import edu.uci.ics.crawler4j.fetcher.PageFetcher;
import edu.uci.ics.crawler4j.robotstxt.RobotstxtConfig;
import edu.uci.ics.crawler4j.robotstxt.RobotstxtServer;

import java.awt.Desktop;
import java.net.URI;
import textExtraction.Utilities;

public class PdfCrawController {
    public static void main(String[] args) throws Exception {

     Parameters parameters = Parameters.getInstance();//global system config

     if (parameters.parsCitHome().isEmpty()) {
      System.out.println("You must provide the path to ParsCit installation in order
         to be able of execute the script associates with ParsCit!");
      return;
     }

     if ( parameters.neo4jHome().isEmpty()) {
      System.out.println("Warning: You must provide the path to Neo4j installation in
         order to be able of visualize the system output.");
     }

     java.util.Date date= new java.util.Date();
     String dateSufix= date.toString().replace(' ', '_').replace(':', '_');
     String rootFolder = System.getProperty("user.dir")+ "/TLOCrawler_" + dateSufix;//
        args[0];

           CrawlConfig config = new CrawlConfig();

           config.setCrawlStorageFolder(rootFolder);

           /*
```

```
38              * numberOfCrawlers shows the number of concurrent threads that should
                * be initiated for crawling.
40              */
               int numberOfCrawlers = 2;
42             /*
                * Be polite: Make sure that we don't send more than 1 request per
44              * second (1000 milliseconds between requests).
                */
46             config.setPolitenessDelay(1000);

48             /*
                * You can set the maximum crawl depth here. The default value is -1 for
50              * unlimited depth
                */
52             config.setMaxDepthOfCrawling(-1);

54             /*
                * You can set the maximum number of pages to crawl. The default value
56              * is -1 for unlimited number of pages
                */
58             config.setMaxPagesToFetch(10000);

60             /*
                * This config parameter can be used to set your crawl to be resumable
62              * (meaning that you can resume the crawl from a previously
                * interrupted/crashed crawl). Note: if you enable resuming feature and
64              * want to start a fresh crawl, you need to delete the contents of
                * rootFolder manually.
66              */
               config.setResumableCrawling(false);
68             config.setConnectionTimeout(30000);
               config.setIncludeHttpsPages(true);
70             config.setFollowRedirects(true);
               config.setCrawlStorageFolder(rootFolder);
72
               /*
74              * Since images are binary content, we need to set this parameter to
                * true to make sure they are included in the crawl.
76              */
               config.setIncludeBinaryContentInCrawling(true);
78
               String[] crawlDomains = parameters.crawlerSeeds();//extract seeds to crawl
80
               PageFetcher pageFetcher = new PageFetcher(config);
82             RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
               RobotstxtServer robotstxtServer = new RobotstxtServer(robotstxtConfig,
                   pageFetcher);
84             CrawlController controller = new CrawlController(config, pageFetcher,
                   robotstxtServer);
```

```
            for (String domain : crawlDomains) {
                  controller.addSeed(domain);
            }

            PdfCrawler.configure(crawlDomains, rootFolder,parameters.crawlOnlySubpaths
               ());//set the domains and the folder to persist pdfs files

            //this call is blocking
            controller.start(PdfCrawler.class, numberOfCrawlers);
      //End crawling job

            DBTraverseHelper.getInstance().traversePdfFilesInDB(); //post-crawling
               processing
            EmbeddedNeo4j.getInstance().shutDown();//shutdown embebbed database

      Process proc;
       try {//set config properties and open web interface of neo4j on server mode
        String newConfig = Utilities.defaultNeo4jConfig.replace("<DBPATHTOREPLACE>",
           rootFolder+"/Neo4J");
        String newPathConfigFile = parameters.neo4jHome() + "/conf/neo4j-server-" +
           dateSufix + ".properties";

        Utilities.saveStringToFile(newConfig,newPathConfigFile);

        String pathConfig = parameters.neo4jHome() + "/conf/neo4j-wrapper.conf";
        String oldWrapperConfig = Utilities.getFileContents(pathConfig);
        String strToFind = "wrapper.java.additional=-Dorg.neo4j.server.properties=";
        int indStart = oldWrapperConfig.indexOf(strToFind);
        int indEnd = oldWrapperConfig.indexOf("\n", indStart);
        indStart+=strToFind.length();

        String strToReplace = oldWrapperConfig.substring(indStart, indEnd);
        oldWrapperConfig = oldWrapperConfig.replaceFirst(strToReplace,newPathConfigFile
           +"\n");

        Utilities.saveStringToFile(oldWrapperConfig, pathConfig);

        //stop the server
        String[] strCmd = new String[]{parameters.neo4jHome() + "/bin/neo4j","stop"};
        System.out.println(strCmd);
        proc = Runtime.getRuntime().exec(strCmd);
        proc.waitFor();
        //start the server
        strCmd = new String[]{parameters.neo4jHome() + "/bin/neo4j","start"};
        System.out.println(strCmd);
        proc = Runtime.getRuntime().exec(strCmd);
        proc.waitFor();

        proc.destroy();
```

```
        proc = null;
130
        String neo4jWebClientUrl = "http://localhost:7474";
132
              if (Desktop.isDesktopSupported()) {
134               // Windows
                  Desktop.getDesktop().browse(new URI(neo4jWebClientUrl));
136           } else {
                  // Ubuntu
138               Runtime runtime = Runtime.getRuntime();
                  runtime.exec("/usr/bin/firefox -new-window " + neo4jWebClientUrl);
140           }
      } catch (Exception e) {
142    e.printStackTrace();
      }
144   }
}
```

A.2. BDOG Embebida

Archivo 2: EmbeddedNeo4j.java

```
1  package dbBroker;

3  import java.io.File;
   import java.io.IOException;
5  import java.lang.reflect.Field;
   import java.util.Map;
7  import org.apache.log4j.Level;
   import org.neo4j.graphdb.*;
9  import org.neo4j.graphdb.factory.GraphDatabaseFactory;
   import org.neo4j.graphdb.index.Index;
11 import org.neo4j.graphdb.index.UniqueFactory;
   import org.neo4j.kernel.impl.util.FileUtils;
13 import textLOCrawler.CrawlUtilities;

15 public class EmbeddedNeo4j {

17  private static volatile EmbeddedNeo4j _instance = null;

19  private static String DB_PATH = null;
    public static UniqueFactory<Node> factory;
21
    public static GraphDatabaseService graphDb;
23
    public static org.apache.log4j.Logger logger;
```

```
25
   public static Index<Node> allDbPagesIndex;//index for speed the search and traverse
       over all nodes created
27     private static final String NAME_PAGES_INDEX = "pages";

29     public static Index<Node> pdfIndex;//index for speed the search and traverse over
          all pdf nodes created
       private static final String NAME_PDF_INDEX = "pdf";
31
       public static final String FIELDNAME_KEY = "Key";
33
   public static final String FILIATION_NAME = "Filiation";
35 public static final String AUTHOR_NAME = "Author";

37 public static EmbeddedNeo4j getInstance() {
       if(_instance == null)
39      return getInstance(System.getProperty("user.dir"));
       else
41      return _instance;
   }
43
       public static EmbeddedNeo4j getInstance(String rootDBPath) {
45           if(_instance == null){
                       _instance = new EmbeddedNeo4j();
47                     DB_PATH = rootDBPath + "/Neo4J";
                       File file = new File(DB_PATH);
49                     file.mkdirs();
                       clearDb();
51                     createDb();
                       registerShutdownHook( graphDb );
53              }
               return _instance;
55     }

57     public static enum RelTypes implements RelationshipType {Links,HasFiliation,
          HasAuthor,HasConnection}

59     public void saveObject(IDbNeo4jObject nodeToSave) throws Exception{
        Transaction tx = graphDb.beginTx();
61         try
           {
63          //get or create the unique node for the key (implemented by the interface)
            Node nodo = factory.getOrCreate( FIELDNAME_KEY, nodeToSave.getKey() );
65
            logger.log(Level.INFO, "Saving␣nodo,␣Key:␣" + nodeToSave.getKey() + "␣nodo␣ID
               :␣" + nodo.getId());
67
            //recover all the public fields from the pojo
69          Field[] fields = nodeToSave.getClass().getDeclaredFields();
```

```
        for(Field f : fields){
71       if (f.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue; //skip
             implementation

73       //Current Property Value
         Object propertyValue = f.get(nodeToSave);
75       if (propertyValue == null) continue;

77       //if the field is not a collection set it in the node
         if (!java.lang.Iterable.class.isAssignableFrom(propertyValue.getClass())) {
79
          //if it is a dbNeoObject, save it (like a page or file)
81        if (IDbNeo4jObject.class.isAssignableFrom(propertyValue.getClass())) {
           Node propertyNode = factory.getOrCreate( FIELDNAME_KEY, ((IDbNeo4jObject)
               propertyValue).getKey() );
83         logger.log(Level.INFO, "Saving Node Key: " + ((IDbNeo4jObject)
               propertyValue).getKey() + " nodo ID: " + propertyNode.getId());
           for(Field innerPropF : propertyValue.getClass().getDeclaredFields()){
85          if (innerPropF.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue;
                Object innerPropertyValue = innerPropF.get(propertyValue);
87              //Skip if the inner object implements iterators
                if (innerPropertyValue == null) continue;
89              if (java.lang.Iterable.class.isAssignableFrom(innerPropertyValue.
                    getClass())) continue;
                logger.log(Level.INFO, "Traverse intern property of node " + nodo.
                    getId() + " => Field: " + innerPropF.getName() + " Value: " +
                    innerPropertyValue);
91              propertyNode.setProperty(innerPropF.getName(), innerPropertyValue)
                    ;
                }
93         /* In the model, only the DbPage has a ParentDbPage:IDbNeo4jObject
               property, so setting this relationship is redundant cause the RelType.
               Links
            * */
95         //nodo.createRelationshipTo(propertyNode, RelTypes.Linked);
          }
97        else
          {
99         //just a property
           nodo.setProperty(f.getName(), propertyValue);
101       }
          continue; //move next for the next field
103      }

105   //if the object implements an iterator loop it (i.e. outgoin links)
      java.lang.Iterable coleccion =  (Iterable) f.get(nodeToSave);
107
         //Loop the colection
109      java.util.Iterator iter = coleccion.iterator();
```

```
        while (iter.hasNext()) {
111      Object innerObject = iter.next();

113      //see if implements the interfaz for save in the DB
         if (!IDbNeo4jObject.class.isAssignableFrom(innerObject.getClass())) continue;
115
         Node innerNode = factory.getOrCreate( FIELDNAME_KEY, ((IDbNeo4jObject)
            innerObject).getKey() );
117
         logger.log(Level.INFO, "Saving nodo, Key: " + ((IDbNeo4jObject)innerObject).
            getKey() + " nodo ID: " + innerNode.getId());
119
            for(Field innerF : innerObject.getClass().getDeclaredFields()){
121
             if (innerF.getName().equalsIgnoreCase(FIELDNAME_KEY)) continue;
123
             Object innerPropertyValue = innerF.get(innerObject);
125
             //Skip if the inner object implements iterators
127          if (innerPropertyValue == null) continue;
             if (java.lang.Iterable.class.isAssignableFrom(innerPropertyValue.
                getClass())) continue;
129
             logger.log(Level.INFO, "Traver intern properties intern node: " + nodo.
                getId() + " => Field: " + innerF.getName() + " Value: " +
                innerPropertyValue);
131
          innerNode.setProperty(innerF.getName(), innerPropertyValue);
133          }

135          //finaly, create the relationship

137          logger.log(Level.INFO, "Node "  + nodo.getId() + " links " + innerNode.
                getId() );
             nodo.createRelationshipTo(innerNode, RelTypes.Links);
139      }
           }
141
           //If it is a pdf file, we add it to the pdfIndex
143        if (nodeToSave.getKey().endsWith(".pdf")) //TODO: improve and abstract this
              logic
            if ( pdfIndex.get(FIELDNAME_KEY, nodeToSave.getKey()) != null )
145          pdfIndex.add( nodo, FIELDNAME_KEY, nodeToSave.getKey() );

147        tx.success();

149      } catch (Exception e) {
           logger.log(Level.INFO, "Neo4J unsuccessful transaction.\nError: " );
151        logger.log(Level.INFO, e);
```

```
    e.printStackTrace();
153     throw e;
    }
155         finally
        {
157             tx.finish();
        }
159     }

161     private static void createDb()
    {
163         graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
        registerShutdownHook( graphDb );
165
        try ( Transaction tx = graphDb.beginTx() )
167     {
          pdfIndex=graphDb.index().forNodes(NAME_PDF_INDEX);
169         allDbPagesIndex=graphDb.index().forNodes(NAME_PAGES_INDEX);
            //http://docs.neo4j.org/chunked/stable/tutorials-java-embedded-unique-
                nodes.html#tutorials-java-embedded-unique-get-or-create
171         //https://github.com/neo4j/neo4j/issues/73
            factory = new UniqueFactory.UniqueNodeFactory( allDbPagesIndex )
173       {
            @Override
175             protected void initialize( Node created, Map<String, Object>
                    properties )
                {
177          /*
              * Implement this method to initialize the Node or Relationship created
                  for being stored in the index.
179           * This method will be invoked exactly once per created unique entity.
              * The created entity might be discarded if another thread creates an
                  entity concurrently.
181           * This method will however only be invoked in the transaction that
                  succeeds in creating the node.
              * */
183           String key = (String) properties.get( FIELDNAME_KEY);
              if (key!=null) {
185            created.setProperty( FIELDNAME_KEY, properties.get( FIELDNAME_KEY ) );
                           String extension = CrawlUtilities.getExtensionFromPathName((
                               String)properties.get( FIELDNAME_KEY ));
187                        if (extension.endsWith(".pdf")) //TODO: improve and abstract
                               this logic
                            created.addLabel( DynamicLabel.label( "Pdf" ) );
189                        else {
                            created.addLabel( DynamicLabel.label( "Html" ) );
191           }
       return;
193           }
```

```
195             String author  = (String) properties.get( AUTHOR_NAME);
                if (author != null) {
197              created.setProperty( AUTHOR_NAME, properties.get( AUTHOR_NAME ) );
                        created.addLabel( DynamicLabel.label( "Author" ) );
199         return;
                }
201
                String filiation  = (String) properties.get( FILIATION_NAME);
203             if (filiation != null) {
                 created.setProperty( FILIATION_NAME, properties.get( FILIATION_NAME ) )
                    ;
205                     created.addLabel( DynamicLabel.label( "Filiation" ) );
        return;
207             }
                    }
209           };

211           tx.success();
         }
213     }

215     private static void clearDb()
        {
217         try
            {
219             FileUtils.deleteRecursively( new File( DB_PATH ) );
            }
221         catch ( IOException e )
            {
223             throw new RuntimeException( e );
            }
225     }

227     public void shutDown()
        {
229         System.out.println();
            System.out.println( "Shutting down database ..." );
231         graphDb.shutdown();
        }
233
        private static void registerShutdownHook( final GraphDatabaseService graphDb )
235      {
             // Registers a shutdown hook for the Neo4j instance so that it
237          // shuts down nicely when the VM exits (even if you "Ctrl-C" the
             // running application).
239          Runtime.getRuntime().addShutdownHook( new Thread()
             {
241              @Override
```

```
            public void run()
243         {
                graphDb.shutdown();
245         }
        } );
247 }

249 public void setLogger(org.apache.log4j.Logger oLogger) {
 logger = oLogger;
251 }

253 }
```

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"
        + "|wav|avi|mov|mpeg|ram|m4v"
        + "|rm|smil|wmv|swf|wma|zip|rar|gz))$");

private static File storageFolder;
private static String[] crawlDomains;
private static String[] crawlBaseDomains;

 public static void configure(String[] domains, String storageFolderName, boolean
    onlySubpaths) {
  storageFolder = new File(storageFolderName);
     if (!storageFolder.exists()) {
             storageFolder.mkdirs();
     }

       EmbeddedNeo4j db = EmbeddedNeo4j.getInstance(storageFolderName);
       db.setLogger(PdfCrawler.logger);

       logger.log(Level.INFO, "Seeds To Crawl  ");
       for(String seed : domains)
        logger.log(Level.INFO, "Seed: " + seed);
       logger.log(Level.INFO, "Only Subpaths: " + onlySubpaths);

       PdfCrawler.crawlDomains = domains;

  PdfCrawler.crawlBaseDomains = new String[domains.length];
  for(int i=0; i<domains.length ;i++) {
   if (onlySubpaths)
    PdfCrawler.crawlBaseDomains[i] = PdfCrawler.crawlDomains[i];
   else
    PdfCrawler.crawlBaseDomains[i] = CrawlUtilities.getDomainWithoutPath(PdfCrawler.
       crawlDomains[i]);
  }
    }

 @Override
 public boolean shouldVisit(WebURL url) {
     String href = url.getURL().toLowerCase();

     if (FILTERS.matcher(href).matches()) {//not interested file to visit
      logger.log(Level.INFO, "shouldVisit false: FILTERS " + href);
       return false;
   }

     if (pdfPatterns.matcher(href).matches()) {// is a pdf file
      logger.log(Level.INFO, "shouldVisit true: pdfPattern " + href);
      return true;
     }
```

```
77      for (String baseDomain : crawlBaseDomains) {
            if (href.indexOf(baseDomain )>=0 ){
79           logger.log(Level.INFO, "shouldVisit true: crawlBaseDomains " + href);
             return true;
81          }
        }
83
        logger.log(Level.INFO, "shouldVisit false: crawlBaseDomains " + href);
85       return false;
     }
87
    private boolean urlShouldBePersisted(String url) {
89
     if (FILTERS.matcher(url).matches()) { //not interested file to persist
91        return false;
      }
93
     if (pdfPatterns.matcher(url).matches()) // is a pdf file
95          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
        if ((page.getParseData() instanceof HtmlParseData)){
113
            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
            HTMLTextExtractor textFromHtmlextractor = new HTMLTextExtractor();
143             textFromHtmlextractor.processHtmlString(html,url.toString());
                 stringCSV = CrawlUtilities.encodeListAsCSVString(textFromHtmlextractor.
                    getListOfBibsRef());
145         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

      // We are only interested in processing pdfs
187   if ((page.getParseData() instanceof BinaryParseData)) {

189    if (!pdfPatterns.matcher(url.getURL()).matches()) return;

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

197 // store pdfs
      //TODO: check if the file exits
199   String pdfPath = storageFolder.getAbsolutePath() + "/" + hashedName;
      String textFromPdfPath = storageFolder.getAbsolutePath() + "/" + Utilities.
         removeExension(hashedName) + ".txt";
201
    IO.writeBytesToFile(page.getContentData(),pdfPath);
203
    DbPage pageParent = new DbPage();
205 pageParent.setKey(CrawlUtilities.createHashForString(url.getParentUrl()));

207 DbPdf pdf = new DbPdf();
    pdf.setKey(hashedName);
209 pdf.setSize(page.getContentData().length);
    pdf.setUrl(url.getURL());
211 pdf.setParentDbPage(pageParent);
    pdf.setKeyParentDbPage(pageParent.getKey());
213
    //TODO: should be better dispatch this job in a different thread
215 PDFTextAndMetadataExtractorWrapper wrapperPdfToText= new
       PDFTextAndMetadataExtractorWrapper();
    try {
```

```
217             wrapperPdfToText.processPdfFile(new File(pdfPath), -1, -1);
                wrapperPdfToText.saveToFile(textFromPdfPath);
219             wrapperPdfToText.prettyPrintResult();
                pdf.setPdfboxAuthorList(CrawlUtilities.encodeListAsCSVString(
                    wrapperPdfToText.getAuthorsList()));
221             pdf.setPdfboxTitleList(CrawlUtilities.encodeListAsCSVString(
                    wrapperPdfToText.getTitlesList()));
                pdf.setPdfboxRights(wrapperPdfToText.getRigths()==null?"":
                    wrapperPdfToText.getRigths());
223             pdf.setPdfboxSubjectsList(CrawlUtilities.encodeListAsCSVString(
                    wrapperPdfToText.getSubjectsList()));
                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) {
                 //ALchemy API extraction
229              AlchemyapiWrapperExtractor alchemyWrapper = new
                     AlchemyapiWrapperExtractor();
                 try {
231               pdf.setLanguage(alchemyWrapper.getLanguageOfText(wrapperPdfToText.
                      getFirstTextSegment()));
              } catch (Exception e) {
233            e.printStackTrace();
              }
235           } else {
               pdf.setLanguage(wrapperPdfToText.getLanguage());
237           }

239           //RegexMail extraction
            List<String>mails = RegExMailExtractor.listMailsExtractedFromString(
                wrapperPdfToText.getFirstTextSegment());
241         pdf.setMailList(CrawlUtilities.encodeListAsCSVString(mails));

243     } catch (Exception e) {
         System.err.println( "Error:␣extracting␣text␣for␣document:" +  pdfPath );
245              e.printStackTrace();
                 return;
247     }

249     //TODO: add this limit to parameters files
        if (page.getContentData().length < 150 *1024* 1024) {//ParsCit fails with very
            large files
251      ParsCitWrapper parsCitWrapper = new ParsCitWrapper();
            parsCitWrapper.parseFile(new File(textFromPdfPath),storageFolder);
253         parsCitWrapper.saveToFile(Utilities.removeExension(textFromPdfPath) + ".
                parsCitOut" );
```

```
            parsCitWrapper.prettyPrintResult();
255         pdf.setParsCitAbstractList(CrawlUtilities.encodePairListAsCSVString(
                parsCitWrapper.getPotentialAbstract()));
            pdf.setParsCitAuthorList(CrawlUtilities.encodePairListAsCSVString(
                parsCitWrapper.getPotentialAuthorsList()));
257         pdf.setParsCitFiliationsList(CrawlUtilities.encodePairListAsCSVString(
                parsCitWrapper.getFiliationsList()));
            pdf.setParsCitMailList(CrawlUtilities.encodeListAsCSVString(parsCitWrapper.
                getMailList()));
259         pdf.setParsCitTitleList(CrawlUtilities.encodePairListAsCSVString(
                parsCitWrapper.getPotentialTitle()));
            pdf.setParsCitValidCitesList(CrawlUtilities.encodeListAsCSVString(
                parsCitWrapper.getValidCitesList()));
261     }

263     logger.log(Level.INFO,"pdf␣persisted␣url:␣" +  url.getURL() + "␣-␣parent:␣" + url.
            getParentUrl());

265         try {
        EmbeddedNeo4j.getInstance().saveObject(pdf);
267     } catch (Exception e) {
        e.printStackTrace();
269     logger.log(Level.INFO,"Error␣saving␣nodo:␣" + e);
        logger.log(Level.INFO,"Error␣saving␣nodo:␣" + e.getMessage());
271     logger.log(Level.INFO,"Error␣saving␣nodo:␣" + e.getStackTrace());
    }
273

    logger.trace("saved␣node:␣" + url);
275       }
  }
277 }
```

A.4. Post-Procesador

Archivo 4: DBTraverseHelper.java

```
1  package dbBroker;

3  import java.util.ArrayList;
   import java.util.Arrays;
5  import java.util.Iterator;
   import java.util.List;
7  import java.util.regex.Matcher;
   import org.apache.log4j.Level;
9  import org.neo4j.graphdb.Direction;
   import org.neo4j.graphdb.Node;
```

```
11 import org.neo4j.graphdb.ReturnableEvaluator;
   import org.neo4j.graphdb.StopEvaluator;
13 import org.neo4j.graphdb.Transaction;
   import org.neo4j.graphdb.Traverser;
15 import org.neo4j.graphdb.index.Index;
   import org.neo4j.graphdb.index.IndexHits;
17 import textLOCrawler.CrawlUtilities;
   import textLOCrawler.Parameters;
19 import textLOCrawler.TLOLabel;
   import dbBroker.EmbeddedNeo4j.RelTypes;
21
   public class DBTraverseHelper {
23
    private static volatile DBTraverseHelper _instance = null;
25
    private static Index<Node> authorsIndex;
27  private static final String AUTHORS_INDEX = "authorsIndex";

29  private static Index<Node> filiationIndex;
    private static final String FILIATION_INDEX = "filiationIndex";
31
    public static DBTraverseHelper getInstance() {
33     if(_instance == null) {
                _instance = new DBTraverseHelper();
35     }
       return _instance;
37  }

39     public void traversePdfFilesInDB() {
        Transaction tx = EmbeddedNeo4j.graphDb.beginTx();
41        try
          {
43          Parameters parameters = Parameters.getInstance();

45          authorsIndex=EmbeddedNeo4j.graphDb.index().forNodes(AUTHORS_INDEX);
            filiationIndex=EmbeddedNeo4j.graphDb.index().forNodes(FILIATION_INDEX);
47
            //Graph traversal parameters
49          int mailDepth = parameters.maxDepthForKey(Parameters.KEY_MAX_DEPTH_EMAILLIST
                );
            int authorDepth = parameters.maxDepthForKey(Parameters.
               KEY_MAX_DEPTH_POTENTIALAUTHORLIST);
51          int filiationDepth = parameters.maxDepthForKey(Parameters.
               KEY_MAX_DEPTH_POTENTIALFILIATIONLIST);
            int maxOfAllDepth =  Math.max(mailDepth,Math.max(authorDepth,filiationDepth)
                );
53          float authorConfidenceLowerBound  = parameters.confidenceLowerBoundForKey(
               Parameters.KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST);
```

```
        float filiationConfidenceLowerBound = parameters.confidenceLowerBoundForKey(
            Parameters.KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIATIONLIST);
55
        //first travel
57      for ( Node pdfNode : EmbeddedNeo4j.pdfIndex.query( EmbeddedNeo4j.
            FIELDNAME_KEY, "*" ) )
            {
59       /*Graph traversal
          *
61        */
            Traverser traverse = pdfNode.traverse(Traverser.Order.BREADTH_FIRST,
63                  StopEvaluator.END_OF_GRAPH,
                    ReturnableEvaluator.ALL_BUT_START_NODE,
65                  EmbeddedNeo4j.RelTypes.Links,
                    Direction.BOTH);
67       String RelMailList = "";//list of mails separated by ';' and "(depth)" of
             the relationship, i.e. (2) means 2 degrees of distance
         String RelMailList_justPairs = "";//list of mails separated by ';' (used to
              duplicated values)
69         String RelAuthorsList = "";//list of "entity person=confidence value"
               separated by ';' and "(depth)" of the relationship, i.e. (2)
              means 2 degrees of distance
           String RelAuthorsList_justPairs = "";//list of "entity person=
              confidence value" separated by ';' (used to duplicated values)
71       String RelFiliationsList = "";//list of "entity organization=confidence
             value" separated by ';'and "(depth)" of the relationship, i.e. (2)
            means 2 degrees of distance
           String RelFiliationsList_justPairs = "";//list of "entity person=
              confidence value" separated by ';' (used to duplicated values)
73       //Start traverse
         for(Node page:traverse) {
75        int currentDepth = traverse.currentPosition().depth();
          if (currentDepth > maxOfAllDepth  )
77         break;//BREADTH_FIRST => other nodes from now on are deeper

79              if ( ((String)page.getProperty( "Url","")).endsWith(".pdf")) //
                   TODO: improve and abstract this logic
                 continue;//we don't examine pdf relatives nodes
81
          if (currentDepth <=  mailDepth) {//add mails
83         String nodeMailList = page.getProperty("MailList","").toString();
           if ( nodeMailList!=null && nodeMailList.length()>0 ) {
85          String newValues = CrawlUtilities.filterDuplicatedValuesSimpleList(
               nodeMailList.substring(0, Math.max(nodeMailList.length() - 1,0)),
               RelMailList_justPairs);
            if ( newValues!=null
87            && newValues.length()>0) {
               RelMailList_justPairs += (RelMailList_justPairs.length()>0?";":"") +
                    newValues;
```

```
                    RelMailList += newValues +  " (" + (new Integer(currentDepth))
                        .toString() + ");";
                }
               }
              }

              if (currentDepth <=  authorDepth) {//add potential authors
               String nodeAuthorList = page.getProperty("AapiAuthorList","").toString();
               if ( nodeAuthorList!=null && nodeAuthorList.length()>0 ) {
                String newValues = CrawlUtilities.filterStringPairList(nodeAuthorList.
                    substring(0,Math.max(nodeAuthorList.length()-1,0)),
                    authorConfidenceLowerBound,RelAuthorsList_justPairs);
                if ( newValues!= null
                     && newValues.length()>0 ) {
                RelAuthorsList_justPairs += (RelAuthorsList_justPairs.length()>0 ? ";":"
                    ") + newValues;
                   RelAuthorsList += newValues +  " ("+ (new Integer(currentDepth)).
                       toString() + ");";
                }
               }
              }

              if (currentDepth <=  filiationDepth) {//add potential filiations
               String nodeFiliationList = page.getProperty("AapiFiliationList","").
                  toString();
               if ( nodeFiliationList!=null && nodeFiliationList.length()>0 ) {
                String newValues  = CrawlUtilities.filterStringPairList(
                    nodeFiliationList.substring(0,Math.max(nodeFiliationList.length()
                    -1,0)),filiationConfidenceLowerBound,RelFiliationsList_justPairs);
                if ( newValues!= null
                         && newValues.length()>0 ) {
                    RelFiliationsList_justPairs += (RelFiliationsList_justPairs.length()
                       >0 ? ";":"") + newValues;
                    RelFiliationsList += newValues + " (" + (new Integer(currentDepth)).
                       toString() + ");";
                }
               }
              }
             }
             if (RelMailList.length()>0)
              pdfNode.setProperty("RelMailList", RelMailList);
             if (RelAuthorsList.length()>0)
              pdfNode.setProperty("RelAuthorsList", RelAuthorsList);
             if (RelFiliationsList.length()>0)
              pdfNode.setProperty("RelFiliationsList", RelFiliationsList);

             /*
              * Look if the related entities appear on the first fragment of text
              * */
```

```
        String textFragment = pdfNode.getProperty("TextFragment","").toString();
129     CrawlUtilities.getKeysFromPairList( RelAuthorsList_justPairs);

131     List<String> entitiesFiliationMatched = CrawlUtilities.findMatch(textFragment,
           CrawlUtilities.getKeysFromPairList(RelFiliationsList_justPairs));
        if (entitiesFiliationMatched.size()>0) {
133      String strMatched = CrawlUtilities.encodeListAsCSVString(
            entitiesFiliationMatched);
         pdfNode.setProperty("RelFiliationsListMatchedInFirstTextFragment", strMatched);
135     }

137     List<String> entitiesAuthorsMatched = CrawlUtilities.findMatch(textFragment,
           CrawlUtilities.getKeysFromPairList(RelAuthorsList_justPairs));
        if (entitiesAuthorsMatched.size()>0) {
139      String strMatched =  CrawlUtilities.encodeListAsCSVString(entitiesAuthorsMatched
            );
               pdfNode.setProperty("RelAuthorsListMatchedInFirstTextFragment", strMatched
                  );
141     }

143     this.createAuthorsAndFiliationNodes(pdfNode);//create authors and filiations
           nodes

145     this.performLabelingJob(pdfNode);
         }
147
          EmbeddedNeo4j.logger.log(Level.INFO,"\n*******Second travel pdf Node****");
149       //second travel
           for ( Node pdfNode : EmbeddedNeo4j.pdfIndex.query( EmbeddedNeo4j.
              FIELDNAME_KEY, "*" ) )
151           {
            this.findAuthorsFromOtherDocuments(pdfNode);
153         this.findFiliationsFromOtherDocuments(pdfNode);

155

        EmbeddedNeo4j.logger.log(Level.INFO,
157       "\n\n\n--{\n\nPdf file with URL: " + pdfNode.getProperty( "Url","")+
          "\n\nKey: " + pdfNode.getProperty( EmbeddedNeo4j.FIELDNAME_KEY,"")
159      + "\n\nLanguage "+ pdfNode.getProperty( "Language","")
            + "\n\n---LABELING FIELDS"
161      + "\n\nCategory: "+ pdfNode.getProperty( "category","")
         + "\n\nSubcategory: "+ pdfNode.getProperty( "subCategory","")
163
         + "\n\n---MAIL"
165      + "\n\nRel Mail list: "+ pdfNode.getProperty( "RelMailList","")
            + "\n\nParsCit Mail list: "+ pdfNode.getProperty( "ParsCitMailList","")
167
         + "\n\n---RIGHTS"
169         + "\n\nPdfbox Rights: " + pdfNode.getProperty( "PdfboxRights","")
```

```
171         + "\n\n---TITLE"
                + "\n\nPdfbox␣Title:␣"+ pdfNode.getProperty( "PdfboxTitleList","")
173             + "\n\nParsCit␣Title:␣"+ pdfNode.getProperty( "ParsCitTitleList","")

175         + "\n\n---AUTHOR␣FIELDS"
                + "\n\nPdfbox␣Author␣list:␣" + pdfNode.getProperty("PdfboxAuthorList","")
177             + "\n\nRelatives␣nodes'␣Author␣list:␣" + pdfNode.getProperty( "
                    RelAuthorsList","")
                + "\n\nParsCit␣Author␣list:␣" + pdfNode.getProperty("ParsCitAuthorList","")
179             + "\n\nAuthor␣from␣AAPI␣Matched␣in␣First␣Text␣Fragment␣list:␣" + pdfNode.
                    getProperty( "RelAuthorsListMatchedInFirstTextFragment","")
                + "\n\nAuthor␣From␣Other␣Docs␣Matched␣in␣First␣Text␣Fragment␣list:␣" +
                    pdfNode.getProperty( "RelAuthorsFromOthersDocMatched","")
181             + "\n\nTotal␣Author␣list:␣" + pdfNode.getProperty("TotalAuthorsList","")

183             + "\n\n---FILIATION␣FIELDS"
                + "\n\nParsCit␣Filiation␣list:␣" + pdfNode.getProperty("
                    ParsCitFiliationsList","")
185             + "\n\nRelatives␣nodes'␣Filiation␣list:␣" +  pdfNode.getProperty( "
                    RelFiliationsList","")
                + "\n\nFiliation␣from␣AAPI␣Matched␣in␣First␣Text␣Fragment␣list:␣"  + pdfNode
                    .getProperty( "RelFiliationsListMatchedInFirstTextFragment","")
187             + "\n\nFiliation␣From␣Other␣Docs␣Matched␣in␣First␣Text␣Fragment␣list:␣" +
                    pdfNode.getProperty( "RelFiliationsFromOthersDocMatched","")
                + "\n\nTotal␣Filiation␣list:␣" + pdfNode.getProperty("TotalFiliationsList","
                    ")
189
                + "\n\n--}\n\n\n\n\n\n");
191                }
        tx.success();
193       }
          catch(Exception ex){
195        ex.printStackTrace();
           ex.printStackTrace();
197        ex.printStackTrace();
           throw ex;
199       }
          finally
201       {
              tx.finish();
203       }
      }
205   /*
       * Look at the parsCit 'filiations', 'authors', the '
          RelFiliationsListMatchedInFirstTextFragment' and '
          RelAuthorsListMatchedInFirstTextFragment'
207    * properties, and make new nodes filiations/authors (if needed) setting the
          bidirectional rel 'conecction' them and
```

```
     * outgoing relationships 'hasFiliation' and 'hasAuthor' from the pdfNode.
     * */
    private void createAuthorsAndFiliationNodes(Node pdfNode) {
     try {
      List<Node>authorNodes = new ArrayList<Node>();
         List<Node>filiationNodes = new ArrayList<Node>();

      String listValuesStr = pdfNode.getProperty("ParsCitAuthorList","").toString();
      List <String> values = CrawlUtilities.getKeysFromPairList(listValuesStr.
         substring(0,Math.max(listValuesStr.length()-1,0)));
      authorNodes.addAll(this.getOrCreateEntityNodes(values,authorsIndex,EmbeddedNeo4j
         .AUTHOR_NAME,pdfNode,RelTypes.HasAuthor));

      listValuesStr = pdfNode.getProperty("RelAuthorsListMatchedInFirstTextFragment","
         ").toString();
      values = CrawlUtilities.decodeListFromCSVString(listValuesStr);
      authorNodes.addAll(this.getOrCreateEntityNodes(values,authorsIndex,EmbeddedNeo4j
         .AUTHOR_NAME,pdfNode,RelTypes.HasAuthor));

      String filiationAuthorList = pdfNode.getProperty("ParsCitFiliationsList","").
         toString();
      values = CrawlUtilities.getKeysFromPairList(filiationAuthorList.substring(0,Math
         .max(filiationAuthorList.length()-1,0)));
      filiationNodes.addAll(this.getOrCreateEntityNodes(values,filiationIndex,
         EmbeddedNeo4j.FILIATION_NAME,pdfNode,RelTypes.HasFiliation));

      listValuesStr = pdfNode.getProperty("RelFiliationsListMatchedInFirstTextFragment
         ","").toString();
      values =CrawlUtilities.decodeListFromCSVString(listValuesStr);
      filiationNodes.addAll(this.getOrCreateEntityNodes(values,filiationIndex,
         EmbeddedNeo4j.FILIATION_NAME,pdfNode,RelTypes.HasFiliation));
      /*
       * make relationships
       * */
      Iterator<Node>authorIter = authorNodes.iterator();
      while(authorIter.hasNext()) {
       Node author = authorIter.next();
       Iterator<Node>filIter = filiationNodes.iterator();
       while(filIter.hasNext()) {
        Node fil = filIter.next();
        fil.createRelationshipTo(author,RelTypes.HasConnection);
       }
      }
  } catch (Exception e) {
   //todo: handle it!
  }
    }
```

```
247     private List<Node> getOrCreateEntityNodes(List <String> values,Index<Node>
            indexLookup,String propertyName,Node pdfNode, RelTypes relantionshipName ) {
        List<Node>nodes = new ArrayList<Node>();
249       if (values.size()<=0) return nodes;
    Iterator<String> iter = values.iterator();
251 while(iter.hasNext()){
     String text = iter.next().toLowerCase().trim();//avoid different nodes with same
            string but different case
253  if (text.length()<=0) continue;
     IndexHits<Node>matches = indexLookup.query( propertyName,text);
255  Node created;
     if (matches.size() > 0) {
257   created = matches.getSingle();
      nodes.add(created);
259  } else {
     created = EmbeddedNeo4j.factory.getOrCreate( propertyName,text);
261  created.setProperty(propertyName, text);
        nodes.add(created);
263     indexLookup.add( created, propertyName,text);//add to the respective index
     }
265  pdfNode.createRelationshipTo(created, relantionshipName);
    }
267 return nodes;
     }
269
     private void findAuthorsFromOtherDocuments(Node pdfNode) {
271 List<String> listToMatch = new ArrayList<String>();

273 String listValuesStr = pdfNode.getProperty("ParsCitAuthorList","").toString();
    List <String> existingValues = CrawlUtilities.getKeysFromPairList(listValuesStr.
        substring(0,Math.max(listValuesStr.length()-1,0)));
275 listValuesStr = pdfNode.getProperty("RelAuthorsListMatchedInFirstTextFragment","").
        toString();
    CrawlUtilities.unionToFirstListIgnoringCase(existingValues,CrawlUtilities.
        decodeListFromCSVString(listValuesStr));
277
       for ( Node node : authorsIndex.query( EmbeddedNeo4j.AUTHOR_NAME, "*" ) ){
279     String authorName = node.getProperty(EmbeddedNeo4j.AUTHOR_NAME,"").toString();
        if (!CrawlUtilities.listStringContainsStringIgnoringCase(existingValues,
            authorName))
281     listToMatch.add(authorName);
       }
283    String textFragment =  pdfNode.getProperty("TextFragment","").toString();
       List<String> entitiesAuthorsMatched = CrawlUtilities.findMatch(textFragment,
           listToMatch);
285 String strMatched;
    if (entitiesAuthorsMatched.size()>0) {
287  strMatched =  CrawlUtilities.encodeListAsCSVString(entitiesAuthorsMatched);
    } else {
```

```
289      strMatched = "";
         }
291      pdfNode.setProperty("RelAuthorsFromOthersDocMatched", strMatched);

293      //Set Union property
         CrawlUtilities.unionToFirstListIgnoringCase(existingValues,entitiesAuthorsMatched);
295      String unionStr =  CrawlUtilities.encodeListAsCSVString(existingValues);
         pdfNode.setProperty("TotalAuthorsList", unionStr);
297       }

299       private void findFiliationsFromOtherDocuments(Node pdfNode) {
         List<String> listToMatch = new ArrayList<String>();
301
         String listValuesStr = pdfNode.getProperty("ParsCitFiliationsList","").toString();
303      List <String> existingValues = CrawlUtilities.getKeysFromPairList(listValuesStr.
             substring(0,Math.max(listValuesStr.length()-1,0)));
         listValuesStr = pdfNode.getProperty("RelFiliationsListMatchedInFirstTextFragment","
             ").toString();
305      CrawlUtilities.unionToFirstListIgnoringCase(existingValues,CrawlUtilities.
             decodeListFromCSVString(listValuesStr));
            for ( Node node : filiationIndex.query( EmbeddedNeo4j.FILIATION_NAME, "*" ) ){
307            String authorName = node.getProperty(EmbeddedNeo4j.FILIATION_NAME,"").
                   toString();
             if (!CrawlUtilities.listStringContainsStringIgnoringCase(existingValues,
                 authorName))
309          listToMatch.add(authorName);
            }
311        String textFragment =  pdfNode.getProperty("TextFragment","").toString();
            List<String> entitiesMatched = CrawlUtilities.findMatch(textFragment,
               listToMatch);
313      String strMatched;
         if (entitiesMatched.size()>0) {
315       strMatched =  CrawlUtilities.encodeListAsCSVString(entitiesMatched);
         } else {
317       strMatched = "";
         }
319      pdfNode.setProperty("RelFiliationsFromOthersDocMatched", strMatched);

321      //Set Union property
         CrawlUtilities.unionToFirstListIgnoringCase(existingValues,entitiesMatched);
323      String unionStr = CrawlUtilities.encodeListAsCSVString(existingValues);
         pdfNode.setProperty("TotalFiliationsList", unionStr);
325       }

327    /* Labeling: here we just look at the html parent PDFFile's text
        *
329     */
         private void performLabelingJob(Node pdfNode) {
```

```
331         Iterator<TLOLabel> iter = Parameters.getInstance().labelingList().iterator();
                //Labeling parameters
            String category = "";
333         String subCategory = "";
            for ( Node parentNode : EmbeddedNeo4j.allDbPagesIndex.query( EmbeddedNeo4j.
               FIELDNAME_KEY, pdfNode.getProperty("KeyParentDbPage",""))) {//this query
               returns at most one entry
335          String previousSearchCategory = (String) parentNode.getProperty("
                categoryMatched", "");
             String previousSearchSubCategory = (String) parentNode.getProperty("
                subcategoryMatched", "");
337          if (previousSearchCategory!=null && previousSearchCategory.length()>0) {
              category = previousSearchCategory.equalsIgnoreCase("none")?"":
                 previousSearchCategory;
339           subCategory = previousSearchSubCategory.equalsIgnoreCase("none")?"":
                 previousSearchSubCategory;
             } else {
341           String strToScrap1 = (String) parentNode.getProperty("PlainText","");
                String strToScrap2 = (String) pdfNode.getProperty("TextFragment","");
343             while(iter.hasNext()) {
                 TLOLabel label = iter.next();
345              Matcher m = label.getRegExpPattern().matcher(strToScrap1);
                 boolean findSuccess = false;
347              if (m.find())
                  findSuccess = true;
349              else {
                  m = label.getRegExpPattern().matcher(strToScrap2);
351               if (m.find())
                   findSuccess = true;
353              }
                 if (findSuccess) {
355               category+=label.category + ";";
                  subCategory+=label.subcategory + ";";
357              }
                }
359           //persist the match to avoid search again on eventually brother node
              parentNode.setProperty("categoryMatched", category.length()>0?category:"
                 none");
361           parentNode.setProperty("subcategoryMatched", subCategory.length()>0?
                 subCategory:"none");
             }
363          //finally set the category and sub-category fields
             pdfNode.setProperty("category", (category.length()>0)?category:"");
365         pdfNode.setProperty("subCategory",(subCategory.length()>0)?subCategory:"");
            }
367     }
}
```

A.5. Configuración caso de prueba 1

Archivo 5: Paramameters.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<general>

  <crawlerParameters>
   <SEEDS>http://www.fceia.unr.edu.ar/~acasali/</SEEDS>
 <ONLY_SUBPATHS>true</ONLY_SUBPATHS>
  </crawlerParameters>

  <parsCit_Home>
  /home/zeta/Documentos/ParsCit-master/bin
  </parsCit_Home>

  <neo4j_Home>
  /home/zeta/Documentos/neo4j-community-2.0.0
  </neo4j_Home>

  <postCrawlGraphTraversal>
 <MAX_DEPTH_EMAILLIST>5</MAX_DEPTH_EMAILLIST>
 <KEY_MAX_DEPTH_POTENTIALAUTHORLIST>3</KEY_MAX_DEPTH_POTENTIALAUTHORLIST>
 <KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST>0.0</
    KEY_CONFIDENCE_LOWER_BOUND_POTENTIALAUTHORLIST>
 <MAX_POTENTIALFILIATIONLIST>3</MAX_POTENTIALFILIATIONLIST>
 <KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIATIONLIST>0.0</
    KEY_CONFIDENCE_LOWER_BOUND_POTENTIALFILIATIONLIST>
  </postCrawlGraphTraversal>

  <stopWords_authors>
  el,la,las,del,lo,los,del,rosario,práctica,et al,tla,vocabulario
  </stopWords_authors>

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

  <postCrawlLabeling>
    <label>
        <category>Material Educativo</category>
        <subcategory>Práctica</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
        <matchingWords>práctica,prácticas,ejercicio,ejercicios,ejercitación,práctico</
           matchingWords>
        <shouldApplyParsCitRule>false</shouldApplyParsCitRule>
 </label>
 <label>
```

```
42          <category>Material Educativo</category>
            <subcategory>Apunte de Clase</subcategory>
44          <maxCharactersInDoc>1500</maxCharactersInDoc>
            <matchingWords>apunte,apuntes,notas,material,clase,capítulo,capítulos</
               matchingWords>
46   <shouldApplyParsCitRule>false</shouldApplyParsCitRule>
  </label>
48  <label>
            <category>Tesis</category>
50          <subcategory>null</subcategory>
            <maxCharactersInDoc>1500</maxCharactersInDoc>
52          <matchingWords>tesis,tesina</matchingWords>
   <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
54  </label>
  <label>
56          <category>Publicación</category>
            <subcategory>null</subcategory>
58          <maxCharactersInDoc>1500</maxCharactersInDoc>
            <matchingWords>publicación,publicaciones</matchingWords>
60   <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
  </label>
62    </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/zeta/Documentos/ParsCit-master/bin
11   </parsCit_Home>

13   <neo4j_Home>
     /home/zeta/Documentos/neo4j-community-2.0.0
15   </neo4j_Home>

17   <postCrawlGraphTraversal>
  <max_depth_emaillist>5</max_depth_emaillist>
```

```
<key_max_depth_potentialauthorlist>3</key_max_depth_potentialauthorlist>
<key_confidence_lower_bound_potentialauthorlist_aapi>0.0</
   key_confidence_lower_bound_potentialauthorlist_aapi>
<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>

<max_potentialfiliationlist>3</max_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>

 <stopWords_authors>
 el,la,las,del,lo,los,ese,esa,del,rosario,mit,tesis,tésis,unidad,páginas,práctica,
    practica,prácticas,et al,clements,vocabulario,for
 </stopWords_authors>

 <stopWords_filiations>
 el,la,las,del,lo,los,ese,esa,del,csp,abril,mit,tesis,tésis,unidad,páginas,práctica,
    practica,prácticas,et al,for,vocabulario
 </stopWords_filiations>

 <postCrawlLabeling>
   <label>
       <category>Material Educativo</category>
       <subcategory>Práctica</subcategory>
       <maxCharactersInDoc>1500</maxCharactersInDoc>
       <matchingWords>práctica,prácticas,ejercicio,ejercicios,ejercitación,práctico</
          matchingWords>
       <shouldApplyParsCitRule>false</shouldApplyParsCitRule>
</label>
<label>
       <category>Material Educativo</category>
       <subcategory>Apunte de Clase</subcategory>
       <maxCharactersInDoc>1500</maxCharactersInDoc>
       <matchingWords>apunte,apuntes,notas,material,clase,capítulo,capítulos</
          matchingWords>
 <shouldApplyParsCitRule>false</shouldApplyParsCitRule>
</label>
<label>
       <category>Tesis</category>
       <subcategory>null</subcategory>
       <maxCharactersInDoc>1500</maxCharactersInDoc>
```

```
        <matchingWords>tesis,tesina</matchingWords>
 <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
</label>
<label>
        <category>Publicación</category>
        <subcategory>null</subcategory>
        <maxCharactersInDoc>1500</maxCharactersInDoc>
        <matchingWords>publicación,publicaciones</matchingWords>
 <shouldApplyParsCitRule>true</shouldApplyParsCitRule>
</label>
 </postCrawlLabeling>
</general>
```