Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Diseño e Implementación de Mecanismos de Búsqueda Contextualizada y Anotado a través de la Web Semántica

Tesis que presenta

José Manuel Romero Chávez

para obtener el Grado de

Maestro en Ciencias en Computación

Directores de Tesis: **José Guadalupe Rodríguez García**

Maricela Claudia Bravo Contreras

México, Distrito Federal **Octubre 2012**

Resumen

Actualmente los motores de búsqueda más populares (Google por ejemplo) realizan recorridos a través de toda la Web, con el objetivo de indexar las páginas y documentos disponibles en la red, para proporcionar mejores resultados a las consultas del usuario; sin embargo, considerando el ritmo de crecimiento de la Web y la heterogeneidad de la misma, este recorrido resulta costoso en recursos como memoria, procesamiento, ancho de banda y almacenamiento. Los *crawlers*[1] focalizados abordan de cierta manera esta problemática permitiendo hacer exploraciones enfocadas a un tema específico.

En el presente proyecto de tesis se presenta el diseño y la implementación de un mecanismo de búsqueda que permite encontrar la información acorde al contexto semántico del usuario. Este mecanismo hace uso de dos *crawlers* específicos para llevar a cabo la extracción de ontologías y documentos de texto para su posterior organización de forma semántica. Se aborda el problema de la heterogeneidad de documentos existentes en la Web en la medida de lo posible. El caso de uso es la reorganización de la información científica (artículos, libros, etc.) a la que tiene acceso la comunidad del CINVESTAV con la finalidad de facilitar el manejo de la misma.

[1]Buscadores que rastrean las páginas web en busca de información (araña Web)

Abstract

In order to index the pages and files available on the network and provide better results for user queries, the most used search engines (Google for example) make explorations across the Web. However, considering the Web's growth rate and its heterogeneity, these explorations are costly in resources like memory, processor, bandwidth and storage. Focused crawlers address these problems allowing to execute more focused explorations on the Web.

In this thesis project we propose the implementation of a search mechanisms which allow us to find information according to the user's semantic context. We propose the use of two focused crawlers for the extraction of ontologies and text documents, the information retrieved by the crawlers is organized in a semantic structure. With this proposal we pretend to adress the problem of heterogeneity of documents found on the Web. A use case is presented related with the reorganization of scientific information that is accessed by the community of CINVESTAV in order to facilitate the management of all that information.

Agradecimientos

Doy gracias a mi familia por el apoyo que me han proporcionado en estos años en que he estado haciendo este posgrado, por sus enseñanzas, que me han servido para tratar a diario de ser una mejor persona y de hacer lo mejor posible mi trabajo.

AL CINVESTAV IPN, por proveerme de las instalaciones y el equipo apropiado para el desarrollo de mi proyecto. A mis compañeros y profesores por el ambiente de convivencia con ellos.

A Sofía Reza por su apoyo y ayuda, a los profesores de los seminarios de tesis por sus consejos en la correcta escritura del documento.

A mis asesores de tesis, los doctores Maricela Claudia Bravo Contreras y José Guadalupe Rodríguez García, por proporcionarme la orientación adecuada en el proyecto y poner a mi disposición su experiencia y conocimientos.

A mis revisores de tesis, los doctores Iván López Arevalo, Sonia Guadalupe Mendoza Chapa y María Lizbeth Gallardo López por tomarse el tiempo de examinar mi documento y aconsejarme en la correcta redacción del mismo.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo financiero que se me otorgo durante el posgrado.

Tabla de contenido

Índice de figuras

Índice de tablas

Índice de algoritmos

Índice de listados

Capítulo 1

Introducción

Desde sus orígenes la Web ha estado en constante crecimiento. La cantidad de información disponible es cada vez mayor y existe una clara tendencia a migrar muchos sistemas conocidos a la nube[1]. Este crecimiento implica afrontar nuevos retos en la búsqueda y recuperación de información.

Actualmente los buscadores más populares (Google por ejemplo) realizan recorridos a través de toda la Web para indexar las páginas y proveer de respuestas rápidas a los usuarios. Sin embargo, tales respuestas están constituidas de miles o millones de sitios sugeridos para los tópicos y el orden en el que aparecen no siempre facilita la recuperación de la información que el usuario requiere.

Existen otros enfoques que buscan mejorar la calidad de estos resultados mediante minado de la información, con el objetivo de determinar si la información realmente es relevante. Con esto se pretende disminuir el número de respuestas y aumentar la calidad de las mismas. Sin embargo, estos enfoques carecen de rapidez en la recuperación.

Por otro lado existe mucho interés e investigación en la creación de la *Web Semántica* [Daconta et al., 2003], cuya información se encuentre anotada con metadatos ontológicos. Con esto se pretende mejorar la interoperabilidad entre sistemas y mejorar la recuperación de información de manera automática.

En el presente proyecto de tesis se describe el desarrollo de un mecanismo de recuperación de información que es capaz de proporcionar resultados más acorde a las necesidades del usuario. Además, el desarrollo de un mecanismo de anotado semántico que enriquece la información recuperada mediante ontologías. Entonces, la implementación es capaz de generar un repositorio de documentos enriquecidos de forma semántica, con lo cual se facilita y agiliza su acceso posterior de forma automática.

En este capítulo se exponen el planteamiento del problema a resolver, los objetivos

[1]Nube se refiere de manera figurada a la red Internet

del proyecto y se explica la organización de los capítulos posteriores.

1.1 Planteamiento del problema

El presente proyecto se enfoca en el diseño e implementación de un mecanismo de búsqueda contextualizado, es decir, se diseñó y desarrolló un software capaz de extraer información textual y semántica de páginas web, de acuerdo a un contexto proporcionado por un conjunto de palabras clave, y de organizarla de forma semántica, utilizando ontologías para facilitar su acceso y aprovechamiento posterior. El mecanismo de búsqueda puede modelar un dominio de conocimiento con la información extraída.

El problema que se aborda en esta tesis se resume en la siguiente pregunta:

¿Puede la implementación de un mecanismo de búsqueda contextualizada facilitar y agilizar el acceso a la información disponible en la Web extrayendo la información (usando crawlers) y organizándola de forma semántica (mediante ontologías) para poblar una ontología global que modele el dominio específico de conocimiento establecido por el contexto?

La pregunta planteada es digna de investigación, ya que la respuesta permite probar en un ambiente real el alcance de las ontologías para el manejo y modelado de información; además el resultado agiliza el acceso a la información reduciendo los tiempos de búsquedas bibliográficas.

1.2 Objetivo general y objetivos específicos

General

El objetivo general del presente proyecto es la construcción de un mecanismo de búsqueda que, mediante un conjunto de palabras clave y un tema específico, recopile información de la Web (ontologías y documentos de texto como HTML y PDF) referente al contexto establecido a través de las palabras clave proporcionadas y que permita la elaboración de una ontología, que modele el dominio de conocimiento en cuestión, permitiendo la reutilización de las ontologías obtenidas de la Web para complementar dicho conocimiento.

Particulares

- Elaborar las herramientas para la extracción de documentos de la Web (HTML y PDF) y de ontologías.

- Crear un mecanismo que permita la correcta interpretación de la información obtenida, a través de la búsqueda dentro de los documentos de términos específicos del dominio de conocimiento que se esté tratando.
- Crear de mecanismo que permita la explotación de la información obtenida (minado de documentos recuperados).
- Desarrollar un mecanismo para el anotado semántico de los documentos.
- Organizar la información a través de ontologías.
- Construir una ontología global que modele de forma adecuada el dominio de conocimiento tratado.

1.3 Organización de la tesis

La presente tesis se organiza en 6 capitulos con la siguiente estructura:

- **Capítulo 2:** presenta los fundamentos teóricos relacionados con la temática de esta investigación, además de una serie de trabajos relacionados que fueron presentados hasta la fecha de realización de este proyecto.
- **Capítulo 3:** desarrolla la propuesta de solución planteada y muestra los detalles de diseño de cada componente del sistema, utilizando el lenguaje de modelado estándar UML.
- **Capítulo 4:** describe los detalles de implementación y la infraestructura utilizada.
- **Capítulo 5:** explica cada una de las pruebas realizadas al sistema y se reportan los resultados obtenidos en las mismas.
- **Capítulo 6:** muestra las conclusiones generales obtenidas de este proyecto de investigación y describe el trabajo futuro.

Capítulo 2

Estado del arte y trabajos relacionados

La evolución de la Web, desde su creación [Berners-Lee et al., 1992] hasta la actualidad, ha dejado ver una clara tendencia hacia el manejo de los sistemas en la nube. Esto nos permite administrar los recursos de software sin depender de una ubicación física. Sin embargo, el crecimiento que ha tenido la Web ha aumentado la complejidad de la búsqueda y entrega de información a los usuarios, por ejemplo *worldwidewebsize*[1] estima que existen alrededor de 7.39 mil millones de páginas indexadas. Los buscadores más populares hoy en día realizan periódicamente recorridos en toda la Web para indexar los contenidos de las páginas y así poder dar respuestas más rápidas al usuario. Este tipo de recorridos (hechos mediante *crawlers*) consumen muchos recursos computacionales y deben enfrentar el constante crecimiento y comportamiento dinámico de la Web.

Con el objetivo de afrontar esta problemática han surgido algunos enfoques que pretenden reducir estas caminatas en la Web con el uso de *crawlers* focalizados, es decir, *crawlers* que recorran un subconjunto de páginas de acuerdo a ciertos tópicos. También hay una fuerte tendencia hacia una **Web semántica** en donde el contenido se encuentra enriquecido con descripciones semánticas y es más fácil de entender y manejar, esta representación del conocimiento se hace mediante ontologías.

En este capítulo se describen los principales conceptos relacionados con esta investigación, así como una revisión de los trabajos relacionados que se han publicado en los ultimos años. Se abordan primero los conceptos teóricos (referentes las los temas de *ontologías, web crawlers, minería de datos, minería web y sistemas conscientes del contexto*), posteriormente se presentan los trabajos relacionados con el área y por último una discusión acerca del estado del arte relacionado al proyecto.

[1]Sitio www.worldwidewebsize.com que lleva un conteo estadístico de las páginas Web indexadas (consultado el 18 de septiembre del 2012)

2.1 Ontologías

La palabra ontología tiene su origen en la filosofía. Aristóteles la definió como la ciencia del ser o una explicación sistemática de la existencia [Staab and Studer, 2004, Gómez-Pérez, 1999]. En el campo de la Inteligencia Artificial fueron Neches *et al.* quienes definieron el término ontología como "Una ontología define los términos básicos y las relaciones que comprende el vocabulario de un área temática, así como las reglas para combinar términos y relaciones para definir extensiones para el vocabulario"[Neches et al., 1991]. Tiempo después se propusieron algunas modificaciones a la definición citadas a continuación:

- Gruber *et al.* la definen como "Una especificación explícita de una conceptualización"[Gruber et al., 1993].

- Borst hace cambios a la definición proporcionada por Gruber y establece el término ontología como "Una especificación formal de una conceptualización compartida"[Borst, 1997].

La definición dada por Gruber ha sido la más citada en la literatura referente a ontologías. R. Studer *et al.* sugieren que la esencia de las ontologías está basada en las dos definiciones anteriores y explican los términos clave contenidos en ellas como sigue [Studer et al., 1998]:

- **Conceptualización:** se refiere a un modelo abstracto de un fenómeno del mundo real teniendo en cuenta los conceptos relevantes de dicho fenómeno.

- **Explícita:** significa que los conceptos usados y las limitaciones de su uso están explícitamente definidos.

Las ontologías representan un avance importante en el manejo semántico de la información, ya que proveen una forma de estructurar el conocimiento explícitamente, sin dejar lugar a ambigüedades, y permitiéndonos modelar dominios del conocimiento de manera estándar [Uschold and Grüninger, 1996], al proporcionar un entendimiento compartido.

Retomando la definición de Borst, que sugiere que la especificación es *formal*, sabemos entonces que una ontología es legible por una máquina y por lo tanto, además de proporcionar un entendimiento compartido, también facilita el intercambio de información entre sistemas. En referencia a la Web semántica ya se han propuesto herramientas de desarrollo [Motik et al., 2008].

Gómez-Pérez y Corcho mencionan algunos lenguajes que son utilizados para la construcción de ontologías [Gómez-Pérez and Corcho, 2002], de los cuales se incluye una breve descripción:

- **XOL (*XML-based Ontology Exchange Language*):** fue diseñado por la comunidad de bioinformática de los Estados Unidos, para hacer el intercambio de ontologías entre un conjunto heterogéneo de sistemas de software del dominio de bioinformática. Se seleccionaron *Ontolingua* y OML (*Ontology Markup Language*) como base para su creación.

- **SHOE (*Simple HTML Ontology Extension*):** fue desarrollado en la universidad de de Maryland como una extensión de HTML (*HyperText Markup Language*), incorporando conocimiento semántico legible por maquina en documentos web.

- **OML (*Ontology Markup Language*):** fue desarrollado en la universidad de Washington, está parcialmente basado en SHOE.

- **RDF (*Resource Description Framework*):** desarrollado por el W3C para describir recursos de la Web. Permite la especificación semántica de datos basado en XML de una forma estandarizada e interoperable.

- **OIL (*Ontology Interchange Language*):** desarrollado en el proyecto OntoKnowledge, permite la interoperabilidad entre recursos web.

- **DAML+OIL (*DARPA Agent Markup Language+OIL*):** desarrollado por un comité conjunto de los Estados Unidos y la Unión Europea, para permitir la interoperabilidad semantica en XML.

Ademas de los lenguajes mencionados por Gómez-Pérez existe OWL (*Web Ontology Language*), que es un lenguaje de marcado semántico para publicar y compartir ontologías en la Web, está desarrollado como una extensión del vocabulario de RDF y derivado de DAML+OIL.

2.2 Web *crawlers*

Los Web *crawlers* (también conocidos como robots o arañas [Liu, 2006]) son programas que se caracterizan por explorar de forma automática los sitios Web, analizar su información y hacer exploraciones de las páginas que encuentran referenciadas mediante una URL. Esta definición se ajusta al concepto más general o básico de un *crawler*, donde el proceso se repite hasta que algún objetivo se alcanza (por ejemplo una cantidad límite de páginas visitadas).

La Web puede ser vista como un grafo dirigido, donde cada nodo representa una página Web y los hipervínculos representan los arcos o aristas que enlazan dichos nodos. En la Figura 2.1 se puede observar una representación simbólica de la estructura de la Web. Puede observarse que el grafo presenta ciclos, esto se debe a que una página Web puede tener un hipervínculo hacia cualquier otra en la Web incluyéndose

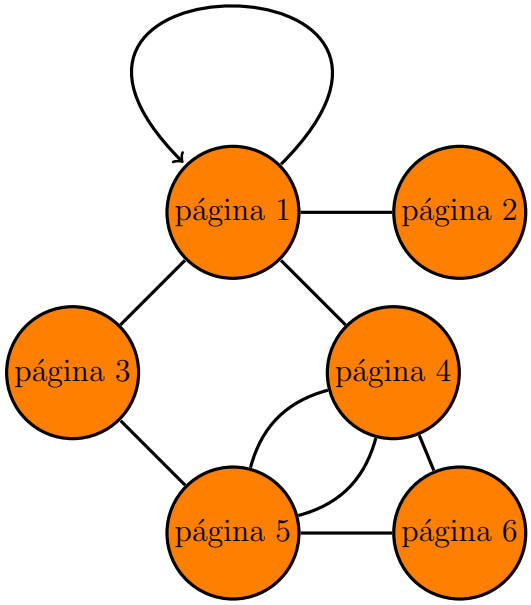


Figura 2.1: Representación simbólica de la Web

a sí misma, tal característica complica los recorridos de los *crawlers*.

Los *crawlers* han sido mejorados para realizar sus recorridos de forma más eficiente en la Web, considerando distintos intereses en la recuperación de información. A continuación se describen los principales tipos de *crawlers* y sus principales características:

2.2.1 *Crawlers* generales

También conocidos como *crawlers* de propósito general, son la forma más básica de un *crawler* y se caracterizan por visitar todos los sitios y páginas Web, sin hacer ninguna distinción entre ellos. De manera general puede describirse el funcionamiento de estos *crawlers* de acuerdo a los siguientes pasos:

1. Recibe un conjunto de URLs semillas[2] y son colocadas en una cola.

2. Por cada URL en la cola descarga la página correspondiente, obtiene las URLs que ésta contiene y las agrega a la cola para procesarlas después.

Este proceso se repite hasta que se alcance un límite previamente fijado.

Al no hacer distinción alguna de los documentos, estos *crawlers* presentan grandes desventajas, por ejemplo: debido a que no pueden determinar la relevancia de los

[2]URLs a partir de las cuales un *crawler* inicia la exploración

documentos, se hacen descargas indiscriminadas incurriendo en un alto gasto de ancho de banda y espacio de almacenamiento. Además los recorridos realizados pueden descargar documentos con muy poca o ninguna relación entre sí, debido a la estructura de hipervínculos la Web.

2.2.2 *Crawlers* focalizados

Estos *crawlers* poseen una estrategia diferente a los mencionados en el apartado 2.2.1. De acuerdo con Liu, hacen la implementación de la frontera como una cola de prioridades, es decir, asignan una prioridad a cada enlace no visitado basado en un estimado del valor de la página enlazada [Liu, 2006]. Los *crawlers* focalizados tienen como objetivo recuperar documentos relacionados a un conjunto de palabras clave de la manera más precisa posible.

Con este tipo de *crawlers* se busca determinar el grado de relevancia de una página antes de descargarla y así evitar descargas innecesarias. Se han buscado varias formas de mejorar estos *crawlers*, por ejemplo, Dong *et al.* proponen una clasificación de los *crawlers* focalizados semánticos [Dong et al., 2009], ellos establecen los siguientes grupos o clases para diferenciarlos:

- ***Crawlers* focalizados basados en ontologías:** en este grupo se encuentran los *crawlers* que hacen uso de ontologías para vincular un documento Web con un conjunto de tópicos, el objetivo es categorizar dicho documento y filtrar páginas no relevantes al dominio de la ontología.
- ***Crawlers* focalizados por abstracción de metadatos:** son *crawlers* que, además de hacer un filtrado de documentos no relevantes, pueden abstraer y anotar metadatos de los documentos recuperados.
- **Otros:** en esta clase los autores colocan a los *crawlers* que no se apoyan en ontologías ni metadatos [Betin-Can and Baykal, 2007, Liu et al., 2004].

Los *crawlers* focalizados presentan como desventaja que el cálculo de la relevancia de los documentos puede llegar a ser costosa en procesamiento, aumentando con ello el tiempo de *crawling*. Aunque el cálculo de relevancia es una actividad necesaria en el *crawler*, ésta se señala como desventaja porque existen distintos tipos de comparaciones, de las cuales algunas incurren en un mayor gasto de procesamiento que otras y es necesario tener una conpensación entre el tiempo de *crawling* y la calidad de de los resultados.

2.3 Minería de datos

Según Liu la minería de datos, también llamada KDD (Knowledge Discovery in Databases), se define comúnmente como: "el proceso de descubrimiento de patrones útiles o conocimiento de fuentes de datos (bases de datos, textos, imágenes

etc.)"[Liu, 2006]. Sumathi y Sivanandam definen esta disciplina como "el proceso de descubrir nuevas correlaciones significativas, patrones, tendencias mediante exploración en grandes conjuntos de datos almacenados en depositos, utilizando estadística, aprendizaje de máquina, Inteligencia Artificial y técnicas de visualización de datos"[Sumathi and Sivanandam, 2006]. Puede notarse que las definiciones citadas difieren básicamente en el nivel de detalle con que son descritas. En este sentido la segunda definición nos provee un panorama más amplio de lo que es minería de datos.

El análisis de los datos que se hace mediante esta disciplina permite aprovechar de mejor forma la información que se encuentra implícita en estas fuentes de datos, posibilitando hacer predicciones y sirviendo como soporte a la toma de decisiones. Liu establece tres pasos o etapas que comprenden el proceso de minado de datos [Liu, 2006]:

- **Pre-procesado:** los datos en bruto pueden no ser adecuados para realizar el minado debido a que puede ser muy grande la cantidad de los mismos y pueden existir atributos irrelevantes al estudio. Esta etapa se encarga de hacer una reducción de los datos mediante muestreo y eliminación de atributos no necesarios, además en este paso se hace remoción de ruido y anormalidades presentes en los datos.
- **Minado de datos:** un algoritmo de minado de datos lleva a cabo la extracción de patrones y/o conocimiento de los datos procesados.
- **Post-procesado:** la utilidad de los patrones encontrados depende del dominio de la aplicación, por lo que es necesario utilizar algunas técnicas de visualización y evaluación para determinar cuáles son útiles a dicha aplicación.

Todo el proceso que implica minería de datos se hace de manera iterativa y normalmente se requieren muchas repeticiones para alcanzar resultados satisfactorios e incorporables a la toma de desiciones en problemas reales.

Fayyad *et al.* hacen una diferencia entre KDD y minería de datos, afirman que KDD se refiere a todo el proceso de descubrimiento de conocimiento útil de los datos [Fayyad et al., 1996], mientras que la minería de datos es un paso particular de esta tarea. De acuerdo con ellos la minería de datos es la aplicación de algoritmos específicos para la extracción de patrones presentes en los datos. De esta manera mencionan que KDD cuenta con otros pasos ademas de la minería de datos:

- Preparación de los datos
- Selección de los datos
- Limpieza de los datos
- Incorporación de los conocimientos previos adecuados
- Interpretación apropiada de los resultados obtenidos

2.4 Minería Web

Con el aumento de información disponible en la Web, se hace cada vez más común el uso de este medio para consulta y búsqueda por parte de los usuarios. Según Liu la búsqueda web tiene sus bases en la recuperación de información (IR) [Liu, 2006], la cual se refiere a encontrar un conjunto de documentos que son relevantes a la consulta del usuario. De manera técnica se establece que IR se ocupa de la adquisición, organización, almacenamiento, recuperación y distribución de la información, donde se enfatiza como unidad básica a un documento.

La minería web se refiere al uso de las técnicas conocidas en minería de datos para la extracción de información útil de la Web (contenido de páginas Web, datos de uso de la Web, entre otros) [Kosala and Blockeel, 2000]. Sin embargo, la naturaleza dinámica y exponencialmente creciente que posee la Web complica el uso de algunas técnicas tradicionales conocidas en esta disciplina, lo cual ha llevado a la creación de algoritmos de minado propios de la minería Web [Liu, 2006].

La principal diferencia entre el minado de datos y el minado Web es que en el primero los datos ya se encuentran disponibles de forma estática en un almacén (base de datos por ejemplo), mientras que en la Web los datos son muy dinámicos y se presentan en una amplia variedad de formatos y por lo tanto es necesario hacer extracciones de tales datos periódicamente [Liu, 2006].

En [Liu, 2006, Kosala and Blockeel, 2000] establecen tres categorías para clasificar las tareas que le corresponden a la minería Web:

- **Minado de la estructura:** se refiere al análisis de hipervínculos que conforman la estructura de la Web (véase la Figura 2.1). De dicho análisis puede obtenerse información útil como: descubrimiento de páginas Web importantes y/o comunidades de usuarios compartiendo intereses comunes.

- **Minado de contenido:** esta tarea es similar a las tareas tradicionales que conciernen a la minería de datos. Ésta se ocupa de minar el contenido de páginas web en busca de patrones para clasificarlas y agruparlas de acuerdo a los tópicos tratados en ellas. Además pueden minarse las revisiones o comentarios que hacen los consumidores de productos o contenidos ofrecidos por la Web, con el objetivo de conocer las preferencias del usuario.

- **Minado de uso:** esta tarea se refiere a la extracción de datos de navegación del usuario para el descubrimiento de sus patrones de acceso a las páginas Web, lo cual puede llevarse a cabo realizando un registro de los *clicks* que hace el usuario sobre los hipervínculos disponibles.

2.5 Sistemas conscientes del contexto

Los humanos son capaces de determinar la situación implícita cuando establecen una comunicación; sin embargo, no sucede lo mismo cuando se interactua con una máquina. Abowd *et al.* definen contexto como: "cualquier información que puede ser utilizada para caracterizar la situación de una entidad"[Abowd et al., 1999], donde entidad puede referirse a: persona, lugar u objeto (físico o computacional). En el área de la computación la conciencia del contexto (*context-awareness*) se refiere a la capacidad de las computadoras de sensar y reaccionar de acuerdo a su ambiente.

Los mismos autores mencionan tres comportamientos importantes de *context-awareness*, que son la presentación de la información y los servicios al usuario, la ejecución automática de un servicio y el etiquetado de contexto a la información para su recuperación posterior. Además identifican como principales retos en el area:

- Desarrollo de una taxonomia y representación uniformes de los tipos de contexto.
- Infraestructura para promover el diseño, implementación y evolución de las aplicaciones conscientes del contexto.
- Imponer aplicaciones conscientes del contexto que asistan las interacciones con los servicios de cómputo ubicuo.

Estos sistemas son en su mayoría desarrollados para ambientes móviles. De manera general, estos sistemas se caracterizan por actuar de acuerdo al contexto del usuario (el cual puede variar). La computación consciente del contexto fue introducida por Mark Weiser, donde afirma que la tecnología más inteligente es aquella que desaparece, es decir, que se hace invisible al usuario [Weiser, 1991]. Hoareau y Satoh establecen que la computación consciente del contexto debe ocuparse de [Hoareau and Satoh, 2009]:

- Adquisición del contexto.
- Abstracción y entendimiento del contexto.
- Aplicación de un comportamiento basado en el contexto reconocido.

2.6 Trabajos relacionados

En esta sección del documento se describen los trabajos relacionados con el presente proyecto y las aportaciones de cada uno de ellos. Tales trabajos serán descritos en orden de acuerdo a su semajanza con el proyecto, es decir, aquellos considerados más parecidos serán descritos primero. Estos trabajos fueron seleccionados debido a la cercanía con los temas desarrollados en esta tesis, que de manera general son: recuperación de información textual y semántica, minado de textos, anotado semántico de documentos y poblado de ontologías. Existen trabajos presentados en esta sección

cuyos enfoques difieren del trabajo realizado en esta tesis, sin embargo, fueron incluidos para presentar un panorama general del estado del arte de este proyecto.

Luong *et al.* propusieron y construyeron un *framework* para el aprendizaje y el enriquecimiento de ontologías, utilizando como dominio de conocimiento la morfología de los anfibios [Luong et al., 2009]. Los autores resaltan el uso de SVM (*Support Vector Machine*) para hacer el filtrado de documentos no relevantes y la alta precisión obtenida (sobre el 77.5 %). Además señalan que dicho *framework* permite el poblado de ontologías tanto de forma automática como semi-automática.

El funcionamiento de este *framework* se resume en las siguientes actividades propuestas por los autores:

- Construcción de una ontología pequeña referente al dominio y generación automática de consultas para cada concepto.
- Uso de un *crawler* focalizado para envíar las consultas a motores de búsqueda y bibliotecas digitales.
- Aplicar SVM para filtrar documentos que coinciden con la consulta pero son menos relevantes al dominio.
- Uso de un sistema de extracción de información para minar los documentos recuperados y enriquecer la ontología con sus contenidos.

En este trabajo el uso de SVM representa una desventaja, debido a la necesidad de entrenar previamente al algoritmo con muestras etiquetadas para obtener una buena precisión. Además, dado que SVM es el que determina la relevancia de los documentos, se hacen descargas innecesarias en el *crawler* previo al uso de SVM.

Van de Maele *et al.* proponen un *crawler* focalizado guiado por ontologías, este *crawler* fue diseñado para indagar en la Web semántica en busca de ontologías [Van de Maele et al., 2008]. Sin embargo, debido a que la disponibilidad de las ontologías en la Web semántica aún es limitada, el *crawler* se evaluó de acuerdo a su capacidad para recuperar "buenas"páginas. De manera general el *framework* consiste de modelar una ontología inicial utilizando el *framework* DOGMA *Studio Workbench*, y dicha ontología establece el dominio de interes para el *crawler*. El *crawler* funciona de manera cíclica, obteniendo las páginas, preprocesándolas y calculando su relevancia en comparación con la ontología de tópicos previamente creada. Los autores resaltan el uso de DMatch *Ontology Matcher* para comparar los documentos web semánticos con la ontología.

Este trabajo carece de una evaluación adecuada del framework. Se partió de la suposición de que si era capaz de obtener "buenas páginas"lo haría tambien con ontologías; además los autores mencionan que no se calcularon las medidas *precision* y

recall [1], debido a que éstas requieren un ambiente bien controlado y definido para el *crawler*.

Dahiwale *et al.* proponen un nuevo algoritmo de *crawling* con la finalidad de abordar el reto de priorizar la cola de URLs para recuperar las páginas más relevantes basándose en una ontología dependiente del dominio, además exploran la posibilidad de utilizar la naturaleza semántica de la URL para determinar la relevancia [Dahiwale et al., 2010]. El algoritmo que proponen queda descrito en los siguientes pasos:

1. Recorre el árbol de la ontología para determinar la trayectoria y la almacena.
2. Toma la URL y descarga la página asociada
3. Extrae todos los vínculos de la página y los guarda en la cola de URLs (frontera).
4. Verifica la asociatividad de las palabras clave del recorrido en la URL.
5. Toma la URL de la frontera para procesarla
6. Encuentra la métrica de asociación (métrica de similaridad) de la URL en la cola.
7. Si obtuvo una URL más relevante, descarga la página asociada, extrae todos los vínculos y anula la cola de URLs con la nueva.
8. Repite el proceso de verificación de las URLs hasta que todas las URLs en la frontera sean las más relevantes.

Los autores afirman que su algoritmo es superior a otros propuestos en el sentido que no necesita hacer uso de la relevancia de la retroalimentación ni de una fase de entrenamiento. Señalan también que hubo una reducción notable en la cantidad de documentos recuperados y en el tiempo requerido en el proceso de *crawling*.

Dong *et al.* implementaron un *crawler* focalizado basado en una ontología de servicio de transporte. En dicho *crawler* se identifican 3 partes principales: un *crawler* focalizado, una base ontológica y una base de metadatos de servicio de transporte [Dong et al., 2008]. El flujo de trabajo general consiste de los siguientes pasos.

1. El *crawler* focalizado descarga todas las páginas web de un sitio en Internet.
2. El *crawler* extrae la información de las páginas descargadas y la información de desempeño de la base de metadatos de servicios de transporte; tales metadatos son almacenados en la base existente.

[1] En reconocimiento de patrones e IR, *precision* se refiere a la fracción de instacias recuperadas que son relevantes y *recall* se refiere a la fracción de instancias relevantes que son recuperadas

3. Se calculan los valores de similaridad entre cada concepto concreto de la base ontológica y la base de metadatos, si este valor de similaridad se encuentra por encima del umbral definido decimos que están lógicamente relacionados.

Una diferencia de este trabajo con otros es que se hace uso de una base de metadatos en conjunto con una ontología en el dominio de servicios de transporte. Los autores afirman que su trabajo proporciona resultados más detallados, los cuales muestran la capacidad de las ontologías en relación a otros trabajos. Mencionan además que su trabajo presenta la limitante de no haber sido probado en grandes volúmenes de información y que aún se basa en la independencia de los términos índice.

Huang *et al.* implementan un *crawler* focalizado basado en ontologías para la recuperación de información referente a comercio electrónico [Huang et al., 2009]. Se hace uso de cálculo de relevancia del contenido y predicción de *links*. Esta propuesta consta de un módulo de preprocesamiento para analizar los documentos y limpiarlos de etiquetas HTML (esto permite la obtención de la estructura de vínculos de la página y su contenido). Después del preprocesamiento un módulo de análisis de enlaces determina la prioridad de los mismos, utilizando un predictor, el cálculo de la relevancia del contenido se lleva a cabo haciendo un mapeo del mismo a la ontología de *e-commerce*.

Los autores señalan que su propuesta supera a otros enfoques de *crawling* conocidos (en cuanto a la razón de captura), como por ejemplo: basado en palabras clave, *breadth-first* y *pagerank*, teniéndose como limitación el gran consumo de tiempo del *framework* debido al alto costo en tiempo de los módulos de aprendizaje de ontologías y análisis de vínculos. Sin embargo, mencionan que si se tienen cuenta los buenos resultados que obtienen en la recuperación de información, su enfoque presenta la mejor eficiencia.

Kumar y Vig proponen una arquitectura de *crawler* focalizado denominado *CORE crawler* haciendo uso de una puntuación de relevancia ontológica y *tunneling* [Kumar and Vig, 2009]. El trabajo que realiza *CORE crawler* se define de acuerdo a los siguientes pasos:

1. Repetir estos pasos hasta que el límite máximo del *crawler* sea alcanzado.

2. Tomar la consulta del usuario e inicializar las semillas junto con su prioridad de acuerdo al algoritmo *SeedInitializer*. Fijar la relevancia mínima ontologica (*Min_OntRel*), y la relevancia mínima de búsqueda (*Min_LaRel*).

3. Descargar las páginas semilla de acuerdo a su prioridad y para cada una ir al paso 4.

4. Por cada enlace en la página ir al paso 5.

5. Calcular *ORBS* y *LARS* y colocarlos en una cola de prioridad de acuerdo a *ORBS*.

6. Recuperar el enlace con el *ORBS* más alto y si *ORBS*>*Min_OntRel* ir al paso 4, en caso contrario pasar al paso 7.

7. Si *LARS*>*Min_LaRel* descargar la página.

Donde el ORBS (*Ontology Based Relevant Score*) es un generador que hace uso de una ontología relevante, un contexto de enlaces y una tabla para almacenar la importancia de cada término. LARS (*Look Ahead Relevancy Score*) es un generador asistido por reglas adaptativas, definición de clases y un clasificador.

Este trabajo fue comparado con enfoques base de *crawling* focalizado, obteniendo una tasa de captura mayor a los mismos.

Yuvarani *et al.* proponen un *framework* para mejorar el *crawling* focalizado basado en semántica de enlaces denominado *LSCrawler* [Yuvarani et al., 2006]. Su objetivo fue determinar la relevancia de los documentos antes de que sean descargados analizando las URLs. *LSCrawler* se compone de los siguientes módulos:

- **Detector de semillas:** Su finalidad es determinar las URL semillas, consultando tres motores de búsqueda populares (Google, Yahoo, MSN).

  1. Envía el término T a los motores de búsqueda y obtiene URL_{ij} (URLs obtenidas de los motores de de búsqueda respectivos).
  2. Determina la prioridad de las URLs
     a) *Alta:* más de una ocurrencia a través de los motores de búsqueda.
     b) *Media:* repetido dentro de un motor de búsqueda.
     c) *Baja:* solo ocurre una vez dentro de un motor de búsqueda.
  3. N URLs son elegidas para ser el conjunto de semillas.

- **Gestor de crawlers:** se encarga de revisar el *buffer* de URLs y crear hilos para los *crawlers*, para que descarguen los documentos de dichas URLs, además realiza balanceo de cargas.

- **Crawler:** es un programa multihilo, hecho en Java, capaz de descargar documentos de la Web y almacenarlos en un repositorio, cada *crawler* hace peticiones por URLs a servidores.

- **Módulo de protocolo HTTP:** una vez que recibe la URL de la cola, envía una petición por el documento y una vez recibido marca la URL como procesada y guarda un registro de tiempo de recepción del mismo.

- **Extractor de enlaces:** hace un *parsing* del documento obtenido y extrae los hipervínculos, posteriormente revisa si los enlaces no han sido marcados como procesados y elimina las *Stop Keywords* del documento.

- **Extractor de ontología relevante:** procesa las ontologías relevantes del repositorio y realiza un *parsing* de las mismas, utilizando el mismo *parser* de *LSCrawler* para generar la jerarquía taxonómica que contiene clases y propiedades.
- **Analizador de hipertexto:** recibe la ontología relevante del componente anterior y determina la relevancia de los términos recibidos del extractor de enlaces, haciendo referencia a la jerarquía taxonómica.

Este trabajo fue comparado contra la técnica tradicional de recuperación de información (*full-text indexed search*), obteniendo mejores resultados en la métrica *recall*. Sin embargo, ésta ha sido la única evaluación realizada y sería útil hacer una más detallada.

Zheng *et al.* hicieron uso de una red neuronal artificial como soporte para el *crawler* focalizado con el objetivo de obtener mejores resultados que los enfoques que habían sido propuestos hasta entonces [Zheng et al., 2008]. De manera general el *framework* consta de un mecanismo de *crawling* y uno de cálculo de relevancia y consiste de las siguientes etapas:

- **Preparación de datos:** en esta etapa se construye un conjunto de ejemplos de entrenamiento positivos y negativos, obtenidos para determinar la relevancia de las páginas revisadas respecto a los tópicos dados.
- **Entrenamiento:** los ejemplos obtenidos en la primera etapa alimentan y entrenan la red neuronal artificial, utilizando un conjunto de conceptos obtenidos de una ontología de un dominio específico.
- **Etapa de obtención de páginas web:** en esta etapa se obtienen las páginas web y se calculan sus relevancias respecto a los tópicos del *crawler* con ayuda de la red neuronal artificial.

Los autores mencionan que su enfoque (utilizando tanto red neuronal como ontología) supera a otras propuestas como:

- *Crawler* focalizado basado en ontologías
- Basado en palabras clave
- *Breadth-first*
- Basado en red neuronal artificial

Sin embargo, presenta como limitante que su desempeño depende de la ontología que se esté utilizando, la comprensividad de la misma y qué tan relacionada está con los tópicos de *crawling*.

Liu *et al.* proponen el uso de un grafo de contexto de similaridad de conceptos para determinar la prioridad con que el *crawler* visitará las páginas [Liu et al., 2011]. Tal grafo corresponde a un enrejado de conceptos, donde cada nodo representa un concepto y cada arista las relaciones entre los mismos. El funcionamiento de este *framework* se describe de acuerdo a lo siguiente:

- **Recopilación de datos del tema:** esta etapa consiste en recopilar los intereses del usuario para calcular la relación de similaridad de los tópicos con las páginas.
- **Procesamiento de contenido:** las páginas descargadas son analizadas léxicamente y reducidas a vectores. Cada término en el vector es representado de acuerdo a su TF-IDF (*Term Frecuency-Inverse Document Frecuency*) en VSM (*Vector Support Machine*). Al mismo tiempo se clasifican los términos en orden descendente de acuerdo a su peso y se eligen los N primeros como términos característicos. Finalmente éstos son tratados como un conjunto de conceptos para construir un contexto formal y formar un enrejado de conceptos.
- **Construcción de gráfica de contexto de similaridad de conceptos:** del enrejado de conceptos obtenido en la etapa anterior se construye una gráfica de similaridad nuclear, usando **WordNet** como ontología de dominio. La combinación del enrejado básico con el actual nos proporciona la gráfica de contexto de similaridad de conceptos.
- **Asignación de prioridad:** con ayuda de la gráfica construida en la etapa anterior se asigna una puntuación a las URLs no visitadas, con el fin de determinar la prioridad de las mismas y visitar aquellas que tienen más relación con el contexto de tópicos proporcionados.

Se resalta de este trabajo buenos resultados en la medida *recall* con respecto a otros.

Jung propone el uso de una estrategia evolutiva para un *crawler* cooperativo con el objetivo de construir una estructura semántica de espacios web [Jung, 2009]. El sistema hace uso de *crawlers* y *meta-crawlers*, es decir, cada vez que un usuario nuevo ingresa a la Web semántica, un determinado *crawler* descubre una nueva clase semántica y puede compartirla con los demás *crawlers* a través de un *meta-crawler* centralizado. La arquitectura de esta propuesta se define a continuación:

- ***Picking*:** mientras un *crawler* accesa a un espacio web, la estructura semántica parcial puede ser extraída. Si después de la operación de fusión el valor de la razón de coincidencia está por encima del valor de umbral, la información debe ser enviada al *meta-crawler*.
- ***Querying*:** un *crawler* puede actualizar su información semántica acerca de los recursos, cuya razón de desigualdad supere un umbral establecido para el *meta-crawler*, para una generación eficiente de predicados semánticos.

Los pasos que realiza el *meta-crawler* son:

- ***Knitting*:** construir la ontología local de cada espacio web semántico. El *meta-crawler* puede agregar instancias semánticas descubiertas por *Picking* de forma incremental.
- ***Merging*:** las ontologías locales de dos espacios web pueden ser fusionadas cuando tales espacios necesitan ser integrados.
- ***Translating*:** para la consulta de un *crawler* (*Querying*) el *meta-crawler* puede recomendar las correspondencias de dos elementos asociados y sus relaciones.
- ***Comparing*:** el *meta-crawler* puede medir la similaridad semántica entre el contexto del usuario y las anotaciones semánticas de un espacio web en un determinado momento.

La razón del uso de la estrategia evolutiva fue para mejorar el desempeño del *framework* y para resaltar las siguientes contribuciones:

1. Minimizar los conflictos semánticos del conocimiento obtenido mediante el *crawler*.
2. Capturar los cambios más recientes en la evolución de las ontologías.

Yang y Hsu diseñaron una ontología para una página Web de CFP (*Call for Papers*) e hicieron el análisis de las clases de la ontología de esta página Web en Protégé [Yang and Hsu, 2009]. Posteriormente en conjunto con MS SQL *server* se conformó una plataforma de intercambio ontológico de algunas palabras clave de páginas CFP. Finalmente con Java se construyó el *OntoCrawlerII* (*Ontology-Supported Focused Crawler*).

De este trabajo se resalta el uso de comparación de palabras clave y ontología de CFP para filtrar páginas no relevantes, mejorar las medidas *precision* y *recall* y proveer soporte a las consultas del usuario y al núcleo de los sistemas de búsqueda Web.

Gacitua *et al.* propusieron un *framework* para aprendizaje de ontologías desde texto [Gacitua et al., 2008], el cual provee un proceso cíclico que involucra la aplicación sucesiva de varias técnicas de NLP (*Natural Language Processing*) y algoritmos de aprendizaje para extracción de conceptos y modelado de ontologías. Este trabajo proporciona soporte para evaluar la utilidad y precisión de las diferentes técnicas y sus posibles combinaciones en procesos específicos. Las siguientes etapas describen su funcionamiento:

(a) Abstracción y clasificación semi-automática de los conceptos de dominio.

(b) Codificación de dichos conceptos en OWL (lenguaje ontológico).

(c) Edición de los mismos usando una versión mejorada de Protégé.

Hacene *et al.* proponen un enfoque para construcción de ontologías (desde texto) de manera semi-automática [Hacene et al., 2008]. El componente más importante de este trabajo es el análisis relacional de conceptos. El enfoque extrae contextos formales del cuerpo del documento utilizando análisis lingüístico y deriva en dos jerarquías *is-a* (de conceptos ontológicos y sus relaciones transversales asociadas). Como preprocesamiento se realiza el análisis de texto, el cual es utilizado para transformar una colección de documentos en un conjunto de tablas de datos y sus inter-relaciones. Posteriormente el análisis relacional de conceptos los transforma en un conjunto de enrejados de conceptos, con conceptos inter-relacionados; de tales enrejados se genera una ontología de forma semi-automática, transladando los elementos relevantes a conceptos y relaciones. Finalmente la ontología se refina abstrayendo nuevas relaciones de las definidas inicialmente.

El análisis relacional de conceptos utilizado en este trabajo es una extensión del FCA (*Formal Concept Analysis*), esta extensión integra el procesamiento de conceptos de dominio y las relaciones transversales en un solo paso. Los autores mencionan que su trabajo supera a otros enfoques concurrentes a este, donde solo se utiliza FCA para construir solo jerarquías de conceptos. Como incoveniente se tiene la obtención de una ontología muy extensa, aunque plantean como solución hipotética el uso combinado de podado automático y refactorización para obtener una ontología focalizada.

Nie y Zhou proponen un *framework* para aprendizaje de ontologías denominado *OntoExtractor* [Nie and Zhou, 2008]. Este trabajo incluye extracción de conceptos semilla (son los conceptos más representativos en el dominio específico), construcción de relaciones semánticas y refinamiento de ontologías. Los siguientes pasos describen el funcionamiento general de *OntoExtractor*:

1. Preprocesar del cuerpo del documento y extracción de información textual.
2. Extraer de los conceptos semilla originales con enfoques estadísticos.
3. Por cada concepto semilla, extraer todas las sentencias que lo contengan y hacer análisis sintáctico.
4. Extraer nuevos conceptos basándose en la información sintáctica.
5. Aplicar las plantillas semánticas predefinidas y extraer las relaciones taxonómicas, no taxonómicas y los axiomas.
6. Refinar y reorganizar la ontología resultante.

Para este trabajo se realizaron mediciones con las métricas *precision* y *recall* y se compararon con un ontología base creada por un humano. En los experimentos llevados a cabo se observó una buena adaptabilidad al dominio por parte de OntoExtractor, aunque los resultados obtenidos en *precision* y *recall* no fueron satisfactorios.

Entre otros beneficios se mencionan la eliminación de un cuello de botella en la adquisición de conocimiento y la dependencia de dominio en la construcción de ontologías.

Rudolph *et al.* describen una forma de construcción y refinamiento de ontologías combinando técnicas de NLP y análisis formal de conceptos [Rudolph et al., 2007]. Concretamente este trabajo es la combinación en forma sinérgica de los enfoques *Text2Onto* (*framework* para aprendizaje de ontologías) y la técnica de exploración relacional (RE).

- **Text2Onto:** ofrece algoritmos para generar conceptos, relaciones taxonómicas y no taxonómicas, así como axiomas de disyunción basado en la evidencia léxica. El proceso es semi-automatizado, debido a que no hay garantía de que el modelo de conocimiento generado sea correcto y lo suficientemente preciso para caracterizar al dominio en cuestión; es por esto que se deja la desición final al ingeniero de ontologías.
- **Exploración Relacional:** está basada en un algoritmo de exploración de atributos de FCA. La ventaja de su uso es que los resultados obtenidos son lógicamnente nítidos y naturalmente consistentes.

Suponiendo que se tiene una base de conocimiento que ha sido refinada de acuerdo a un término específico, Text2Onto puede ser utilizado para extraer axiomas hipotéticos relacionados con dicho término. Estos axiomas potenciales son agregados a la base de conocimiento despues de ser validados por el experto. Posteriormente se aplicará RE para especificar las interdependencias del término investigado y los conceptos importantes relacionados.

Gulla *et al.* presentaron un sistema de extracción de frases clave (conjunto óptimo de frases para describir un documento) que ha sido utilizado para acelerar la construcción de ontologías de búsqueda [Gulla et al., 2007]; estas frases sirven como conceptos candidatos en la ontología e incluso pueden indicar cómo deben ser definidas las relaciones semánticas.

De este enfoque no supervisado se tiene como ventaja que es más ampliamente aplicable, debido a que no requiere ningún conocimiento de dominio ni consultar a expertos de dominio, mientras que el supervisado posee un conjunto de frases predefinidas para entrenar al algoritmo. Por otro lado el no supervisado normalmente produce frases clave menos relevantes que el supervisado.

Magnini *et.al* propusieron poblado de ontologías desde menciones textuales, la cual definen como una subtarea de la población de ontologías desde texto. En esta subtarea asumen que las menciones de varios tipos de entidades ya han sido extraídas de la colección de documentos [Magnini et al., 2006]. Este trabajo denominado OPTM (*Ontology Population from Textual Mentions*) considera tres subtareas descritas a continuación:

1. **Reconocimiento y clasificación de los atributos de la entidad:** el material textual expresado en una mención es extraído y distribuido entre los pares *atributo-valor* definidos por la clase de la mención, por ejemplo dada la mención *"U.S. President Bush"* se espera que el atributo *Last_Name* sea ocupado por *Bush* y el atributo *Role* por *U.S. President*.

2. **Normalización:** el material extraído en el paso 1 es asignado a conceptos y relaciones ya definidos en la ontología.

3. **Resolución de correferencia del texto interno de la entidad:** una mención m_i sólo puede ser asignada a una entidad individual perteneciente a una clase C.

Giannopoulos *et al.* presentaron una herramienta para anotado semántico y búsqueda [Giannopoulos et al., 2010]. Esta herramienta soporta anotado manual y automático de documentos en distintos formatos (doc, pdf, rtf, txt, odt, sxw) utilizando clases de ontologías; además provee facilidades de búsqueda basada en ontologías. Los autores señalan que su trabajo contrasta con otros al tratarse de una herramienta fácil de usar con soporte completo para:

- Anotado semántico de tipos de documentos comunes sin alterar su formato.
- Compartir esas anotaciones.
- Búsqueda combinando palabras clave y búsqueda semántica.

La arquitectura que proponen está dividida en 4 componentes que se describen a continuación:

a) **Anotado semántico:** facilita el anotado semántico, consiste de un visor de documentos, un visor de ontologías y un editor de anotado.

b) **Servidor de ontología:** almacena las anotaciones de los documentos en forma de instancias de ontología (OWL).

c) **Indexado:** responsable de indexar los documentos utilizando un índice invertido.

d) **Búsqueda:** permite al usuario realizar consultas usando tanto información textual como semántica.

Roberts *et al.* presentaron un trabajo de anotado semántico sobre documentos de texto y registros estructurados del Royal Marsden Hospital [Roberts et al., 2008], relacionados con pacientes muertos por cáncer. Este proyecto fue denominado *CLEF Corpus* y realiza el anotado de distintos tipos de textos con multiples entidades y sus relaciones. Los autores mencionan haber analizado la aplicabilidad de este sistema de anotado para diversos subgeneros clínicos, concluyendo que puede ser aplicado a otros subgeneros y ofrecer buenos resultados.

2.7 Discusión

En este capítulo se describieron los elementos teóricos necesarios en la presente investigación. Además, se realizó un análisis de los trabajos relacionados que se han publicado en los últimos años, con el objetivo de determinar la factibilidad y originalidad del proyecto. Este capítulo contiene los conocimientos necesarios para el desarrollo propuesto y representa una herramienta fundamental en el mismo.

Tabla 2.1: Análisis de trabajos relacionados con poblado de ontologías

Trabajo	Característica (s) relevante	Ventaja(s)	Desventaja(s)
Ontology-Based Focused Crawling	–Uso de SVM (*Support Vector Machine*) –Poblado automático y semiautomático	–Buena precisión (sobre el 77.5 %)	–Necesidad de entrenamiento del algoritmo –Descargas innecesarias antes del uso de SVM
A flexible framework to experiment with ontology learning techniques	Aplicación de técnicas de NLP	Mejora la construcción de ontologías de dominio	No se mencionan
Ontology Learning from Text Using Relational Concept Analysis	Análisis relacional de conceptos	Supera los enfoques concurrentes a este trabajo	Obtención de una ontología muy extensa
A domain adaptive ontology learning framework	Refinamiento de ontologías	–Buena adaptabilidad al dominio –Eliminación de un cuello de botella en la adquisición de conocimiento	*Precision* y *recall* no satisfactorios
Supporting Lexical Ontology Learning by Relational Exploration	Técnicas de NLP y analisis formal de conceptos	No se mencionan	Faltan evaluaciones
Ontology Learning for Search Applications	Sistema de extracción de frases clave	No requiere de conocimiento o experto de dominio	Produce frases menos relevantes que el enfoque supervisado
Ontology population from textual mentions	Uso de menciones textuales	No se mencionan	No se mencionan

Las tablas 2.1, 2.2 y 2.3 muestran un resumen de los trabajos revisados en esta sección. Se resaltan las características más relevantes de cada trabajo y, de acuerdo a lo escrito por los autores correspondientes, se denotan las ventajas y desventajas de cada uno de los trabajos.

De acuerdo al análisis realizado en los trabajos relacionados se determina que es factible su desarrollo, ya que los trabajos presentados hasta la fecha, representan módulos en la propuesta realizada por esta tesis, por ejemplo, un módulo de este proyecto es el desarrollo de un *crawler* para la extracción de ontologías de la Web y Van

Tabla 2.2: Análisis de trabajos relacionados con *crawlers* focalizados

Trabajo	Característica (s) relevante	Ventaja(s)	Desventaja(s)
An Ontology-Based Crawler for the Semantic Web	–Guiado por ontologías –Uso de *DMatch Ontology Matcher*	Capaz de encontrar relaciones que no se encuentran con similaridad de textos	Carece de la evaluación adecuada
Intelligent web crawler	Verificación semántica de la URL	–No hay necesidad de entrenamiento –Reducción notable en la cantidad de documentos recuperados	No se mencionan
A Transport Service Ontology-based Focused Crawler	Uso de una base de metadatos junto con una ontología de dominio	Proporciona resultados más detallados en relación a otros trabajos	No ha sido probado en grandes volúmenes de información
Focused Crawling for Retrieving E-commerce Information Based on Learnable Ontology and Link Prediction	Uso de predicción de *links*	Mejor razón de captura que *breadth-first* y *pagerank*	Gran consumo de tiempo
CORE focused web crawler	Uso de puntuación de relevancia ontológica y *tunneling*	Mejor razon de cosecha que los enfoques base de *crawling* focalizado	Necesidad de entrenamiento
LSCrawler	–Semántica de *links* –Uso de hilos	Mejore que *full-text indexed search*	No se han realizado más evaluaciones
An ontology-based approach to learnable focused crawling	Uso de red neuronal y ontologías	Supera los enfoques: –basado en ontologías –basado en palabras clave –*Breadth-first* –Basado en red neuronal artificial	Necesidad de fase de entrenamiento
Focused Crawler Based on Domain Ontology and FCA	Uso de un grafo de contexto de similaridad de conceptos	Supera enfoques contemporáneos en *recall*	No se mencionan
Using evolution strategy for cooperative focused crawling on semantic web	Uso de una estrategia evolutiva	–Minimiza conflictos semánticos del conocimiento obtenido –Captura los cambios más recientes en las ontologías	No se mencionan
OntoCrawlerII	Uso de comparación de palabras clave y ontología de CFP	Mejor que *OntoCrawlerI*	No se mencionan

Tabla 2.3: Análisis de trabajos relacionados con anotado semántico

Trabajo	Característica (s) relevante	Ventaja(s)	Desventaja(s)
GoNTogle	Anotado manual y automático de distintos formatos de texto	Herramienta fácil de usar	Sólo presentan evaluación preliminar
Semantic annotation of clinical text: The CLEF corpus	Anotado de registros estructurados	Mejor framework en información clínica	No tienen permisos para liberar el material

de Maele *et al.* ya han hecho una propuesta para ello. Además, no ha sido publicado un trabajo igual al de esta tesis, lo cual dota de originalidad al mismo.

Capítulo 3

Arquitectura del sistema

Es importante contar con el diseño adecuado cuando se va a desarrollar un software. El diseño permite generar un producto de calidad y evita desperdicios de tiempo. Un diseño bien definido permite al desarrollador identificar de manera precisa los requerimientos del software y posibilita hacer un análisis costo-beneficio para determinar la factibilidad del proyecto. El diseño debe estar bien estructurado y proporcionar una idea completa del software.

En este capítulo se describen los componentes de la solución propuesta. Se incluyen diagramas de clases, de flujo y de secuencia para detallar dichos componentes. También se describe el funcionamiento de exploración y descarga concurrentes de los *crawlers*.

Como soporte al diseño de la solución que se presenta en este capítulo, se hizo una revisión del libro *UML Distilled* de Martin Fowler [Fowler, 2003], esto con el objetivo de hacer un diseño apegado a los estándares de modelado de *software*.

3.1 Propuesta de solución

En la Figura 3.1 se muestran los principales componentes de software que se diseãron para dar solución al problema planteado. De dichos componentes, ambos *crawlers* pueden ejecutarse a la par. Después se hace el *parsing* de los documentos para eliminar información irrelevante. Por último se realiza el anotado semántico.

El componente "*crawler* textual" se encarga de la recuperación de documentos de texto utilizando un contexto base. El cual se construye extrayendo palabras clave de un documento base. El "*crawler* ontológico" se encarga de recuperar ontologías similares al contexto mencionado, el *parser* se encarga de limpiar los documentos de texto de símbolos y etiquetas (HTML) para disminuir el ruido durante la comparación y el módulo de anotado semántico se encarga de agregar etiquetas a los documentos para vincularlos con información semántica. Además, este módulo también se encarga de

poblar una ontología con la información etiquetada.

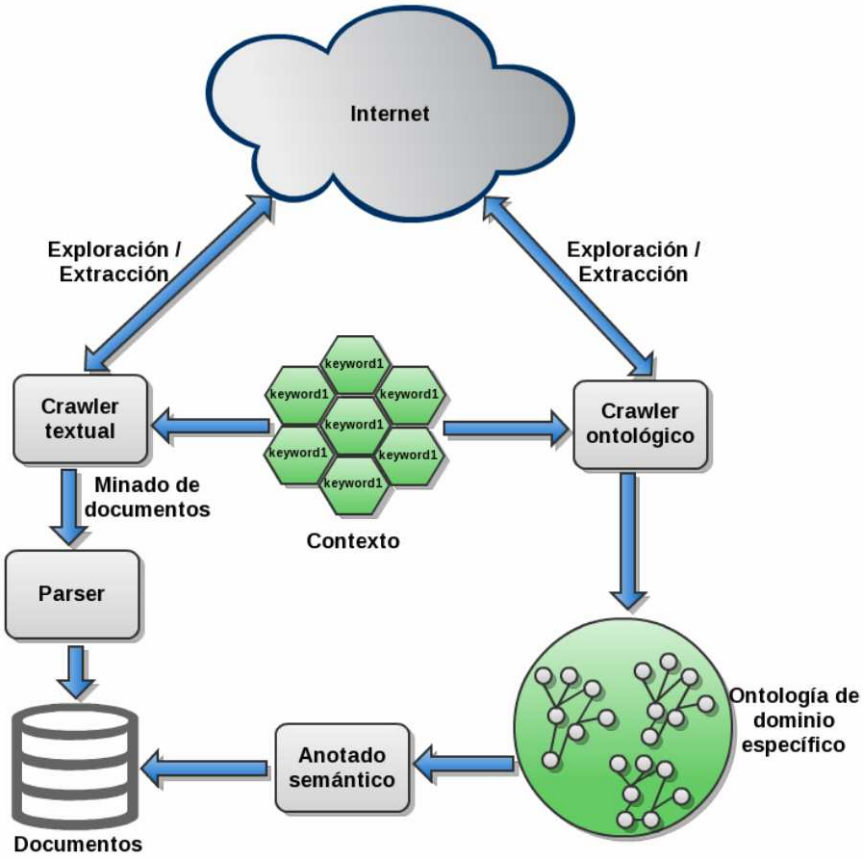


Figura 3.1: Estructura propuesta

Los *crawlers* mencionados realizan una comparación entre el contexto base y el recurso (documento u ontología) que se quiere recuperar, esta comparación establece un coeficiente de similaridad entre ambos recursos. El recurso se descarga si el coeficiente de similaridad obtenido es mayor o igual a un determinado umbral. La propuesta de solución está estructurada de acuerdo a los siguientes componentes:

- Definición del contexto en la Subsección 3.1.1.
- *Crawler* textual en la Subsección 3.1.2.
- *Crawler* ontológico en la Subsección 3.1.3.
- Analizador de texto (*parser*) en la Subsección 3.1.4.
- Anotado semántico y poblado de ontologías en la Subsección 3.1.5.

3.1.1 Definición del contexto

En esta tesis la búsqueda de información se realiza sobre un área y un contexto de investigación delimitado, el cual está definido por un conjunto de palabras clave y es susceptible de ser modificado en tiempo de ejecución. De esta manera, la búsqueda se hace sobre un tema o área de investigación en específico.

Se debe tener en cuenta que existen palabras que, dentro de un dominio de conocimiento, tienen más peso que otras; de acuerdo con esto, es adecuado asociar un índice numérico a cada término, como una manera de determinar el grado de importancia de cada palabra clave dentro del contexto que se quiere definir. Los números asociados deben encontrarse en una escala bien definida, por ejemplo: 1 a 10, 1 a 100, 0.0 a 1.0.

Una forma alterna de definir el contexto es utilizando uno o más documentos que pertenecen al dominio de interés, de los cuales se pueden extraer los términos más representativos mediante una serie de técnicas que se describen a continuación:

- **Obtener *tokens*:** consiste en convertir el documento de texto en un listado de palabras. Para facilitar su manejo, también se convierten todas las letras a minusculas, por ejemplo: del siguiente texto "*An ontology is an explicit formal conceptualization*", se obtiene esta lista [*"an", "ontology", "is", "an", "explicit", "formal", "conceptualization"*].

- **Remover *StopWords*:** se refiere a eliminar palabras que son muy comunes en la mayoría de los documentos, pero que no son representativas de un dominio de conocimiento, por ejemplo: *"the, is, at, of"*. Retomando el texto de ejemplo señalado en el punto anterior, la lista queda como sigue: [*"ontology", "explicit", "formal", "conceptualization"*].

- **Lematización:** consiste en llevar las palabras a su raíz, por ejemplo: "*cars*" a "*car*", "*says*" a "*say*". Con esto se logra que las palabras que provienen de una misma raíz sean tratadas como iguales.

- **Construcción de bolsa de palabras:** se refiere a asignar un peso a cada uno de los términos en la lista obtenida de los procesos anteriores. En este caso, el número que se asocia a cada palabra es la cantidad de repeticiones de la misma en la lista. Al final se obtiene un diccionario con cada palabra clave y su frecuencia correspondiente. Para el ejemplo mencionado en los puntos anteriores la bolsa queda como sigue: {*"ontology":1, "explicit":1, "formal":1, "conceptualization":1*}.

Una manera de mejorar la evaluación de similaridad es hacer uso de la secuencia de estas palabras formando *n-gramas*. Por ejemplo, una bolsa de *bigramas* utilizando el mismo ejemplo queda como sigue: {*("ontology", "explicit"):1, ("explicit", "formal"):1, ("formal", "conceptualization"):1*}. Esto permite considerar, además de las

palabras clave, la secuencia en la que se encuentran en el documento, lo cual hace más exigente la comparación conforme crece el *n-grama* (trigramas, tetragramas, etc.).

En caso de tratarse de más de un documento de donde se quiere modelar el contexto, es recomendable hacer uso de TF-IDF (*Term Frequency Inverse Document Frequency*) [Aizawa, 2003], que además de considerar la frecuencia de cada término en un documento, toma en cuenta la cantidad de documentos en los que dicho término está presente.

La bolsa de palabras queda expresada mediante la ecuacion (3.1):

$$B = \{(w_1, f_1), (w_2, f_2), \ldots, (w_n, f_n)\}, \tag{3.1}$$

donde w_i es una palabra del documento y f_i es el numero de veces que dicha palabra se repite dentro del documento.

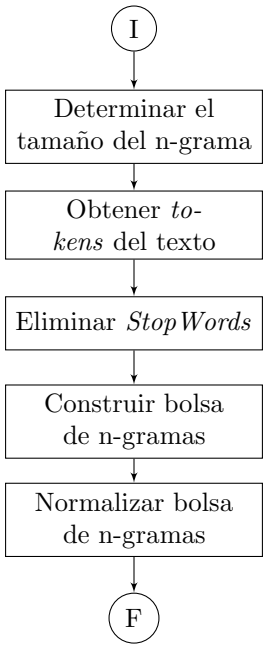


Figura 3.2: Secuencia general de actividades para modelado de contexto

El diagrama de flujo de la Figura 3.2 muestra de manera general, la secuencia de las principales actividades que debe realizar este módulo.

La Figura 3.3 muestra el diseño de la clase con los atributos y operaciones necesarias para el tratamiento del texto. Esta clase se encarga de extraer los *tokens* del texto, eliminar las *stopwords* (palabras muy comunes en el lenguaje) y lematizar las

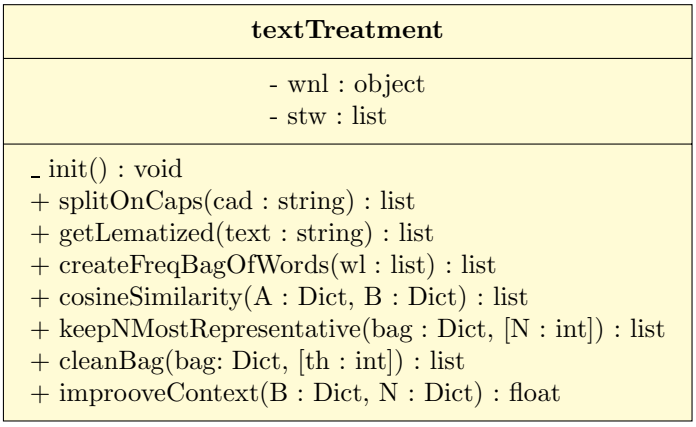


Figura 3.3: Clase para tratamiento de texto

palabras resultantes (operacion `getLematized()`); además, cuenta con las operaciones sobre vectores necesarias para crear las bolsas de palabras, realizar podado de las mismas y obtener la métrica de similaridad.

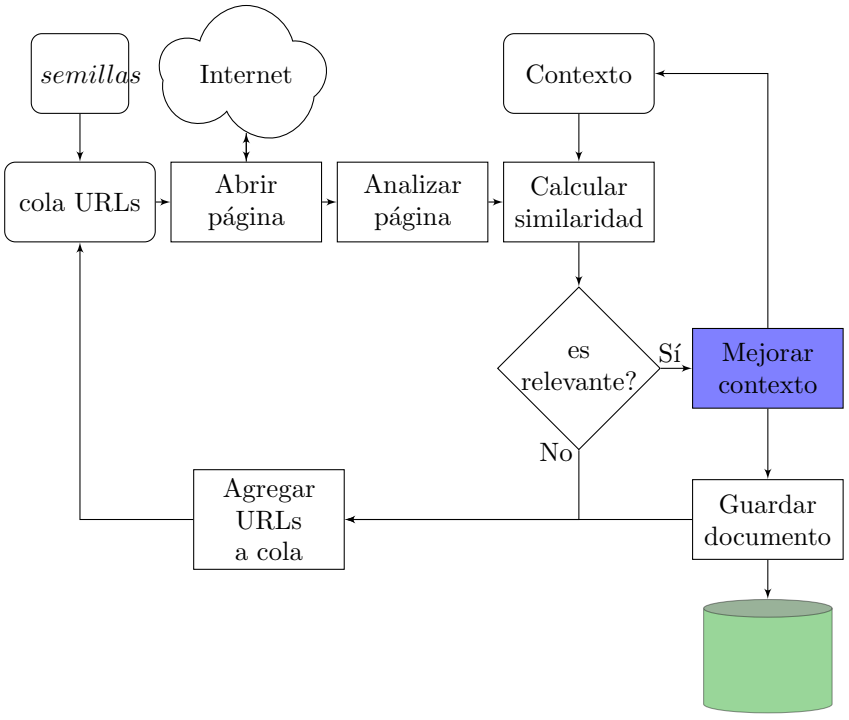


Figura 3.4: Inclusión de mejora de contexto en funcionamiento general de *crawler*

En este proyecto se incluye la propuesta de manejo de un contexto dinámico, el cual consiste en una mejora del contexto en tiempo de ejecución del *crawler*. El funcionamiento general del *crawler* con mejora del contexto se muestra en la Figura 3.4. Esta adaptación del contexto a los nuevos recursos relevantes se apoya en la

ecuación (3.2):

$$B' = \begin{cases} \{(w_i, f_i) | (w_i, f_i) = (w_{iB}, f_{iB}) \text{ si } \forall\ w_i \in B \wedge w_i \notin D\} \\ \{(w_i, f_i) | (w_i, f_i) = (w_{iD}, f_{iD}) \text{ si } \forall\ w_i \in D \wedge w_i \notin B\} \\ \{(w_i, f_i) | w_i = w_{iB} \text{ y } f_i = (f_{iB} + f_{iD})/2 \text{ si } \forall\ w_i \in B \cap D\}, \end{cases} \tag{3.2}$$

donde B' es la nueva bolsa de palabras, (w_i, f_i) son los pares que componen al conjunto de acuerdo con (3.1), w_{iB} se refiere a una palabra que pertenece al conjunto B, lo mismo que w_{iD} para el conjunto D y las frecuencias representadas con estos sufijos ($_{iB}$ y $_{iD}$). En nuestra propuesta hacemos uso de 2 umbrales definidos como:

- α : con el se determina si se descarga el documento
- β : utilizado para determinar si se mejora el contexto

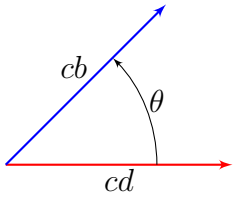


Figura 3.5: Representación de los contextos como vectores

Los vectores involucrados en la comparación se pueden ilustrar de forma representativa en la Figura 3.5. En esta figura podemos observar dos vectores que representan al contexto base (cb) y al contexto del nuevo documento abierto (cd); también se tiene a θ que denota el ángulo entre ambos vectores.

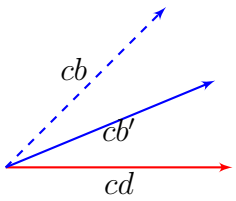


Figura 3.6: Representación de la mejora del contexto cb a cb'

Si el $\cos(\theta)$ arroja un valor mayor al umbral definido, se mejorará el contexto obteniendo un promedio entre los dos vectores. La Figura 3.6 muestra como el contexto base (cb) se acerca al contexto del documento (cd) que se ha determinado como relevante, generando un nuevo contexto base cb'.

3.1.2 *Crawler* específico para extracción de texto

Este componente de la arquitectura tiene como objetivo obtener documentos de texto de Internet. Este *crawler* toma como base el contexto previamente definido para hacer las búsquedas. Es importante señalar que la exploración está limitada a un conjunto de dominios específicos.

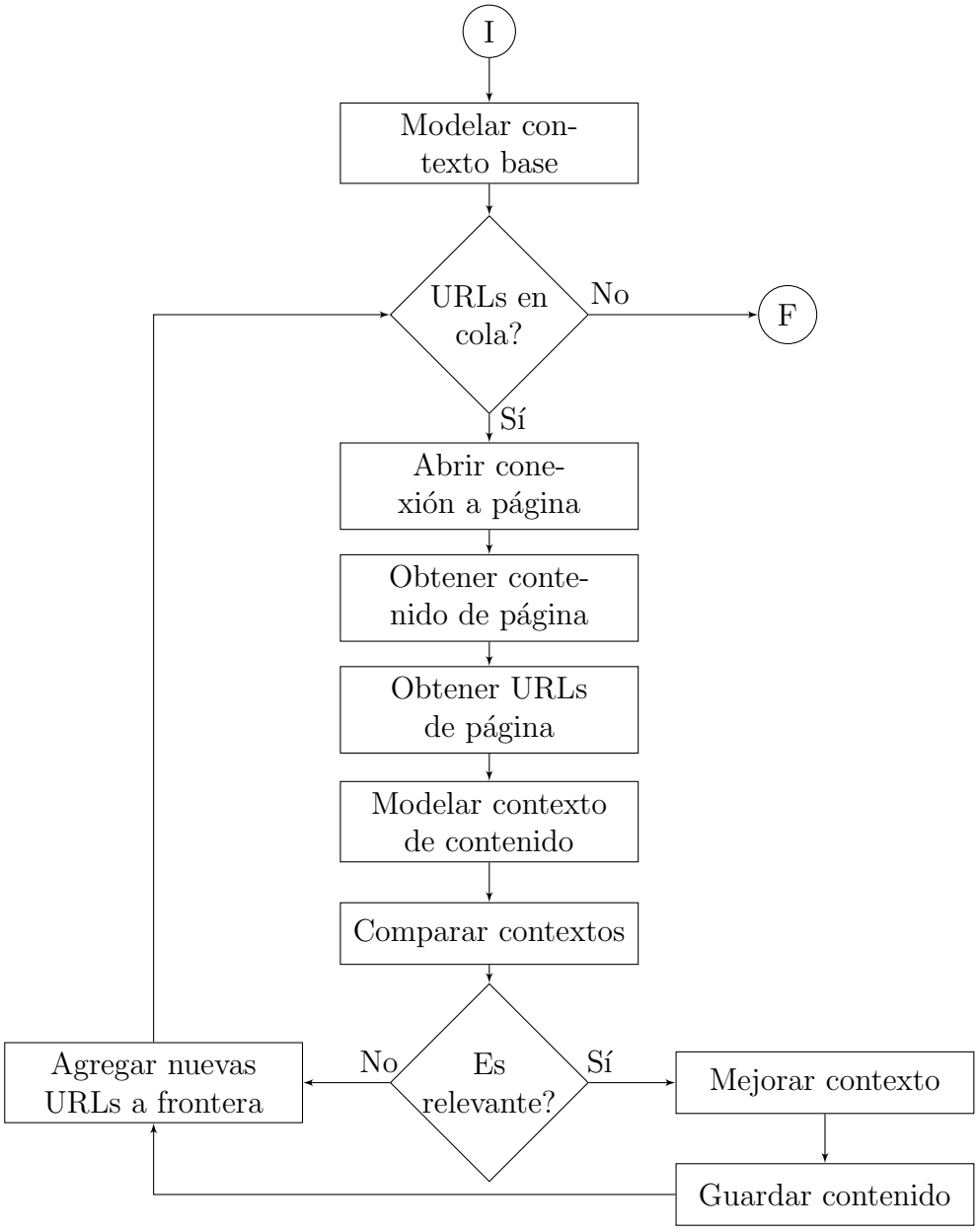


Figura 3.7: Flujo de actividades del *Crawler* textual

El diagrama de flujo correspondiente a este proceso se muestra en la Figura 3.7. En el se identifican las actividades más importantes para que dicho proceso arroje los

resultados deseados. Algunas de las actividades mostradas en este diagrama corresponden a operaciones de la clase *textTreatment*, que son llamadas desde el crawler textual.

Los documentos que se van recuperar incluyen formatos PDF y HTML, los cuales deberán pertenecer al dominio de conocimiento determinado por el contexto. Para evaluar esta pertenencia se utiliza la bolsa de palabras descrita en la subsección 3.1.1; la similitud se evalúa construyendo otra bolsa de palabras a partir del documento que se quiere recuperar y calculando el porcentaje de similaridad entre ambas. Esto es posible utilizando medidas de similitud de vectores, ya que la bolsa de palabras puede verse como un vector multidimencional, donde cada palabra clave establece una dimensión y su frecuencia asociada denota su valor. Los coeficientes de similaridad utilizados son los siguientes, de acuerdo con Rorvig (1999) [Lee et al., 2005]:

- **Jaccard extendido o Tanimoto:** mide la similaridad de dos conjuntos dividiendo el tamaño de la intersección entre el de la unión, la versión extendida permite trabajar con datos continuos:

$$simT_{ij} = \frac{\sum_k x_{ik}x_{jk}}{\sum_k x_{ik}^2 + \sum_k x_{jk}^2 - \sum_k x_{ik}x_{jk}}, \tag{3.3}$$

  donde $simT_{ij}$ es el coeficiente de similaridad obtenido y x_{ik} es la frecuencia del término k en el documento i.

- **Dice:** proporciona un valor en referencia a qué tan similar es un conjunto a otro:

$$simD_{ij} = \frac{2\sum_k x_{ik}x_{jk}}{\sum_k x_{ik}^2 + \sum_k x_{jk}^2}, \tag{3.4}$$

  donde $simD_{ij}$ es el coeficiente de similaridad obtenido.

- **Modelo Overlap:** calcula la superposición entre dos conjuntos:

$$simO_{ij} = \frac{\sum_k x_{ik}x_{jk}}{min(\sum_k x_{ik}^2, \sum_k x_{jk}^2)}, \tag{3.5}$$

  donde $simO_{ij}$ es el coeficiente de similaridad obtenido.

- **Similaridad de coseno:** calcula el coseno del ángulo entre dos vectores. Si el ángulo es 0 el coseno será 1 (documentos idénticos):

$$simC_{ij} = \frac{\sum_k x_{ik} x_{jk}}{(\sum_k x_{ik}^2 \sum_k x_{jk}^2)^{\frac{1}{2}}}, \tag{3.6}$$

  donde $simC_{ij}$ es el coeficiente de similaridad obtenido.

Se sabe que al normalizar un vector se obtiene un vector igual al primero pero con longitud igual a 1. Teniendo esto en cuenta, se pueden reducir las fórmulas anteriores sustituyendo los elementos del tipo $\sum_k x_{ik}^2$ por 1. Al trabajar con vectores normalizados podemos observar que los coeficientes *Dice*, *Overlap* y coseno son iguales, reduciendo el coeficiente a un producto punto entre los vectores.

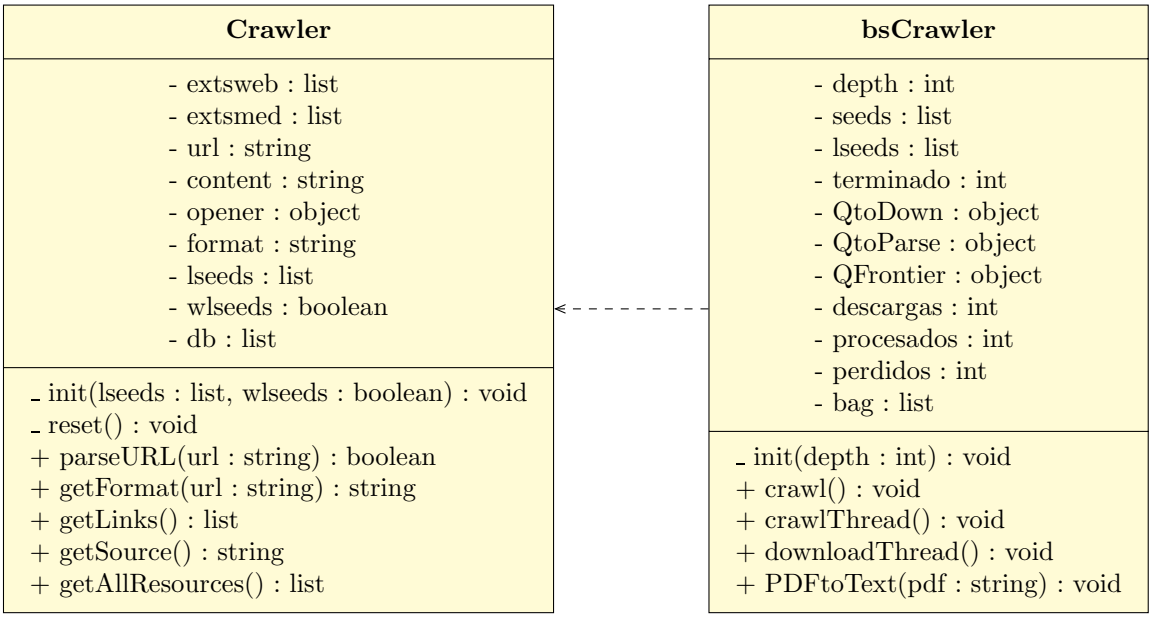


Figura 3.8: Clase de *crawler* específico para la extracción de texto

En la Figura 3.8 se muestran las clases necesarias para el funcionamiento del *crawler* mencionado. La clase *Crawler* provee operaciones básicas para la exploración y recuperación de información, mientras que *bsCrawler* se encarga de administrar la profundidad a la que se quiere llegar y de gestionar los hilos para hacer exploración y descarga concurrentes.

Además, debido a que este es uno de los procesos principales dentro del proyecto, se propone un diagrama de secuencia que muestra la interacción de los objetos involucrados en el mismo. Este diagrama se muestra en la Figura 3.9. En él se exponen

las interacciones necesarias para la exploración de la Web en busca de documentos relevantes al contexto.

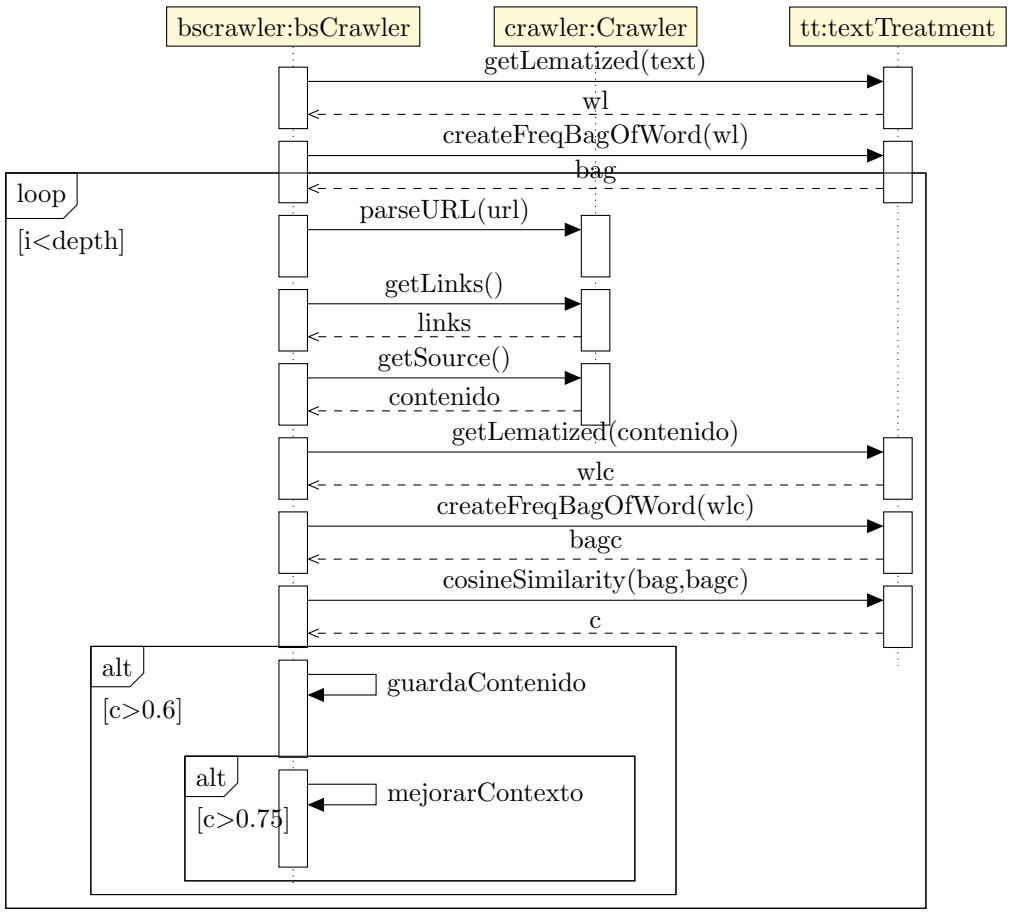


Figura 3.9: Diagrama de secuencia para la exploración en busca de texto

Puede observarse que una parte del proceso se encuentra en un ciclo (*loop*), lo cual indica que el proceso se repetirá hasta que se alcance la profundidad deseada en la exploración. Además, se señala en otro bloque (*alt*) que se descargarán sólo los documentos que superen un cierto umbral en el coeficiente de similaridad; todos los documentos se comparan con el contexto base que se modela antes del ciclo.

La primera parte del diagrama (comprendido por las actividades: `getLematized()`, `createFreqBagOfWord()` y `normalizeVector()`) muestra las operaciones de inicialización necesarias para modelar el contexto base. Estas operaciones son muy importantes porque la obtención de los documentos deseados depende de un buen modelado de este contexto. Además, estas operaciones se llevan a cabo como un bloque de inicialización porque sólo es necesario llevarse a cabo una vez, es decir, se modela el contexto y se carga en memoria para su uso posterior en el cálculo de similaridad.

Las actividades que se encuentran dentro del ciclo establecen la secuencia a seguir para realizar la exploración. Se abre una conexión a cada URL, empezando por las URLs semilla proporcionadas (actividad `parseURL()`). Esta apertura permite cargar todo el documento en memoria y extraer el contenido de interés (URLs con la actividad `getLinks()` y texto de interés con la actividad `getSource()`). Ya separado el texto que nos interesa se modela un contexto del mismo, una vez que se tiene el nuevo contexto modelado se compara con el contexto base (actividad `cosineSimilarity()`) y se obtiene un coeficiente entre 0 y 1, que nos indica qué tan similares son ambos contextos. Por último, mediante un bloque de decisión, se determina si se guarda el nuevo documento en disco (si el coeficiente de similaridad supera un umbral definido).

3.1.3 *Crawler* específico para extracción de ontologías

Este *crawler* lleva a cabo la extracción de ontologías de repositorios en la Web y su almacenamiento en un repositorio local. Para realizar dicha exploración, este *crawler* también se basa en el contexto previamente definido.

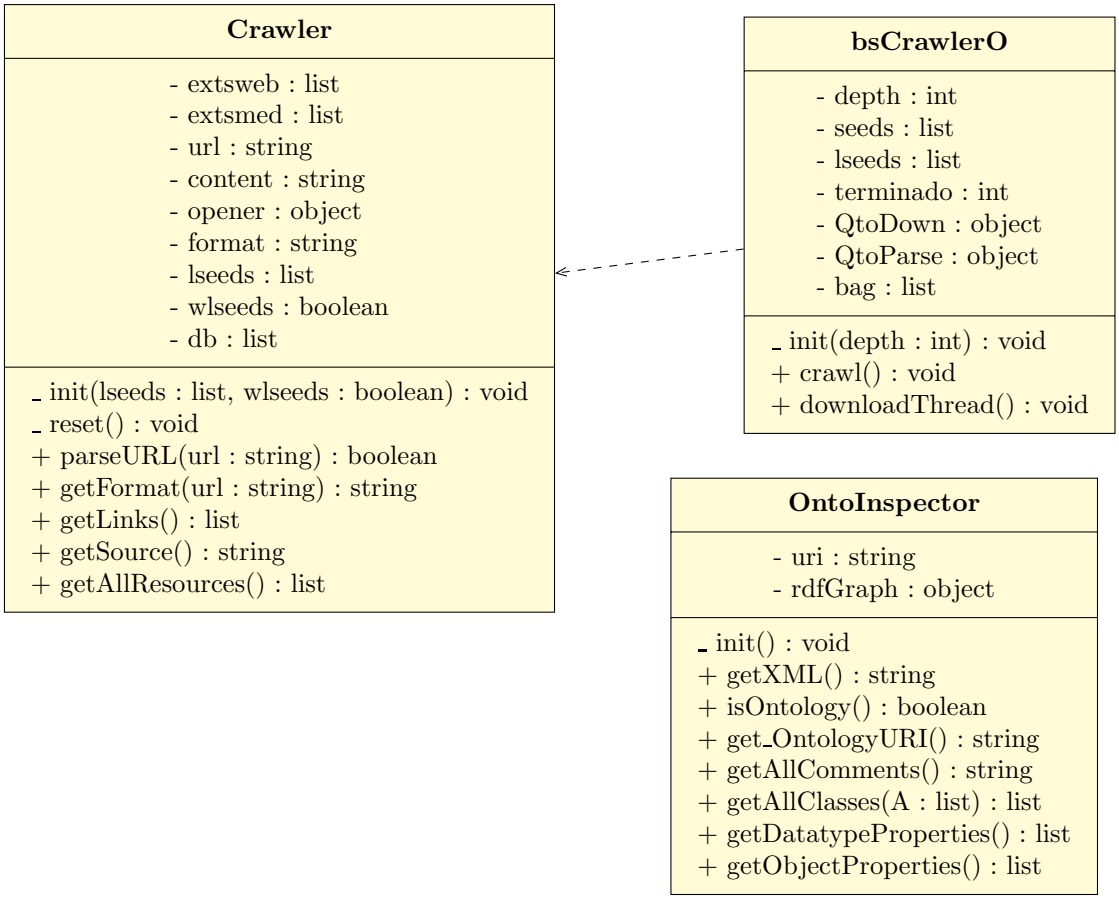


Figura 3.10: Clases para el *crawler* específico para la extracción de ontologías

En la Figura 3.10 se puede observar que el diseño de las clases para este *crawler* contiene también la clase *Crawler*. Además, se incluye la clase *OntoInspector* que permite la extracción de información de las ontologías. El diagrama de flujo modelado para este proceso se muestra en la Figura 3.11.

El objetivo de este *crawler* es obtener las ontologías adecuadas para modelar el dominio de conocimiento del tema o área de investigación delimitada por dicho contexto.

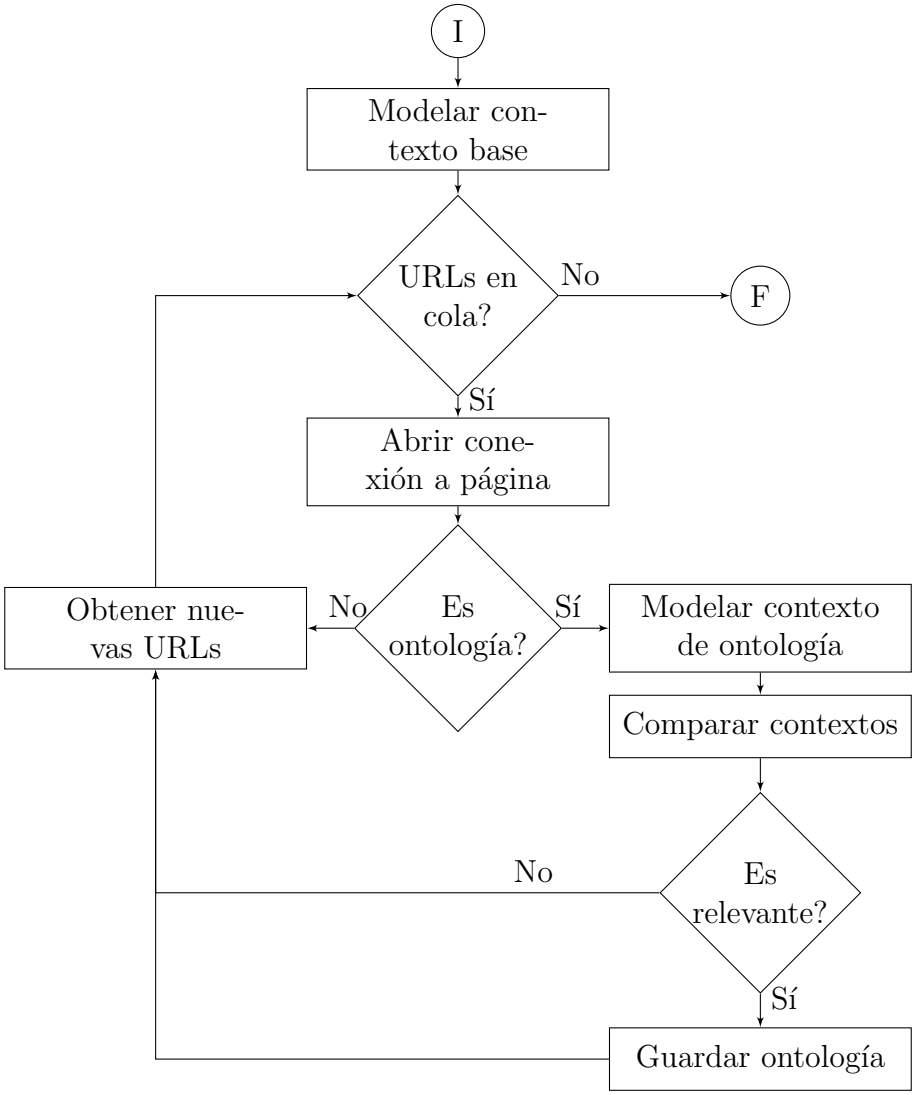


Figura 3.11: Flujo de actividades del *Crawler* ontológico

Para determinar la similaridad de las ontologías con el contexto previamente modelado, se hace uso de los mismos coeficientes mencionados en la subsección 3.1.2. Sin embargo, ya que la ontología está estructurada como un archivo XML, es necesario delimitar la extracción de palabras clave de la misma. En este caso, se decidió to-

mar en cuenta todos los nombres de las clases, ya que son los más representativos y, además, se consideran los comentarios del archivo, que proveen de una descripción a dichas clases y enriquecen el contexto de la ontología.

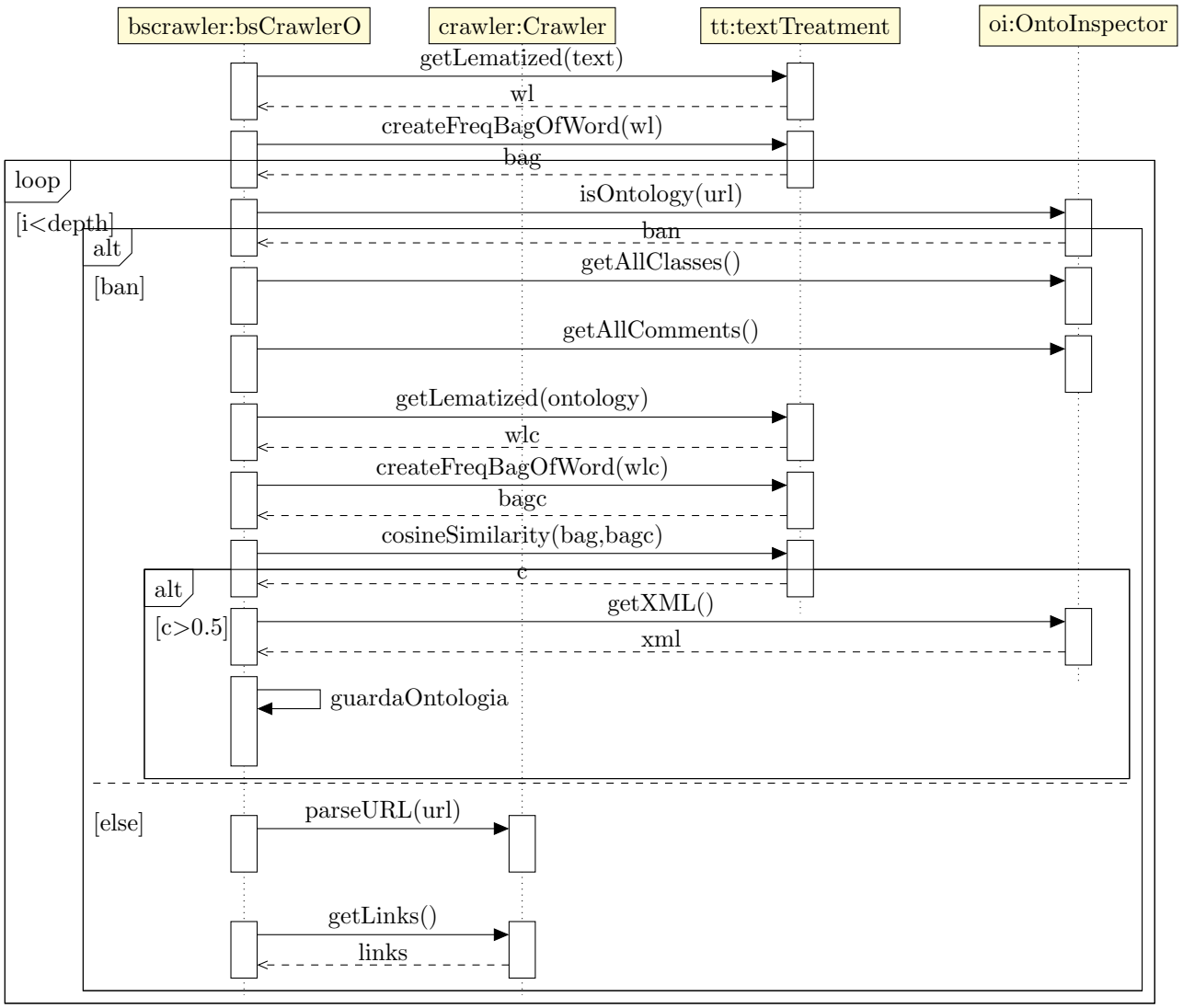


Figura 3.12: Diagrama de secuencia para la exploración en busca de ontologías

Para dicho proceso también se propone un diagrama de secuencia para detallar la interacción y comunicación entre los objetos involucrados. Este diagrama se muestra en la Figura 3.12 e ilustra el proceso de exploración en busca de ontologías.

El diagrama es similar al de la Figura 3.9, ya que la exploración de la Web se lleva de la misma manera, es decir, se exploran cada una de las URLs obtenidas empezando por las semillas. El modelado de los contextos y el cálculo del coeficiente de similaridad se hace de la misma manera.

Las ontologías deben tratarse de manera diferente para extraer de manera correcta su información, esto debido a que se trata de documentos con una estructura totalmente diferente, es decir, a diferencia de los documentos de texto que son no estructurados, las ontologías tienen una estructura bien definida y semejante a la de un árbol. En este diagrama se observa que en lugar de abrir una conexión, como se hace en el *crawler* textual, se revisa si es una ontología (actividad `isOntology()`), es decir, utilizando la biblioteca *RDFLib* se intenta hacer un *parsing* del contenido tratandolo como *xml*, *ntriples* o *n3*, si no se logra el *parsing* correctamente se determina que no es una ontología; en caso de tratarse de una ontología se extraen los nombres de las clases y los comentarios de la misma (actividades `getAllClasses()` y `getAllComments()`) para modelar el contexto. Una vez extraída esta información, el modelado y la comparación se llevan a cabo de la misma manera que en el diagrama de la Figura 3.9. La descarga también se hace de manera diferente, obteniendo la fuente XML íntegra de la ontología.

Las operaciones mencionadas constituyen el primer bloque de la condición que evalúa si se trata de una ontología. Respecto al bloque que se refiere al caso contrario puede observarse que se realiza la apertura a la conexión para extraer los hipervínculos y continuar la exploración.

3.1.4 *Parser*

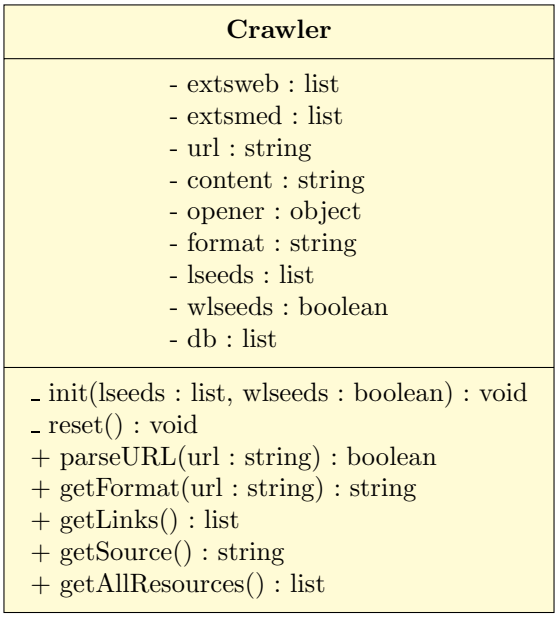


Figura 3.13: Clase para *parser*

Este componente se hace cargo de limpiar la información de los documentos que

han sido recuperados por el *crawler*, es decir, de remover las partes del documento que no sean de importancia. No se hace un manejo de ambigüedad o de los sinónimos en el texto. La limpieza mencionada consiste en remover del contenido símbolos, imagenes, etiquetas, etc. Por ejemplo: en el caso de los documentos HTML debe eliminarse todas las etiquetas que éstos contengan, de manera que sólo quede el texto, es decir, al eliminar las etiquetas del fragmento <`span class="field_title"`> `A survey of ontology evaluation techniques`</`span`> el texto resultante es `A survey of ontology evaluation techniques`.

Debido a que en el proceso de *crawling* es necesario leer el documento para llevar a cabo la comparación con el contexto, se decidió incluir el *parsing* dentro del mismo y así aprovechar que el documento ya está cargado en memoria.

Al final de este proceso deben quedar los documentos solo con la información correspondiente al dominio establecido por el contexto.

La clase que incluye estas funciones se muestra en la Figura 3.13 y es la misma que aparece en las Figuras 3.8 y 3.10. Esta clase permite obtener de manera separada las URLs encontradas en el documento, el formato del documento y su contenido libre de etiquetas.

3.1.5 Anotado semántico y poblado ontológico

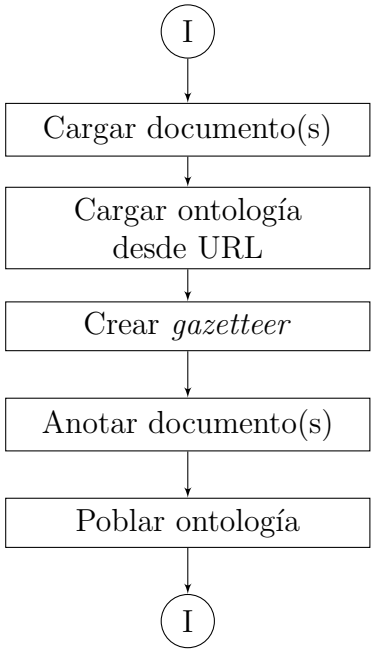


Figura 3.14: Diagrama de flujo para el anotado semántico

Esta es la última tarea a realizar para la solución al problema planteado. Este

proceso consiste en hacer el anotado semántico de los documentos que se han obtenido con el *crawler* específico para texto, después que hayan pasado por el *parser*. Además en este bloque de la propuesta se lleva a cabo el poblado de una ontología utilizando las anotaciones generadas. La ontología utilizada para anotado y poblado de la misma puede ser una de las ontologías descargadas o una construida desde cero.

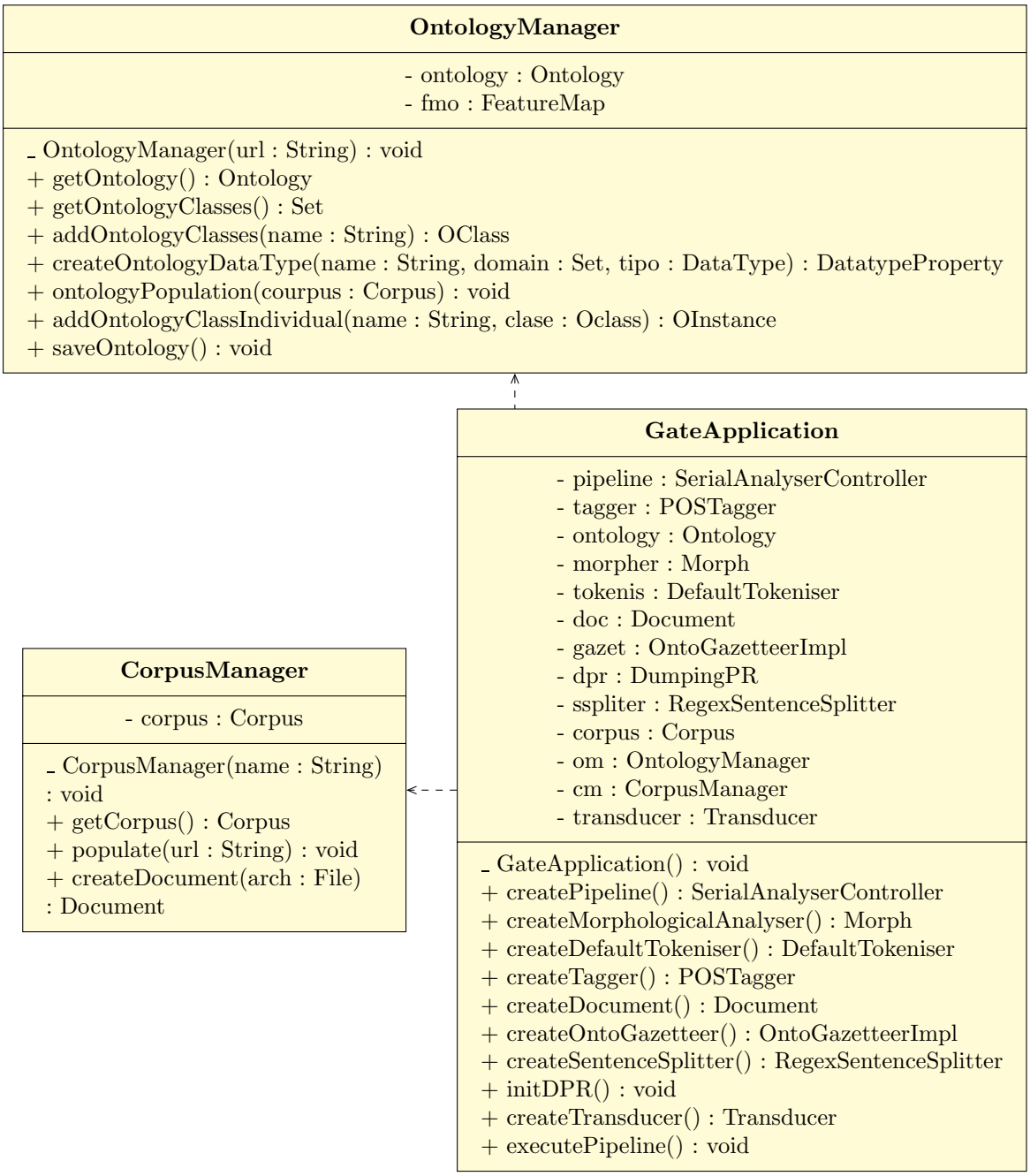


Figura 3.15: Clases para anotado semántico

En la Figura 3.14 se muestran las actividades generales necesarias para llevar a cabo el anotado semántico de los documentos. Como se muestra en el diagrama, puede tratarse de uno o más documentos, es decir, puede hacerse el anotado sobre un *corpus* de documentos en lugar de uno solo.

Para este proceso se identificaron las clases que se muestran en la Figura 3.15. El proceso de anotado semántico se lleva a cabo con las operaciones de la clase *GateApplication*. Las clases *OntologyManager* y *CorpusManager* proveen operaciones para la gestión de la ontología en la cual se basa y la gestión del *corpus*.

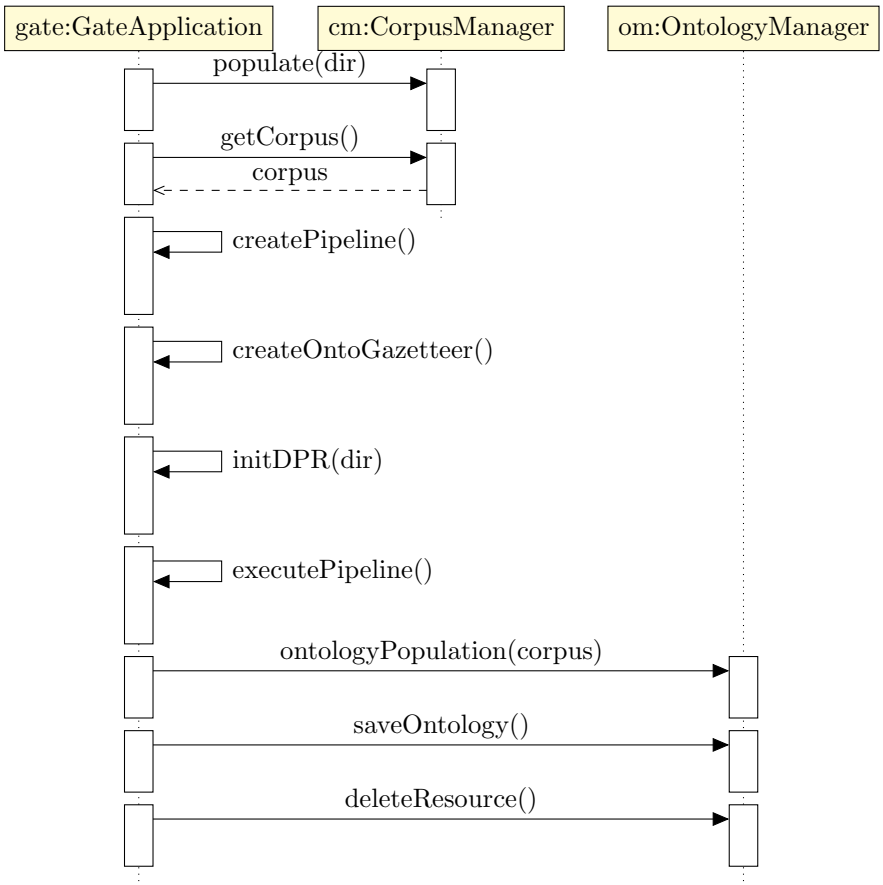


Figura 3.16: Diagrama de secuencia para el anotado semántico

En la Figura 3.16 se muestra la secuencia de actividades y la interacción entre los objetos involucrados en el proceso de anotado semántico. En ella podemos observar que se hace uso de la clase *CorpusManager* sólo para crear y poblar un corpus con uno o más documentos, es decir, el parámetro *dir* puede contener la URL de un archivo o directorio de archivos. Una vez creado y poblado el *corpus*, éste es obtenido por el objeto *gate*, lo cual es necesario para incluirlo en los recursos de procesamiento.

Las operaciones para crear dichos recursos de procesamiento, necesarias para el anotado, son: `createPipeline()`, `createOntoGazetteer()` e `initDPR()`. El *pipeline* es un controlador serial que permite cargar otros recursos de procesamiento en un orden determinado y ejecutarlos en ese mismo orden. Otro elemento clave es el *OntoGazetteer*, que permite vincular las anotaciones a clases de una ontología. La operación *executePipeline*() ejecuta los recursos de procesamiento y anota los documentos del *corpus*.

Por último, las siguientes tres operaciones permiten respectivamente: poblar la ontología con individuos en las clases vinculadas a anotaciones, guardar la ontología poblada y eliminar el recurso para liberar la memoria. Los individuos mencionados se generan con la información de las anotaciones que son vinculadas a una clase, es decir, si un fragmento del texto es etiquetado y vinculado a una clase, se crea un individuo para dicha clase y se crea una propiedad *hasName* cuyo valor es el fragmento de texto anotado.

3.2 Búsqueda y almacenamiento concurrente

El objetivo de los diagramas de flujo mostrados en esta sección es detallar el uso de hilos en la solución y exponer como pueden mejorar el desempeño del sistema y las posibles desventajas de su uso.

En la Figura 3.17 se muestra el esquema de ejecución para la apertura de URLs y obtención de recursos de las mismas. Debido a que algunos servidores pueden tardar más en responder a una petición que otros, esta arquitectura permite un mejor aprovechamiento del tiempo, al posibilitar abrir tantas URLs como hilos se estén ejecutando. De esta manera, si un hilo se tarda demasiado o no puede abrir una URL determinada, los demas estarán entregando recursos para continuar con la ejecución del *crawler*. Sin embargo, es importante tener en cuenta que una cantidad muy grande de hilos puede ocasionar un uso excesivo de algún dominio, ya que puede haber más de un hilo o incluso todos haciendo peticiones a dicho dominio.

La cantidad de hilos debe regularse para evitar incurrir en el uso excesivo mencionado y el alto consumo de ancho de banda. Además, puede resultar útil un reordenamiento en la cola de URLs, de forma que los hilos tomen URLs de dominios diferentes en caso de haber más de uno.

En la Figura 3.18 se muestra una estructura parecida a la anterior, donde se ilustra el flujo necesario para realizar la descarga de los recursos. En este caso no se incurre en un uso excesivo de los dominios, ya que se utilizan los recursos que ya han sido obtenidos mediante la parte del proceso mostrada de la Figura 3.17. Sin embargo, debe hacerse un análisis sobre la cantidad necesaria de hilos por las siguientes razones:

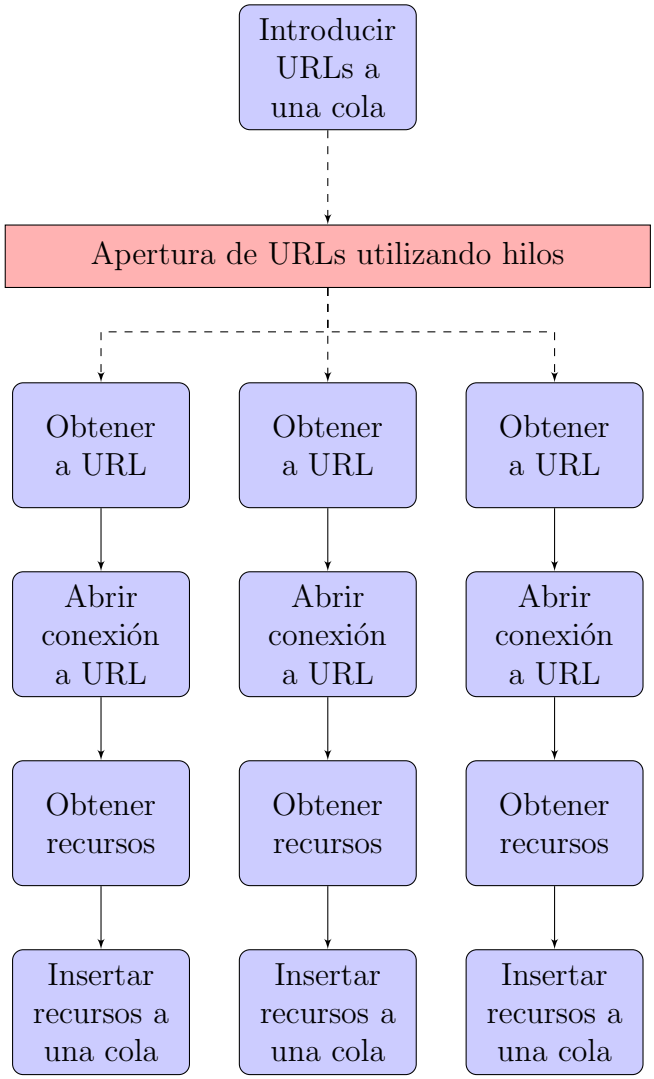


Figura 3.17: Flujo de ejecución para apertura de conexiones

- Si la cola que contiene los recursos está vacía mucho tiempo, los hilos pasarán la mayor parte del tiempo a la espera de tales recursos. Esto indica que una cantidad menor es suficiente y evita tener hilos ejecutándose sin aportar al proceso general.

- Por otro lado, si la cantidad de hilos es muy pequeña y no pueden seguir el paso al que los recursos entran en la cola, ésta continuará creciendo durante la ejecución consumiendo cada vez más memoria. En este caso, una cantidad mayor de hilos es recomendable para aprovechar correctamente los recursos.

Puede observarse que en la Figura 3.18 no se incluye un bloque condicional para indicar que la similaridad debe superar un umbral definido para poder descargar el recurso, esto se debe a que tales diagramas son sólo para ilustrar de manera general

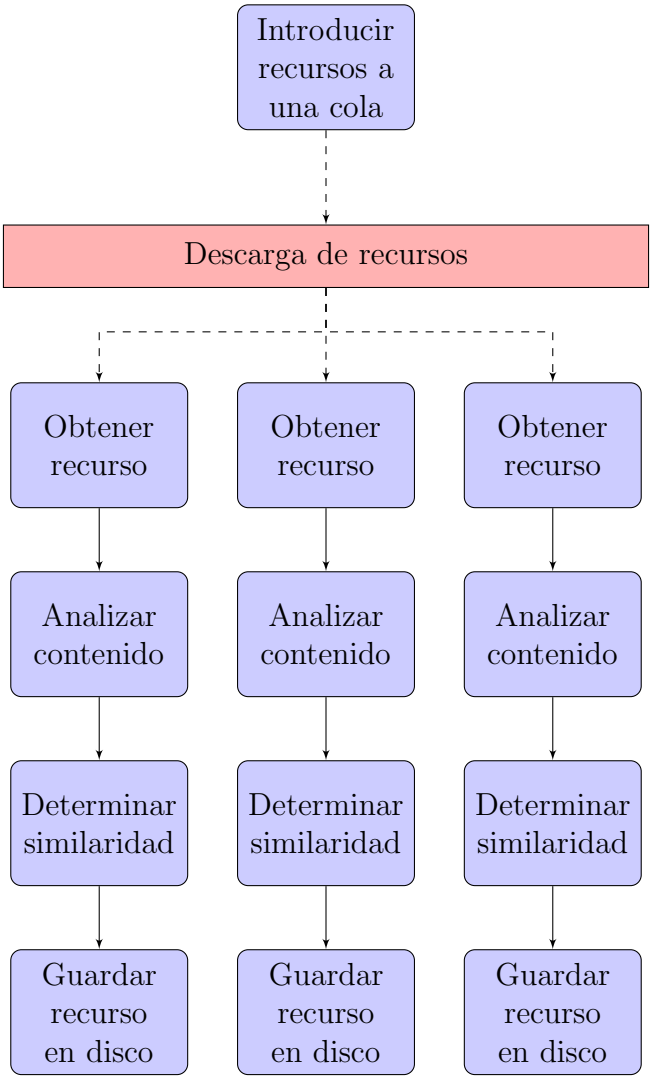


Figura 3.18: Flujo de ejecución para descargas

la arquitectura utilizando hilos.

El diagrama de la Figura 3.19 muestra, también de manera general, el flujo principal del sistema de exploración. Este es conducido por el hilo principal de ejecución y tiene como objetivo llevar un control de la profundidad que se lleva en la exploración. Además, en este proceso se gestionan los hilos que realizan la exploración y descarga concurrentes.

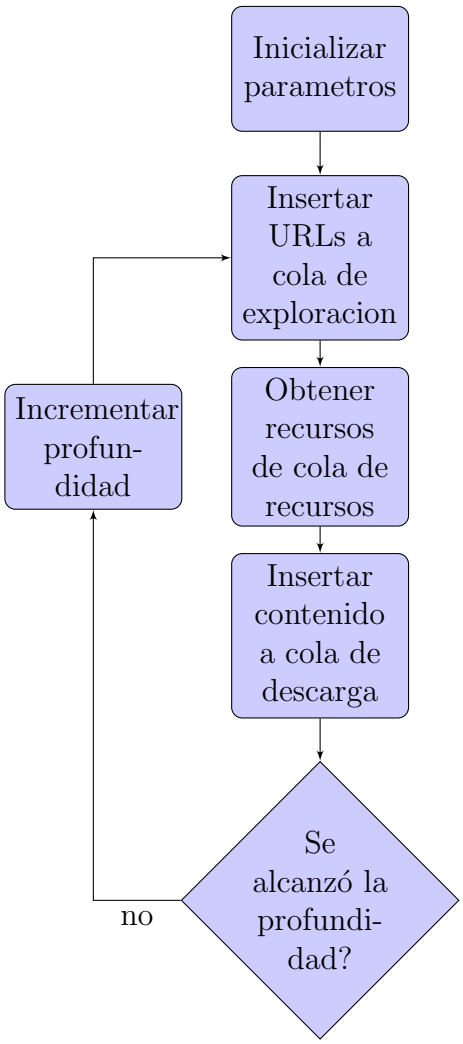


Figura 3.19: Flujo principal de la exploración

Capítulo 4

Implementación del sistema

Es importante contar con la infraestructura adecuada para la implementación de un determinado sistema de software, dicha infraestructura debe cubrir los requerimientos de ejecución del sistema. Además, es de igual importacia la selección del lenguaje de programación adecuado. Esto permite una ejecución eficiente del software y facilita la obtención de los resultados correspondientes.

También deben documentarse de manera adecuada los códigos y especificarse los algoritmos utilizados. Con el objetivo de generar una implementación que sea fácil de entender y reutilizar. En este capítulo se describen los detalles de la infraestructura utilizada y de la implementación.

4.1 Infraestructura

A continuación se describen en detalle las especificaciones de software y hardware necesarios para la correcta ejecución del sistema.

4.1.1 Hardware

Para dar solución a la problemática expuesta en esta tesis se utilizó un equipo de cómputo de escritorio con las siguientes características:

- Procesador Intel Core i7 modelo 9200 con 4 núcleos a 2.67GHz y capacidad para 8 hilos en Hyperthreading[1].
- 8GB de memoria RAM a 1067 MHz
- Disco duro con capacidad de 750GB

[1]Marca de Intel para denominar su implementación de la tecnología Multithreading Simultáneo. Permite el procesamiento en paralelo de los programas preparados para ejecutarse en hilos, dentro de un solo procesador

Es importante mencionar que se señalan sólo las características de importancia para el desarrollo de este proyecto, ya que el sistema incurre en un alto gasto de memoria principal, almacenamiento, procesamiento y ancho de banda. Para el caso del ancho de banda, se cuenta con una IP estática asignada a la máquina mediante la red cableada, dicha IP ha sido proveida por el CINVESTAV, esto permite que se tenga una conexión estable a todas horas.

4.1.2 Software

El equipo de cómputo designado para el trabajo cuenta con el sistema operativo Ubuntu 11.10, elegido por ser una plataforma libre y tener compatibilidad con las bibliotecas requeridas. Para el desarrollo de la propuesta se eligieron como lenguajes de programación: **Python** para programar los *crawlers* y el *parser*, necesarios para la recuperación de información textual y ontológica, y **Java** para la rutina de anotado semántico y poblado de ontologías que actúa sobre los documentos de texto descargados, utilizando una de las ontologías descargadas o una nueva específica para dicho propósito.

Se optó por trabajar con python para desarrollar los primeros módulos de la solución por ser un lenguaje de alto nivel, el cual cuenta con un gran número de bibliotecas que resultan útiles y benefician esta implementación. En cuanto a la parte de anotado semántico, se eligió Java como lenguaje para utilizar los *plugins*[2] de GATE (por sus siglas en ingles *General Architecture for Text Engineering*) y facilitar la tarea.

El equipo de cómputo cuenta con la versión 2.7.2 de python, y se instalaron las siguientes bibliotecas como soporte al desarrollo:

- **Beautifulsoup:** es un analizador HTML con tolerancia a errores. Esta biblioteca facilita el tratamiento de los archivos HTML, permitiendo una fácil extracción de las URLs al *crawler* y facilitando la interpretación del documento en diferentes codificaciones.

- **NLTK (Natural Language ToolKit):** es una biblioteca para el procesamiento del lenguaje natural. En este desarrollo se hace uso de herramientas (*Stemmer, Lematizer, Tokenizer*) para hacer un tratamiento al texto, antes de calcular el coeficiente de similaridad.

- **RDFlib:** esta biblioteca permite el tratamiento y exploración de ontologías tratándolas como grafos.

[2]Es un complemento que se relaciona con una aplicación para aportarle una función nueva y generalmente muy especifica

En el caso de la aplicación para anotado semántico y poblado de ontologías, se cargaron al proyecto los *jars*[3] contenidos en el directorio del mismo. Específicamente se incluyeron *gate.jar* encontrado en el subdirectorio *GATE-DIR/bin/*, *tools.jar* encontrado en *GATE-DIR/plugins/Tools/* y todos los archivos *jar* contenidos en *GATE-DIR/lib/*.

4.2 Modelado del contexto

Como se mencionó en la sección 3.1.1 de este documento, se define como contexto la representación vectorial de los documentos utilizando bolsas de palabras. La implementación de este módulo consiste en el tratamiento necesario del texto para crear dicha bolsa de palabras. Los pasos necesarios para realizar este proceso se muestran en el Algoritmo 1.

Este algoritmo detalla las operaciones necesarias para modelar el contexto a partir de un documento. El proceso inicia con la apertura y lectura de un archivo de texto, después se eliminan del texto los símbolos que pudieran generar ruido, es decir, dado que sólo interesa el texto, se elimina todo símbolo que no sea una letra; una vez hecho esto se separa el texto en palabras generando un arreglo con las mismas. Por último se recorre el arreglo palabra por palabra y se aplica a cada una de ellas un proceso de *stemming*[4] y otro para convertirlas a minúsculas, se verifica si la palabra ya ha sido agregada a la bolsa y, de ser así, se incrementa su frecuencia en 1, en caso contrario se agrega a la bolsa con una frecuencia igual a 1.

Las estructuras de datos utilizadas en el Algoritmo 1 se describen a continuación:

- *Archivo*: es la URL del archivo de entrada, este parametro es de tipo cadena.
- *texto*: se refiere al texto leído del archivo que fué cargado en memoria como una cadena de caracteres.
- *arreglo*: es un arreglo de palabras obtenido al extraer los términos (*tokens*) del documento.
- *bag*: bolsa de palabras que retorna el algoritmo despues de construirla, la cual es un diccionario de la forma {*"palabra1":frecuencia1, "palabra2":frecuencia2, . . . , "palabran":frecuencian*}.
- *palabra*: es cada una de las palabras de acuerdo con el recorrido del ciclo *para todo* y es de tipo cadena.

[3]JAR (por sus siglas en inglés, Java ARchive) es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java o empaquetar bibliotecas Java

[4]Llevar las palabras a su raíz (lematización)

Algoritmo 1 Modelado del contexto a partir de un documento

Entrada: *Archivo* de texto
Salida: Bolsa de palabras *bag*

1: Abrir *Archivo* // `Se carga el archivo a memoria`
2: *texto* ← leer *Archivo* // `Leer el texto contenido en el archivo`
3: Eliminar símbolos inválidos // `Se eliminan del texto símbolos que no interesan, dejando solo letras`
4: *arreglo* ← Obtener *tokens* de *texto* // `Se crea un arreglo de palabras a partir del texto`
5: Eliminar *StopWords* // `Se eliminan las palabras que son muy comunes en el lenguaje`
6: *bag* ← {}
7: **Para Todo** *palabra* ∈ *arreglo* **Hacer** // `Recorrido de cada una de las palabras`
8: *palabra* ← Stemming(*palabra*) // `Se lleva a las palabras a su raíz`
9: *palabra* ← minusculas(*palabra*) // `Se convierte el texto de la palabra a minúsculas`
10: **Si** *palabra* ∈ *bag* **Entonces** // `Construcción de la bolsa de palabras`
11: *bag*[*palabra*]+ = 1 // `Si la palabra ya ha sido agregada a` *bag* `sumarle 1 a su frecuencia`
12: **Si No**
13: *bag*[*palabra*] ← 1 // `Si no ha sido agregada la palabra, se agrega con frecuencia 1`
14: **Fin Si**
15: **Fin Para**
16: **Devolver** *bag*

El algoritmo presentado para modelar el contexto corresponde a las funciones `getLematized()` y `createFreqBagOfWords()` de la clase *textTreatment*, la primera se encarga de limpiar el texto, separarlo en palabras, eliminar las *StopWords* y llevar cada palabra a su raíz, y la segunda se encarga de construir la bolsa de palabras utilizando su frecuencia. El proceso completo de modelado del contexto es referido en las secciones siguientes mediante una línea dentro de algunos algoritmos.

En la subsección 3.1.2 se mencionan algunas métricas de similitud de vectores, de las cuales se eligió la de cosenos para su implementación en la comparación de bolsas de palabras, de acuerdo con los resultados de correlación mostrados en [Lee et al., 2005], donde se puede apreciar cómo esta métrica supera a las otras tres consideradas. Dado que las bolsas de palabras son vectores multidimensionales y no tienen un orden específico en su estructura, el cálculo del coeficiente de similaridad se implementó de acuerdo al Algoritmo 2. Este algoritmo corresponde a la función `cosineSimilarity()` de la clase *textTreatment*.

Algoritmo 2 Cálculo de coeficiente de similaridad

Entrada: A,B Bolsas de palabras que se quieren comparar
Salida: Coeficiente de similaridad cs

1: $ppunto \leftarrow 0$
2: $sumaA \leftarrow 0$
3: $sumaB \leftarrow 0$
4: **Para Todo** $dimension, valor \in A$ **Hacer** // `Recorrido sobre la bolsa A`
5: $sumaA += valor * valor$ // `Cálculo del producto punto de` $A * A$
6: **Si** $dimension \in B$ **Entonces** // `Verificar si la segunda bolsa contiene la misma dimensión`
7: $ppunto += valor * B[dimension]$ // `Cálculo del producto punto de` $A * B$
8: **Fin Si**
9: **Fin Para**
10: **Para Todo** $dimension, valor \in B$ **Hacer** // `Recorrido sobre la bolsa B`
11: $sumaB += valor * valor$ // `Cálculo del producto punto de` $B * B$
12: **Fin Para**
13: $cs \leftarrow ppunto/((sumaA * sumaB)^{(1/2)})$ // `Cálculo del coeficiente de similaridad de acuerdo con` (3.6)
14: **Devolver** cs

Como puede observarse en el algoritmo, existen dos bloques *Para todo* para obtener la suma requerida para el cálculo de la magnitud de cada uno de los vectores A y B. Para el producto punto entre ambos vectores se toman en cuenta sólo las dimensiones en común, lo cual puede ser ejemplificado como sigue: suponiendo que se tengan dos vectores $< x, y >$ y $< x, y, z >$, cuyos valores sean $< a, b >$ y $< c, d, e >$ respectivamente el producto punto es entonces $a * c + b * d + 0 * e$, sin embargo, los elementos multiplicados por 0 se descartan y por ende sólo se requieren los elementos comunes. Las estructuras de datos manejadas en este algoritmo se describen en los siguientes puntos:

- A **y** B**:** son las bolsas de palabras que seran comparadas y son de tipo Dict (diccionario en python).
- *ppunto***:** es un punto flotante donde se guarda el producto punto de los vectores A y B.
- *sumaA***:** es un punto flotante donde se guarda el producto punto de $A * A$.
- *sumaB***:** es un punto flotante donde se guarda el producto punto de $B * B$.
- *dimension***:** es una cadena que representa a cada una de las palabras en la bolsa en las iteraciones de los ciclos *Para Todo*.

- *dimension*: es un punto flotante que representa a cada una de las frecuencias en la bolsa en las iteraciones de los ciclos *Para Todo*. Esta frecuencia es la correspondiente a cada palabra (*dimension*) en las iteraciones.
- *cs*: es un punto flotante que contiene el coeficiente de similaridad obtenido.

Este algoritmo también es referido en secciones posteriores como una función `coeficienteSimilaridad()` en los algoritmos.

Algoritmo 3 Mejora de contexto base en tiempo de ejecución

Entrada: A,B Contexto base y nuevo contexto respectivamente
Salida: Conexto mejorado A

1: **Para Todo** $dimension, valor \in B$ **Hacer** // `Recorrido sobre la bolsa B`
2: **Si** $dimension \in A$ **Entonces** // `Verificar si` A `ya contiene la palabra`
3: $A[dimension] = (valor + B[dimension])/2$ // `Se promedian las dimensiones`
4: **Si No**
5: $A[dimension] = B[dimension]$ // `Se agrega como nueva dimensión`
6: **Fin Si**
7: **Fin Para**
8: **Devolver** A

En cuanto a la mejora del contexto mencionada en la subsección 3.1.1, detallada mediante la ecuación (3.2) e incluida en la clase *textTreatment* como la operación `improoveContext()`, la implementación correspondiente funciona de acuerdo al Algoritmo 3.

En este algoritmo se utilizan las mismas variables que en el Algoritmo 2. En este caso se hace un solo recorrido sobre el contexto nuevo encontrado relevante. Se verifica si cada palabra en el nuevo contexto ya está en él base, de ser asi, se calcula un promedio entre sus pesos, dicho promedio es asignado como valor a la dimensión representada por la palabra actual. En caso de no encontrarse la palabra en el contexto base, se agrega como una dimensión nueva cuyo valor es el peso que ésta tiene en el nuevo contexto.

4.3 *Crawler* textual

El funcionamiento del *crawler* textual puede resumirse como la exploración de URLs en busca de documentos similares al contexto. El pseudocódigo mostrado en el Algoritmo 4 describe de manera general este funcionamiento, posteriormente se muestra cómo se estructuran algunos bloques del pseudocódigo para ejecutarse utilizando hilos de ejecución. Estos algoritmos corresponden a las funciones de navegación que

Algoritmo 4 *Crawler* para recuperación de archivos de texto

Entrada: *Archivo* de texto (servirá como contexto), profundidad P, URLs semilla S

1: $cBase \leftarrow$ bolsaPalabras($Archivo$) // Representación vectorial del documento de acuerdo con (3.1)
2: $paginas \leftarrow S$ // Se asignan las semillas a una lista iterable
3: $\alpha \leftarrow 0.6$ // Se designa un umbral para decidir si un documento es relevante
4: $\beta \leftarrow 0.75$ // Se designa un umbral para decidir si se mejora el contexto
5: $\delta \leftarrow 0.2$ // Se designa un delta para determinar si las nuevas URLs son relevantes (predicción de relevancia)
6: **Mientras** No se alcance la profundidad P **Hacer** // Recorrido en profundidad
7: $urls \leftarrow []$ // Inicializar cola temporal de URLs
8: **Para Todo** $j \in paginas$ **Hacer** // Recorrido en amplitud
9: Abrir j // Abrir conexión a j y cargar su contenido a memoria
10: $C \leftarrow$extraer contenido de j // Asignar el texto fuente a C
11: $links \leftarrow$ extraer enlaces de j // Asignar los enlaces contenidos en j a $links$
12: **Para Todo** $link \in links$ **Hacer** // Revisar contexto de las URLs
13: **Si** Similaridad de contexto de $link >= \delta$ **Entonces**
14: Agregar $link$ a $urls$ // Se agregan las nuevas URLs a $urls$
15: **Fin Si**
16: **Fin Para**
17: $cDoc \leftarrow$ bolsaPalabras(C) // Crear bolsa de palabras a partir del contenido según (3.1)
18: $cs \leftarrow$ coeficienteSimilaridad($cDoc$,$cBase$) // Calcular la similitud entre las bolsas de palabras
19: **Si** $cs >= \alpha$ **Entonces**
20: Guardar documento C en disco duro // Descarga de documento de memoria principal a disco duro
21: **Si** $cs >= \beta$ **Entonces**
22: $cBase \leftarrow$ Mezclar($cBase$, $cDoc$) // Mejora de contexto mediante (3.2)
23: **Fin Si**
24: **Fin Si**
25: **Fin Para**
26: $paginas \leftarrow urls$ // Se agregan las nuevas URLs a la cola de exploración
27: **Fin Mientras**

provee la clase *Crawler* y la gestión de hilos de ejecución que se lleva a cabo con las funciones de la clase *bsCrawler*.

El Algoritmo 4 recibe como entradas: *Archivo*, el cual es texto que servirá para modelar el contexto base; P, que es un número entero que establece la profundidad hasta la que se quiere explorar; y S, el cual es un conjunto de URLs a partir de las cuales se inicia la navegación. El resto de variables utilizadas en el proceso se describen en los siguientes puntos:

- *cBase*: contexto base utilizado, formado a partir del archivo de entrada de acuerdo con el Algoritmo 1.
- *paginas*: cola que funciona como frontera del *crawler* donde se agregan nuevas URLs obtenidas de las páginas visitadas.
- α: porcentaje que debe ser superado por el coeficiente de similaridad para considerar al documento como relevante.
- β: porcentaje que debe ser superado por el coeficiente de similaridad para considerar al documento en la mejora del contexto base.
- j: variable utilizada para iterar en el ciclo *para todo* de la línea 8, representando cada una de las páginas contenidas en el arreglo *páginas*.
- *urls*: variable de carácter temporal que funciona como cola para almacenar las URLs extraídas de las páginas en la $i$ésima profundidad.
- *link*: variable utilizada para iterar en el ciclo *para todo* de la línea 12, representando cada una de las URLs en *links*.
- C: variable de tipo cadena donde se almacena el texto del documento abierto
- *links*: conjunto de URLs extraidas de la $j$ésima página
- *cDoc*: contexto del nuevo documento
- *cs*: coeficiente de similaridad entre los dos contextos
- δ: es un valor que debe ser superado en la de predicción de relevancia para agregar el nuevo enlace a la frontera.

En este algoritmo se inicia el *crawler* con la construcción de una bolsa de palabras a partir del documento de entrada, se agregan las semillas a la variable *paginas* y se fija el umbral. Después, mediante un ciclo *Mientras*, se explora hasta alcanzar una profundidad P. Un segundo ciclo (*para todo*) es utilizado para realizar una exploración en amplitud.

En este algoritmo se hace una referencia mínima al bloque de predicción de links (linea 12 del algoritmo), que de acuerdo con [Cheng et al., 2008], consiste en analizar el contexto de la URL en el archivo HTML, es decir, extraer el contenido textual que lo rodea y compararlo con el contexto base para determinar su relevancia. Se hace esta consideración teniendo en cuenta que el texto que rodea a dicha URL generalmente trata el tema referenciado por la URL. El Algoritmo 6 contiene pasos más detallados de este bloque.

El cómputo del coeficiente de similaridad y el proceso de descarga del archivo (indicado en las líneas 18 y 20 del Algoritmo 4) se realizan en hilos de ejecución separados al hilo principal de ejecución, con el objetivo de acelerar la exploración del *crawler*. Para hacer este manejo se utiliza el objeto *Queue* de Python, el cual es una cola diseñada para trabajar con hilos de ejecución sin la necesidad de bloquear el recurso (cola) para utilizarlo. El agoritmo 5 detalla el pseudocódigo de esta función.

Algoritmo 5 Función de descarga de contenido del *crawler*

```
1:  Mientras No se alcance la profundidad o no esté vacía la cola QtoDown Hacer
2:    Extraer url, formato y contenido de la cola    // Cada posición es un
      arreglo con estos 3 datos
3:    cDoc ← bolsaPalabras(contenido)    // Se genera la bolsa de palabras a
      partir del documento explorado
4:    cs ← coeficienteSimilaridad(cDoc,cBase)    // Cálculo de la similaridad
      entre las bolsas de palabras
5:    Si cs >= α Entonces
6:      Si cs >= β Entonces
7:        cBase ← Mezclar(cBase, cDoc)    // Si se supera β se realiza una
          mezcla de los contextos para enriquecer al contexto base
8:      Fin Si
9:      nombreArchivo ← url + "." + formato    // Se estructura el nombre
        del archivo
10:     Guardar contenido en disco duro    // Descarga del documento a disco
        duro
11:   Fin Si
12: Fin Mientras
```

En el Algoritmo 5 la condición del bucle *mientras* se valida utilizando un atributo del objeto (profundidad), que se actualiza desde el hilo principal de ejecución conforme la exploración avanza. Las estructuras de datos utilizadas por este algoritmo se describen a continuación:

- *url*: cadena que denota la URL de donde se extrajo el contenido
- *formato*: cadena que describe el formato del documento obtenido

- *contenido*: cadena que contiene el texto extraído del documento
- *cBase*: contexto base utilizado, formado a partir del archivo de entrada de acuerdo con el Algoritmo 1.
- *cDoc*: contexto del nuevo documento
- *cs*: coeficiente de similaridad entre los dos contextos
- α: porcentaje que debe ser superado por el coeficiente de similaridad para considerar al documento como relevante.
- β: porcentaje que debe ser superado por el coeficiente de similaridad para considerar al documento en la mejora del contexto base.

Puede observarse que este algoritmo hace uso de algunas variables del algoritmo 4.

Como se mencionó en la sección 3.2, la apertura de páginas también puede hacerse utilizando multiples hilos de ejecución. Para llevar a cabo esto, se separan algunas operaciones a una función de apertura de conexiones; dicha función se describe en detalle en el Algoritmo 6. Puede observarse que estos hilos se encargan de extraer la información útil de cada documento y la agregan a la cola de descarga, además agregan las URLs a una cola frontera, la cual es utilizada por el hilo principal para extraer los nuevos conjuntos de hipervínculos que se explorarán.

En el Algoritmo 6 también se hace uso de la variable *cBase* perteneciente al Algoritmo 4. Las variables C, j y δ son las mismas utilizadas en el Algoritmo 4 y *QtoDown* es la variable antes mencionada como cola de descarga del *crawler*. Este algoritmo se encarga de abrir las conexiones a cada una de las URLs de la frontera y obtener los datos de interes de las mismas. Las variables utilizadas se describen en los siguientes puntos:

- *url*: es la URL siguiente en la frontera para ser tratada
- *html*: árbol creado a partir de la estructura de etiquetas del documento HTML
- *padre*: etiqueta padre de la URL en la estructura de árbol *html*
- *linkContext*: texto que rodea a la etiqueta $< a >$ que contiene la URL
- *bLink*: bolsa de palabras creada a partir de *linkContext*
- *links*: arreglo temporal para almacenar URLs
- $QFrontier$: es un objeto de tipo *Queue* que funciona como frontera del *crawler*
- $Visitados$: arreglo donde se graban las URLs que ya han sido exploradas para evitar abrirlas de nuevo.

Algoritmo 6 Función de apertura de conexiones a las páginas

1: **Mientras** no esté vacía la cola *QtoParse* **Hacer**
2: Extraer *url* de la cola // Cada posición contiene una url
3: Abrir conexion a *url* // Se abre la conexión a la página correspondiente
4: Extraer contenido C // Se extrae la información de interés
5: Generar arbol *html* // Se crea el arbol correspondiente al contenido HTML
6: $links \leftarrow []$
7: **Para Todo** $j \in html$ **Hacer** // Tratamiento de las nuevas URLs
8: Extraer formato // Se busca por una extensión dentro de la URL
9: **Si** (Se identificó un formato **&** (formato es HTML **O** formato es PDF))**O** formato es desconocido **Entonces** // Se verifica que sea un formato de interés o desconocido para agregarlo
10: *padre* ← Navegar a etiqueta padre de j // Se busca en el arbol el nodo padre de la etiqueta $< a >$ que contiene la url
11: *linkContext* ←Extraer contenido de *padre* // Se extrae el texto que rodea a la URL
12: *bLink* ←bolsaPalabras(*linkContext*) // Se crea una bolsa de palabras para el texto que rodea la URL
13: $cs \leftarrow$ coeficienteSimilaridad(*bLink*,*cBase*) // Calcular la similitud entre los contextos
14: **Si** $cs >= \delta$ **Entonces**
15: Agregar j a la *links* // Se agregan la URL y su correspondiente contexto
16: **Fin Si**
17: **Fin Si**
18: **Fin Para**
19: Agregar *links* a *QFrontier* // Se agregan los nuevos vínculo-contexto a la frontera
20: Agregar datos extraidos a *QtoDown* // Se agrega el contenido a la cola de descarga
21: Agregar *url* a *Visitados* // La URL procesada se agrega a un arreglo de URLs visitadas para evitar volver a tratarlas
22: **Fin Mientras**

Existen algunos detalles importantes en esta implementación que se mencionan en los siguientes puntos:

- **Filtrado de extensiones:** se determina el formato el contenido de una determinada URL buscando una extensión, y se descartan aquellas URLs que contengan extensiones como: jpg, jpeg, gif, bmp, png, wmv, mpg, entre otras. Con esto se evita la exploración innecesaria de este tipo de archivos, ya que no

son de interés para este proyecto. Este filtro puede observarse en el bloque *para todo* iniciado en la línea 9 del Algoritmo 6.

- **Tratamiento de archivos PDF:** los archivos PDF son convertidos a texto plano (utilizando un comando de sistema para convertirlo de formato PDF a txt) para poder construir la bolsa de palabras y llevar a cabo el cálculo del coeficiente de similaridad.

- **Análisis de contexto de los enlaces:** se lleva a cabo un cómputo de similaridad del contexto base con el texto que rodea al enlace. Con esto se realiza una predicción aproximada de la relevancia de los enlaces que están por explorarse. En los algoritmos 4 y 6 se observa esta característica en los bloques iniciados por las líneas 12 y 9 respectivamente.

- ***Tunneling*:** con la ayuda de la predicción de enlaces y la relevancia de los documentos, se descarta la exploración, si se encuentra un enlace no relevente cuyos dos antecesores inmediatos fueron no relevantes.

4.4 *Crawler* ontológico

Esta parte de la implementación comparte muchas de sus características con el *crawler* textual, por ejemplo el uso de multiples hilos de ejecución para la apertura de varias conexiones a la vez, además de el cálculo de similaridad con el contexto y la descarga de recursos al disco duro.

Este *crawler* difiere del textual en el orden de las operaciones, es decir, en este caso en lugar de realizar la apertura del recurso con una operación de tratamiento de HTML, se lleva a cabo utilizando la biblioteca **rdflib** con el objetivo de determinar si se trata de una ontología. En el caso de la descarga de recursos, el modelado del contexto se hace extrayendo los nombres de las clases, propiedades (tanto de tipo de dato como de objeto) y los comentarios de la ontología. El Algoritmo 7 muestra estas diferencias ilustrando el funcionamiento general del *crawler* para recuperación de ontologías.

La extracción de información de las ontologías consiste en realizar una navegación en su estructura tratándola como árbol. Dicha estructura consiste de un conjunto de nodos (clases, individuos, propiedades) conectados entre sí mediate relaciones, por ejemplo: un individuo está conectado a una o más clases mediante una relación jerárquica que establece la o las clases a las que pertenece dicho individuo.

Es importante señalar que el análisis de la información de la ontología para determinar su relevenacia se lleva a cabo mediante un metodo sintáctico, el cual consiste en la extracción de las etiquetas que establecen los nombres para cada uno de los

Algoritmo 7 *Crawler* para recurperación de ontologías

Entrada: *Archivo* de texto (servirá como contexto), profundidad P, URLs semilla S

1: $cBase \leftarrow$ bolsaPalabras($Archivo$)
2: $paginas \leftarrow S$
3: $\alpha = 0.6$
4: **Mientras** No se alcance la profundidad P **Hacer** // Recorrido en profundidad
5: $urls \leftarrow []$
6: **Para Todo** $j \in paginas$ **Hacer**
7: **Si** j es ontología **Entonces** // Comprueba si el recurso que se está abriendo es una ontología
8: $C \leftarrow$ Recurso j
9: $clases \leftarrow$ getAllClasses(C) // Extracción de los nombres de las clases de la ontología
10: $propiedades \leftarrow$ getAllProperties(C) // Extracción de las propiedades de la ontología
11: $comentarios \leftarrow$ getAllComments(C) // Extracción de los comentarios de la ontología
12: $cDoc \leftarrow$ bolsaPalabras($clases$ + $propiedades$ + $comentarios$) // Construcción de bolsa de palabras a partir de la información extraída
13: $cs \leftarrow$ coeficienteSimilaridad($cDoc$,$cBase$)
14: **Si** $cs >= \alpha$ **Entonces**
15: Guardar ontología C en disco duro // Se guarda el recurso en formato RDF
16: **Fin Si**
17: **Si No** // En caso de no ser ontología, se extraen los links de la página
18: Abrir página j
19: Extraer enlaces $links$
20: Agregar $links$ a $urls$
21: **Fin Si**
22: **Fin Para**
23: $paginas \leftarrow urls$ // Se agregan las nuevas urls a la cola del crawler
24: **Fin Mientras**

elementos de la ontología (clases, propiedades, individuos) y la extracción de la información contenida en los comentarios. El texto obtenido es tratado de la misma manera que los documentos en el *crawler* textual para formar una bolsa de palabras que pueda ser comparada con el contexto base.

En lo que se refiere al tratamiento de las páginas (HTML) visitadas, sólo se lleva a cabo una extracción de URLs para continuar con la navegación, ya que para este *crawler* sólo son de interés las ontologías. De acuerdo con esto no es necesario extraer el contenido de los HTML porque no es de interés a este módulo. El tratamiento del contexto de las URLs se hace de la misma manera que para el *crawler* ontológico.

Las variables utilizadas en el algoritmo se describen en los siguientes puntos:

- *cBase*: contexto base utilizado, formado a partir del archivo de entrada de acuerdo con el Algoritmo 1.
- *paginas*: cola que funciona como frontera del *crawler* donde se agregan nuevos enlaces (URIs y URLs) obtenidos de las páginas visitadas.
- α: porcentaje que debe ser superado por el coeficiente de similaridad para considerar a la ontología como relevante.
- j: variable utilizada para iterar en el ciclo *para todo* de la línea 6, representando cada una de las páginas contenidas en el arreglo *páginas*.
- *urls*: variable de carácter temporal que funciona como cola para almacenar los enlaces (URLs y URIs) extraídos de las páginas en la $i$ésima profundidad.
- C: variable de tipo cadena donde se almacena el recurso
- *links*: conjunto de enlaces (URLs y URIs) extraidos de la $j$ésima página
- *cDoc*: contexto de la ontología
- *cs*: coeficiente de similaridad entre los dos contextos

En el Algoritmo 7 se muestran tres operaciones sobre ontologías (`getAllClasses()`, `getAllProperties()` y `getAllComments()`) cuya especificación se detalla en los algoritmos 8, 9 y 10 respectivamente.

Algoritmo 8 Obtencion de nombres de clases de la ontología

Entrada: O ontología
Salida: $Clases$ Nombres de las clases
1: $Clases \leftarrow []$
2: **Para Todo** $a, b, c \in O.triples((None, RDF.type, OWL.Class))$ **Hacer** // `Recorrido por las tripletas de la ontología`
3: Agregar $s.split(\text{“\#”})[1]$ a $Clases$
4: **Fin Para**
5: **Devolver** $Clases$

Estos 3 algoritmos realizan operaciones similares en busca de datos distintos, es decir, para la extracción de los datos de la ontología se hace un recorrido de las tripletas de la misma. El bloque *Para Todo* en cada algoritmo hace el recorrido sobre

Algoritmo 9 Obtencion de nombres de propiedades de la ontología

Entrada: O ontología
Salida: *Propiedades* Nombres de las propiedades

1: $Propiedades \leftarrow []$
2: **Para Todo** $a, b, c \in O.triples((None, RDF.type, OWL.DatatypeProperty$**O** $OWL.ObjectProperty))$ **Hacer** // `Recorrido por las tripletas de la ontología`
3: Agregar $s.split(\text{“\#”})[1]$ a $Propiedades$
4: **Fin Para**
5: **Devolver** $Propiedades$

Algoritmo 10 Obtencion de comentarios de la ontología

Entrada: O ontología
Salida: *comentarios* Textos de los comentarios

1: $comentarios \leftarrow []$
2: **Para Todo** $a, b, c \in O.triples((None, None, None))$ **Hacer** // `Recorrido por las tripletas de la ontología`
3: Agregar $O.comment(a)$ a $comentarios$
4: **Fin Para**
5: **Devolver** $comentarios$

dichas tripletas, cuya estructura puede ser vista como sigue: $< a, b, c >$ donde a representa el dato de interés, b indica que se trata de un “tipo” y c establece qué tipo, por ejemplo:

- $a \leftarrow$ `http://www.owl-ontologies.com/Humano.owl#Alumno`
- $b \leftarrow$ `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
- $c \leftarrow$ `http://www.w3.org/2002/07/owl#Class`

Teniendo esto en cuenta, puede entenderse la estructura del bloque *Para Todo*, donde por ejemplo: la estructura $O.triples((None, RDF.type, OWL.Class))$ indica que se buscan tripletas (*triples*) de la ontología O cuyo tipo ($RDF.type$) sea clase ($OWL.Class$). Instrucciones como $RDF.type$ y $OWL.Class$ son propias de la biblioteca utilizada y no determinan los formatos considerados, es decir, el intérprete carga la ontología (*xml, ntriples* o *n3*) a memoria y la recorre con esas instrucciones.

Las variables *Clases*, *Propiedades* y *comentarios* de los algoritmos 8, 9 y 10 respectivamente representan el listado de recursos obtenidos en cada uno de los casos y que son devueltos por cada algoritmo. En el caso particular de la obtencion puede observarse que no se indica el tipo en la búsqueda de tripletas, esto se debe a que nos interesa un recorrido por todas ellas en lugar de un tipo específico y la instrucción $O.comment(a)$ obtiene el comentario correspondiente al dato a.

4.5 *Parser*

Se implementó un analizador con la finalidad de remover información no deseada de los recursos que busca obtener. Este *parser* se incluyó como parte del crawler textual en distintas operaciones dentro del código del mismo, ver Algoritmo 11.

Algoritmo 11 Analizador de recursos (*parser*)

Entrada: *URL*
Salida: *contenido*,*links*

1: Abrir conexión a *URL* web
2: Cargar *contenido* a un objeto BeautifulSoup *html* // `Intérprete HTML`
3: Determinar $formato$ del contenido // `Se hace uso de los metadatos para extraer el ‘‘Content-Type"`
4: **Si** *contenido* es HTML **Entonces** // `Tipos html, asp, php, etc. son tratados como html`
5: *links* ←Extraer hipervinculos del objeto // `Nuevos links a incluirse en la frontera`
6: *contenidoT* ← “”
7: **Para Todo** *tag* ∈ *html.body* **Hacer** //
8: *contenidoT*+ = *tag.content*() // `Se obtiene solo el texto contenido en la etiqueta <body> limpio de etiquetas`
9: **Fin Para**
10: *contenido* ← *contenidoT*
11: **Si No**, **Si** *contenido* es PDF **Entonces**
12: Convertir PDF a texto plano // `Facilita el tratamiento del texto`
13: Normalizar a una codificación // `Se eliminan símbolos no útiles para evitar ruido en el cálculo de similitud`
14: **Fin Si**
15: **Devolver** *contenido*, *links*

Como se observa en el algoritmo, las operaciones corresponden a las que se mencionan en la sección 4.3. Con este parser se logra obtener el texto relevante a la búsqueda libre de etiquetas y/o símbolos que no son de interés. Esto permite realizar el cálculo de similaridad de manera más precisa, ya que la eliminación de dicha información reduce el ruido en el cálculo.

Este módulo de la implementación carga el documento HTML en memoria mediante un intérprete (en este caso **BeautifulSoup**) que permite tratar al documento como un árbol de acuerdo a la estructura de sus etiquetas. El formato del documento se determina haciendo uso de los metadatos del mismo, en este caso del metadato *Content-Type*. En caso de tratarse de un PDF se convierte a texto plano y se eliminan símbolos e imágenes del mismo dejando únicamente el texto. En caso de tratarse de un documento HTML el tratamiento como árbol del mismo permite unir los contenidos de cada etiqueta (bloque *Para Todo* del algoritmo) evitando a las etiquetas

mismas, además, el intérprete permite identificar los *htmlentities*[5], lo cual permite identificar por ejemplo: tratar al segmento `<` como el símbolo <, que es lo que significa en HTML, y de esta manera facilitar su tratamiento.

Una vez que se ha extraído el texto plano del documento tratado, este proceso termina retornando el texto limpio y los links encontrados. Los siguientes puntos describen las variables utilizadas en el proceso:

- *URL*: cadena que contiene el enlace para abrir la conexión y obtener el documento.
- *contenido*: variable de tipo cadena para guardar el contenido en cada una de las etapas de su tratamiento.
- *contenidoT*: variable temporal de tipo cadena para concatenar el contenido de cada una de las etiquetas.
- *tag*: cada una de las etiquetas dentro de la etiqueta $< body >$
- *links*: arreglo que se usa para guardar las URLs encontradas en el documento.
- $formato$: cadena que contiene el formato del documento, obtenido de la información de los metadatos del mismo.

4.6 Anotado semántico y poblado de ontologías

La implementación del módulo de anotado semántico y poblado de ontologías se hizo en Java. Se hizo uso de las bibliotecas y herramientas de GATE para tratamiento de texto y ontologías. Dichas herramientas, como *gazetteers*, permiten la identificación de entidades (personas, organizaciones, etc) dentro de un documento, además, permiten establecer un vínculo entre tales entidades y clases dentro de una ontología.

Este módulo permite el etiquetado de los documentos que han sido obtenidos mediante el *crawler* específico para la extracción de texto, dichas etiquetas son vinculadas a clases de una ontología, que pudo ser obtenida mediante el *crawler* ontológico o creada a la medida de acuerdo al interés del usuario. Además, permite la creación de nuevos individuos dentro de la ontología, así como atributos y relaciones para dichos individuos.

En el Algoritmo 12 se muestra la secuencia de operaciones necesarias para realizar el anotado semántico de documentos. Este algoritmo utiliza las siguientes estructuras de datos:

[5]En HTML permiten desplegar caracteres reservados como el <, letras acentuadas o con diéresis, etc.

Algoritmo 12 Anotado semántico de documentos

Entrada: URL de la ubicación de la ontología *ontoURL*, URL del documento o directorio de documentos *Dir* que se quieren anotar, Archivo de mapeo *mapURL* y archivo de listados *listURL*

1: *ontologia* ← abrirOntologia(*ontoURL*) // `Carga la ontología a memoria para su uso`
2: *corpus* ← crearCorpus()
3: **Para Todo** *docURL* ∈ *Dir* **Hacer** // `Agregar todos los documentos contenidos en Dir`
4: *documento* ← abrirDocumento(*docURL*)
5: Agregar *documento* a *corpus*
6: **Fin Para**
7: *pipeline* ← crearRecursoProcesamiento() // `Permite una ejecución ordenada de los recursos`
8: *gazetteer* ← crearOntoGazetteer(*mapURL*, *listURL*) // `Recurso de procesamiento que genera las anotaciones`
9: Agregar *gazetteer* a *pipeline*
10: Ejecutar *pipeline*
11: **Para Todo** *cdoc* ∈ *corpus* **Hacer** // `Por cada documento en` *corpus* `guardar una copia anotada del mismo`
12: Guardar documento *cdoc* anotado
13: **Fin Para**
14: poblarOntologia(*corpus*) // `Se crean nuevos individuos y relaciones en la ontología`
15: guardar cambios en *ontologia* // `Se guardan los cambios realizados por la operación poblarOntologia`

- *mapURL*: es la URL de un archivo de mapeo que permite vincular las entidades que se identifiquen en el documento con clases de alguna ontología. Este archivo es texto plano y sus líneas tienen el formato: *person_first.lst: /home/jmromero/Ontologies/demo.owl: Person*, donde *person_first.lst* es un archivo de texto plano, que contiene una lista de nombres que serán identificados como entidades, */home/jmromero/Ontologies/demo.owl* es la URL (proporcionada al mecanismo de anotado) de la ontología a la que se vincularán las anotaciones y *Person* es una clase de dicha ontología con la cual son vinculados los nombres encontrados en el listado.

- *listURL*: se refiere a la ubicación de un archivo de metadatos de los listados. En este archivo las líneas son del tipo *country.lst:location:country*, donde *country.lst* identifica al archivo de listado correspondiente, *location* indica el tipo de entidad y *country* se refiere a que la entidad es un país (*country* en inglés). El archivo debe ser así de explícito, ya que también pueden existir entidades de tipo *location* que no sean *country*, por ejemplo: *city*, *province*, etc.

- *corpus*: comprende un conjunto de documentos sobre los que se hace el anotado semántico.
- *documento*: es un documento cargado a memoria para ser agregado al corpus.
- *pipeline*: recurso de procesamiento que permite la ejecución ordenada (en el orden en que son agregados) de los demas recursos.
- *gazetteer*: es el recurso que realiza las anotaciones basado en el archivo de mapeo y el archivo de metadatos de los listados.
- *docURL* **y** *cdoc*: objetos que permiten iterar en el directorio de archivos y el *corpus* respectivamente.

El etiquetado de personas y localizaciones se menciona como ejemplo, es decir, los archivos de configuración pueden editarse para enriquecer el anotado con otros tipos de entidades y mientras más explícita sea su información se obtienen mejores resultados.

Algoritmo 13 Poblado de ontologías

Entrada: *corpus*

1: **Para Todo** $doc \in corpus$ **Hacer** // Recorrer el corpus documento por documento
2: $anotaciones \leftarrow$ extraerAnotaciones(doc) // Se obtiene el conjunto de anotaciones que se generaron
3: **Para Todo** $a \in anotaciones$ **Hacer** // Revisar cada anotación para extraer información útil
4: $clase \leftarrow$ extraerClase(a) // Se extrae el nombre de la clase que vincula a la anotación
5: **Si** $clase \in ontologia$ **Entonces**
6: $ind \leftarrow$ crearIndividuo($clase$) // Se crea un nuevo individuo para la clase encontrada
7: $dato \leftarrow$ extraerInformacion(a) // Se obtiene la subcadena del documento que fue etiquetada
8: $atributo \leftarrow$ crearDatatypeProperty(ind) // Se crea un nuevo tipo de dato
9: Agregar valor $dato$ a $atributo$ // Al tipo de dato se le asigna como valor la subcadena obtenida
10: **Fin Si**
11: **Fin Para**
12: **Fin Para**

La variable de entrada es el *corpus*, como se indica en el Algoritmo 12. El resto de las variables utilizadas en el proceso se describen a continuación:

- *doc*: documento dentro del corpus
- *anotaciones*: conjunto de anotaciones generadas
- *a*: cada anotación dentro del conjunto
- *clase*: nombre de la clase que se vinculó a una determinada subcadena en el documento.
- *ind*: nuevo individuo creado para *clase*
- *dato*: subcadena del documento invulucrada en la anotación *a*
- *atributo*: nueva relacion de tipo de dato creada para el individuo *ind*

La operación `poblarOntologia(`*corpus*`)` (línea 14 del Algoritmo 12) se detalla en el Algoritmo 13. En este proceso se lleva a cabo la creación de individuos y relaciones con tipos de datos a partir de las entidades identificadas en el proceso de anotado. El algoritmo verifica, documento por documento, cada una de las anotaciones generadas previamente, con esto se logra identificar las clases vinculadas a la ontología y crear nuevos individuos para las mismas.

El algoritmo hace un recorrido por todas las anotaciones generadas en el proceso de anotado. Por cada anotación extrae la URL de ontología que vincula dicha anotación, ya que sobre esta ontología deberá agregarse el individuo y/o las propiedades. Una vez identificada la ontología se extrae el nombre de la clase que ha sido vinculada y dentro de la ontología se crea un individuo perteneciente a dicha clase. Para asignar una propiedad nombre al individuo se extrae el fragmento de texto anotado del documento, y siguiendo una notación en inglés se crea una propiedad *hasName*, a la cual se asigna como valor el fragmento de texto anotado. Por último mediante las propiedades *minorType* y *majorType* de la anotación se busca enriquecer el poblado, por ejemplo: en el caso de entidades de tipo *Person*, *minorType* contiene el genero (*male* o *female*) de la persona, lo que permite agregar una propiedad adicional que podriamos llamar *hasGender*.

Los archivos de mapeo y de metadatos de los listados, mencionados como parte del algoritmo 12, son componentes clave en los procesos de anotado semántico y poblado de ontologías. El Listado 4.1 muestra un ejemplo del contenido de mapping.def (archivo de mapeo).

Listado 4.1: Contenido del archivo de mapeo

```
university_lower.lst:/home/jmromero/ontologiaGlobal/Global.owl:University
institute_upper.lst:/home/jmromero/ontologiaGlobal/Global.owl:Institute
author_male.lst:/home/jmromero/ontologiaGlobal/Global.owl:Researcher
country_cap.lst:/home/jmromero/ontologiaGlobal/Global.owl:Country
topicfc.lst:/home/jmromero/ontologiaGlobal/Global.owl:Topic
topicc.lst:/home/jmromero/ontologiaGlobal/Global.owl:Topic
```

En la estructura se puede observar que para un listado, *topicfc.lst* por ejemplo, se realizan anotaciones vinculadas a la ontología */home/jmromero/ontologiaGlobal/-Global.owl* con la clase *Topic* perteneciente a dicha ontología.

Listado 4.2: Contenido del archivo de metadatos

```
topicfc.lst:Topic:is a web crawler that attempts to download only web
    pages that are relevant to a pre-defined topic or set of topics.
topicc.lst:Topic:is a computer program that browses the World Wide Web
    in a methodical, automated manner or in an orderly fashion.
```

El Listado 4.2 contiene las relaciones de cada uno de los listados, *topicfc.lst* por ejemplo, con un conjunto de características específicas de acuerdo al interés en el anotado. En este caso La primera característica del ejemplo citado es *Topic* y es la que aparece como *majorType* en las anotaciones generadas. La segunda característica aparece como *minorType* en las anotaciones y en este caso es una definición de *crawler* focalizado. El listado mencionado (*topicfc.lst*) contiene un listado de cadenas de texto que denotan los fragmentos del documento que pueden ser etiquetados y dotados de las características mencionadas. El Listado 4.3 muestra el contenido de *topicfc.lst*.

Listado 4.3: Contenido del archivo topicfc.lst

```
Focused Crawler
Focused crawler
focused crawler
FOCUSED CRAWLER
Focused Crawling
Focused crawling
focused crawling
FOCUSED CRAWLING
```

De acuerdo esto fragmentos como *Focused Crawler* o *Focused Crawling* dentro de un documento son etiquetados con las características mencionadas y vinculados a la clase *Topic* de la ontología *Global.owl*.

Capítulo 5

Pruebas y resultados obtenidos

Una buena presentación de los resultados de un proyecto de software permite visualizar si han cumplido los objetivos planteados para el mismo. Además permite exponer las dificultades encontradas a lo largo del desarrollo, lo cual resulta de gran utilidad para la evaluación del enfoque utilizado en la solución.

En este capítulo se presentan los resultados de la implementación propuesta para resolver el problema planteado en esta tesis. El capítulo presenta, para cada módulo de la implementación, una descripción de las pruebas realizadas y los resultados obtenidos; por último, al final del capítulo se presenta un análisis general de los resultados de la solución implementada y un boceto de la posible interfaz de acuerdo al orden de las operaciones.

Para obtener los resultados que se presentan en este capítulo hay algunas dependencias que deben ser satisfechas en la ejecución, es decir, algunos de los módulos reciben como entradas los resultados de otros. A continuación se enumera la secuencia de ejecución para obtener los resultados correctos:

1. Primero debe modelarse el contexto base que sirve como dominio de búsqueda de ambos *crawlers*. Para las pruebas realizadas el contexto se construye a partir de un documento de texto, sin embargo, éste puede modelarse manualmente con un conjunto de palabras clave y una frecuencia asociada a cada una de ellas.

2. Los *crawlers* pueden ejecutarse a la vez siempre cuando ya se haya modelado el contexto, el cual es utilizado por ambos; siendo éste la única dependencia de los crawlers.

3. El *parser* está incluido en el *crawler* textual por lo que no es necesaria una ejecución por separado.

4. El módulo de anotado semántico y poblado de ontologías debe ejecutarse una vez que ambos *crawlers* han terminado, ya que este módulo utiliza los documentos y ontologías descargados. También existe la opción de modelar una ontología

específica para el anotado en lugar de utilizar una de las descargadas, y de ser este el caso sólo es necesario esperar a que termine el *crawler textual*.

5.1 *Crawler* textual

La calidad de un *crawler* focalizado está determinada por la calidad de los recursos que es capaz de recuperar. En este caso se evalúa la pertenencia al dominio de los recursos obtenidos por el *crawler*. Las métricas utilizadas para dicha evaluación son: *precision*, la cual evalúa la precisión del *crawler* y se calcula mediante la ecuación (5.1) y *recall*, que se refiere a la capacidad del *crawler* para predecir y seleccionar los documentos relevantes de un determinado conjunto de documentos y se calcula mediante (5.2).

$$Precision = \frac{|\{\text{Documentos relevantes}\} \cap \{\text{Documentos recuperados}\}|}{|\{\text{Documentos recuperados}\}|} \tag{5.1}$$

$$Recall = \frac{|\{\text{Documentos relevantes}\} \cap \{\text{Documentos recuperados}\}|}{|\{\text{Documentos relevantes}\}|} \tag{5.2}$$

Debido a que es necesario conocer cuáles documentos son o no relevantes al dominio de búsqueda, se realizaron las pruebas sobre conjuntos de documentos locales. Las ejecuciones en ambientes controlados simulan el recorrido del crawler en la Web, abriendo y analizando cada documento para determinar su relevancia al contexto. Hacer uso de conjuntos de documentos locales nos permite hacer un juicio propio sobre la relevancia de los mismos, lo cual es útil para determinar si el programa está recuperando los documentos considerados correctos.

Para las pruebas realizadas, el contexto base del *crawler* se modeló a partir de un documento. El *crawler* se probó en tres conjuntos con 125 documentos cada uno. Dichos documentos fueron revisados cuidadosamente para determinar su similaridad con un documento base para cada conjunto. Para cada conjunto el documento base utilizado es diferente. Cada conjunto tiene un número significativo de documentos que pertenecen al área de conocimiento del documento base, pero que no necesariamente tratan los mismos tópicos. Además, se incluyen documentos de áreas distintas, dichos documentos fueron agregados con la finalidad de observar dónde serán ubicados por el crawler con respecto al umbral.

Es importante tener en cuenta que, como se mencionó en la sección 3.1.2, se hace uso de dos umbrales diferentes:

- α**:** con él se determina si se descarga el documento

- β**:** utilizado para determinar si se mejora el contexto

Se realizaron ejecuciones utilizando diferentes valores en los umbrales para determinar la combinación más adecuada para cada conjunto, de acuerdo a las medidas de *precision* y *recall*. Se calculó también la media armónica (F-measure), calculada con (5.3), para cada una de las prubeas. La Tabla 5.1 muestra los valores obtenidos para cada una de las pruebas utilizando un $\beta = 0.70$ y un α variable.

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

Tabla 5.1: Resultados para $\beta = 0.70$

$\beta = 0.70$	Conjunto 1			Conjunto 2			Conjunto 3		
$\alpha =$	**0.55**	**0.60**	**0.65**	**0.55**	**0.60**	**0.65**	**0.55**	**0.60**	**0.65**
Precision	0.6511	0.8333	1.0	0.6585	0.7222	0.913	0.9	0.9166	1.0
Recall	0.9032	0.8064	0.5483	0.9310	0.8965	0.7241	0.7823	0.4782	0.2173
F-measure	0.7567	0.8196	0.7083	0.7714	0.8	0.8076	0.8372	0.6285	0.3571

Se busca un valor alto en la media armónica y a la vez mantener equilibrados los valores de *precision* y *recall*. Teniendo esto en cuenta, en la Tabla 5.1 los mejores resultados para el conjunto 1 $\alpha = 0.6$, para el conjunto 2 $\alpha = 0.65$ y para el conjunto 3 $\alpha = 0.55$.

Tabla 5.2: Resultados para $\beta = 0.75$

$\beta = 0.75$	Conjunto 1			Conjunto 2			Conjunto 3		
$\alpha =$	**0.55**	**0.60**	**0.65**	**0.55**	**0.60**	**0.65**	**0.55**	**0.60**	**0.65**
Precision	0.8055	0.8846	1.0	0.5918	0.6923	0.8666	0.9545	1.0	1.0
Recall	0.9354	0.7419	0.4838	1.0	0.9310	0.8965	0.913	0.7826	0.3478
F-measure	0.8656	0.8070	0.6521	0.7435	0.7941	0.8813	0.9333	0.878	0.5161

En el caso de la Tabla 5.2 los mejores valores se obtienen para el conjunto 1 con $\alpha = 0.55$, para el conjunto 2 $\alpha = 0.65$ y para el conjunto 3 $\alpha = 0.55$.

El valor de α en ambas pruebas denota una compensación entre *precision* y *recall*, es decir, a medida que el valor de α aumenta, lo hace también el de *precision* y disminuye el de *recall*; y sucede lo contrario cuando α disminuye. Por otro lado, un valor alto de β proporciona una mejora en el valor de *precision*, al permitir que sólo los elementos relevantes influyan, mientras que un valor muy bajo puede alejar al contexto de los elementos relevantes, debido a la mezcla que se da con elementos no relevantes en este caso.

Los resultados sugieren que el valor de α capaz de arrojar buenos resultados en los 3 conjuntos es el 0.6, por lo cual se determinó fijar $\alpha = 0.6$ como valor a ser superado para decidir si se descarga el documento. Además, elevar α afecta de manera negativa los resultados en los conjuntos 1 y 3 para ambos valores de β, mientras que en el 2 se ve afectado negativamente cuando α disminuye y $\beta = 0.75$; esto sugiere que el umbral se mantenga cercano a 0.6.

Se observa entonces que, usando el umbral de 0.6, existe poco equilibrio entre *precision* y *recall* cuando $\beta = 0.70$. Por otro lado, utilizando 0.75, la media armónica se incrementa en algunos casos y el equilibrio entre las medidas es bueno. De acuerdo con esto, se determinó como adecuado el valor de 0.75 para β como umbral a ser superado para decidir si se realiza la mezcla de los contextos.

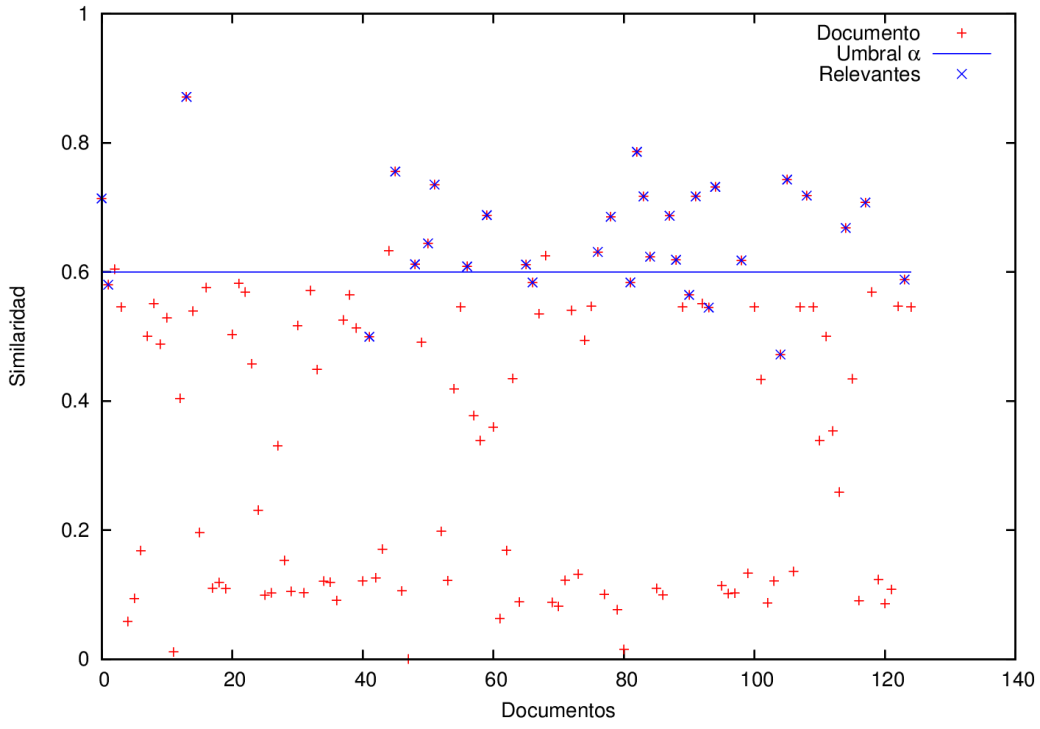
Tabla 5.3: Resultados obtenidos sin el uso de mejora de contexto

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.6$			
Precision	0.8846	0.6585	1.0
Recall	0.7419	0.931	0.6086
F-measure	0.8070	0.7714	0.7567

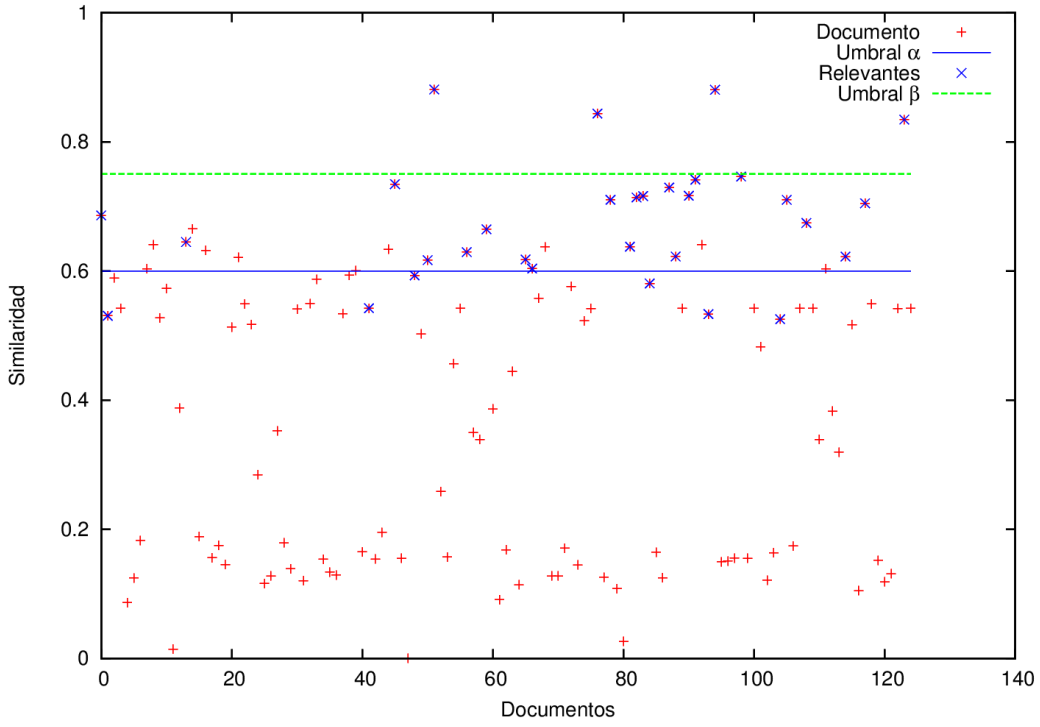
Los resultados que se muestran en la Tabla 5.3 se obtuvieron sin el uso de la mejora en tiempo de ejecución, en ella se utilizó el umbral seleccionado para determinar si es relevante $alpha = 0.6$. Comparando los resultados contra los de la Tabla 5.2, el enfoque propuesto obtiene mejores resultados en los conjuntos 2 y 3 y se mantiene igual en el conjunto 1, esto se debe a que con el incremento de β se reduce la cantidad de documentos que influyen en el contexto, y por ello es menos notorio el cambio; sin embargo, con un β alto se asegura que el contexto solo sea afectado por elementos relevantes.

Comparado con la Tabla 5.1, la propuesta proporciona un mejor equilibrio entre *precision* y *recall* para los conjuntos 1 y 2 pero pierde en el caso del conjunto 3, esto se debe a que, con un valor de β pequeño, el contexto puede verse influido por elementos poco relevantes o no relevantes acercándose a los mismos, lo que propicia que se acepten documentos no relevantes y se descarten documentos relevantes. Con esto se resalta la importancia de manejar un valor alto para β y evitar un impacto negativo sobre el contexto. La mejora es poco notoria debido a que: la propuesta encuentra mejores resultados a medida que el contexto se enriquece con informacion de nuevos elementos, por lo que depende de que en el recorrido encuentre elementos que superen el umbral β en el coeficiente de similaridad.

Las gráficas mostradas en las figuras 5.1, 5.2 y 5.3 muestran los resultados de los 125 elementos dentro de cada uno de los conjuntos (1, 2 y 3 respectivamente). La ejecución con mejora de contexto en este caso consistió de una mezcla previa con todos los documentos que superan a β en lugar de mezclarlo en tiempo de ejecución,

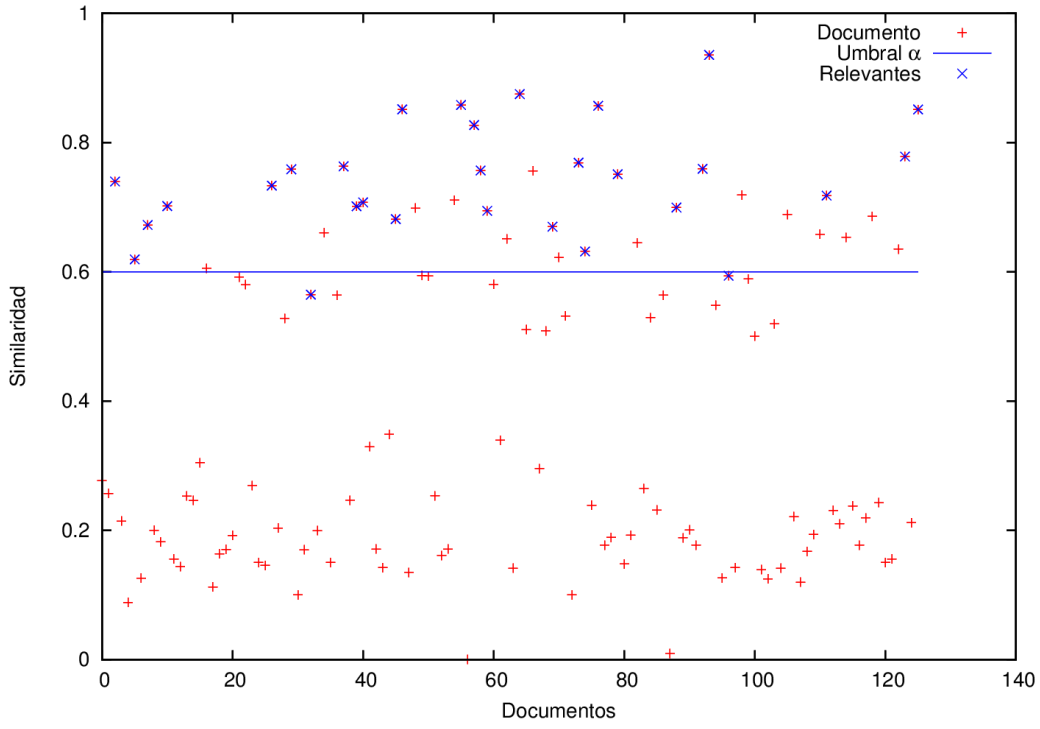


a)

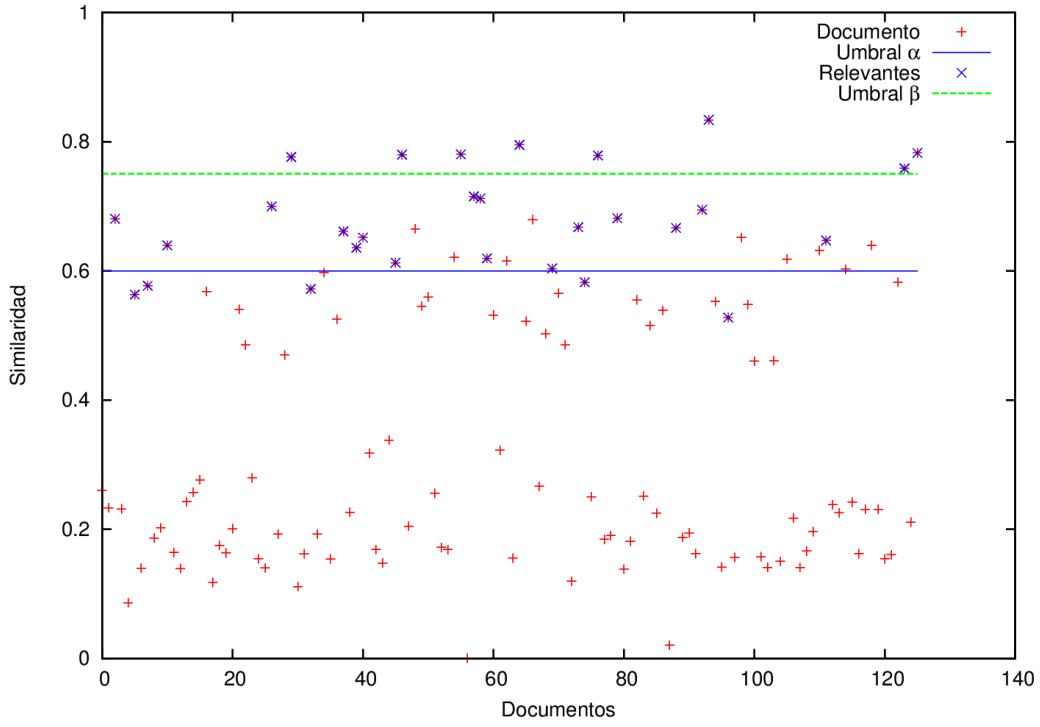


b)

Figura 5.1: Conjunto 1 sin contexto mejorado (a)) y con contexto mejorado (b)).

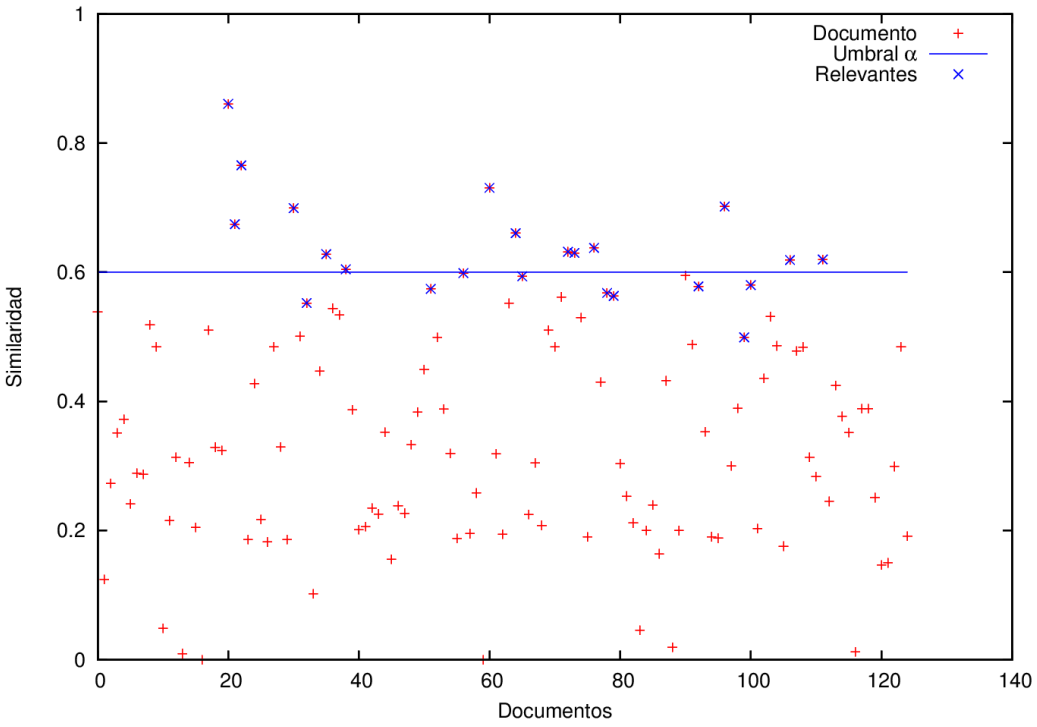


a)

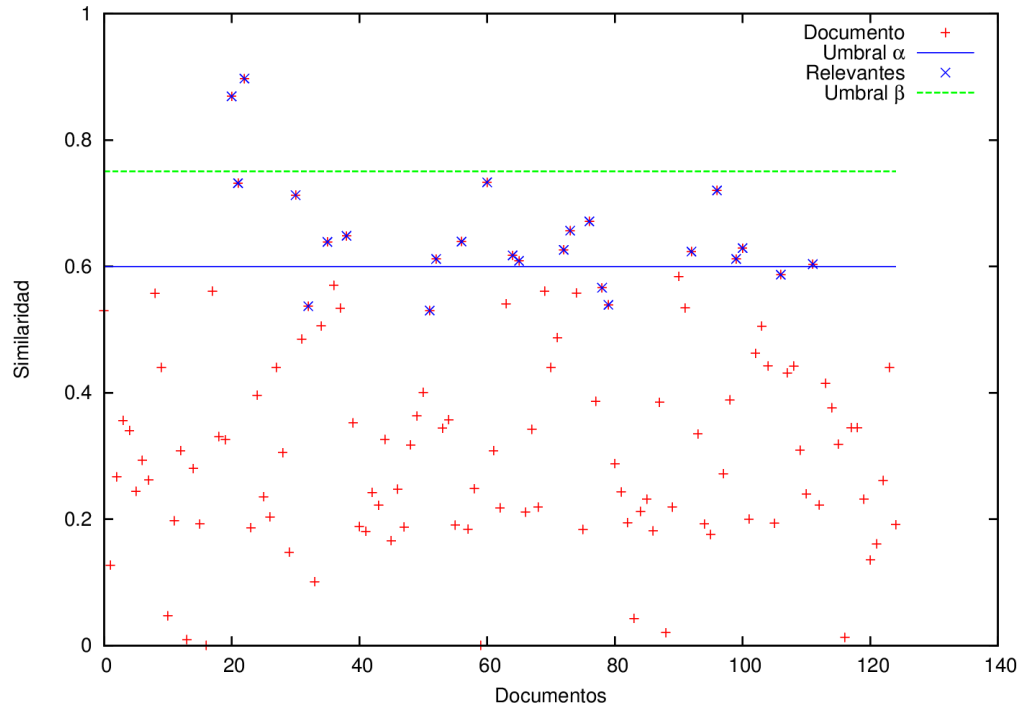


b)

Figura 5.2: Conjunto 2 sin contexto mejorado (a)) y con contexto mejorado (b)).



a)



b)

Figura 5.3: Conjunto 3 sin contexto mejorado (a)) y con contexto mejorado (b)).

por lo que los resultados con mejora de contexto pueden diferir de los mostrados en las tablas, el objetivo es hacer más notorio el movimiento del contexto base después de ser mezclado con otros. En estas gráficas se observa:

- **Conjunto 1 (Figura 5.1):** en *a*) (sin el uso de mejora de contexto) se obtienen 26 documentos como relevantes, mientras que en *b*) (con mejora de contexto) se obtienen 35. En este caso se ha logrado aumentar el número de documentos obtenidos, sin embargo, se han atraído algunos no relevantes dentro de los mismos. Se denota entonces una mejora en *recall*.

- **Conjunto 2 (Figura 5.2):** en *a*) (sin el uso de mejora de contexto) se obtienen 41 documentos como relevantes, mientras que en *b*) (con mejora de contexto) se obtienen 32. En este caso solo se han descartado 5 documentos no relevantes que se recuperan en *a*) y se han perdido 3 relevantes.

- **Conjunto 3 (Figura 5.3):** en *a*) (sin el uso de mejora de contexto) se obtienen 14 documentos como relevantes, mientras que en *b*) (con mejora de contexto) se obtienen. Se mantienen los mismos documentos, mas 5 relevantes que han sido recuperados con la mejora de contexto.

En estas gráficas se ilustra cómo con el uso de la mejora de contexto, se logra determinar como releventes a documentos que, sin el uso de dicha propuesta, quedan fuera. Además, se observa que es capaz de excluir documentos. Habiendo mostrado que la propuesta proporciona una mejora en las pruebas y de acuerdo a lo mostrado en las graficas se puede afirmar que dicho enfoque es capaz de acercarse a los documentos más relevantes y de alejarse de los menos relevantes.

5.2 *Crawler* ontológico

Este módulo se encarga de la recuperación de recursos semánticos, particularmente de la recuperación de ontologías a través de la Web. Este *crawler* funciona de la misma manera que el *textual*.

El *crawler* textual, como se describió en la sección 5.1, es capaz de obtener muy buenos resultados en el proceso de recuperación de información. Los módulos difieren en las siguientes características:

- El *crawler* textual hace uso de un contexto dinámico y el ontológico no.

- La extracción de información del recurso a recuperar difiere debido a que el *crawler* textual lo hace sobre documentos semiestructurados no estructurados (PDF y HTML), mientras que el ontológico, lo hace sobre documentos estructurados (XML).

El intérprete se encarga de diferenciar una ontología de otros archivos XML que no son ontologías. Las ontologías contienen información estructurada que puede ser vista como un árbol (jerarquía de clases por ejemplo). Además, las ontologías contienen relaciones semánticas entre los elementos, lo cual hace menos significativos los métodos sintácticos de comparación. Por otro lado, debido a que ambos *crawlers* deben utilizar el mismo contexto, es necesario realizar la comparación de la misma manera utilizando la métrica de cosenos.

Se preparó un conjunto de 160 ontologías para llevar a cabo las pruebas en un repositorio local y, de esta manera, poder establecer un juicio adecuado de las ontologías que el *crawler* es capaz de recuperar en cada ejecución. Es importante mencionar que, debido a que existen pocas ontologías disponibles en la Web, el conjunto es muy heterogéneo, es decir, los temas modelados por las ontologías del conjunto son muy variados. Esta heterogeneidad genera cierta incertidumbre en la correcta afinación del umbral correspondiente debido a que se recuperan pocos resultados en las ejecuciones.

Las pruebas realizadas consisten en 3 ejecuciones del *crawler* sobre el repositorio local de ontologías, utilizando un contexto base diferente en cada una de ellas. Los títulos de los documentos utilizados para modelar el contexto base fueron: *"A Survey of Ontology Evaluation Techniques"* para la primera ejecución, *"Methodology for the Design and Evaluation of Ontologies"* para la segunda y *"A formal ontology of subcellular neuroanatomy"* para la tercera ejecución.

Los resultados obtenidos en las ejecuciones mostraron coeficientes de similaridad bajos con el contexto base, lo cual es razonable debido a que el contexto base fue modelado a partir de un documento de texto plano, mientras que el contexto de cada ontología se modeló extrayendo elementos específicos de la misma como nombres de clases, relaciones, etc. Esta disminución observada en la similaridad sugiere el uso de un umbral menor al utilizado en el *crawler* textual, por lo que se hicieron los cálculos para umbrales con valores 0.35, 0.4 y 0.45. Es importante mencionar que para el crawler ontológico no se hace uso de mejora del contexto, por lo que sólo se utiliza el umbral α para determinar si se descarga la ontología.

Tabla 5.4: Resultados obtenidos del *crawler* ontológico

	Ejecución 1			Ejecución 2			Ejecución 3		
$\alpha =$	**0.35**	**0.40**	**0.45**	**0.35**	**0.40**	**0.45**	**0.35**	**0.40**	**0.45**
Precision	0.6	0.6666	0.6666	1.0	1.0	1.0	1.0	—-	—-
Recall	1.0	0.6666	0.6666	0.75	0.5	0.25	0.75	—-	—-
F-measure	0.75	0.6666	0.6666	0.8571	0.6666	0.4	0.8571	—-	—-

Para evaluar el funcionamiento del *crawler* ontológico se hizo uso de las medidas de *precision* (5.1), *recall*(5.2) y la media armónica *F-measure* (5.3). En la Tabla 5.4 se muestran los resultados de las pruebas realizadas para cada una de las ejecuciones

y los distintos valores de α.

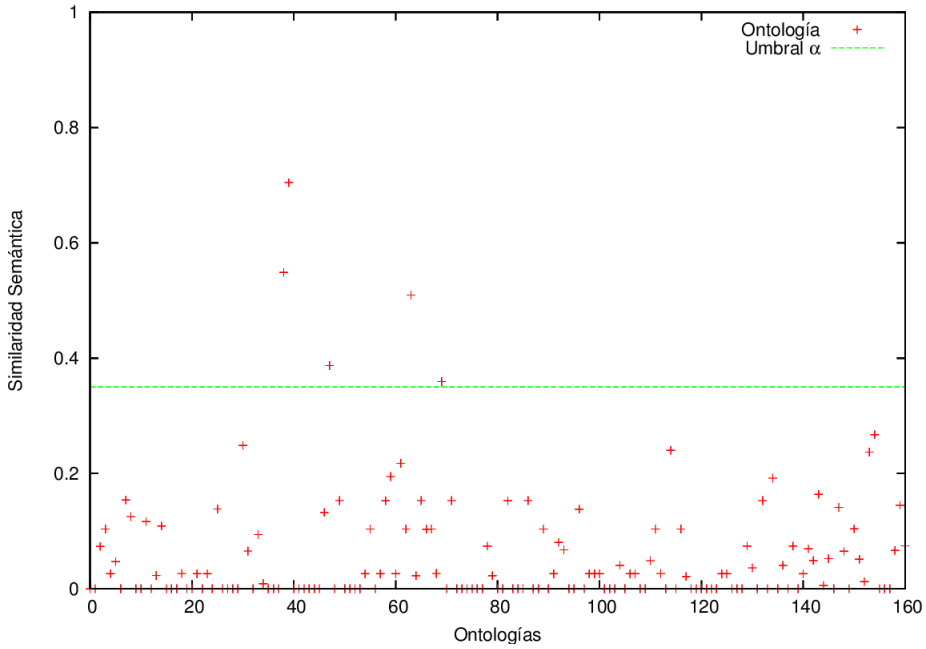


Figura 5.4: Resultados del *crawler* ontológico (ejecución 1)

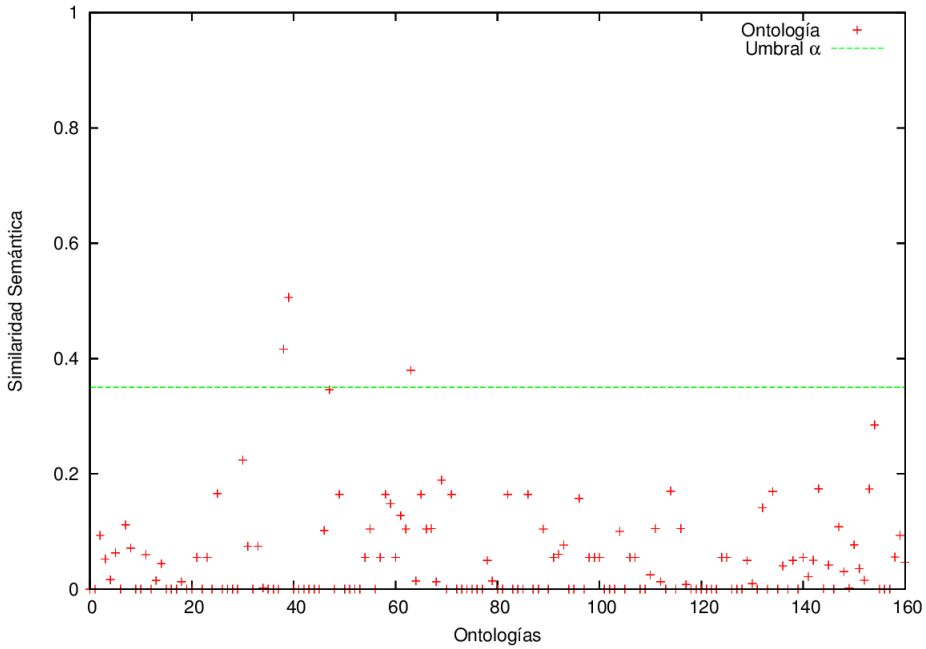


Figura 5.5: Resultados del *crawler* ontológico (ejecución 2)

Puede observarse en la tabla que los mejores valores se obtienen utilizando un $\alpha = 0.35$, por lo que se determinó fijar este valor como umbral para decidir si se

descarga el recurso o se descarta. Como se ilustra en la Tabla 5.4, los valores de las medidas son bastante buenos, por lo que se puede concluir que el *crawler* es capaz de recuperar buenos resultados utilizando el umbral mencionado.

La Figura 5.4 muestra las 160 ontologías (eje x) mencionadas, denotando el valor de similaridad que obtuvieron (eje y). Como se observa, el *crawler* fue capaz de recuperar 5 ontologías del total. De acuerdo con la revisión realizada a cada una de las ontologías, sólo 3 de las 5 recuperadas son relevantes. Estos resultados corresponden a los mostrados en la Tabla 5.4 (ejecución 1 $\alpha = 0.35$).

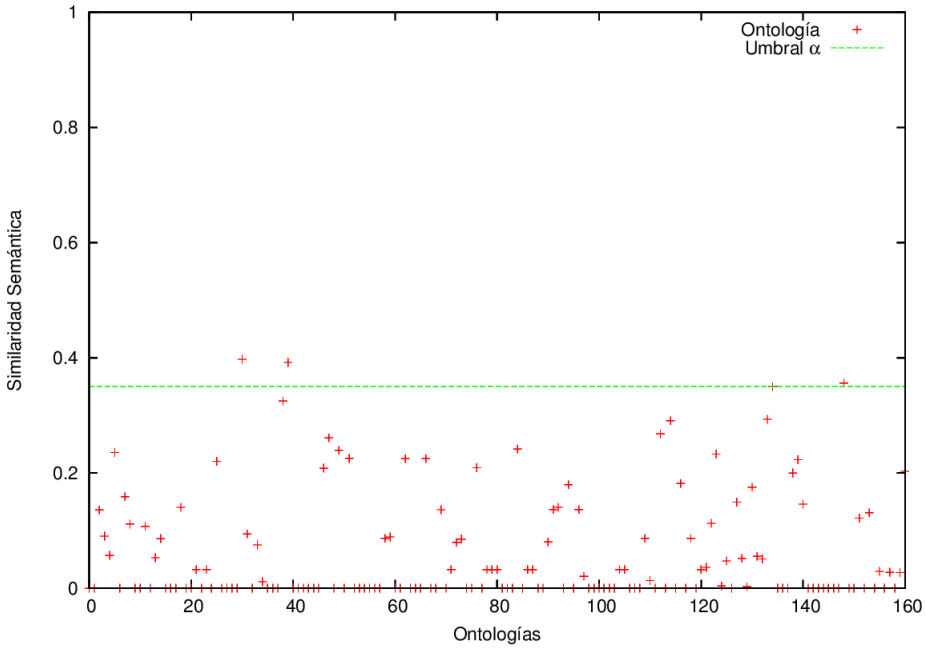


Figura 5.6: Resultados del *crawler* ontológico (ejecución 3)

En el caso de la Figura 5.5, cuyos resultados corresponden a la ejecución 2, el *crawler* solo fue capaz de recuperar 3 ontologías. En este caso las 3 ontologías recuperadas son relevantes al contexto utilizado en la prueba, sin embargo, la revisión realizada de las ontologías determina que existen 4 ontologías relevantes a este dominio. La ontología relevante que el *crawler* no fue capaz de recuperar tiene un valor de similaridad de 0.34, lo cual se refleja en el valor obtenido para la medida *recall* en la Tabla 5.4 (ejecución 2 $\alpha = 0.35$).

Para la tercera ejecución se obtuvo la gráfica mostrada en la Figura 5.6, la cual muestra un caso igual al de la ejecución 2 en un sentido cuantitativo, es decir, el *crawler* recuperó 3 ontologías relevantes y fallo en recuperar una relevante con valor de similaridad 0.349 (bastante cercano al umbral). Por otro lado, no se recuperó ningún falso positivo, sin embargo, se observa mayor cercanía de los elementos no relevantes

en relación con los resultados de las ejecuciones 1 y 2.

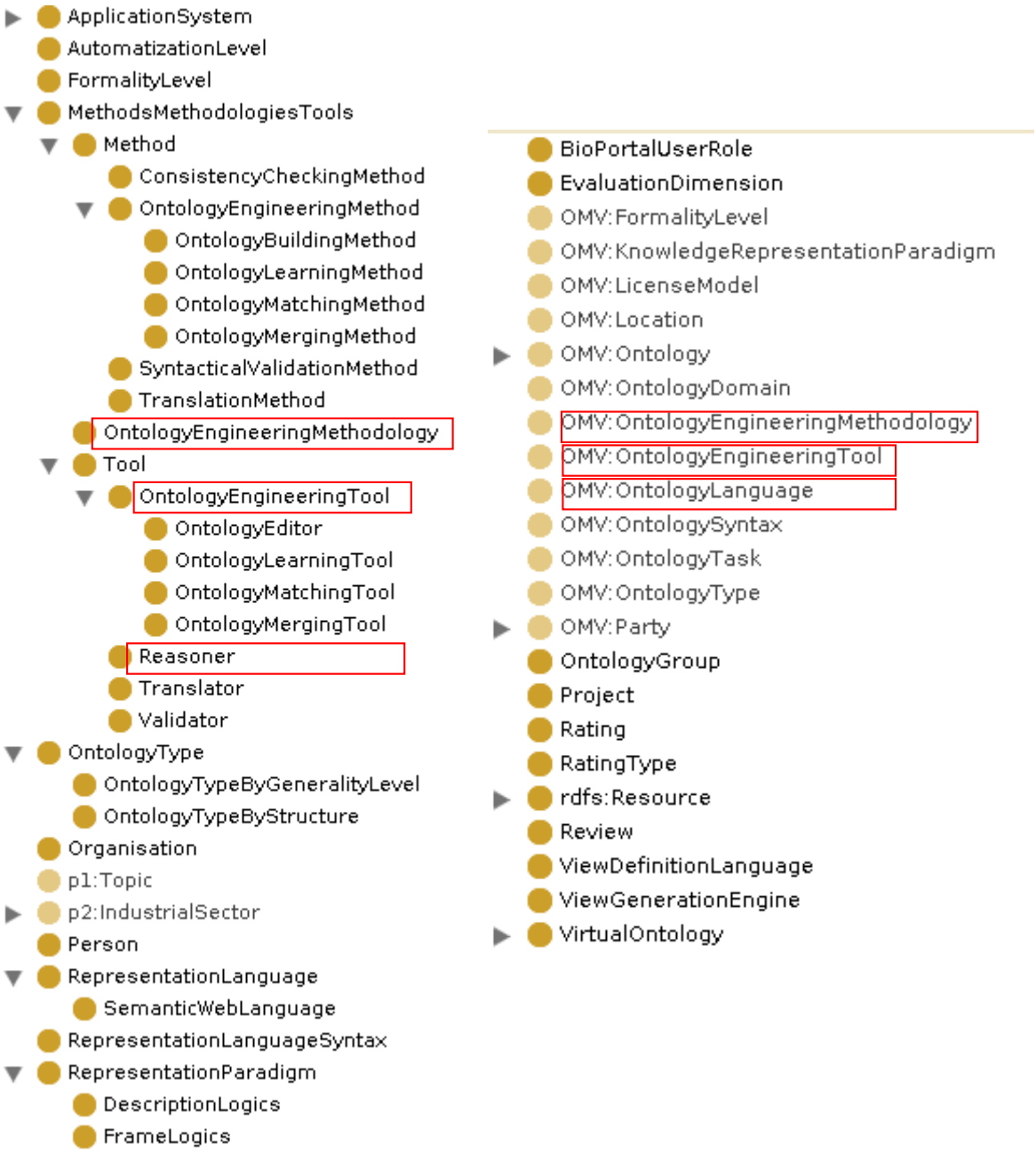


Figura 5.7: Ontologías relevantes

Como caso de análisis particular se retomaron los resultados de la ejecución 1, donde se tienen dos elementos recuperados que no son relevantes al contexto base. La revisión detallada de estos casos muestra que la ontología recuperada con un valor de similaridad de 0.359 es ajena al contexto base, sin embargo, la que se recuperó con un valor de 0.5 presenta una cercanía semántica que se describe a continuación.

La Figura 5.7 muestra dos capturas de pantalla de Protégé de las estructuras de clases de 2 de las ontologías recuperadas como relevantes, particularmente de las dos con los coeficientes de similaridad más altos. Como se puede apreciar en la figura las clases de ambas ontologías hacen alusión a temas relacionados con ontologías, por ejemplo: su estructura, evaluación de las mismas, ingeniería de ontologías entre otros, estos temas pueden observarse en los nombres de algunas clases (*OntologyEngineeringTool, OntologyEngineeringMethodology, Reasoner, OntologyLanguage*, etc.) que se resaltan en la figura. Claramente están muy relacionadas con el contexto base construido a partir del documento con título *"A Survey of Ontology Evaluation Techniques"*.

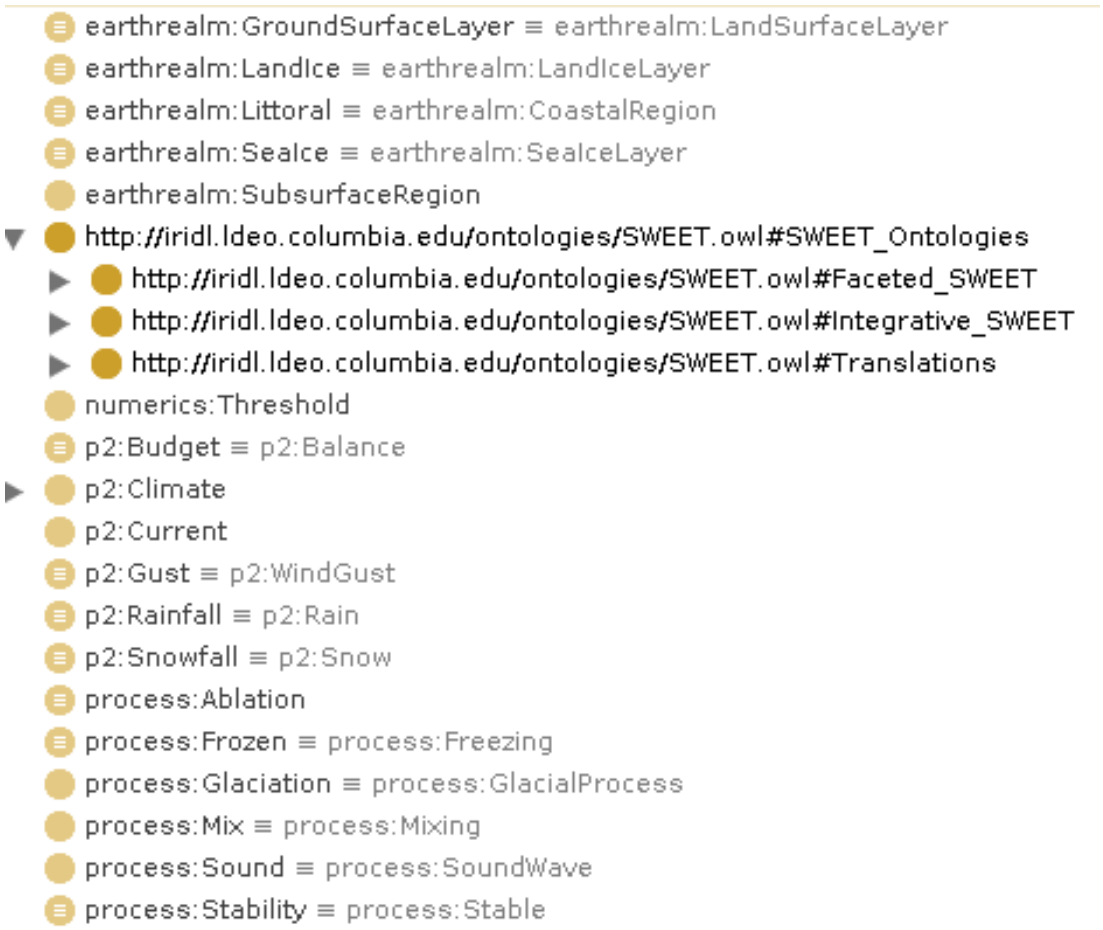


Figura 5.8: Ontología no relevante

La Figura 5.8 muestra la estructura de clases de la ontología no relevante con el coeficiente de similaridad más alto (0.5), puede notarse en los nombres de las clases los diversos temas tratados. Sin embargo, el alto coeficiente de similaridad obtenido se debe a que se trata de una ontologia que describe la estructura organizacional

(reflejando su documentación) de las ontologías SWEET, es decir, dado que contiene información de la estructura de otras ontologías es justificable que el *crawler* la determine relevante por la naturaleza del contexto base (evaluación de ontologías).

Las pruebas y observaciones expuestas sobre este módulo de la implementación determinan el buen funcionamiento del mismo. Se establece entonces que este *crawler* cumple con el objetivo de la recuperación de información semántica (ontologías) de la Web.

5.3 *Parser*

Este módulo se encarga de extraer el texto plano de los documentos para su tratamiento. El objetivo es extraer sólo el texto de interés al dominio de búsqueda establecido por el contexto base. De esta manera, el *parser* se encarga de:

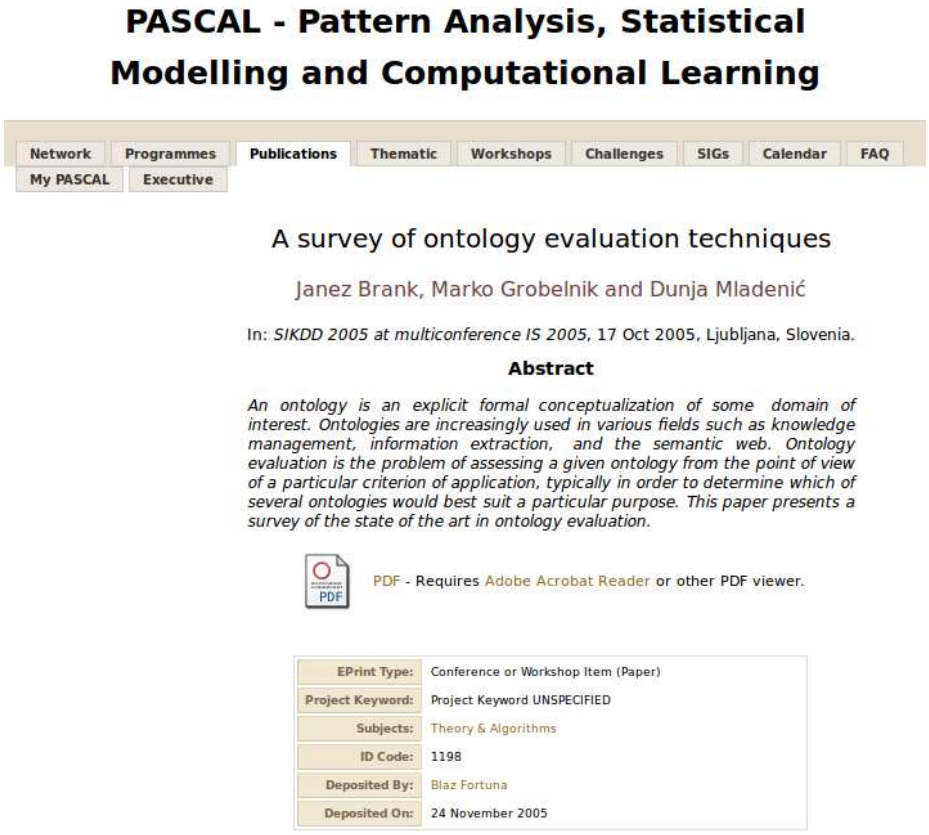


Figura 5.9: Apariencia de resumen de artículo

- Remover etiquetas y código en el caso de archivos con formato HTML
- Extraer el texto plano independiente de la codificación en archivos PDF

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html  xmlns:ep="http://eprints.org/ep2/template" xmlns="http://www.w3.org/1999/xhtml">
  <head >
    <title >PASCAL - </title>
    <meta  http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link  rel="stylesheet" type="text/css" href="http://www.pascal-network.org/pascal.css" />
    <link  rel="stylesheet" type="text/css" href="http://eprints.pascal-network.org/eprints.css"
    <link  rel="Top" href="http://eprints.pascal-network.org/" />
    <link  rel="Search" href="http://eprints.pascal-network.org/perl/search" />
    <link  rel="Help" href="http://eprints.pascal-network.org/help" />
    <link  rel="schema.DC" href="http://purl.org/DC/elements/1.0/" /><meta  content="A survey of
domain of interest. Ontologies are increasingly used in various
fields such as knowledge management, information extraction,
and the semantic web. Ontology evaluation is the problem of
assessing a given ontology from the point of view of a
particular criterion of application, typically in order to
determine which of several ontologies would best suit a
particular purpose. This paper presents a survey of the state
of the art in ontology evaluation." name="DC.description" /><meta  name="DC.publisher" /><meta  c
  </head>
  <body >

    <div  id="title">
     <center ><table >
     <tr ><td  id="tdt"><a  href="http://www.pascal-network.org/"><img  src="http://www.pascal-ne
     </table></center></div>
     <ul  id="tabnav">
      <li ><a  href="http://www.pascal-network.org/Network/">Network</a></li>
      <li ><a  href="http://www.pascal-network.org/Programmes/">Programmes</a></li>
      <li ><a  class="active" href="http://eprints.pascal-network.org/">Publications</a></li>
      <li ><a  href="http://www.pascal-network.org/Thematic/">Thematic</a></li>
      <li ><a  href="http://www.pascal-network.org/Workshops/">Workshops</a></li>
      <li ><a  href="http://www.pascal-network.org/Challenges/">Challenges</a></li>
      <li ><a  href="http://www.pascal-network.org/sigwiki/">SIGs</a></li>
      <li ><a  href="http://www.pascal-network.org/Calendar/">Calendar</a></li>
      <li ><a  href="http://www.pascal-network.org/FAQ/">FAQ</a></li>
      <li ><a  href="http://www.pascal-network.org/My_PASCAL/">My PASCAL</a></li>
      <li ><a  href="http://www.pascal-network.org/Executive/">Executive</a></li>
     </ul>
<div  id="cont">
      <div  align="center"><table  width="800"><tr ><td  align="left">
        <h1  class="pagetitle"></h1>
        <p ><span  class="citation" xmlns:ep="http://eprints.org/ep2/citations" xmlns="http://www
<em ><span  class="field_title">A survey of ontology evaluation techniques</span></em><br  />
<span  class="field_creators">Janez Brank, Marko Grobelnik and Dunja Mladeni&#x107;</span><br  />
In: <em ><span  class="field_event_title">SIKDD 2005 at multiconference IS 2005</span></em>, <spa

</span></p><h2  class="abstitle">Abstract</h2><p  class="abstext">An ontology is an explicit form
domain of interest. Ontologies are increasingly used in various&#13;
fields such as knowledge management, information extraction,&#13;
and the semantic web. Ontology evaluation is the problem of&#13;
assessing a given ontology from the point of view of a&#13;
```

Figura 5.10: Código fuente de la página

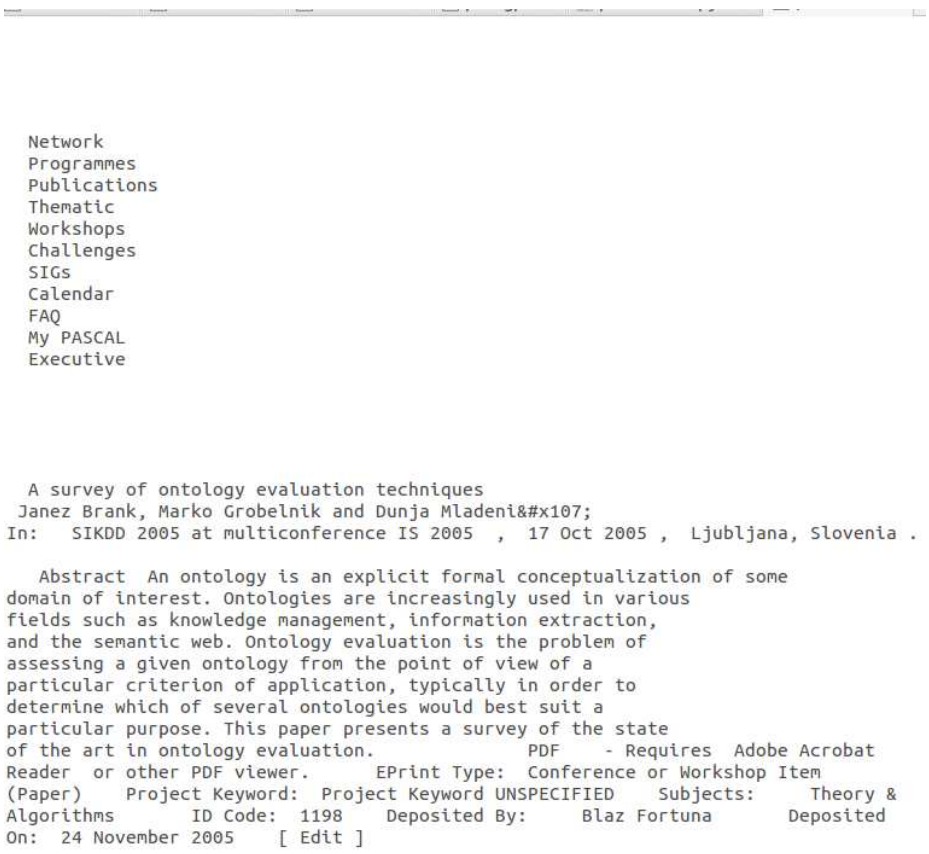
- Remover símbolos que puedan interferir en la extracción adecuada de los *tokens*

En este caso las pruebas consistieron en procesar varias entradas de texto diferentes, cada una de las cuales contenía varios de los elementos mencionados para su remoción del texto. Como ejemplo del tratamiento de un documento HTML se tiene un resumen del artículo *"A survey of ontology evaluation techniques"* mostrado en el sitio *http://eprints.pascal-network.org/*, cuya apariencia se muestra en la Figura 5.9.

El documento fuente que se obtiene al generar la conexión a esta página es el código HTML que genera dicha visualización. La captura mostrada en la Figura 5.10 muestra una fracción de esta estructura de código donde se encuentra el resumen.

El trabajo del *parser* consiste en eliminar todas las etiquetas, ya que éstas no son de utilidad para la búsqueda. Además, el parser se encarga de hacer la interpretación

adecuada de los *HTML-entities*, los cuales son códigos estándar para la generación de algunos símbolos, por ejemplo: el espacio en blanco se genera mediante &*nbsp*;, el símbolo < se genera con &*lt*;, etc.



```
  Network
  Programmes
  Publications
  Thematic
  Workshops
  Challenges
  SIGs
  Calendar
  FAQ
  My PASCAL
  Executive



  A survey of ontology evaluation techniques
 Janez Brank, Marko Grobelnik and Dunja Mladeni&#x107;
In:   SIKDD 2005 at multiconference IS 2005  ,  17 Oct 2005 ,  Ljubljana, Slovenia .

   Abstract  An ontology is an explicit formal conceptualization of some
domain of interest. Ontologies are increasingly used in various
fields such as knowledge management, information extraction,
and the semantic web. Ontology evaluation is the problem of
assessing a given ontology from the point of view of a
particular criterion of application, typically in order to
determine which of several ontologies would best suit a
particular purpose. This paper presents a survey of the state
of the art in ontology evaluation.              PDF    - Requires  Adobe Acrobat
Reader  or other PDF viewer.      EPrint Type:  Conference or Workshop Item
(Paper)    Project Keyword:  Project Keyword UNSPECIFIED    Subjects:     Theory &
Algorithms       ID Code:  1198    Deposited By:     Blaz Fortuna       Deposited
On:  24 November 2005    [ Edit ]
```

Figura 5.11: Resultado obtenido por el *parser*

Los resultados que obtiene el *parser*, al tratar la página que se muestra en las figuras 5.9 (apariencia) y 5.10 (código HTML), se muestran en la Figura 5.11. Como puede observarse en la Figura 5.11, el archivo fuente ha sido limpiado de etiquetas y símbolos no utiles al interés de la búsqueda, manteniendo unicamente el texto contenido en el archivo HTML.

Estos resultados exponen el buen funcionamiento del *parser* implementado, permitiendo que sólo la información de interés al contexto de búsqueda se conserve. Con esto, el *parser* ayuda a que el repositorio de documentos que se quiere construir contenga sólo la información significativa para el contexto. Además, al conservar sólo la información de interés al contexto mejora su aprovechamiento posterior al proporcionar al usuario sólo esta información.

5.4 Anotado semántico y poblado de ontologías

Con este módulo se logra anotar de forma semántica los documentos obtenidos por el *crawler*. Mediante el uso de ontologías se vinculan las entidades identificadas en los documentos con clases dentro de la ontología. Además, se lleva a cabo la creación de individuos dentro de las clases vinculadas para cada una de las entidades identificadas.

El anotado semántico y poblado de ontologías se lleva a cabo de manera automática, por lo que las pruebas realizadas sobre este módulo consistieron en: la ejecución de dicho módulo sobre un conjunto de documentos, buscando identificar entidades de tipo organización y persona; después se hizo una revisión manual de los documentos para verificar que las anotaciones creadas fueron correctas y que se generaron los individuos correspondientes dentro de la ontología, evitando la generación de más de un individuo con el mismo valor en su propiedad "nombre".

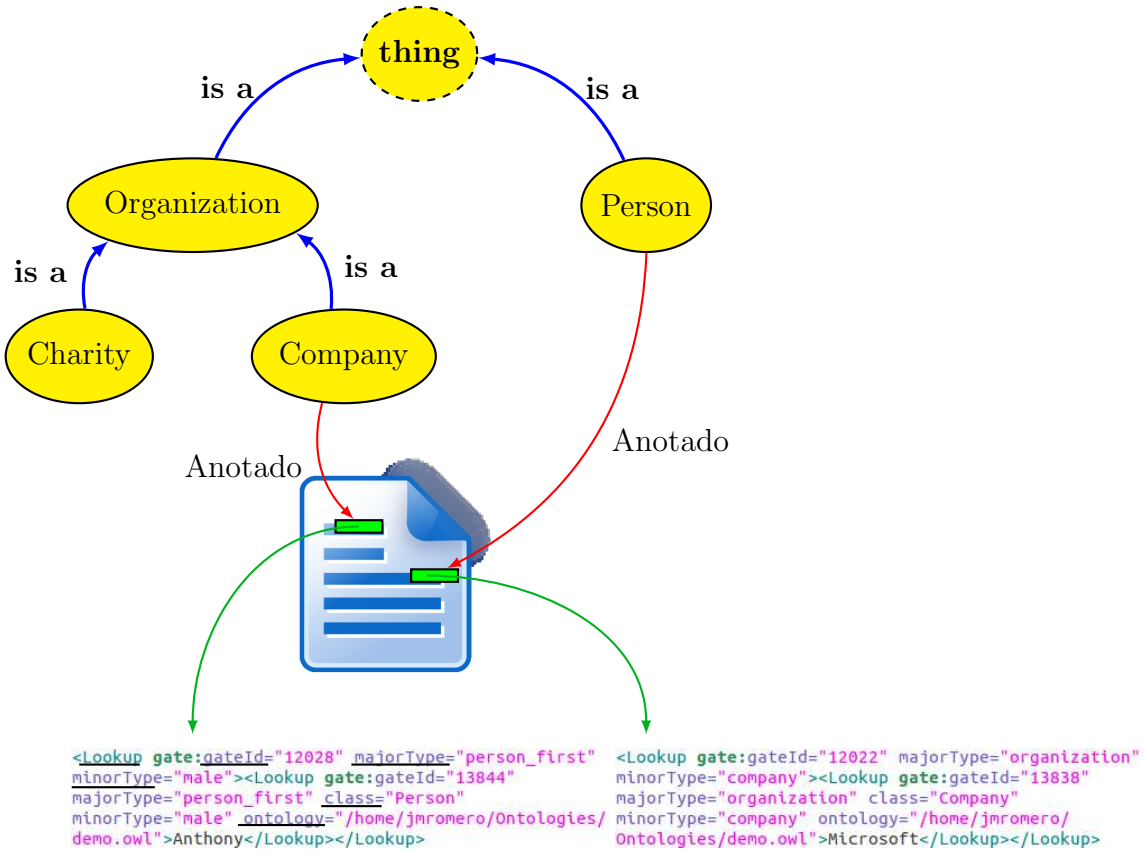


Figura 5.12: Ontología de ejemplo

La Figura 5.12 representa, de manera simbólica, los vínculos que se establecen entre las clases de una ontología determinada y los fragmentos de texto de un documento sobre el que se realiza el anotado semántico. En esta imagen la ontología

está representada por las elipses y sus interconexiones (relaciones jerárquicas), los vínculos se muestran mediante las líneas etiquetadas como "Anotado" y los textos señalados por estos bloques anotados mediante flechas son ejemplos de los resultados obtenidos por el mecanismo de anotado.

La estructura del archivo generado mediante el proceso de anotado tiene etiquetas (subrayadas en un fragmento de texto de la figura) con las siguientes propiedades:

- **Lookup:** es un nombre genérico para denominar a la etiqueta de anotado y puede ser cambiado por otro. En este caso *Lookup* es el nombre por omisión que establece GATE.
- **gateId:** identificador único asignado por GATE a la etiqueta
- **majorType:** tipo de entidad identificado como principal.
- **minorType:** tipo menor o subtipo, esto puede verse como una clasificación mas específica, por ejemplo: el anotado sobre "Microsoft" como *company* cuyo *majorType* es *organization*.
- **class:** en el se establece el nombre de la clase con la que se creo el vínculo
- **ontology:** URL de la ontología con la que se creo el vínculo

Las características *majorType* y *minorType* se especifican en el archivo de configuración denominado como *listURL* en la sección 4.6. En este archivo las líneas del tipo *country.lst:location:country* establecen que *majorType* será *location* y *minorType* será *country*.

De acuerdo con uno de los objetivos planteados en este proyecto de tesis, se propuso la construcción de una ontología que modelara el dominio de conocimiento establecido por el contexto. La idea era utilizar dicha ontología para realizar el anotado semántico de los documentos. También se mencionó que una de las ontologías descargadas relevante al contexto podía servir para realizar el anotado semántico de los documentos.

Para las pruebas de este módulo se decidió construir la ontología mencionada, sin embargo, se creó una ontología más general con el objetivo de establecer relaciones entre los documentos y los autores; en ella se incluyeron clasificaciones de localizaciones y organizaciones para identificar las ciudades o países que son referidos en el documento y las universidades, instituciones de investigación, instituciones gubernamentales o empresas que se mencionan. Además, con la idea de establecer el dominio conocimiento del documento, se incluyó la identificación de tópicos pertenecientes a distintas áreas para poder relacionar a cada documento con uno o mas de dichos tópicos.

Figura 5.13: Estructura de la ontología

La Figura 5.13 muestra la estructura de clases que se generó para la ontología mencionada. La estructura de clases fue lo único que se construyó manualmente como un modelo a seguir de lo que se quería identificar, de manera que el módulo de anotado semántico estableciera un vínculo con una de las clases de la ontología mediante etiquetas. Las etiquetas agregadas al documento de manera automática constituyen una forma de dotar de características adicionales a distintos fragmentos del texto, con lo cual se logra enriquecer semánticamente a dicho documento.

El pseudocódigo mostrado en el Listado 5.1 muestra las primeras líneas de uno de los archivos que han sido anotados semánticamente. En él se muestran algunos identificadores (*gateId*) asignados por GATE y metadatos que describen al documento.

Listado 5.1: Cabeceras de documento anotado

```
<html xmlns:gate="http://www.gate.ac.uk" gate:gateId="0"
    gate:annotMaxId="33386">
<head gate:gateId="1">
<meta gate:gateId="2" content="22" name="xmpTPg:NPages">
```

En el bloque de pseudocódigo mostrado en el Listado 5.2 se muestra la estructura de etiquetado de los fragmentos del texto. El caso particular que se ilustra es el etiquetado de los fragmentos *"Soumen Chakrabarti"* y *"Chakrabarti"*, los cuales han sido identificados para pertenecer a la clase *Researcher* de la ontología. Es importante señalar que la identificación de tales fragmentos depende de los archivos de configuración *mapURL* y *listURL* mencionados al principio de la sección 4.6.

Listado 5.2: Etiquetas de anotado semántico

```
<Lookup gate:gateId="28775" majorType="Person" minorType="male">
<Lookup gate:gateId="32477" majorType="Person" class="Researcher"
    minorType="male"
    ontology="/home/jmromero/ontologiaGlobal/Global.owl">Soumen
<Lookup gate:gateId="28776" majorType="Person" minorType="male">
<Lookup gate:gateId="32478" majorType="Person" class="Researcher"
    minorType="male"
    ontology="/home/jmromero/ontologiaGlobal/Global.owl"> Chakrabarti
</Lookup></Lookup></Lookup></Lookup>
```

El vínculo que se establece mediante los atributos *class* y *ontology* permite enlazar el fragmento de texto a una clase en la ontología señalada y de esta manera se enriquece semánticamente la información contenida en el documento. Tales atributos permiten el poblado automático de ontologías, ya que estas etiquetas pueden ser leídas por el programa y éste puede identificar por ejemplo: en el Listado 5.2 que el fragmento de texto *Soumen Chakrabarti* es de la clase *Researcher* en la ontología señalada. Los resultados mostrados exponen la calidad del módulo de anotado semántico mediante la correcta identificación de entidades.

Listado 5.3: Poblado ontológico (generación de individuo *Researcher*)

```
<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Researcher_1">
    <rdf:type rdf:resource =
        "http://www.owl-ontologies.com/Global.owl#Researcher"/>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Researcher_1">
  <ResearcherhasName
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Soumen
      Chakrabarti</ResearcherhasName>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Researcher_1">
  <ResearcherhasGender
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">male
    </ResearcherhasGender>
</rdf:Description>
```

Por otro lado, el poblado de la ontología se lleva a cabo tomando las anotaciones generadas como base, es decir, cada fragmento de texto vinculado a una clase dentro de la ontología se traduce en la creación de un individuo dentro de la misma. Por cada clase se crea al menos una propiedad de tipo de dato, compuesta por la concatenación del nombre de la clase (*Researcher* en este caso) y un sufijo que indica que tiene un nombre (*hasName*). Para cada individuo creado se agrega como valor

de la propiedad (*ResearcherhasName* en este caso) el fragmento de texto etiquetado (*Soumen Chakrabarti* por ejemplo).

En algunos casos se establecen características adicionales en los archivos de configuración, esto se hace agregando ":" (separador) seguido de la nueva propiedad que se quiere agregar; en este caso se designó el género para aparecer como *minorType* y se puede notar que se indentifica como masculino (*male*) a "Soumen Chakrabarti". Los nombres para los individuos generados se componen de la concatenación del nombre de la clase, un guión bajo y un índice numérico que se incrementa con cada individuo nuevo creado, en el caso de ejemplo tratado un nombre de individuo puede ser *Researcher_2*.

El Listado 5.3 muestra un fragmento de la ontología generada en RDF, donde se especifica que el individuo *Researcher_1* pertenece a la clase *Researcher*, y que tiene dos propiedades *ResearcherhasName* y *ResearcherhasGender* de tipo *String* y con valores *Soumen Chakrabarti* y *male* respectivamente.

Listado 5.4: Generación de individuo para *Document*

```
<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Document_0">
  <rdf:type
      rdf:resource="http://www.owl-ontologies.com/Global.owl#Document"/>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Document_0">
  <DocumenthasTitle
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Focused
      Crawling: A New Approach to Topic-Specific Resource
      Discovery</DocumenthasTitle>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Document_0">
  <hasAuthor rdf:resource=
      "http://www.owl-ontologies.com/Global.owl#Researcher_1"/>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Document_0">
  <hasTopic
      rdf:resource="http://www.owl-ontologies.com/Global.owl#Topic_2"/>
  <hasTopic
      rdf:resource="http://www.owl-ontologies.com/Global.owl#Topic_1"/>
  <hasTopic
      rdf:resource="http://www.owl-ontologies.com/Global.owl#Topic_0"/>
</rdf:Description>
```

Como se mencionó anteriormente la ontología generada debe contener individuos para representar a los documentos (mediante su título), además, debe establecer las relaciones hacia el autor o autores correspondientes (clase *Research*) y hacia los tópicos que hayan sido identificados. El bloque de pseudocódigo mostrado en el Listado 5.4 muestra la generación de un individuo de la clase *Document* y sus propiedades y relaciones.

En el Listado 5.4 se observan las relaciones del documento (*Document_0*) con *Researcher_1* mediante la propiedad *hasAuthor* y con los individuos de *Topic* mediante la propiedad *hasTopic*. Además, en lugar de la propiedad con sufijo *hasName* se colocó una con el sufijo *hasTitle* con el objetivo de hacerlo mas propio de un documento.

Listado 5.5: Estructura de individuo *Topic_1*

```
<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Topic_1">
  <rdf:type
      rdf:resource="http://www.owl-ontologies.com/Global.owl#Topic"/>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Topic_1">
  <TopichasName
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">focused
      crawling</TopichasName>
</rdf:Description>

<rdf:Description
    rdf:about="http://www.owl-ontologies.com/Global.owl#Topic_1">
  <hasDefinition
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string">is a web
      crawler that attempts to download only web pages that are
      relevant to a pre-defined topic or set of topics.</hasDefinition>
</rdf:Description>
```

En el Listado 5.5 se muestra la estructura generada para los individuos pertenecientes a la clase *Topic*, tomando como ejemplo uno de los individuos relacionados con el documento expuesto en el Listado 5.4.

Como complemento a los listados de pseudocódigo explicados en esta sección se incluyen las figuras 5.14 y 5.15, las cuales muestran de manera gráfica (con la ayuda de Protégé) el documento tomando como ejemplo y sus relaciones con el autor y los tópicos.

De acuerdo con los resultados obtenidos por este módulo se logró el anotado semántico de los documentos, a los cuales se añade la extensión *.html* para facilitar su visualización sin etiquetas utilizando un navegador, sin embargo, tales documentos

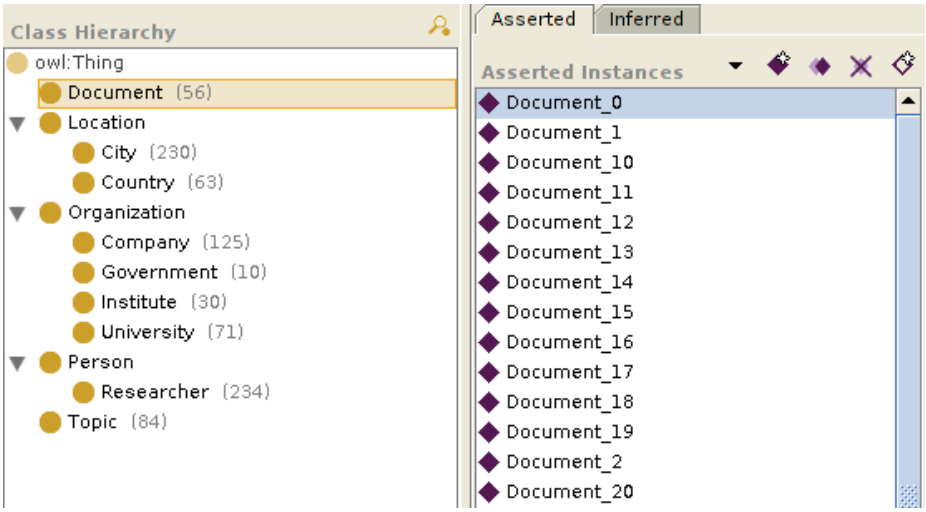


Figura 5.14: Generación de individuo *Document_0*

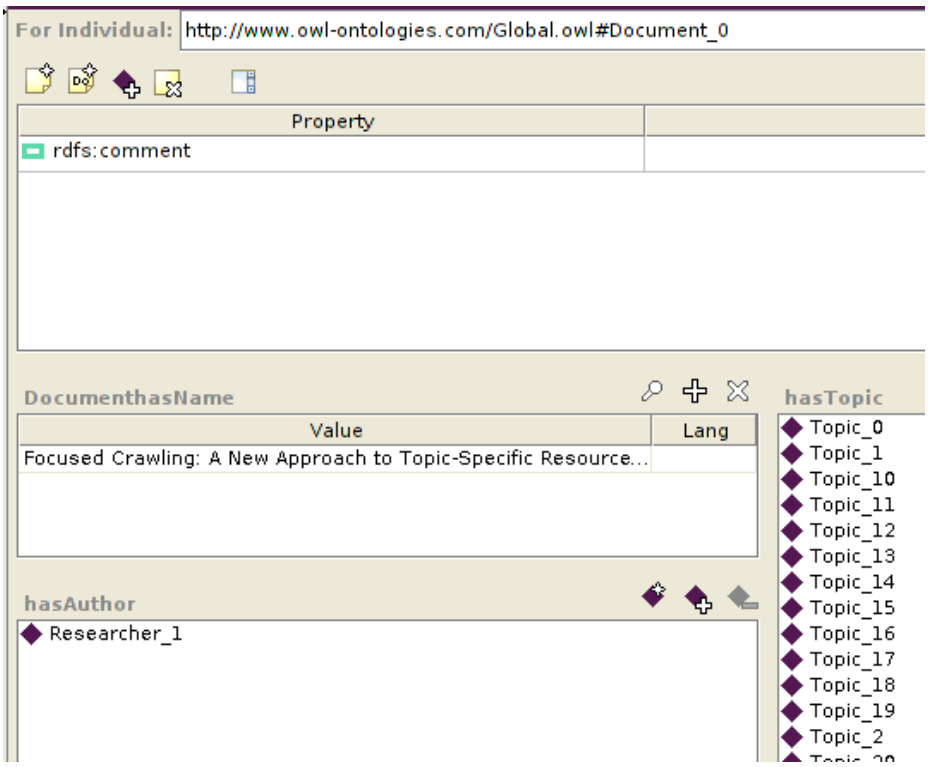


Figura 5.15: Propiedades y relaciones de individuo *Document_0*

se encuentran enriquecidos con la información ontológica. Por otro lado, se logró también la generacion de individuos y propiedades de tipo de dato en la ontología.

Con los resultados obtenidos por este módulo se logra la creación de un repositorio de documentos semánticamente enriquecidos. Esta adición de información semántica

y los vínculos con las ontologías dotan de un mayor significado a los documentos mejorando su aprovechamiento posterior.

5.5 Análisis de resultados

Los resultados presentados en este capítulo demuestran el cumplimiento de los objetivos de este proyecto. Para cada módulo se detallaron las características de las pruebas realizadas y se describió la calidad de los resultados obtenidos. De esto se concluye que la propuesta presentada y desarrollada es un mecanismo eficiente en la recuperación de información, anotado semántico y poblado de ontologías.

El diseño modular permitió la generación de componentes reutilizables, los cuales pueden utilizarse de manera separada, por ejemplo: si sólo se quiere recuperar un conjunto de documentos relevantes a un dominio de búsqueda, bastará con utilizar el *crawler* textual. Por otro lado, utilizados en conjunto de acuerdo a la propuesta presentada, se obtiene como resultado un conjunto de documentos enriquecidos semánticamente y una ontología vinculada a los mismos; con esto se mejora el aprovechamiento posterior de los documentos agilizando el acceso a los mismos.

La realización de pruebas y obtención de los resultados permitieron identificar algunas dificultades y formas de mejora para los distintos módulos de la implementación. Algunas de las principales dificultades encontradas son: la evaluación del *crawler* ontológico, el uso de las herramientas de GATE de forma adecuada, el tratamiento de las ontologías con python, entre otras.

Respecto al uso de hilos para la ejecución de los *crawlers*, no se incluyó una subsección de pruebas debido a que en las ejecuciones iniciales su uso implicó un consumo excesivo del ancho de banda disponible y abuso de peticiones a un servidor. Una vez observado este comportamiento y para evitar abusar de los recursos, se decidió dejar la propuesta de uso de hilos como una opción explorada dentro de este proyecto de tesis. Sin embargo, en estas ejecuciones se observó una mejora significativa en el funcionamiento de los *crawlers* en lo referente a los tiempos de recuperación de la información.

De acuerdo al análisis que se hizo de los resultados se concluye que la solución implementada es la adecuada a la problemática presentada en este proyecto de tesis. El diseño modular que se implemento permite que sus componentes puedan reutilizarse y facilita la adición de nuevo código o modificaciones al mismo sin afectar el funcionamiento general de la solución. También se pudo observar que aun puede mejorarse la solución en varios aspectos.

No se desarrolló una interfaz gráfica para la implementación, sin embargo, se elaboró un boceto de la misma que ilustra las etapas, mediante pestañas, en que deben ejecutarse cada uno de los módulos. Las figuras 5.16, 5.17 y 5.18 muestran las tres

pantallas del boceto elaborado.

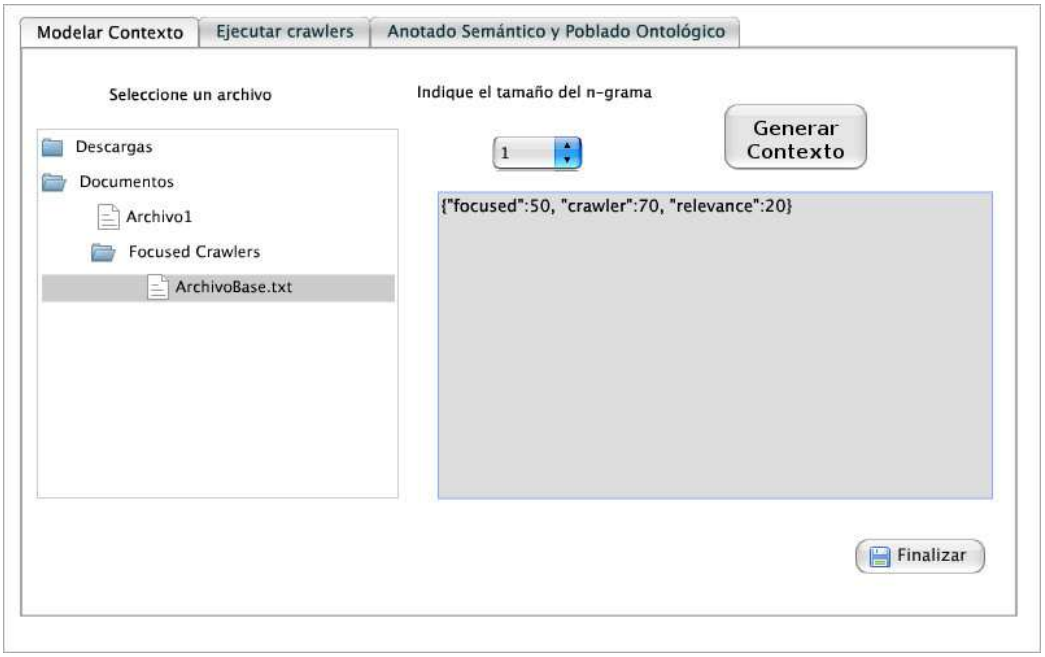


Figura 5.16: Interfaz para modelado de contexto

La interfaz gráfica de la propuesta se presenta en la Figura 5.16, en donde se muestra la etapa de modelado de contexto, la cual debe realizarse en primer lugar. Como se muestra el contexto puede elaborarse a partir de un archivo de texto y se muestra una opción para modificar el tamaño del *n-grama*. Además, se muestra un area de texto donde puede modelarse manualmente el contexto, siempre que se siga el formato adecuado.

La Figura 5.17 muestra la etapa de ejecución de los *crawlers*, los cuales utilizan el contexto previamente modelado. Como parámetros se pueden modificar la cantidad de hilos para exploración y descarga, además, se indica la profundidad a la que llegará el *crawler*. Tales parametros pueden modificarse de manera independiente para cada *crawler*.

La Figura 5.18 muestra la etapa final correspondiente al módulo de anotado semántico y poblado de ontologías. En este caso sólo es necesario cargar los archivos de mapeo, de metadatos de listados y la ontología. Este módulo hace uso de los documentos descargados por el *crawler* textual.

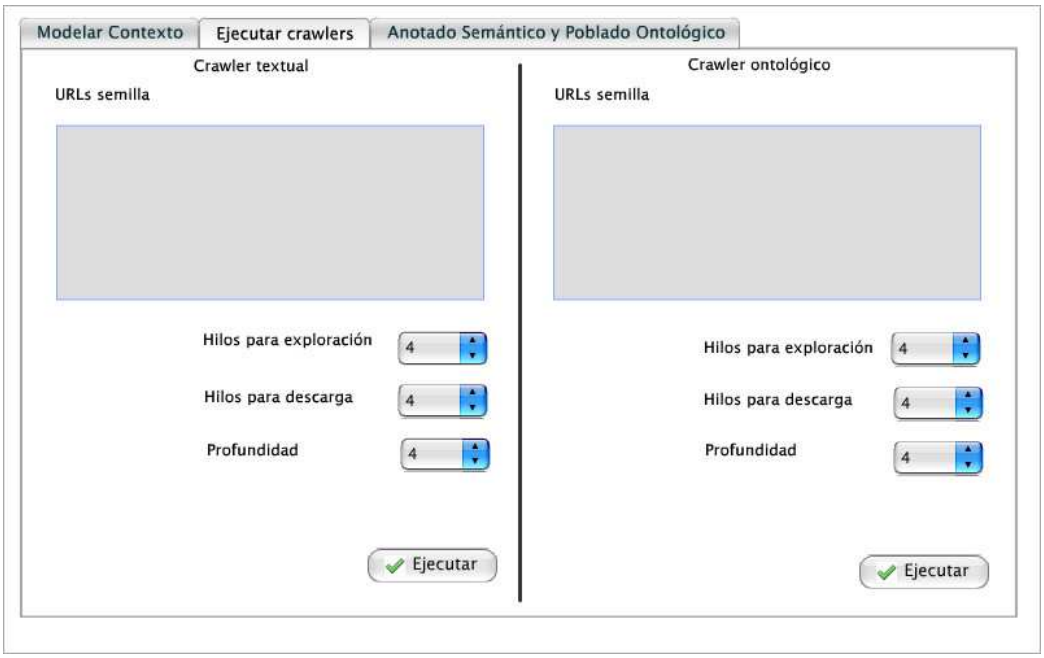


Figura 5.17: Interfaz para ejecución de *crawlers*

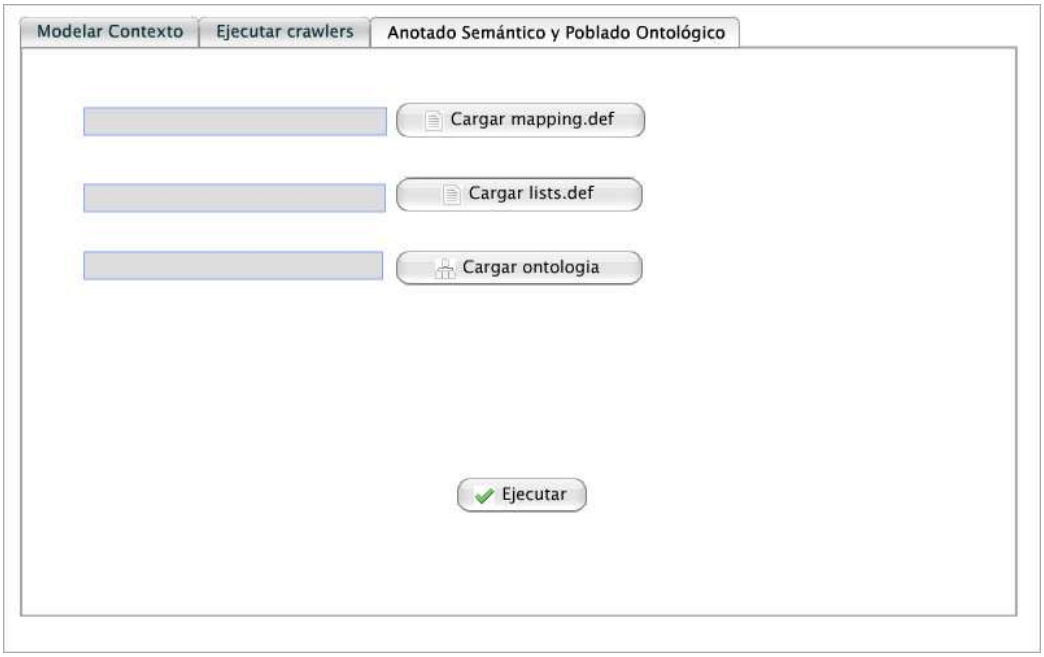


Figura 5.18: Interfaz para anotado semántico y poblado de ontologías

Capítulo 6

Conclusiones y trabajo futuro

Los capítulos anteriores constituyen la descripcion del problema plateado por la tesis y el desarrollo de la solución propuesta. Habiendo concluido con el desarrollo y la obtención de resultados favorales, en este capítulo se exponen las conclusiones a las que se llegaron con el desarrollo del proyecto planteado por esta tesis. Además, se describe el trabajo futuro necesario para la mejora y/o expansión de este proyecto.

6.1 Conclusiones

En el presente trabajo de tesis se propuso el desarrollo de un mecanismo que permitiera la recuperación de información (documentos y ontologías) y el anotado semántico de la información recuperada utilizando ontologías. La combinación de técnicas de modelado de contexto, recuperación de información y anotado semántico permitieron la generación de un conjunto de módulos de software que permiten cumplir los objetivos de este trabajo.

El diseño modular que se hizo para dar solución al problema permitió la generación de componentes reutilizables que pueden ser modificados sin afectar al funcionamiento general de la propuesta. Este diseño es el mostrado en la figura 3.1 del capítulo 3. El modelado del contexto separado del funcionamiento de los mecanismos de recuperación de información (*crawlers*) permite al usuario generar un contexto de manera independiente y que sirve para ambos mecanismos.

El desarrollo separado de los *crawlers* permite realizar exploraciones acorde a cada necesidad (recuperación de texto y ontologías respectivamente), al tratar de manera separada a cada recurso debido a la estructura de los mismos. En el caso del *parser* para la limpieza de los documentos de texto, permite un cálculo de similitud con el mínimo de ruido y sin interferir en el resto de procesos.

Particularmente en el *crawler* textual se propuso un enfoque nuevo que mejora de manera significativa la obtención de documentos relevantes conforme se avanza en

la exploración. Este enfoque se basa en el uso de un "contexto dinámico", el cual es modificado en tiempo de ejecución para mejorarlo y que modele de mejor manera el dominio de búsqueda del usuario. Como se observa en el la sección 5.1, los resultados obtenidos por este modulo denotan una mejora con respecto al método tradicional utilizando el espacio vectorial para la definición de las características (bolsas de palabras). De esta manera, el enfoque propuesto es capaz de refinar la búsqueda a medida que avanza en la exploración de la web.

Las pruebas realizadas para el *crawler* ontológico se realizaron sobre un conjunto de ontologías muy heterogéneo, lo cual generó cierta incertidumbre en el proceso de afinación del umbral adecuado. Sin embargo, se lograron obtener buenos resultados con el umbral elegido, lo cual demuestra el buen funcionamiento de este módulo. Puede concluirse que el *crawler* ontológico es capaz de recuperar "buenas ontologías", es decir, puede recuperar ontologías relevantes al dominio de búsqueda que se establezca.

El módulo de anotado semántico y poblado de ontologías trabaja de manera automática y permite la identificación de las entidades de un documento y la generación de individuos, propiedades y relaciones dentro de la ontología para enriquecerla. El módulo se implementó para trabajar de manera automática pensando en que pocos usuarios tienen los conocimientos necesarios para tratar con las ontologías, de esta manera, el usuario sólo tiene que preocuparse por definir los listados de entidades que le interesa que sean identificados y establecer la clasificación utilizando los nombres de las clases de la ontología que se quiere utilizar, mientras que la generación de individuos, propiedades y relaciones dentro de la ontología, así como la adición de etiquetas (metadatos semánticos) al documento es realizada de forma automática por dicho módulo.

De manera general, los resultados presentados en el capítulo 5 permiten resaltar el buen funcionamiento del desarrollo realizado. En el caso del *crawler* textual, como ya se mencionó antes en esta sección, se obtuvo una mejora significativa en referencia a los metodos sintácticos convencionales mediante el uso de una propuesta propia de este proyecto y que consiste en adaptar el contexto base enriqueciendolo con nuevas características (palabras de los documentos recuperados muy relevantes). Del *crawler* ontológico, se obtuvieron buenos resultados en la recuperación de las ontologías, sin embargo, no se está considerando la estructura de la ontología porque se está utilizando el espacio vectorial para realizar la comparación, esto debido a que así se encuentra modelado el contexto base y debe ser el mismo para ambos *crawlers* según lo especificado en el proyecto. Se muestran también con imágenes y explicación a detalle los resultados del módulo de anotado semántico y poblado de ontologias descrito en el párrafo anterior.

Los resultados obtenidos por el *parser*, como se describieron en el capítulo 5, cumplen con el objetivo del mismo, es decir, este módulo es capaz de obtener de forma

correcta el texto de interés de un determinado artículo.

Respecto al uso de descarga concurrente de documentos se observo que permite un uso eficiente de la memoria al evitar que ésta se sobrecargue de documentos en cola para ser descargados. Por otro lado, la apertura concurrente de conexiones debe manejarse con cuidado, ya que se observó que se puede incurrir en un exceso de demanda de conexiones a un mismo sitio web, lo cual puede ser visto como un ataque DoS (*Denial of Service*).

6.2 Trabajo futuro

El trabajo presentado en este proyecto aun puede ser mejorado y ampliado para obtener resultados de mayor calidad. Estas mejoras quedan como trabajo futuro del proyecto.

Dentro de los aspectos importantes que se dejan como trabajo futuro se encuentra el uso de una métrica de similaridad híbrida, es decir, una mezcla entre sintáctica (actualmente utilizada) y semántica, por ejemplo: una forma de implementar esta mezcla es dotar a las palabras del vector con un conjunto de sinónimos. Se sugiere como trabajo futuro un metodo híbrido en lugar de uno completamente semántico porque este último aumenta mucho los tiempos de recuperación de documentos, por lo tanto es preferible buscar una buena opción entre ambos. También puede buscarse una manera diferente de modelar el contexto del dominio de búsqueda.

El manejo dinámico del contexto propuesto en este proyecto puede ser mejorado al encontrar una cantidad adecuada de mezclas de contextos donde se obtenga la bolsa de palabras óptima para una determinada búsqueda, con lo cual se reduciría el número de mezclas durante la ejecución y por lo tanto se ahorraría tiempo de ejecución.

Respecto al *parser*, es posible hacer uno más exhaustivo dotándolo de información semántica para hacer una mejor limpieza de la información de interes. Sin embargo, en este punto debe considerarse que esta mejora aumenta los tiempos de recuperación de documentos, al tratarse de una operación que se realiza sobre cada documento analizado. Por otro lado un *parser* más exhaustivo elimina más ruido que pudiera afectar a la métrica de similaridad, lo cual se traduciría en una mejora significativa en la *precision* y *recall* del *crawler*.

En el caso del *crawler* ontológico es necesario mejorar el cálculo de similaridad para que sea más acorde a la estructura de arbol de la ontología. Además, considerar la taxonomía en la comparación puede aumentar significativamente la calidad de los resultados. También es necesario abordar la dificultad encontrada en la realización de las pruebas debido a las pocas ontologías que se encontraron para probarlo.

El mecanismo de anotado semántico y poblado de ontologías puede mejorarse adicionándolo a un mecanismo de identificación de relaciones entre entidades para enriquecer la generación de individuos dentro de la ontología.

El uso de hilos para la recuperación de información de explorarse más a detalle, con el objetivo de encontrar la configuración adecuada que permita la ejecución concurrente y evite el abuso de los recursos disponibles.

Otro punto importante a tratar en el trabajo futuro es la construcción de un mecanismo que explote el repositorio de documentos semánticamente anotados que este trabajo es capaz de generar. Un ejemplo de esto es un editor inteligente, capaz de sugerir bibliografía relacionada a quien redacta un documento de acuerdo al contexto de su redacción.

Bibliografía

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In Gellersen, H.-W., editor, *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307, Karlsruhe, Germany. Springer.

[Aizawa, 2003] Aizawa, A. N. (2003). An information-theoretic perspective of tf-idf measures. *Information Processing & Management*, 39(1):45–65.

[Berners-Lee et al., 1992] Berners-Lee, T., Cailliau, R., and Groff, J.-F. (1992). The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5):454–459.

[Betin-Can and Baykal, 2007] Betin-Can, A. and Baykal, N. (2007). Medicoport: A medical search engine for all. *Computer Methods and Programs in Biomedicine*, 86(1):73–86.

[Borst, 1997] Borst, W. N. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Universiteit Twente, Enschede.

[Cheng et al., 2008] Cheng, Q., Beizhan, W., and Pianpian, W. (2008). Efficient focused crawling strategy using combination of link structure and content similarity. In *IT in Medicine and Education, 2008. ITME 2008. IEEE International Symposium on*, pages 1045 –1048.

[Daconta et al., 2003] Daconta, M. C., Obrst, L., and Smith, K. T. (2003). *The semantic web - a guide to the future of XML, web services, and knowledge management*. Wiley.

[Dahiwale et al., 2010] Dahiwale, P., Mokhade, A., and Raghuwanshi, M. M. (2010). Intelligent web crawler. In Mishra, B. K., editor, *Proceedings of the ICWET 10 International Conference and Workshop on Emerging Trends in Technology*, ICWET 10, pages 613–617, Mumbai, Maharashtra, India. ACM.

[Dong et al., 2008] Dong, H., Hussain, F. K., and Chang, E. (2008). A transport service ontology-based focused crawler. In *Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid*, SKG '08, pages 49–56, Los Alamitos, CA, USA. IEEE Computer Society.

[Dong et al., 2009] Dong, H., Hussain, F. K., and Chang, E. (2009). State of the art in semantic focused crawlers. In *Proceedings of the International Conference on Computational Science and Its Applications: Part II*, ICCSA '09, pages 910–924, Seoul, Korea. Springer-Verlag.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–54.

[Fowler, 2003] Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition.

[Gacitua et al., 2008] Gacitua, R., Sawyer, P., and Rayson, P. (2008). A flexible framework to experiment with ontology learning techniques. *Knowledge-Based Systems*, 21(3):192–199.

[Giannopoulos et al., 2010] Giannopoulos, G., Bikakis, N., Dalamagas, T., and Sellis, T. (2010). Gontogle: A tool for semantic annotation and search. In Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., and Tudorache, T., editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference*, volume 6089 of *Lecture Notes in Computer Science*, pages 376–380, Heraklion, Crete, Greece. Springer.

[Gómez-Pérez, 1999] Gómez-Pérez, A. (1999). Ontological engineering: A state of the art. *Expert Update: Knowledge Based Systems and Applied Artificial Intelligence*, 2(3):33–43.

[Gómez-Pérez and Corcho, 2002] Gómez-Pérez, A. and Corcho, O. (2002). Ontology specification languages for the semantic web. *IEEE Intelligent Systems*, 17(1):54–60.

[Gruber et al., 1993] Gruber, T. et al. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220.

[Gulla et al., 2007] Gulla, J., Borch, H., and Ingvaldsen, J. (2007). Ontology learning for search applications. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007*, volume 4803 of *Lecture Notes in Computer Science*, pages 1050–1062, Vilamoura, Portugal. Springer.

[Hacene et al., 2008] Hacene, M. R., Napoli, A., Valtchev, P., Toussaint, Y., and Bendaoud, R. (2008). Ontology learning from text using relational concept analysis. In Peter Kropf, Morad Benyoucef, H. M., editor, *Proceedings of the 2008 International MCETECH Conference on e-Technologies*, MCETECH '08, pages 154–163, Montreal, Canada. IEEE Computer Society.

[Hoareau and Satoh, 2009] Hoareau, C. and Satoh, I. (2009). Modeling and processing information for context-aware computing: A survey. *New Generation Computing*, 27(3):177–196.

[Huang et al., 2009] Huang, W., Zhang, L., Zhang, J., and Zhu, M. (2009). Focused crawling for retrieving e-commerce information based on learnable ontology and link prediction. In V.E. Muhin, Z. Y., editor, *Proceedings of the 2009 International Symposium on Information Engineering and Electronic Commerce*, IEEC '09, pages 574–579, Los Alamitos, CA. IEEE Computer Society.

[Jung, 2009] Jung, J. (2009). Using evolution strategy for cooperative focused crawling on semantic web. *Neural Computing & Applications*, 18(3):213–221.

[Kosala and Blockeel, 2000] Kosala, R. and Blockeel, H. (2000). Web mining research: a survey. *SIGKDD Explor. Newsl.*, 2(1):1–15.

[Kumar and Vig, 2009] Kumar, M. and Vig, R. (2009). Design of core: context ontology rule enhanced focused web crawler. In *Proceedings of the International Conference on Advances in Computing, Communication and Control*, ICAC3 '09, pages 494–497, Mumbai, Maharashtra, India. ACM.

[Lee et al., 2005] Lee, M. D., Pincombe, B. M., and Welsh, M. B. (2005). An empirical evaluation of models of text document similarity. In Bara, B. G., Barsalou, L., and Bucciarelli, M., editors, *Proceedings of the XXVII Annual Conference of the Cognitive Science Society*, pages 1254–1259.

[Liu, 2006] Liu, B. (2006). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer, New York, USA, first edition.

[Liu et al., 2004] Liu, H., Milios, E. E., and Janssen, J. (2004). Focused crawling by learning hmm from user's topic-specific browsing. In *Web Intelligence*, pages 732–732.

[Liu et al., 2011] Liu, Z., Du, Y., and Zhao, Y. (2011). Focused crawler based on domain ontology and fca. *Journal of Information & Computational Science*, 8(10):1909–1917.

[Luong et al., 2009] Luong, H. P., Gauch, S., and Wang, Q. (2009). Ontology-based focused crawling. In Andrew Kusiak, S.-g. L., editor, *International Conference on Information, Process, and Knowledge Management*, EKNOW '09, pages 123–128, Cancun, México. IEEE Computer Society.

[Magnini et al., 2006] Magnini, B., Pianta, E., Popescu, O., and Speranza, M. (2006). Ontology population from textual mentions: Task definition and benchmark. In Paul Buitelaar, P. C. and Loos, B., editors, *Proceedings of the OLP2 workshop on Ontology Population and Learning*, pages 26–32, Sidney, Australia. Association for Computational Linguistics.

[Motik et al., 2008] Motik, B., Patel-Schneider, P., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., and Smith, M. (2008). Owl 2 web ontology language: structural specification and functional-style syntax. *W3C Recommendation*, 27.

[Neches et al., 1991] Neches, R., Fikes, R., Finin, T. W., Gruber, T. R., Patil, R. S., Senator, T. E., and Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56.

[Nie and Zhou, 2008] Nie, X. and Zhou, J. (2008). A domain adaptive ontology learning framework. In *Proceedings of the IEEE International Conference on Networking, Sensing and Control*, ICNSC 2008, pages 1726–1729, Hainan, China. IEEE.

[Roberts et al., 2008] Roberts, A., Gaizauskas, R., Hepple, M., Demetriou, G., Guo, Y., Setzer, A., and Roberts, I. (2008). Semantic annotation of clinical text: The clef corpus. In *LREC 2008 Workshop - Building and evaluating resources for biomedical text mining*, Marrakech, Morocco.

[Rudolph et al., 2007] Rudolph, S., Völker, J., and Hitzler, P. (2007). Supporting lexical ontology learning by relational exploration. In Priss, U., Polovina, S., and Hill, R., editors, *Conceptual Structures: Knowledge Architectures for Smart Applications, 15th International Conference on Conceptual Structures*, volume 4604 of *Lecture Notes in Computer Science*, pages 488–491, Sheffield, UK. Springer.

[Staab and Studer, 2004] Staab, S. and Studer, R., editors (2004). *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, New York, USA, first edition.

[Studer et al., 1998] Studer, R., Benjamins, V., and Fensel, D. (1998). Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197.

[Sumathi and Sivanandam, 2006] Sumathi, S. and Sivanandam, S. N. (2006). *Introduction to Data Mining and its Applications*, volume 29 of *Studies in Computational Intelligence*. Springer, New York, USA, first edition.

[Uschold and Grüninger, 1996] Uschold, M. and Grüninger, M. (1996). Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136.

[Van de Maele et al., 2008] Van de Maele, F., Spyns, P., and Meersman, R. (2008). An ontology-based crawler for the semantic web. In Meersman, R., Tari, Z., and Herrero, P., editors, *On the Move to Meaningful Internet Systems: OTM 2008 Workshops, OTM Confederated International Workshops and Posters, ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS 2008*, volume 5333 of *Lecture Notes in Computer Science*, pages 1056–1065, Monterrey, México. Springer.

[Weiser, 1991] Weiser, M. (1991). The computer for the twenty-first century. *Scientific American*, 265(3):94–104.

[Yang and Hsu, 2009] Yang, S. and Hsu, C. (2009). Ontology-supported web crawler for information integration on call for papers. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 6, pages 3354–3360, Baoding, Hebei, China. IEEE.

[Yuvarani et al., 2006] Yuvarani, M., Iyengar, N. C. S. N., and Kannan, A. (2006). Lscrawler: A framework for an enhanced focused web crawler based on link semantics. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '06, pages 794–800, Hong Kong, China. IEEE Computer Society.

[Zheng et al., 2008] Zheng, H.-T., Kang, B.-Y., and Kim, H.-G. (2008). An ontology-based approach to learnable focused crawling. *Information Sciences: an International Journal*, 178(23):4512–4522.