Universidad Nacional de Educación a Distancia

Máster en Lenguajes y Sistemas Informáticos

PROPUESTA DE BÚSQUEDA SEMÁNTICA: APLICACIÓN AL CATÁLOGO DE MAPAS, PLANOS Y DIBUJOS DEL ARCHIVO GENERAL DE SIMANCAS

Autor: Jose Alberto Benítez Andrades

Directora: Ana Mª García Serrano

Febrero 2013

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los objetivos generales de este trabajo han sido desde el principio: (1) plantear un trabajo de iniciación a la investigación aplicada, (2) establecer una metodología de tratamiento de información estructurada, como es un catálogo ya existente, para proponer su enriquecimiento semántico y (2) probar experimentalmente la propuesta en el **catálogo de mapas, planos y dibujos del Archivo General de Simancas (AGS)**.

El estado del arte se ha planteado sobre la base de dos perspectivas, una tecnológica y una segunda de carácter teórico-práctico que a su vez de enfoca, por una parte (a) al uso de recursos disponibles en el ámbito de la ingeniería lingüística y la recuperación de información y por otra parte (b) al contexto relacionado con el uso de ontologías en el área de la ingeniería del conocimiento, ambos en una tarea de búsqueda de información. Se han estudiado trabajos en los que se ha realizado enriquecimiento o búsqueda semántica con técnicas basadas en análisis lingüístico-semánticos superficiales o bien basadas en estructuras semánticas provenientes del análisis del dominio concreto de la aplicación tipo *case-frames*.

Por lo tanto el índice de la parte de esta memoria dedicada al estado del arte se organiza sobre la base del:

- Panorama tecnológico, que incluye descripciones de las herramientas disponibles necesarias para el trabajo de carácter teórico-práctico que se presenta,
- Trabajos de investigación relacionados con el uso de técnicas (a) lingüísticas para recuperación de información y (b) relacionadas con las ontologías.

Finalmente una vez estudiado el panorama tecnológico relacionado con la recuperación de información se ha decidido experimentar en el dominio concreto de AGS por una parte con su representación en forma de ontología y su indexación basada en facetas, y por otra con una técnica clásica para a partir de su experimentación comparar las tres aproximaciones.

Y a continuación el trabajo que se presenta de carácter teórico-práctico, se divide en los siguientes niveles:

- **Análisis del problema principal a resolver**
  - Análisis exhaustivo del problema partiendo de la base de las 7792 fichas pertenecientes al **Archivo General de Simancas**.
  - Definición del problema a resolver: se parte de información en una base de datos textuales cuyo acceso se quiere mejorar de manera semántica para facilitar su utilización por usuarios no expertos en lenguajes de consulta formales.

- **Estudio de las posibles soluciones existentes para resolver el problema**
  - Necesidad enriquecer semánticamente la base de datos existente.

- **Selección de la solución o soluciones que se van a experimentar en este trabajo de investigación**

- Análisis de la ontología en el dominio particular del **Archivo General de Simancas (AGS)**.
- Creación de un sistema de preguntas.
- Creación de un analizador (parser) semántico de consultas, bajo una interfaz gráfica accesible, usable y funcional.
- Creación de un sistema semi-automático capaz de nutrir automáticamente la base de datos categorizada.

❖ **Elección de las herramientas necesarias para poder aplicar la solución elegida**

Investigación y aprendizaje de la utilización de los siguientes elementos:

- Ficheros en formato **RDF**, **OWL** y lenguaje de consultas **SPARQL**.
- Instalación y puesta en marcha de servidor **Apache SOLR** y del software **Protègè**.
- Creación de un buscador semántico y textual en lenguaje **PHP, JavaScript, AJAX, HTML5** y **CSS3.**

❖ **Desarrollo de las herramientas necesarias para abarcar el problema**

- Necesidad de creación de herramientas intermedias de conversión de ficheros para poder utilizar otras herramientas.

❖ **Fase de aplicación de la solución, realizando la experimentación y analizando los resultados obtenidos tras la prueba.**

- Clasificación de consultas en el dominio particular de la ontología
- Realización de búsquedas de elementos de forma facetada y de forma textual
- Comparación de los resultados obtenidos

Las pruebas experimentales tras el análisis de las fichas del citado catálogo, y la implantación de dos modelos de base para las propuestas de enriquecimiento semántico, llevan a los comentarios y conclusiones del último apartado de esta memoria, principalmente relacionados con la recuperación semántica de información y con la población de ontologías.

Este trabajo de fin de máster ha sido realizado a lo largo de los años 2011, 2012 y parte de 2013, ocupando un total de aproximadamente 760 horas de investigación y desarrollo, compaginando su realización con mi actual puesto laboral en la empresa que trabajo desde Junio de 2010, de la cual también soy socio fundador, INDIPROWEB S.L. (Innovación Diseño y Programación WEB – www.indipro.es).

PARTE 1: PANORAMA TECNOLÓGICO

Se presenta el estado del arte o panorama tecnológico en dos áreas de trabajo: la web semántica y la recuperación de información. En cada una de ellas se presentan los conceptos básicos necesarios para realizar la propuesta diseñada. Además se incluyen los trabajos teórico-prácticos que más han influido en el trabajo realizado.

En concreto sobre la web semántica se ha estudiado e incluido en la memoria:

- ✓ Su concepto, herramientas y estándares disponibles.
- ✓ La definición de ontologías: Protégé
- ✓ Estándares para descripción de contenidos: RDF-Dublin Core y OWL.
- ✓ Un método para extraer relaciones semánticas desde la Wikipedia [12].

El trabajo no ha planteado la extracción de nuevas relaciones semánticas, pero el estudio de este artículo ha sido de mucho interés, para conocer la complejidad de los recursos léxicos.

Respecto a la recuperación de información se ha incluido en la memoria el estudio realizado sobre:

- ✓ Los modelos clásicos de recuperación de información.
- ✓ Estándares y herramientas relacionados con Lucene, Solar, Apache y Sparql.
- ✓ Se ha estudiado el trabajo relacionado con la recuperación ontológica dirigido por P. Castells [13].

  Finalmente en este trabajo no se ha anotado con ontologías por la ausencia de una ontología (o varias), que se adecúen al contenido de las fichas del AGS, pero sobre todo por encontrarse otra línea de investigación mas interesante, que es la que finalmente se siguió.

  Se recuerda que el Archivo General de Simancas, iniciado por Carlos V y finalizado por su hijo Felipe II, guarda toda la documentación producida por los organismos de gobierno de la monarquía hispánica desde la época de los Reyes Católicos (1475) hasta la entrada del Régimen Liberal (1834). En este trabajo solo se ha tenido en cuenta las Colecciones de fichas sobre Mapas, Planos y Dibujos.

- ✓ Los sistemas pregunta-respuesta [14].

  A partir de este interesante trabajo y de [15] se ha realizado la taxonomía de tipos de preguntas del catálogo AGS. La forma de obtención de respuestas ha influido en la definición del modelo utilizado finalmente en el trabajo.

Todas estas actividades de estudio de estándares, herramientas y trabajos relacionados, ha llevado al planteamiento del trabajo a realizar.

1. WEB SEMÁNTICA

La web tal y como se entiende a día de hoy, a pesar de sus conocidas ventajas a priori relativas a la compartición y reusabilidad de información y conocimiento, sufre del estado de los mecanismos de procesado automático para manejar la gran cantidad de páginas existentes y de los problemas derivados de la falta de interoperabilidad entre recursos e información. A continuación se presentan algunos de estos problemas y se verá cómo de la necesidad de resolverlos, se acabarán adecuando de manera natural a la Web Semántica

Aunque no es sencillo calcular el tamaño total de la web, se estima que hay alrededor de unos 4 * 10 documentos disponibles en red, lo que viene a equivaler a unos 28 millones de libros, teniendo en cuenta que la *American Research Libraries*, que agrupa un total de 100 bibliotecas de EEUU tiene unos 3.7 millones de libros y que la biblioteca de Harvard (la mayor de EEUU) tiene catalogados 15 millones de libros, haciéndose una idea del tamaño que tiene la Web. Resulta entonces que en la Web actual, hay un sistema que almacena, se puede decir, prácticamente toda la información del mundo, y que presenta un acceso casi instantáneo a la misma desde cualquier lugar del planeta con una conexión a Internet. Por otro lado, la información disponible no es solamente textual, sino que hay que sumar los documentos formados por imágenes, videos, presentaciones, gráficos, mapas etc. Además, cualquier persona puede, al menos en teoría, añadir más información a la Web (mediante la creación de un nuevo sitio web por ejemplo).

Pues bien, el conjunto de todas estas características es lo que define a la Web actual con el nombre de “Web Sintáctica”. En la “Web Sintáctica”, hay un conjunto de recursos enlazados entre sí (formando un grafo dirigido) tal y como se puede ver en la siguiente figura:

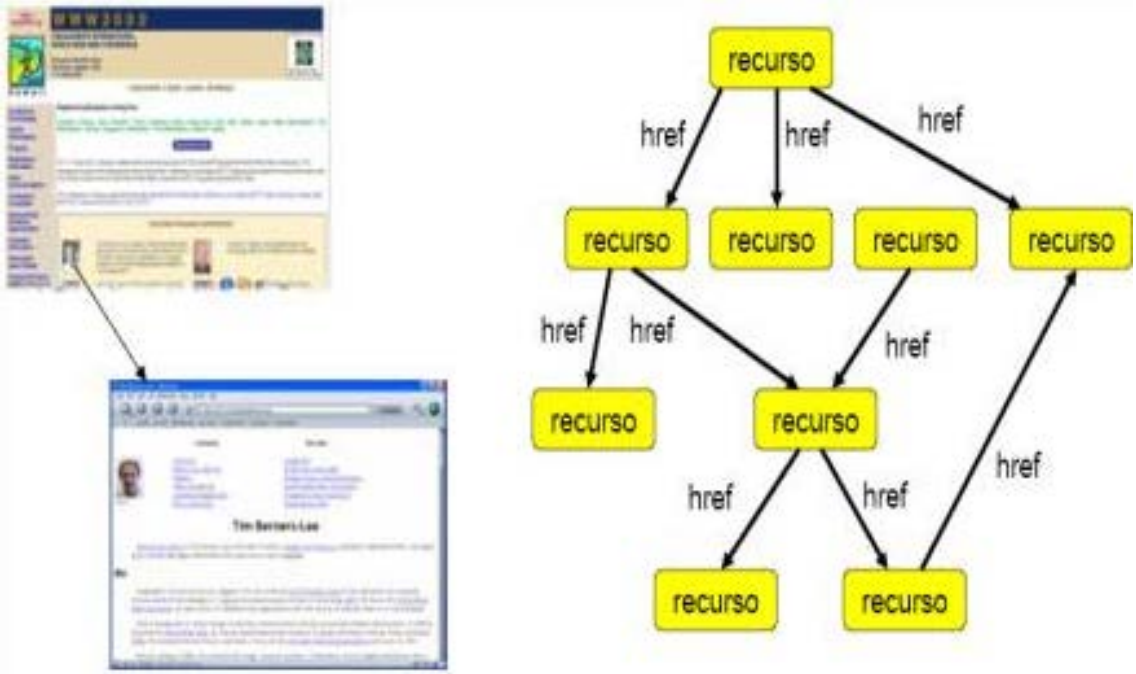

Ilustración 1 La Web Sintáctica

Para gestionar esta gran cantidad de información, han aparecido numerosos buscadores que enlazan las páginas (Google, por ejemplo, indexa cerca de 9.000 millones de páginas), sin

embargo, a pesar de la potencia que demuestran, aún quedan lejos de proporcionar al usuario las respuestas adecuadas y esperadas a partir de las preguntas que realizan, fundamentalmente por tres motivos:

- no enlazan con la totalidad de páginas existentes, con lo que se hace necesaria la actualización constante de los índices mediante robots de búsqueda automáticos
- la escasa precisión de los resultados, ya que algunas consultas devuelven varios millones de resultados distintos.
- y la alta sensibilidad al vocabulario empleado en la búsqueda, es decir, un documento que se busque debe estar descrito con las palabras usadas en la búsqueda.

Por otro lado, tenemos también los problemas de interoperabilidad de aplicaciones, que se deben a la falta de entendimiento técnico, sintáctico y semántico entre ellas, lo que repercute en el coste de los servicios que las empresas proporcionan y ralentiza la implantación de nuevos servicios útiles para el usuario.

La Web Semántica trata de resolver todos estos problemas, añadiendo a la Web Sintáctica la semántica que le falta para crear un entorno en donde poder acceder a la información que satisfaga las necesidades del usuario de un modo lo más exacto y completo posible, a la vez que se facilita el procesado de la misma y se resuelven los problemas de interoperabilidad entre aplicaciones que hemos indicado anteriormente.

La primera de las definiciones sobre Web Semántica viene de la mano del creador del concepto, Tim Berners-Lee:

"El primer paso es colocar los datos en la Web de un modo en que las máquinas puedan entenderlos naturalmente o convertirlos a esa forma. Esto crea lo que yo llamo la Web Semántica: una red de datos que pueden ser procesados directa o indirectamente por máquinas"

[Weaving the Web, 1999]

"La Web Semántica es una extensión de la Web en la cual la información aparece con un significado bien definido, lo que facilita que los ordenadores y la gente trabajen en cooperación"

[The Semantic Web, Scientific American, Mayo de 2001]

Tim Berners-Lee plasmó su idea original de la Web Semántica en la siguiente figura, en donde la Web Semántica se articula en base a conceptos como el enlazado de información, el hipertexto, los sistemas jerárquicos, etc.:

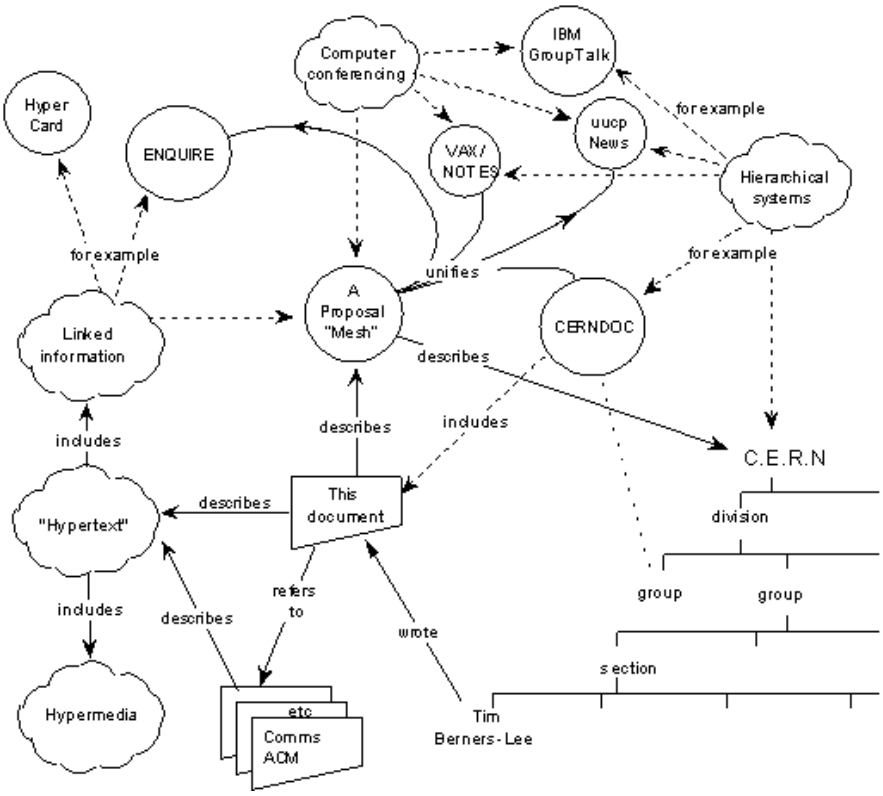

Ilustración 2 La Web Sintáctica por Tim Berners-Lee

La definición oficial que existe en la página del W3C es:

"La Web Semántica es la Web de los datos. Hay muchos datos que usamos cada día y que no son parte de la Web. Podemos ver los apuntes bancarios en la web, e incluso nuestras fotografías y citas en el calendario. Pero ¿podemos ver las fotos en un calendario para ver qué es lo que estaba haciendo cuando las hice? ¿Puedo ver mis apuntes bancarios en el calendario? La respuesta a estas preguntas es no. Y ¿por qué no? La respuesta es porque no tenemos una Web de datos. Y esto es debido a que los datos están controlados por las aplicaciones, y cada una los guarda y trata de manera particular.

La Web Semántica trata sobre dos cosas. Sobre formatos comunes para el intercambio de datos, mientras que en la Web original solamente se intercambian documentos. Y trata sobre los lenguajes que representan los datos como objetos del mundo real. La Web Semántica proporciona un framework común que permite a los datos ser compartidos y reutilizados a través de las límites impuestos por aplicaciones, empresas o comunidades. Es un esfuerzo de desarrollo colaborativo liderado por la W3C y un gran número de investigadores y socios industriales." [http://www.w3.org/2001/sw/]

Dicho de otro modo, la Web Semántica se fundamenta en el hecho de que las máquinas comprendan el significado de la información disponible (o los datos incluidos en la información. Luego la Web Semántica es pura Inteligencia Artificial, área de estudio dedicada a plantear correctamente nuevos problemas (posiblemente relacionados con el comportamiento humano) y buscar su solución, aunque a las máquinas aún les queda un largo camino por recorrer para poder llegar a comprender siguiendo un esquema de razonamiento como el que hacemos los humanos, que sí que son capaces de llegar a conclusiones (deducciones o

inferencia) mediante procesos de lógica-matemática. Ni que decir tiene que las conclusiones a las que se lleguen dependerán de la validez, o de lo buenas que sean, las reglas de deducción que se utilicen para llegar a estas. Gráficamente:

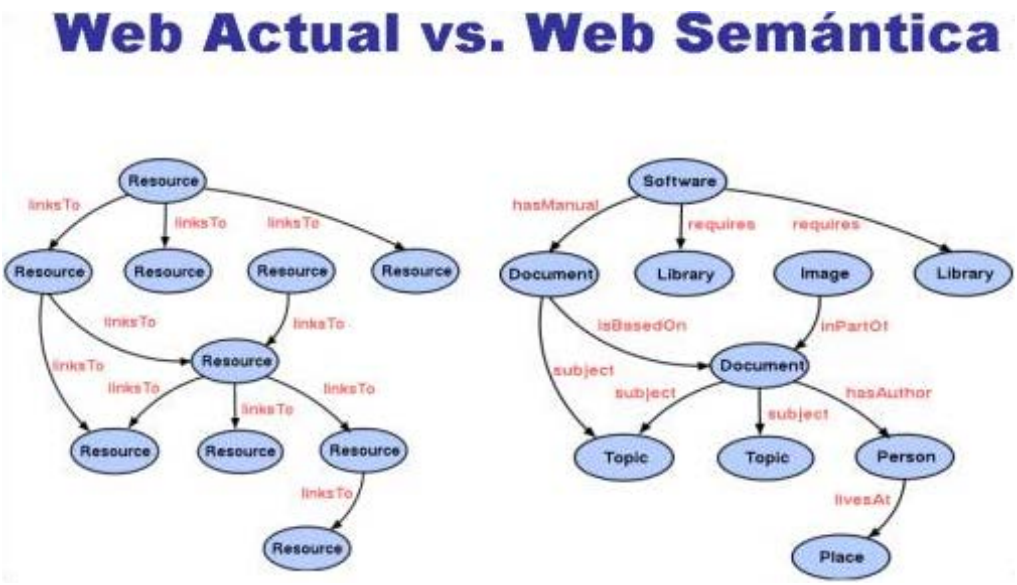

Ilustración 3 Web Actual vs Web Semántica

La figura muestra la forma en la que en la Web Sintáctica se compone de recursos enlazados a través de enlaces (hipertexto) frente a la Web Semántica, en donde los elementos quedan algo más caracterizados, de este modo vemos que una entidad “Software” requiere, de una entidad “Biblioteca”, y que tiene un documento que es su manual de uso, etc. De este modo se puede establecer una estructura semántica de un concepto al que luego se apliquen reglas lógicas para inferir nuevo conocimiento.

La W3C utiliza la siguiente figura para mostrar cuales son los elementos estructurales básicos que forman parte de la Web Semántica:

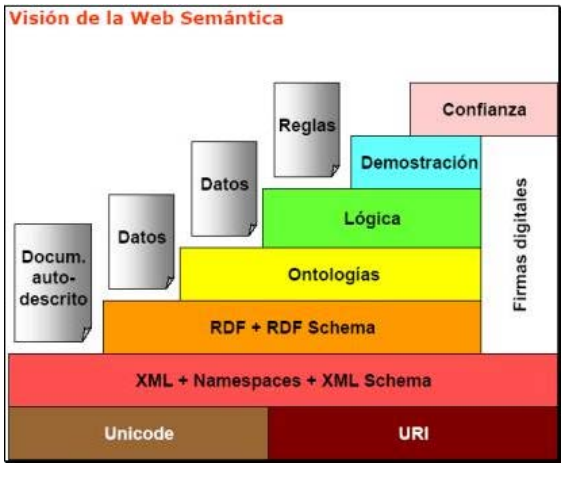

Ilustración 4 Visión de la web semántica

De los elementos anteriores, se reseña lo siguiente:

- Confianza: Se refiere a las técnicas que aseguran la identidad y fiabilidad de los datos y servicios

- Prueba: Explican y verifican los pasos de los razonamientos lógicos
- Lógica: El razonamiento lógico determina si los datos son correctos e infiere las conclusiones oportunas a partir de los mismos.

Aunque como hemos dicho Tim Berners-Lee es el autor original de la Web Semántica, desde hace una década, en Europa, los grupos de investigación que más están actuando como "evangelizadores" de la Web Semántica, entre otros, son los dirigidos por Rudi Studer y Dieter Fensel. Rudi Studer profesor en la Universidad de Karlsruhe (Alemania), es el responsable del grupo de investigación sobre gestión del conocimiento dentro del instituto AIFB (Instituto de Informática Aplicada y Métodos de Descripción Formal de la Universidad de Karlsruhe, Institute of Applied Informatics and Formal Description Methods), aparte de este cargo ostenta la presidencia de la Semantic WebScience Association .

Dieter Frensel, es profesor y director del DERI Digital Enterprise Innsbruck Research Institute, en la Universidad de Innsbruck en Austria. A lo largo de su larga carrera profesional ha impartido gran multitud de seminarios técnicos, dedicándose en los últimos años a la divulgación e investigación de la aplicación de la Web Semántica. En el 2005 fue galardonado con el segundo premio sobre Sistemas Semánticos del FIT-IT, con su proyecto de nombre GRISINO cuyo objetivo era la creación de un sistema Grid Semántico que permitiese la comunicación entre objetos distribuidos de manera inteligente.

Desde entonces ha habido un gran auge en el desarrollo de ontologías, y existen varias herramientas y estándares para ello. A continuación se describen algunas.

1.1. HERRAMIENTAS Y ESTÁNDARES

Probablemente la Web Semántica no podría haberse desarrollado sin la creación de un conjunto de estándares bien definidos. Uno de los resultados del empuje general hacia una estructura más semántica en la Web, fue el desarrollo del lenguaje XML, que permite a los desarrolladores usar su propio conjunto de etiquetas (*markup-tags*).

Pero esto que parece tan simple, esconde una potencia muy grande, ya que XML podemos considerarlo como un metalenguaje, o dicho de otra manera, un lenguaje para definir otros lenguajes de etiquetas estructurados. BPEL y WSMO, son lenguajes que utilizan XML para su definición, del mismo modo, los lenguajes para la definición de ontologías como RDF y OWL también son lenguajes XML. De hecho hoy día se acepta como válido que la Web Semántica se construirá basándose en XML y en las tecnologías asociadas al mismo.

Mientras que HTML es un lenguaje de marcado para documentos de hipertexto (a nivel sintáctico), XML es un lenguaje de marcado para documentos de todas las clases y la base para lenguajes más "semánticos". Se dice que XML es extensible porque permite al programador asociar sus propias etiquetas a los datos.

La motivación original que impulsó la aparición de XML fue la necesidad de la creciente globalización en los negocios electrónicos; el B2B, B2C, C2C necesitan por definición, el intercambio de datos, además las soluciones del tipo EDI han demostrado ser insuficientes en el mundo de hoy en día.

Un documento XML consiste en una serie de etiquetas anidadas abiertas (<) y cerradas (>), y entre estos delimitadores de apertura y cierre se indica un nombre, denominado nombre de la etiqueta, donde cada etiqueta tiene ciertos valores. El programador define el nombre de cada una de las etiquetas y las combinaciones que pueden darse entre ellas, mediante dos técnicas diferentes: los documentos DTD o los XML Schema.

La ventaja que presenta el uso de estos estándares es que facilitan a los analizadores sintácticos o parsers para comprobar si un documento es válido o no, es decir, verificar la gramática del documento y validar si las etiquetas que contiene junto con los anidamientos de las mismas son válidas o no.

El uso de XML aporta una serie de beneficios:

- Cualquier documento y cualquier tipo de dato puede expresarse como un documento XML, de este modo se pueden definir los datos independientemente del lenguaje o plataforma siendo un formato universal para el intercambio de datos. Cualquiera que lo desee puede crear un documento XML conforme a una DTD o XML Schema que necesite y utilizarlo en sus aplicaciones o transacciones. Este beneficio se resume generalmente diciendo que XML es un lenguaje abierto.
- Es auto-descriptivo, se almacenan datos y la estructura de los mismos. Resulta muy fácil entender un documento XML echando simplemente un vistazo al mismo. Dicho de otra forma, XML es un metalenguaje.
- Es compartible a través de la red, los ficheros XML son ficheros de texto, luego cualquier plataforma es capaz de operar con ellos.
- Se basa en el uso de Unicode, lo que permite crear documentos para cualquier idioma.
- Al hacer uso de los DTDs o XML Schema se pueden validar.
- Permite la integración tanto de datos estructurados (como las tablas relacionales) y poco estructurados (como los documentos).
- Finalmente XML es un lenguaje neutro, entendiendo por neutro que es independiente el mecanismo de presentación que se utilice para mostrar los datos del mismo, lo que facilita aún más su interoperabilidad entre dispositivos de distinto tipo (por ejemplo la web vista desde un navegador y vista a través del móvil).

Sin embargo los estándares no aportan una forma de representación estándar de la semántica de la información o conocimiento, por lo que surge un nuevo concepto clave en la web semántica: las ontologías.

1.1.1. LAS ONTOLOGÍAS Y SUS ESTÁNDARES: RDF, OWL Y SPARQL

En el ámbito de la inteligencia artificial, una ontología consiste en una descripción que define formalmente relaciones entre términos. El tipo más común de ontología para la Web está formada por una taxonomía (una organización jerarquizada que se emplea para estructurar contenidos, dividiéndolos en clases de objetos y agrupándolos según sus características) y un conjunto de reglas de inferencia que definen las condiciones sobre las relaciones entre los objetos. Las ontologías son uno de los elementos fundamentales de la Web semántica y con ellas se describen formalmente las relaciones entre términos, esto es piezas de información.

Existen varios lenguajes de definición de ontologías, siendo algunos de los más comunes RDF (http://www.w3.org/RDF/), DAML+IOL (http://www.w3.org/TR/daml+oil-reference) o bien OWL (http://www.w3.org/2004/OWL/), que se explican a continuación.

Resource Description Framework ó RDF (http://www.w3.org/RDF/) es un modelo estándar para el intercambio de datos en la Web. RDF tiene características que facilitan la unión de datos de diferentes fuentes incluso cuando los esquemas que los definen son diferentes, y además soporta la evolución de esquemas en el tiempo sin requerir que los consumidores de los datos cambien.

Una de las características más importantes de RDF es que extiende la estructura de enlaces de la Web para utilizar URIs (*Uniform Resource Identifier*), que son cadenas de caracteres cortas que identifican inequívocamente un recurso (http://tools.ietf.org/html/rfc3986) y que definen relaciones entre sus elementos.

Utilizando este modelo RDF, que se conoce como "triple, o tripleta" o sentencia, permite que datos estructurados y semi-estructurados se mezclen y se compartan entre diferentes aplicaciones.

RDF es hoy el estándar más popular y extendido en la comunidad de la web semántica. (http://www.w3.org/TR/rdf-schema/) es un lenguaje que se utiliza para describir vocabularios con RDF, y se trata de una extensión semántica de RDF. *RDF Schema* proporciona mecanismos para describir grupos de recursos relacionados y las relaciones entre estos recursos. Con los recursos *RDF Schema* se determinan las características de otros recursos, como dominios o rangos de propiedades.

SPARQL (http://www.w3.org/TR/rdf-sparql-query/) es un lenguaje de consulta sobre RDF, que permite hacer búsquedas sobre los recursos de la Web semántica utilizando distintas fuentes de datos.

Al lenguaje RDF le siguieron otros lenguajes de marcas para describir recursos Web, como OIL (*Ontology Inference Language*) y DAML (DARPA *Agent Markup Language*), dos lenguajes muy similares que se terminaron fundiendo en DAML+OIL (http://www.w3.org/TR/daml+oil-reference). Este lenguaje es una extensión de RDF y *RDF Schema* que proporciona primitivas de modelado más ricas. A continuación aparece el *Web Ontology Languaje* u OWL (http://www.w3.org/2004/OWL/) que es un lenguaje para descripción de ontologías Web que pretende aportar:

- Capacidad para ser distribuidas entre diferentes sistemas.
- Escalabilidad para las necesidades de la Web.
- Compatibilidad con los estándares Web para accesibilidad e internacionalización.
- Extensibilidad.

OWL se construye sobre RDF y RDF *Schema*, y añade más vocabulario para describir propiedades y clases: entre otras, relaciones entre clases, cardinalidad, simetría, más tipos de propiedades, características de las propiedades y enumeraciones de clases.

Si bien RDF y OWL son hoy en día los lenguajes más consolidados, existen otros lenguajes interesantes, aunque con menos usuarios, como TopicMaps, OCML o WebODE.

Hay varias herramientas disponibles para el desarrollo de ontologías, una de las primeras y casi ya un estándar es Protégé (http://protege.stanford.edu/), herramienta libre y de código abierto que permite crear y editar ontologías. Protégé tiene su propio lenguaje interno para definir ontologías, pero permite también trabajar con RDF y OWL de modo transparente. Protégé se está convirtiendo en una herramienta muy potente a cuyo desarrollo contribuyen muchas personas con los *plugins*.

A continuación se reflexionará sobre la necesidad y las ventajas de desarrollar una ontología y se presentará brevemente la herramienta Protégé.

1.1.2. ¿POR QUÉ DESARROLLAR UNA ONTOLOGÍA?

En los últimos años las ontologías (especificaciones formales y específicas de términos y relaciones entre ellos (Gruber 1993)) han llegado a ser comunes en el *World-Wide Web* y van desde grandes taxonomías que categorizan sitios Web (tales como en Yahoo!) a categorizaciones de productos para vender y sus características (tales como Amazon.com).

En medicina, por ejemplo, se han producido grandes, estandarizados y estructurados vocabularios tales como Snomed (Price and Spackman 2000) y la red semántica *Unified Medical Language System* (Humphreys and Lindberg 1993). Están surgiendo otras ontologías amplias y de propósito general, como por ejemplo, el programa de desarrollo de las naciones unidas (*United Nations Development Program*) y Dun & Bradstreet unieron esfuerzos para desarrollar la ontología UNSPSC que provee terminología para productos y servicios (www.unspsc.org). Es un sistema de cifrado que clasifica productos y servicios para fines comerciales a escala mundial. La gestión y desarrollo de UNSPSC está coordinado por GS1 US y respaldado por la ONU desde 2003. La versión actual de la clasificación contiene más de 16.000 términos y puede descargarse libremente del portal de UNSPSC.

Una ontología define un vocabulario común para usuarios que necesitan compartir información en un dominio. Ella contiene definiciones de conceptos básicos y sus relaciones que pueden ser interpretadas por una máquina. ¿Por qué alguien desearía desarrollar una ontología? Algunas de las razones son:

- Compartir el entendimiento común de la estructura de información entre personas o agentes de software.
- Permitir la reutilización de conocimiento de un dominio.
- Explicitar suposiciones de un dominio.
- Separar el conocimiento del dominio del conocimiento operacional.
- Analizar el conocimiento de un dominio.

Compartir el entendimiento común de la estructura de información entre personas y agentes de software es una de las más importantes metas al desarrollar ontologías (Musen 1992; Gruber 1993). Por ejemplo, supongamos que varios sitios Web contengan información médica o provean servicios de e-commerce médico. Si esos sitios Web comparten y publican la misma ontología subyacente de los términos que usan, entonces agentes de software podrían extraen y agregar información de esos sitios diferentes. Los agentes podrían usar esta información agregada para responder solicitudes de los usuarios o servir como datos de entrada a otras aplicaciones.

Permitir la reutilización de conocimiento de un dominio fue una de las fuerzas conductoras detrás recientes trabajos en la investigación sobre ontologías. Por ejemplo, modelos para diferentes dominios necesitan representar la noción de tiempo. Esta representación incluye las nociones de intervalo de tiempos, puntos en el tiempo, medidas relativas de tiempo, y cosas por el estilo. Si un grupo de investigadores desarrolla tal ontología en detalle, otros podrían simplemente reusarla en sus dominios. También se puede reusar una ontología general, tal como la ontología UNSPSC, y extenderla para describir nuestro dominio de interés.

La explicitación de suposiciones de un dominio, que subyacen bajo una implementación, permite cambiar esas suposiciones fácilmente si el conocimiento del dominio cambia. Suposiciones modificadas explícitamente acerca del mundo en algún lenguaje de programación hacen que las suposiciones no solo sean difíciles de hallar sino también difíciles de cambiar, en particular para alguien sin competencias en programación. Además, las especificaciones explícitas del dominio de conocimiento son útiles para nuevos usuarios que deben aprender el significado de los términos del dominio.

La separación del conocimiento del dominio del conocimiento operacional es otro uso común de las ontologías. Por ejemplo se puede describir la tarea de configuración de un producto a partir de sus componentes de acuerdo a especificaciones requeridas e implementar un programa que hace independiente esta configuración de los productos y componentes en sí (McGuinness and Wright 1998).

Analizar el conocimiento de un dominio es posible una vez que una especificación declarativa de los términos está disponible. El análisis formal de los términos es imprescindible al intentar reusar ontologías existentes y al extenderlas (McGuinness et al. 2000).

1.1.3. HERRAMIENTA DE DESARROLLO: PROTÉGÉ

Protégé es un editor de ontologías y un entorno basado en conocimiento, gratuito y de código abierto. Provee un conjunto de herramientas para construir modelos de dominio y aplicaciones basadas en conocimiento con ontologías. Básicamente, implementa un conjunto rico de estructuras de modelado de conocimiento y acciones que soportan la creación, visualización, y manipulación de ontologías en varios formatos de representación. Puede ser extendida, para construir herramientas basadas en conocimiento y aplicaciones, por medio de su arquitectura basada en *plugins* y de una (*Application Programming Interfase)* API escrita en el lenguaje de programación Java.

La plataforma de Protégé soporta dos modos de modelado de ontologías a través de los editores Protégé-Frames y Protégé-OWL.

- *Protégé-Frames*: El editor *Protégé-Frames* permite a los usuarios construir ontologías basadas en marcos (frame-based), en concordancia con el protocolo *Open Knowledge Base Connectivity protocol* (OKBC) para acceder a bases de conocimiento almacenadas en sistemas de representación de conocimiento. Los marcos (frames) son las unidades principales para la construcción de una base de conocimiento. En este modelo, una ontología consiste de un conjunto de clases, propiedades, restricciones y axiomas. Las clases (*classes*) representan los conceptos destacados del dominio, las propiedades (*slots*) son asociados a clases para describir sus propiedades o relaciones, las restricciones (*facets*) describen propiedades de las relaciones, y los axiomas

especifican restricciones adicionales. Una base de conocimiento en este modelo, contiene también un conjunto de instancias de esas clases (ejemplares individuales de los conceptos que mantienen valores específicos para sus propiedades).

- Protégé-OWL: El editor Protégé-OWL permite a los usuarios construir ontologías para la Web Semántica en este lenguaje.

Una ontología OWL puede incluir descripciones de clases, propiedades e instancias. La semántica formal de OWL especifica cómo derivar sus consecuencias lógicas, es decir, hechos que no están explícita o literalmente presentes en una ontología, sino implicados por la semántica. Estas implicaciones pueden estar basadas en un sólo documento o en múltiples documentos distribuidos que han sido combinados usando mecanismos definidos dentro de OWL.

La API central (core API) , se utiliza para acceder a la funcionalidad básica de Protégé y bases de conocimiento basadas en marcos (aquellas creadas con Protégé-Frames). También posee una API OWL, que extiende la API central para proveer acceso a ontologías OWL (aquellas creadas con Protégé-OWL). Las APIs de Protégé pueden usarse directamente por aplicaciones externas para acceder a bases de conocimiento creadas con Protégé y hacer uso de los formularios e interfaces de usuarios sin ejecutar Protégé.

1.2. DESCRIPCIÓN DE CONTENIDOS: RDF DUBLIN CORE

Debido a que en este trabajo ha sido necesario realizar distintas pruebas y experimentos con ficheros en formato RDF Dublin Core, a continuación se describen las características de este formato de manera detallada.

La Iniciativa Dublin Core (DCMI) http://dublincore.org/ comenzó su andadura en 1995 en un encuentro en Dublin, Ohio (USA) en el que participaron el NCSA (*National Center for Supercomputing Applications*) y OCLC (*On Line Library Computer Center*) , junto con representantes de la IETF (*Internet Engineering Task Force*) y en el que bibliotecarios, proveedores de contenido y expertos en lenguajes de marcado pretendieron desarrollar estándares para describir los recursos de información y facilitar su recuperación. Así nació un pequeño conjunto de descriptores, en principio pensados para que fuera el propio autor el que los incluyera en el documento o recurso, pero que rápidamente adquirieron alcance global porque también se interesaron en ellos numerosos y variados proveedores de información pertenecientes a diferentes sectores como el de las artes, las ciencias, la educación, los negocios y la administración.

Hoy, los metadatos Dublin Core se han convertido en uno de los estándares más extendidos para la recuperación de información en la *World Wide Web* y el DC se ha convertido en un vocabulario muy utilizado no sólo en el ámbito bibliotecario y documental, sino en otros muchos sectores. Además, este conjunto de metadatos se puede utilizar no solo con HTML, sino sobre otros lenguajes estructurados como XML y conjuntamente con otros lenguajes de descripción como RDF. *The Dublin Core metadata element set* se convirtió en norma ISO 15836/2003 en febrero de 2003.

El conjunto de elementos Dublin Core se centró en 13 elementos, pero concluyó con 15 descriptores como resultado de un consenso y un esfuerzo interdisciplinar e internacional. Ya existen transcripciones a 20 idiomas y ha sido adoptado por el CEN/ISS (European Committee for Standardization / Information Society Standardization System) y posee dos RFCs de Internet (*Requests for Comments*) (RFC2413) y (RFC2731). Es también el estándar oficial del WWW *Consortium* y el estándar del Z39.50. Los metadatos Dublin Core han sido aprobados por el organismo nacional de estandarización norteamericano (ANSI/NISO Z39.85) y los utilizan como base tanto gobiernos como agencias supranacionales y muchas otras iniciativas de metadatos pertenecientes a comunidades específicas como bibliotecas, archivos, en educación, negocios, etc.

Los metadatos Dublin Core tratan de ubicar, dentro de Internet, los datos necesarios para describir, identificar, procesar, encontrar y recuperar un documento introducido en la red. Si este conjunto de elementos Dublin Core se lograra aceptar internacionalmente supondría que todos los procesos que indizan documentos en Internet encontrarían, en la cabecera de los mismos, todos los datos necesarios para su indización y además estos datos serían uniformes y mejoraría la efectividad de los motores de búsqueda.

La *Dublin Core Metadata Initiative* (DCMI) es la responsable del desarrollo, estandarización y promoción del conjunto de los elementos de metadatos Dublin Core. Su objetivo es elaborar normas interoperables sobre metadatos y desarrollar vocabularios especializados en metadatos para la descripción de recursos que permitan sistemas de recuperación más inteligentes. En concreto, la Iniciativa pretende:

- Desarrollar estándares de metadatos para la recuperación de información en Internet a través de distintos dominios
- Definir el marco para la interoperabilidad entre conjuntos de metadatos
- Facilitar el desarrollo de conjuntos de metadatos específicos de una disciplina o comunidad que trabaja dentro del marco de la recuperación de información

La información sobre DCMI está disponible en la URL: http://dublincore.org/ y PURL http://purl.org/metadata/dublin_core , recientemente, se ha creado una traducción de la web al español en Dublin Core Metadata Initiative: http://es.dublincore.org

En España ha sido, principalmente, la RedIris la que, casi en solitario, comenzó a trabajar con los metadatos Dublin Core a través del grupo denominado iris-index (http://www.rediris.es/metadata/) que gestiona la lista de distribución DCMI-ES (foro español del *Dublin Core Metadata Initiative* http://www.rediris.es/list/info/dcmi-es.es.html).En el año 2001 se creó un grupo de trabajo sobre normalización para la recuperación de información en Internet en el marco de SEDIC (Sociedad Española de Documentación e Información Científica): http://www.sedic.es/gt_normalizacion.htm , uno de cuyos temas de interés es, precisamente, los metadatos Dublin Core : http://www.sedic.es/gt_normalizacion_mirror.htm

Entre los estándares DCMI y especificaciones DC figuran las siguientes:

- **Codificación Dublin Core en HTML (IETF RFC 2731):** http://www.ietf.org/rfc/rfc2731.txt
- **Metadatos Dublin Core para la Recuperación de Recursos.**

(IETF RFC 2413). http://www.ietf.org/rfc/rfc2413.txt

- **Conjunto de Elementos de metados Dublin Core, Versión 1.1: Descripción de Referencia**: http://dublincore.org/documents/dces/ Describe el conjunto de los 15 elementos Dublin Core.
- **Términos de metadatos Dublin Core**
  (DCMI Metadata Terms) http://dublincore.org/documents/dcmi-terms/
  Recoge todos los términos de metadatos utilizados por la Iniciativa de Metadatos Dublin Core, incluyendo elementos, elementos refinados, esquemas de codificación y términos del vocabulario.
- **Vocabulario Tipo DCMI**
  (DCMI Type Vocabulary) : http://dublincore.org/documents/dcmi-type-vocabulary/
  El Vocabulario Tipo DCMI proporciona una lista general de términos aprobados que pueden usarse como valores por el elemento Recurso Tipo para identificar el género del recurso.
- **Calificadores Dublin Core**
  (Dublin Core Qualifiers) : http://dublincore.org/documents/dcmes-qualifiers/
  Este documento que, en principio, describía los calificadores principales que regían Dublin Core, las dos categorías de calificadores y ejemplos de listas de calificadores aprobados por la Comisión de Uso de Dublin Core, ahora remite a *DCMI Metadata Terms* http://dublincore.org/documents/dcmi-terms/ pues en esta Especificación es donde quedan recogidos tanto los elementos para refinar, como los esquemas de codificación, los dos tipos de calificadores de los elementos Dublin Core.
- **Esquema de codificación del punto *DCMI***
  (DCMI *Point Encoding Scheme*) . http://dublincore.org/documents/dcmi-point/
  Un punto de localización en el espacio, y métodos para codificarlos en una cadena de texto.
- **Método de Codificación Periódica de DCMI**
  (DCMI *Period Encoding Scheme*) **:** http://dublincore.org/documents/dcmi-period/
  Para especificar los límites de un intervalo de tiempo y los métodos para codificar éste en una cadena de texto.
- **DCMI DCSV** *(Dublin Core Structured Values)* Sintaxis para escribir una lista de valores etiquetados en una cadena de caracteres: http://dublincore.org/documents/dcmi-dcsv/ Se describe un método para grabar una lista de valores etiquetados en una cadena de caracteres, llamado Valores Estructurados Dublin Core, con la etiqueta DCSV. El propósito de esta anotación es ofrecer información estructurada en las descripciones de metadatos Dublin Core.
- **Esquema de Codificación DCMI**
  (DCMI Box Encoding Scheme) : http://dublincore.org/documents/dcmi-box/
  Especificación de los límites espaciales de un lugar y métodos para codificarlo en una cadena de texto.
- **Usar Dublin Core**

  http://dublincore.org/documents/usageguide/

  Este documento es un punto de acceso para los usuarios de Dublin Core, tanto para no especialistas, a quienes les ayudará para la creación de registros descriptivos simples

para fuentes de información; como para especialistas, que encontrarán un punto de referencia útil para la documentación de Dublin Core, con sus cambios y ampliaciones.

- **Política de Espacios de nombre DCMI** (*Namespace Policy for the Dublin Core Metadata Initiative*) **:** http://dublincore.org/documents/dcmi-namespace/ Un namespace XML es una colección de nombres, identificados por una referencia URI, que son usados en documentos XML como elementos tipo y atributos de nombre. El uso de los namespace XML para identificar excepcionalmente términos de metadatos, permite a esos términos no ser utilizados de manera ambigua. DCMI adopta este mecanismo para la identificación de todos los términos DCMI. Este documento especifica los acuerdos realizados para identificar actuales y futuros *namespaces* DCMI.
- **Expresar Dublin Core en meta-elementos HTML/XHTML y elementos de enlace** http://dublincore.org/documents/dcq-html/ Este documento describe cómo usar metadatos Dublin Core codificados en elementos <meta> y <link> de HTML/XHTML.
- **Expresar Simple Dublin Core en RDF/XML** : http://dublincore.org/documents/dcq-html/ El formato Dublin Core puede ser representado en muchos formatos de sintaxis. Este documento explica cómo codificar DCMES en RDF/XML, ofrece una DTD para validar los documentos y describe un método para enlazarlos desde las páginas web.
- **Guía para implementar Dublin Core en**: http://dublincore.org/documents/dc-xml-guidelines/ Este documento ofrece una guía para implementar aplicaciones de metadatos Dublin Core usando XML, tanto en aplicaciones DC simples como calificadas.
- **Declaraciones de términos Dublin Core representadas en lenguaje de esquemas XML** : http://dublincore.org/schemas/xmls/ Este documento muestra los esquemas XML que utilizan metadatos Dublin Core.

1.2.1. ELEMENTOS DUBLIN CORE

El conjunto de elementos de metadatos Dublin Core es un conjunto de metadatos previsto para describir documentos. Cada elemento es opcional y puede repetirse. Además, los elementos pueden aparecer en cualquier orden. Aunque algunos entornos, como HTML, no diferencian entre mayúsculas y minúsculas, es recomendable escribir correctamente cada metadato, según su definición, para evitar conflictos con otros entornos, como SGML y XML.

La descripción de referencia de los 15 elementos principales de los metadatos Dublin Core está en : http://dublincore.org/documents/dces/
(y en español en : http://www.rediris.es/metadata/), aunque con más detalle aparecen estos elementos, junto con el resto de términos utilizados por la Iniciativa de Dublin Core, en *DCMI Metadata Terms* (http://dublincore.org/documents/dcmi-terms/)

Se puede clasificar el conjunto de elementos Dublin Core en 3 grupos que indican la clase o el ámbito de la información que contienen :

- Elementos relacionados principalmente con el contenido del recurso:
  - **Title** (título)
  - **Subject** (tema)
  - **Description** (descripción)

- **Source** (fuente)
- **Languaje** (lenguaje)
- **Relation** (relación)
- **Coverage** (cobertura).

- Elementos relacionados principalmente con el recurso cuando es visto como una propiedad intelectual:
    - **Creator** (autor)
    - **Publisher** (editor) y, otras colaboraciones
    - **Contributor** (otros autores/colaboradores)
    - **Rights** (derechos).

- Elementos relacionados principalmente con la instanciación del recurso:
    - **Date** (fecha)
    - **Type** (tipo de recurso)
    - **Format** (formato)
    - **Identifier** (identificador)

A continuación se muestra con mas detalle cada uno de estos 15 elementos que están recogidos en *DCMI Metadata Terms* (http://dublincore.org/documents/dcmi-terms/):

ETIQUETA DEL ELEMENTO DC.	DESCRIPCIÓN
DC. Title	**Título: El nombre dado a un recurso.** Típicamente, un título es el nombre formal por el que es conocido el recurso.
DC. Creator	**Autor: La entidad primariamente responsable de la creación del contenido intelectual del recurso.** Entre los ejemplos de un creador se incluyen una persona, una organización o un servicio. Típicamente, el nombre del creador podría usarse para indicar la entidad.
DC. Subject	**Materias y palabras clave: El tema del contenido del recurso.** Un tema será expresado como palabras clave, frases clave o códigos de clasificación que describan el tema de un recurso. Se recomienda seleccionar un valor de un vocabulario controlado o un esquema de clasificación formal.
DC. Description	**Descripción: La descripción del contenido del recurso.** La descripción puede incluir, pero no se limita a: un resumen, tabla de contenidos, referencia a una representación gráfica de contenido o una descripción de texto libre del contenido.
DC. Publisher	**Editor: La entidad responsable de hacer que el recurso se encuentre disponible.** Ejemplos de editores son una persona, una organización o un servicio. Típicamente, el nombre de un editor podría usarse para indicar la entidad.
DC. Contributor	**Colaborador. La entidad responsable de hacer colaboraciones al contenido del recurso.** Ejemplos de colaboradores son una persona, una organización o un servicio. Típicamente, el nombre del colaborador podría usarse para indicar la entidad.
DC. Date	**Fecha: Una fecha asociada con un evento en el ciclo de vida del recurso.** Típicamente, la fecha será asociada con la creación o disponibilidad del recurso. Se recomienda utilizar un valor de datos codificado definido en el documento "Date and Time Formats", http://www.w3.org/TR/NOTE-datetime que sigue la norma ISO 8601 que sigue el formato YYYY-MM-DD.
DC. Type	**Tipo: la naturaleza o categoría del contenido del recurso.** El tipo incluye términos que describen las categorías generales, funciones, géneros o

	niveles de agregación del contenido. Se recomienda seleccionar un valor de un vocabulario controlado (por ejemplo, el *DCMI Vocabulary* -DCMITYPE- http://dublincore.org/documents/dcmi-type-vocabulary/). Para describir la manifestación física o digital del recurso, se usa el elemento Formato.
DC. Format	**Formato: la manifestación física o digital del recurso.** El formato puede incluir el tipo de media o dimensiones del recurso. Podría usarse para determinar el *software*, *hardware* u otro equipamiento necesario para ejecutar u operar con el recurso. Ejemplos de las dimensiones son el tamaño y la duración. Se recomienda seleccionar un valor de un vocabulario controlado (por ejemplo, la lista de Internet Media Types (MIME) que define los formatos de medios de ordenador).
DC. Identifier	**Identificación: Una referencia no ambigua para el recurso dentro de un contexto dado.** Se recomienda identificar el recurso por medio de una cadena de números de conformidad con un sistema de identificación formal, tal como un URI (que incluye el Uniform Resource Locator -URL, el Digital Object Identifier (DOI) y el International Standard Book Number (ISBN).
DC. Source	**Fuente**: **Una referencia a un recurso del cual se deriva el recurso actual.** El recurso actual puede derivarse, en todo o en parte, de un recurso fuente. Se recomienda referenciar el recurso por medio de una cadena o número de conformidad con un sistema formal de identificación.
DC. Language	**Lengua**: **La lengua del contenido intelectual del recurso.** Se recomienda usar RFC 3066 http://www.ietf.org/rfc/rfc3066.txt en conjunción con la ISO 639 [ISO639]http://www.loc.gov/standards/iso639-2/ , que define las etiquetas de dos y tres letras primarias para lenguaje, con subetiquetas opcionales. Ejemplo: “en” u “eng” para Inglés, “akk” para Acadio, y “en-GB” para inglés usado en Reino Unido.
DC. Relation	**Relación**: **Una referencia a un recurso relacionado.** Se recomienda referenciar el recurso por medio de una cadena de números de acuerdo con un sistemas de identificación formal.
DC. Coverage	**Cobertura: La extensión o ámbito del contenido del recurso.** La cobertura incluiría la localización espacial (un nombre de lugar o coordenadas geográficas), el período temporal (una etiqueta del período, fecha o rango de datos) o jurisdicción (tal como el nombre de una entidad administrativa). Se recomienda seleccionar un valor de un vocabulario controlado (por ejemplo, del Thesaurus of Geographic Names (TGN) y que, donde sea apropiado, se usen preferentemente los nombres de lugares o períodos de tiempo antes que los identificadores numéricos tales como un conjunto de coordenadas o rangos de datos.
DC. Rights	**Derechos**: **La información sobre los derechos de propiedad y sobre el recurso.** Este elemento podrá contener un estamento de gestión de derechos para el recurso, o referencia a un servicio que provea tal información. La información sobre derechos a menudo corresponde a los derechos de propiedad intelectual, copyright y otros derechos de propiedad.

Tabla 1 Elementos Dublin Core

El DCMI ya indicaba, en sus primeros momentos de desarrollo, que para promover una interoperabilidad global, podría asociarse una descripción del valor de algunos elementos a vocabularios controlados y asumía que serían desarrollados otros vocabularios controlados para asegurar esta interoperabilidad en dominios específicos. Esto es lo que se ha hecho con los elementos DC. Format y DC. Type mediante la creación de una norma ISO 11179 sobre registros de metadatos y la recomendación DCMI Type Vocabulary http://dublincore.org/documents/dcmi-type-vocabulary/ , respectivamente. Y también se utilizan vocabularios controlados para la lengua, fechas, etc.

Como se ha dicho, los elemento del Dublin Core son opcionales y repetibles, pero el esquema, además permite emplear calificadores opcionales para cada elemento que posibilitan indicar la

normativa empleada en caso de haber usado normas de descripción bibliográfica usuales. Los calificadores permiten aumentar la especificidad y precisión de los metadatos, aunque pueden también introducir cierta complejidad que disminuiría la compatibilidad con otras aplicaciones que usen Dublin Core. Debido a esto, los desarrolladores sólo deben escoger elementos del conjunto de calificadores aprobados por Dublin Core.

Para determinar estos calificadores, se dio preferencia a vocabularios, notaciones y términos actualmente mantenidos por agencias ya establecidas, a la espera de que los implementadores desarrollaran calificadores adicionales para sus propios dominios específicos.

Existía una recomendación sobre calificadores DCMI (Dublin Core Qualifiers) donde se establecían 2 tipos de calificadores

- **Refinación de elementos**: Estos calificadores hacen que el significado de un elemento sea más específico. Un elemento refinado comparte el significado del elemento no calificado, pero con un alcance más restrictivo. Si un agente no entiende un término de refinamiento específico para un elemento, debe ser capaz de ignorarlo y tratar el valor del metadato como si estuviese sin calificar. Las definiciones de términos para refinamiento de elementos deben estar públicamente disponibles.
- **Esquema de codificación (scheme)**: Estos calificadores identifican esquemas que ayudan en la interpretación del valor de un elemento. Estos esquemas incluyen vocabularios controlados y notaciones formales o reglas de análisis. Un valor expresado usando un esquema de codificación, será un símbolo (*token*) escogido de un vocabulario controlado (por ejemplo, un término de un sistema de clasificación) o una cadena (*string*) con formato de acuerdo con una notación formal (por ejemplo, "2002-01-01" como la expresión estándar de una fecha). Si un esquema de codificación no es entendido por un agente, el valor será de todas maneras, útil para un lector humano. La descripción definitiva de un esquema de codificación para calificadores debe estar claramente identificada y disponible para uso público.

Por ejemplo, el calificador **Alternative**, esto es, cualquier alternativa al título usada para sustituir al título formal del recurso, es un calificador que refina el elemento **Title**; o los calificadores **Table Of Contents** (Tabla de Contenido) y **Abstract** (Resumen) son calificadores que refinan el elemento **Description**. Y existen calificadores no sólo para refinar el título o la descripción, sino también para refinar la materia, la fecha, el tipo de recurso, el formato, la relación, la cobertura, etc.

Ejemplos de esquemas de codificación serían la clasificación DDC (Dewey Decimal Classification) o LCC (Library of Congress Classification) para materia, o Período DCMI (DCMI Period) para fecha http://dublincore.org/documents/dcmi-period/

Hoy, los calificadores (tanto elementos refinados como esquemas de codificación) según muestra la especificación: http://dublincore.org/documents/dcmes-qualifiers/ han sido incluidos dentro de la Especificación:

"DCMI Metadata Terms"http://dublincore.org/documents/dcmi-terms/

A continuación, se muestran 2 tablas que recogen dichos elementos refinados y los esquemas de codificación, extraídos de "DCMI Metadata Terms" http://dublincore.org/documents/dcmi-terms/ (la traducción es del autor de esta memoria):

Otros elementos y elementos refinados son:

Etiqueta del elemento DC.	Descripción
abstract	Un sumario del contenido del recurso
accessRights	Información acerca de quién puede acceder al recurso o una indicación de su estatus de seguridad.
accrualMethod	El método por el cuál los ítems se añaden a una colección.
accrualPeriodicity	La frecuencia con la que los ítems son añadidos a la colección.
alternative	Otra forma del título usada como un subtítulo o alternativa al título formar del recurso.
audience	Una clase de entidad para la que está indicado el uso del recurso..
available	Fecha (a menudo un rango) en la que el recurso comenzará o estará disponible.
bibliographicCitation	Una referencia bibliográfica para el recurso.
conformsTo	Una referencia a un estándar establecido con el cuál está conforme el recurso.
created	Fecha de creación del recurso.
dateAccepted	Fecha de aceptación del recurso (p. e. de la tesis por un dpto.de universidad, artículo para una revista, etc.).
dateCopyrighted	Fecha del establecimiento del copyright.
issued	Fecha de la puesta en circulación formal (p.e., publicación) de un recurso.
isFormatOf	El recurso descrito tiene el mismo contenido intelectual que el recurso referido, pero presentado en otro formato.
medium	El material o sustancia física del recurso.
modificied	Fecha en que se ha cambiado el recurso.
references	Las referencias de los recursos descritos, citas, u otros puntos de vista que se refieran el recurso.
tableOfContents	Una lista de sub-unidades del contenido del recurso.

Tabla 2 Otros elementos Dublin Core

La tabla anterior es sólo un extracto. La lista completa y detallada tanto de los elementos principales, como de elementos refinados, los esquemas de codificación y los términos del vocabulario (DCMI Type Vocabulary), pueden consultarse en la Especificación de todos los

términos de metadatos mantenidos por el Dublin Core: "DCMI Metadata Terms" http://dublincore.org/documents/dcmi-terms/

En cuanto a los esquemas codificados (scheme), podemos encontrar los siguientes:

Esquemas codificados	Definición
Box	El DCMI Box identifica una región del espacio usando sus límites geográficos.
DCMIType	Una lista de los tipos usados para categorizar la naturaleza o género del contenido de un recurso.
DDC	Clasificación Decimal de Dewey (*Dewey Decimal Classification*).
IMT	Los tipos de media de Internet del recurso.
ISO3166	ISO 3166 Códigos para la representación de nombres de países. http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html
ISO639-2	ISO 639-2: Códigos para la representación de nombres de idiomas. http://lcweb.loc.gov/standards/iso639-2/langhome.html
LCC	Clasificación de la Biblioteca del Congreso (*Library of Congress Classification*). http://lcweb.loc.gov/catdir/cpso/lcco/lcco.html
LCSH	Encabezamientos de materias de la Biblioteca del Congreso (*Library of Congress Subject Headings*). http://www.loc.gov/cds/lcsh.html
MeSH	Encabezamientos de materias médicos (*Medical Subject Headings*). http://www.nlm.nih.gov/mesh/meshhome.html
Period	Una especificación de los límites de un intervalo de tiempo. http://dublincore.org/documents/dcmi-period/
Point	El Punto DCMI identifica un punto en el espacio usando sus coordenadas geográficas. http://dublincore.org/documents/dcmi-point/
RFC1766	La Internet RFC 1766 '*Tags for the identification of Language*' o Etiquetas para la identificación de idioma, especifica un código de dos letras tomado de la ISO 639, seguido opcionalmente por otras dos letras del código del país tomado de la ISO 3166. http://www.ietf.org/rfc/rfc1766.txt
RFC3066	Internet RFC 3066 '*Tags for the Identification of Languages*' especifica una sub-etiqueta que tiene un código de dos letras tomados de la ISO 639 part 1 o un código de tres letras tomado de la ISO 639 part 2, seguido opcionalmente de un código de país de dos letras tomado de la ISO 3166. Cuando un idioma en la ISO 639 tiene dos letras en ambos y un código de tres letras, se usa el código de dos letras; cuando éste tiene solamente un código de tres letras, se usa el código de tres letras Esta RFC reemplaza a la RFC 1766. http://www.ietf.org/rfc/rfc3066.txt
TGN	Tesauro Getty de Nombres Geogoráficos (*The Getty Thesaurus of Geographic Names*). http://www.getty.edu/research/tools/vocabulary/tgn/index.html
UDC	Clasificación Decimal Universal (*Universal Decimal*

	Classification). http://www.udcc.org/
URI	Un URI o *Uniform Resource Identifier.* http://www.ietf.org/rfc/rfc2396.txt
W3CDTF	Reglas de codificación W3C para fechas y tiempos (W3C *Encoding rules for dates and times*) - un perfil basado en la ISO 8601. http://www.w3.org/TR/NOTE-datetime

Tabla 3 Esquemas codificados DC

En cuanto, al Tipo (*The DCMI Type Vocabulary*) http://dublincore.org/documents/dcmi-type-vocabulary/ , los términos empleados en DCMI son los siguientes:

Nombre del término	Definición
Collection	Una **colección** es una suma de ítems. El término colección significa que el recurso se describe como un grupo; sus partes pueden ser descritas y navegadas separadamente.
Dataset	A **conjunto de datos** es una información codificada en una estructura definida (por ejemplo, listas, tablas y bases de datos), con el fin de que puedan ser usadas de forma directa por un proceso mecánico.
Event	Un **evento** es un suceso no persistente basado en el tiempo. El metadato para un evento ofrece información descriptiva que es la base para descubrir el objeto, localización, duración, agentes responsables y enlaces a sucesos relacionados y recursos. El recurso del tipo de evento puede no ser recuperable si la instanciación descrita ha expirado o está todavía por ocurrir. Ejemplos - exhibición, lanzamiento de web, conferencia, taller, estreno, competición, boda, etc.
Image	Una **imagen** es una representación visual primariamente simbólica diferente a un texto. Por ejemplo -imágenes y fotografías de objetos físicos, pinturas, impresiones, dibujos, otras imágenes y gráficos, animaciones y dibujos en movimiento, películas, diagramas, mapas, notaciones musicales. Hay que hacer notar que una imagen puede incluir tanto representaciones electrónicas como físicas.
InteractiveResource	Un **recurso interactivo** es un recurso que requiere interacción del usuario para ser comprendido, ejecutado o experimentado. Por ejemplo - formularios en páginas web, *applets*, objetos de aprendizaje multimedia, servicios de chat, realidad virtual.
MovingImage	Una **imagen en movimiento**. Una serie de representaciones visuales que, cuando se muestran en sucesión, dan una impresión de movimiento. Ejemplos de imágenes en movimiento son: animaciones, películas, programas de televisión, vídeos, zootropos, u otras salidas visuales de una simulación.
PhysicalObject	Una cosa, **objeto** en tres dimensiones o materia. Por ejemplo - un ordenador, la gran pirámide, una escultura. Nótese que las representaciones digitales o los sustitutos de estas cosas usarían *Image*, *Text* u otro tipo.
Service	Un **servicio** es un sistema que ofrece una o más funciones de valor para el uso final. Ejemplos son: un servicio de copias de fotos, un servicio bancario, un servicio de autentificación, préstamo interbibiliotecario, un servidor Z39.50 o un servidor Web.
Software	**Software** es un programa de ordenador en forma de fuente o compilado que puede estar disponible para su instalación en otra máquina. Para *software* que existe solamente para crear un ambiente interactivo, se usa*interactive*.

Sound	Un **sonido** es un recurso cuyo contenido es primariamente entendido para ser reproducido como audio. Por ejemplo - un formato de archivo para reproducir música, un CD de audio, y conversaciones o sonidos grabados.
StillImage	Una **representación visual estática**. Ejemplo de imágenes estáticas son: pinturas, dibujos, diseños gráficos, planos y mapas.
Text	Un **texto** es un recurso cuyo contenido es primordialmente palabras para la lectura. Por ejemplo, -libros, cartas, dissertations, poemas, periódicos, artículos, archivos de listas de correo. Nótese que los facsímiles o imágenes de texto son *still* del género texto.

Tabla 4 Términos DCMI

1.2.2. DESCRIPCIÓN DE OWL

OWL es el acrónimo del inglés *Web Ontology Language* (Lenguaje de Ontologías Web), un lenguaje de marcado para publicar y compartir datos usando ontologías en la WWW. OWL tiene como objetivo facilitar un modelo de marcado construido sobre RDF y codificado en XML. Tiene como antecedente DAML+OIL, en los cuales se inspiraron los creadores de OWL para crear el lenguaje. Junto al entorno RDF y otros componentes, estas herramientas hacen posible el proyecto de web semántica.

El Lenguaje de Ontologías Web (OWL) está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente para representar información para los humanos. OWL facilita un mejor mecanismo de interpretabilidad de contenido Web que los mecanismos de XML, RDF, y esquema RDF (RDF-S) proporcionando vocabulario adicional junto con una semántica formal. OWL proporciona tres sublenguajes, cada uno con nivel de expresividad mayor que el anterior, diseñados para ser usados por comunidades específicas de desarrolladores y usuarios.

- OWL Lite está diseñado para aquellos usuarios que necesitan principalmente una clasificación jerárquica y restricciones simples. Por ejemplo, a la vez que admite restricciones de cardinalidad, sólo permite establecer valores cardinales de 0 ó 1. OWL Lite proporciona una ruta rápida de migración para tesauros y otras taxonomías. OWL Lite tiene también una menor complejidad formal que OWL DL.
- OWL DL está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (todos los cálculos se resolverán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo ciertas restricciones (por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra). OWL DL es denominado de esta forma debido a su correspondencia con la lógica de descripción (*Description Logics)*, en inglés.
- OWL Full está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha.

Cada uno de estos sublenguajes es una extensión de su predecesor más simple, y el siguiente grupo de relaciones se mantienen, pero las relaciones inversas no.

- Cada ontología legal de OWL Lite es una ontología legal de OWL DL
- Cada ontología legal de OWL DL es una ontología legal de OWL Full
- Cada conclusión válida de OWL Lite es una conclusión válida de OWL DL
- Cada conclusión válida de OWL DL es una conclusión válida de OWL Full

OWL Full puede ser considerada como una extensión de RDF, mientras que OWL Lite y OWL DL pueden ser consideradas como extensiones de una visión restringida de RDF. Cada documento OWL (Lite, DL, Full) es un documento RDF, y cada documento RDF es un documento de OWL Full, pero sólo algunos documentos RDF serán legalmente documentos OWL Lite o OWL DL. Por este motivo, se ha de tener cuidado cuando un usuario quiera migrar un documento de RDF a OWL. Cuando se considere que la expresividad de OWL DL o OWL Lite es adecuada, han de tomarse algunas precauciones para asegurar que el documento RDF original cumple con las restricciones adicionales impuestas por OWL DL y OWL Lite. Entre otras: cuando una URI es utilizada como nombre de una clase, debe indicarse explícitamente que ésta URI debe ser una clase del tipo owl:Class (al igual que para las propiedades), cada individuo debe estar definido como perteneciente al menos a una clase (incluso solo con objetos owl:Thing) y las URI usadas para las clases, propiedades e individuos deben ser disjuntos entre ellos.

Se incluyen las siguientes características de OWL Lite relacionadas con el esquema RDF.

- Class: Una clase define un grupo de individuos que pertenecen a la misma porque comparten algunas propiedades. Por ejemplo, “Deborah y Frank son miembros de la clase Persona”. Las clases pueden organizarse en una jerarquía de especialización usando subClassOf. Se puede encontrar una clase general llamada Thing que es una clase de todos los individuos y es una superclase de todas las clases de OWL. También se puede encontrar una clase general llamada Nothing que es la clase que no tiene instancias y es una subclase de todas las clases de OWL.
- rdfs:subClassOf: Las jerarquías de clase deben crearse haciendo una o más indicaciones de que una clase es subclase de otra. Por ejemplo, “la clase Persona podría estar definida como subclase de la clase Mamífero”. De esto podemos deducir que “si un individuo es una Persona, entonces, también es un Mamífero”.
- rdf:Property: las propiedades pueden utilizarse para establecer relaciones entre individuos o de individuos a valores de datos. Ejemplos de propiedades son tieneHijo, tieneFamiliar, tieneHermano, y tieneEdad. Los tres primeros pueden utilizarse para relacionar una instancia de la clase Persona con otra instancia de la clase Persona (siendo casos de ObjectProperty), y el último (tieneEdad) puede ser usado para relacionar una instancia de la clase Persona con una instancia del tipo de datos Entero (siendo un caso de DatatypeProperty). Ambas, owl:ObjectProperty y owl:DatatypeProperty, son subclases de la clase de RDF rdf:Property.
- rdfs:subPropertyOf: Las jerarquías de propiedades pueden crearse haciendo una o más indicaciones de que una propiedad es a su vez subpropiedad de una o más propiedades. Por ejemplo, tieneHermano puede ser una subpropiedad de tieneFamiliar. De esta forma, un razonador puede deducir que si un individuo está

relacionado con otro por la propiedad tieneHermano, entonces está también relacionado con ese otro por la propiedad tieneFamiliar.

- rdfs:domain: Un dominio de propiedad reduce los individuos a los que puede aplicarse la propiedad. Si una propiedad relaciona un individuo con otro individuo, y la propiedad tiene una clase como uno de sus dominios, entonces el individuo debe pertenecer a esa clase. Por ejemplo, puede establecerse que la propiedad tieneHijo tenga como dominio la clase Mamifero. De esto, un razonador puede deducir que si "Frank tieneHijo Anna", entonces Frank debe ser un Mamífero. Obsérvese que rdfs:domain se denomina restricción global debido a que la restricción se refiere a la propiedad y no sólo a la propiedad cuando está asociada con una clase en concreto.
- rdfs:range: El rango de una propiedad reduce los individuos que una propiedad puede tener como su valor. Si una propiedad relaciona a un individuo con otro individuo, y ésta como rango a una clase, entonces el otro individuo debe pertenecer a dicha clase. Por ejemplo, la propiedad tieneHija debe establecerse que tiene como rango la clase Mamífero. A partir de aquí, un razonador puede deducir que "si Louise está relacionada con Deborah mediante la propiedad tieneHija, (por ejemplo, Deborah es hija de Louise), entonces Deborah es un Mamifero". El rango es, al igual que el dominio, una restricción global.
- Individual: Los individuos son instancias de clases, y las propiedades pueden ser usadas para relacionar un individuo con otro. Por ejemplo, un individuo llamado Deborah puede ser descrito como una instancia de la clase Persona y la propiedad tieneEstudiante puede ser usada para relacionar el individuo Deborah con el individuo UniversidadDeStanford.

1.3 TRABAJOS RELACIONADOS

En la bibliografía relacionada con las ontologías se pueden encontrar diferentes aproximaciones y experimentaciones relacionadas de alguna manera, con el propósito y objetivos de este trabajo. Uno de los trabajos relacionados con la identificación de patrones léxicos que representan relaciones semánticas entre conceptos es el [12] titulado *Automatic extraction of semantic relationships for Wordnet by means of pattern learning from Wikipedia* de Maria Ruiz-Casado, Enrique Alfonseca y Pablo Castells (2005).

La Web Semántica (SW) constituye una iniciativa para extender la web con contenidos de lectura mecánica y servicios automatizados mucho más allá de los de las capacidades actuales. Una práctica común es la anotación de contenidos de las páginas web usando una ontología. En la mayoría de los casos, las ontologías están estructuradas como herencias de conceptos con la relación entre significados llamadas hiponimia (es-un, una clase de inclusión o subsumción) y su inversa, hiperonimia, que organiza los conceptos desde lo más general a lo más específico. Adicionalmente, puede haber otras relaciones, como la meronimia (la relación parte-todo) y su inversa, holonimia; telicidad (propuesta), o algún otro que puede ser de interés, como *es-autor-de, es-la-capital-de, es-empleado-de,* etc. En muchos casos, las ontologías distinguen nodos que representan conceptos (clases de cosas, como por ejemplo *persona*) de nodos que representan instancias (por ejemplo *Juan*).

Al igual que la propia web, en ocasiones las ontologías tienen que incluir una gran cantidad de información, o se someten a una rápida evolución. Por lo tanto, es altamente deseable automatizar o semi-automatizar la adquisición de las ontologías.

Respecto a la extracción automática de información de un corpus textual y el enriquecimiento de ontologías automático, se pueden clasificar los enfoques en los siguientes grupos:

- Sistemas basados en las propiedades de distribución de palabras: consiste en estudiar distribuciones coocurrentes de términos para calcular una distancia semántica entre los conceptos representados por estos términos.
- Los sistemas basados en patrones de extracción y coincidencias: estos se basan en patrones léxicos o léxico-semánticos para descubrir relaciones ontológicas y no-taxonómicas entre conceptos en un texto no restringido. Manualmente se definen expresiones regulares para extraer hipónimos y relaciones de parte-de o hiperonimia.
- Los sistemas basados en el análisis de las definiciones de diccionario, se aprovechan de la estructura particular de los diccionarios con el fin de extraer las relaciones de hiperonimia con las que organizan los conceptos en una ontología. También hay varias obras que extraen relaciones adicionales de las glosas de WordNet, por desambiguación de las palabras en las glosas.

En este trabajo se describe un método automático para identificar patrones léxicos que representan relaciones semánticas entre conceptos, desde una enciclopedia online. A continuación, estos patrones pueden ser aplicados para extender ontologías existentes o redes semánticas con nuevas relaciones. Los experimentos se basan en la Wikipedia Simple en Inglés y con WordNet 1.7.

El procedimiento seguido consiste en rastrear la versión de Inglés Simple de la Wikipedia, coleccionando todas las entradas, desambiguándolas y asociando cada una con relaciones. Estos pasos se han llevado a cabo de la siguiente forma:

1. Desambiguación del sentido de la entrada: Este paso consiste en el preprocesamiento de las definiciones de la Wikipedia y asociando cada entrada de la wikipedia con su correspondiente *synset* de WordNet, así el sentido de la entrada es determinado explícitamente.

2. Extracción de patrones: Para cada entrada, la definición es procesada buscando palabras que están conectadas con la entrada de la wikipedia con un hipervínculo. Si hay una relación en WordNet entre la entrada y una de estas palabras, el contexto es analizado y se extrae un patrón para la relación.

3. La generalización de patrón. El procedimiento usado para obtener una métrica de similitud entre dos patrones, consiste en una versión modificada ligeramente del algoritmo dinámico programado para el cálculo *edición-distancia*. La *distancia de edición* entre dos cadenas *A* y *B* es definida como el mínimo número de cambios (inserción de carácter, adición o reemplazamiento) que tiene que ser hecho por la primera cadena en orden de obtener la segunda.

4. Identificación de nuevas relaciones: los patrones son aplicados para descubrir nuevas relaciones distintas a las presentes ya en WordNet.

Algunas de las conclusiones que obtuvieron de este trabajo fueron las siguientes:

- Se describió un nuevo algoritmo de generalización de patrones léxicos, implementados y evaluados. Se basa en el algoritmo de la distancia de edición, que fue modificado para tener en una cuenta las etiquetas gramaticales de las palabras. Este algoritmo es totalmente automático ya que no requiere supervisión humana.
- El conjunto de patrones que fueron encontrados automáticamente de las entradas de la Wikipedia, permitían extraer nuevas relaciones de textos para cada una de las cuatro relaciones: hiperonimia, hiponimia, meronimia y holonimia. Se encontraron más de 1200 nuevas relaciones.
- La precisión de los patrones generados fue similar a la de los patrones escritos a mano. El tipo de patrones de hiponimias lexicosintácticos fueron evaluados, en diferentes contextos, reportando una precisión de 0,65 y 0,39, respectivamente.

Este trabajo planteó las siguientes líneas de investigación: (a) extraer tipos de relaciones, como *lugar, instrumento, télico o autor*; (b) generalizar el experimento a otras ontologías y enciclopedias e incluso aplicarlo a textos no restringidos; y (c) extender el formalismo usado para representar patrones, y codificar características sintácticas.

No se continuó estudiando formas de expandir ontologías o taxonomías pues tras el estudio de algunos trabajos relacionados con la recuperación de información basada en facetas, se decidió hacer otro tipo de experimentación, como se planteará al final de la siguiente sección.

2. RECUPERACIÓN DE INFORMACIÓN

Los avances tecnológicos de los últimos cincuenta años han provocado un aumento exponencial de la información y una mejora de su difusión. Hoy nos hallamos inmersos en la revolución de la información, cada vez tenemos más información disponible y mayores posibilidades para accederla. El proceso de digitalización de los documentos así como el desarrollo de nuevas tecnologías de la información tanto en su creación, como en su distribución, como en su acceso, son dos claros ejemplos de la evolución de la información, lo cual ha permitido su acceso y uso por un número ilimitado de usuarios.

Además, hay que tener en cuenta que el uso masivo de las tecnologías y de los ordenadores no se reduce a la producción editorial, sino que está presente en todos los ámbitos de la vida, sobre todo en el trabajo, y hasta en el hogar donde cada vez es mayor el número de personas que no sólo tienen ordenador sino que poseen equipos multimedia. A ello habría que sumar la distribución de información mediante las llamadas "autopistas de la información", la proliferación de las conexiones de banda ancha y el coste cada vez menor de los medios de almacenamiento. Todo ello nos sitúa dentro de un entorno en desarrollo de información electrónica a la que se puede acceder por medios automáticos.

Otro aspecto que tenemos que considerar es la diversificación de los medios, que trae consigo una mayor cantidad de información no normalizada, imagen, sonido, texto, etc, así como las redes sociales.

Por lo tanto el problema se encuentra en el almacenamiento y recuperación de la información, pues ésta, aparte de lo vasta que es, se encuentra en formatos de muy diferentes características.

Una solución fue la aparición de los buscadores, que sin duda son de gran ayuda para encontrar información requerida por el usuario, pero no es muy precisa, pues la búsqueda se realiza, principalmente, sobre la base de la localización de palabras claves, y no diferencian, por ejemplo, entre páginas personales, académicas, comerciales, etc. Además se recupera, en gran porcentaje, información que no es útil porque no corresponde a lo que estamos buscando. Todo esto es debido principalmente a que los buscadores actuales no están diseñados para "comprender".

Aparece una nueva parte de la Informática que se encarga de la recuperación de la información útil para los usuarios. Existen muchas definiciones al respecto, de las cuales citamos algunas:

- Baeza – Yates (1999): Parte de la informática que estudia la recuperación de la información (no datos) de una colección de documentos escritos. Los documentos recuperados pueden satisfacer una necesidad de información de un usuario expresada normalmente en lenguaje natural.
- Korfhage (1997): La localización y presentación a un usuario de información relevante a una necesidad de información expresada como una pregunta.
- Salton (1989): Un sistema de recuperación de información procesa archivos de registros y peticiones de información, e identifica y recupera de los archivos ciertos registros en respuesta a las peticiones de información.

De éstas podemos resumir que "Es el arte y/o ciencia que se encarga de la búsqueda y presentación de información relevante, de grandes colecciones de documentos, a partir de un usuario que hace una petición de información, normalmente en lenguaje natural".

Desde un punto de vista más procedimental, la Recuperación de Información (RI) se puede definir también como el problema de la selección de información, depositada en un medio de almacenamiento, en respuesta a consultas realizadas por un usuario.

Un Sistema de Recuperación de Información (SRI) debe soportar una serie de operaciones básicas sobre los documentos almacenados, como son: introducción de nuevos documentos, modificación de los que ya estén almacenados y eliminación de los mismos. Tiene que contar con algún método de localización de los documentos (o con varios, generalmente) para presentárselos posteriormente al usuario.

Los SRI implementan estas operaciones de varias formas distintas, lo que provoca una amplia diversidad en los mismos. Por tanto, para estudiarlos es necesario establecer en primer lugar una clasificación de estos sistemas.

En esta sección haremos una introducción a varios de los modelos existentes y analizaremos las componentes que los forman. Los principales modelos clásicos de recuperación de información son: el modelo Booleano, el modelo Espacio Vectorial, el modelo Probabilístico y el modelo Booleano extendido o modelo Difuso.

- **Modelo Booleano**: Este modelo se basa en la teoría del álgebra de Boole. El Algebra Booleana está formada por las reglas algebraicas, basadas en la teoría de conjuntos, para manejar ecuaciones de lógica matemática. La lógica matemática trata con proposiciones, asociadas por medio de operadores como AND, OR, NOT, IF...THEN.
- **Modelo Espacio Vectorial**: fue el primero en proponerse, a finales de los 60, dentro del marco del proyecto SMART. Partiendo de que se pueden representar los documentos como vectores de términos, los documentos podrán situarse en un espacio vectorial de n dimensiones, es decir, con tantas dimensiones como términos tenga el documento. Situado en ese espacio vectorial, cada documento cae entonces en un lugar determinado por sus coordenadas, al igual que en un espacio de tres dimensiones cada objeto queda bien ubicado si se especifican sus tres coordenadas espaciales. Se crean así grupos de documentos que quedan próximos entre sí a causa de las características de sus vectores. Estos grupos o *clusters* están formados, en teoría, por documentos similares, es decir, por grupos de documentos que serían relevantes para la misma clase de necesidades de información. En una base de datos documental organizada de esta manera, resulta muy rápido calcular la relevancia de un documento a una pregunta y es muy rápida también la ordenación por relevancia, ya que, de forma natural, los documentos ya están agrupados por su grado de semejanza. En la fase de la consulta, cuando se formula una pregunta, también se la representa en este espacio vectorial y, así, aquellos documentos que queden más próximos a ella serán, en teoría, los más relevantes para la misma. La representación de los documentos y las consultas se realiza mediante la asociación de un vector de pesos no binarios (un peso por cada término de índice). Por ejemplo, di = (ti1; ti2; ti3; …; tin).

- **Modelo Probabilístico**: El marco del modelo probabilístico está compuesto por conjuntos de variables, operaciones con probabilidades y el teorema de Bayes. Todos los modelos de recuperación probabilísticos están basados en el "Principio de la ordenación por probabilidad", conocido originalmente como "*the probability ranking principle*". Este principio, formulado por Robertson, asegura que el rendimiento óptimo de la recuperación se consigue ordenando los documentos según sus probabilidades de ser relevantes con respecto a una consulta, siendo estas probabilidades calculadas de la forma más precisa posible a partir de la información disponible.
- **Modelo Booleano Extendido**: Cualquier SRI debería ser capaz de tratar con dos características inherentes al pro-ceso de RI: la imprecisión y la subjetividad. Estos dos factores juegan un papel fundamental:
    - en la formulación de las necesidades de información,
    - en la estimación del grado en que cada ítem de información es relevante para las necesidades del usuario, y
    - en la decisión de qué ítems de información deben recuperarse en función a una petición determinada.

  Los SRI Booleanos presentan los siguientes problemas:

    - Uno de sus mayores inconvenientes es la indización de los documentos. Un término puede aparecer en un documento y ser más significativo en éste que en cualquier otro. Sin embargo, no existen mecanismos para representar esta distinción en el modelo Booleano. Este inconveniente afecta directamente al módulo indizador de la base documental.
    - Otra fuente de imprecisión que caracteriza a la RI es el conocimiento vago que el usuario tiene sobre el tema sobre el que está preguntando. Si el usuario es un entendido, le gustaría tener la habilidad de expresar en su consulta la importancia o relevancia que tienen unos términos sobre otros. La incapacidad de realizar esta tarea viene a ser una carencia muy representativa del subsistema de consulta de los SRI Booleanos.
    - Por último, la recuperación es tajante: 1 si el documento es relevante y 0 si no lo es, sin permitir que exista una gradación en la recuperación que maneje mejor la incertidumbre. Este problema se centra en el mecanismo de evaluación.

Sin embargo, a pesar de las carencias anteriores, el modelo Booleano sigue estando muy extendido en el ámbito comercial. Por esta razón, se han llevado a cabo varias extensiones sobre el mismo que permiten salvar algunas de las limitaciones que presenta sin proceder a su completa redefinición. La teoría de conjuntos difusos se ha empleado como herramienta para tal propósito, especialmente por su habilidad para tratar con la imprecisión y la incertidumbre en el proceso de RI. Este hecho se debe fundamentalmente a que:

- es un marco formal diseñado para tratar con imprecisión y vaguedad, y

- facilita la definición de una superestructura del modelo Booleano, de forma que los SRI basados en este modelo pueden modificarse sin tener que ser completamente rediseñados.

A continuación, se describen algunas de las tecnologías relacionadas con los SRI que se van a utilizar en este trabajo.

2.1. HERRAMIENTAS

Apache Solr es una plataforma de búsqueda basada en Apache Lucene, así que, antes de explicar en qué consiste Apache Solr, se presenta brevemente en qué consiste Lucene.

2.1.1. LUCENE

Lucene (http://lucene.apache.org/) es una librería para la recuperación de información escrita en Java que permite añadir una funcionalidad de búsqueda a cualquier aplicación. En los últimos años, se ha convertido en la librería de referencia para implementar herramientas de recuperación de información, llegando a ser casi un estándar (5).

Lucene puede indexar y permitir realizar búsquedas sobre cualquier conjunto de datos que se puedan extraer de un texto, independientemente de cuál sea la fuente, el formato o el idioma. Esto permite que se puedan indexar y buscar datos almacenados en ficheros conteniendo a páginas web o en servidores remotos, documentos almacenados en sistemas de archivos locales, ficheros de texto simples, documentos de Word, XML, HTML, archivos PDF o cualquier otro formato del que se pueda extraer información textual.

Es importante destacar que Lucene no es una herramienta de búsqueda por sí misma, sino una librería que proporciona funcionalidades de recuperación de información que se pueden añadir a cualquier aplicación. Esto queda claro en la Ilustración 5, en la que se pueden ver todos los componentes que forman típicamente una aplicación de búsqueda, y además se puede distinguir sombreados los que implementa Lucene.

La búsqueda es un proceso que consiste en encontrar términos en un índice para acceder a los documentos en los que aparecen. Tanto la recolección de datos como la generación de documentos resultado, son tareas específicas de cada aplicación y que por lo tanto Lucene no implementa. Además, tampoco proporciona ningún tipo de interfaz, por lo que cada aplicación deberá generar las suyas propias. A continuación se detallan algunos de los módulos que Lucene aporta:

- **Análisis de documentos** (*Analyze document*): este paso consiste en convertir los campos de texto en los elementos que se van a indexar, los términos. El proceso de análisis comienza con la tokenización del texto, es decir, con la extracción de cadenas o palabras que lo componen, y que se conocen como tokens. Para generar los tokens, se aplican al texto diferentes procesos, como la extracción de palabras, la eliminación de los signos de puntuación, la normalización (paso a minúsculas), stemming (reducción de las palabras a su forma raíz) o la lemmatizacion (la transformación de las palabras a su forma básica). Los tokens, combinados con el nombre de su campo asociado, forman los términos.

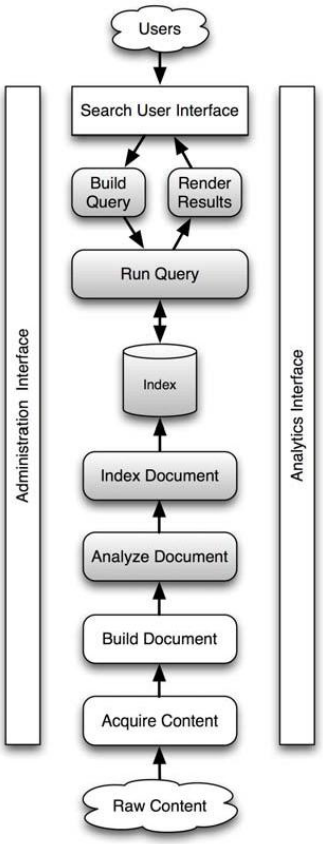

Ilustración 5 Componentes de Lucene

Lucene proporciona diferentes analizadores que permiten controlar este proceso, y también ofrece la posibilidad de personalizar los existentes o crear uno nuevo combinando sus funcionalidades para tokenizar y filtrar los tokens, de manera que el proceso de creación de tokens puede ser personalizado.

- **Indexado de documentos** (*Index document*): en este paso, se añaden los documentos al índice. Lucene almacena los documentos en una estructura de índice invertido, lo que hace más eficiente el almacenamiento del índice en el disco y también más rápidas las búsquedas.
- **Construcción de la consulta** (*Build Query*): cuando un usuario utiliza una aplicación de búsqueda, generalmente lo hace a través de un formulario web en el que introduce una consulta que el navegador envía al servidor web que contiene el motor de búsqueda. Una vez recibida, se debe transformar en una Query del motor de búsqueda, y este proceso es el que engloba el módulo de "Construcción de la consulta".
- **Ejecución de la consulta** (*Run Query*): es el proceso de consultar el índice de búsqueda y recuperar los documentos que concuerdan con la Query. Lucene encapsula este paso, proporcionando una API para realizarlo de forma sencilla, aunque también permite personalizarlo de forma que se pueda modificar el filtrado, el ordenado, etc.

La aproximación de Lucene combina los modelos de espacio vectorial y booleano, y ofrece la posibilidad de decidir cuál de ellos usar a la hora de realizar una consulta. La fase de ejecución

de la consulta termina cuando Lucene devuelve una lista de documentos que se debe transformar para que el usuario del buscador pueda comprenderla y utilizarla.

- **Interpretación de resultados (*Render Results*):** consiste en generar una lista de resultados que el usuario pueda entender y utilizar de forma sencilla. Esta lista se genera a partir de la lista interna de identificadores de documentos ordenados que concuerdan con una consulta.

Lucene permite personalizar la manera de valorar los resultados que nos mostrará al realizar las búsquedas:

- Lucene permite al usuario personalizar su fórmula de puntuación mediante la ampliación de *org.apache.lucene.search.Similarity*. La fórmula que sigue es la siguiente:

score(q,d) = coord(q,d) · queryNorm(q) · ∑t in q (tf(t in d) · idf(t)2 · t.getBoost() · norm(t,d))

Factor	Descripción
tf(t ind)	Factor de frecuencia del término para el término (t) en el documento (d)
idf(t)	Inverso de la frecuencia del término (t)
coord(q,d)	Factor de puntuación basada en cuántos de los términos de consulta se encuentran en el documento especificado.
queryNorm(q)	Factor de normalización utilizado para hacer las puntuaciones entre las consultas comparables.
t.getBoost()	Campo boost
norm(t,d)	Encapsula algunos impulsos y factores de longitud en el tiempo de indexado.

Tabla 5 Explicación de los términos de la ecuación Similarity de Lucene

Para la indexación y recuperación del contenido textual de los documentos que gestiona bastaría con utilizar alguno de los analizadores que proporciona por defecto Lucene, pero si se quieren potenciar las búsquedas de modo que no se produzca demasiado ruido en el resultado y encontrar documentos similares, hay que conseguir que los documentos estén lo más normalizados posibles.

Se puede normalizar los documentos utilizando alguna de las librerías que facilitan a los siguientes procesos:

- **Eliminación de stopwords**: son una lista de palabras de uso frecuente que, tanto en la indexación como en la búsqueda, no se tienen en consideración, se omiten.
- **Stemming**: es un método para obtener una cadena o raíz no necesariamente léxica de una palabra. Las palabras se reducen a su raíz lingüística o a su stem, de modo que, si buscamos por “abandonados” encontrará “abandonados” pero también “abandonadas”, “abandonamos”, ... porque, en realidad, estamos buscando por “abandon”. El proceso de stemming es muy útil para el inglés, aunque para el castellano tiene detractores, porque el stem de una palabra en castellano no tiene por qué coincidir con su raíz léxica.

Lucene provee varios analizadores por defecto, el *StandardAnalyzer* con un listado reducido de stopwords en inglés y podemos obtener una librería (*lucene-analyzers*) con analizadores en bastantes idiomas... casi todos menos en Castellano. Por eso hay que utilizar listas específicas u otras librerías, acudiendo a páginas especializadas en IR o Text Mining (http://snowball.tartarus.org/ , http://www.unine.ch/info/clef/).

Se detalla más información en el Anexo 2.

2.1.2. APACHE SOLR

Apache Solr es una plataforma de búsquedas basada en Apache Lucene, que funciona como un "servidor de búsquedas". Sus principales características incluyen búsquedas de texto completo, clustering dinámico, y manejo de documentos enriquecidos (como Word y PDF). Solr es escalable, permitiendo realizar búsquedas distribuidas y replicación de índices, y actualmente se está usando en muchos de los sitios con más volumen de Internet.

Una característica de Solr (al menos la más útil) es su API estilo REST, ya que en vez de usar drivers o APIs programáticas para comunicarse con Solr se pueden hacer peticiones HTTP y obtener resultados en XML o JSON. (por cierto, se pronuncia "Solar" y no es un acrónimo). Solr no expone una interfaz REST "perfecta" (que use todos los principios de HTTP 1.1), pero los datos tienen una representación simple que viaja entre el cliente y el servidor, sin ninguna encapsulación rara con SOAP u otros. Además, los XML son legibles por personas, y JSON se puede usar con JavaScript y realizar pruebas.

Las ventajas de usar esta interfaz universal (y no propia de un lenguaje) son varias:

- es agnóstico del lenguaje porque usa XML y JSON, que hoy en día pueden ser interpretados por casi cualquiera. La métrica generalmente es JavaScript: si se interpreta con JavaScript dentro de todas las limitaciones que impone un navegador, entonces lo podemos interpretar en cualquier otro lugar. Por supuesto, JavaScript tiene soporte nativo para JSON y XML.
- es agnóstico de los tipos de datos, ya que HTTP sólo transmite textos. Los lenguajes dinámicos como PHP tienen éxito porque su protocolo básico no tiene tipos estrictos. Si la presentación lo va a mostrar por pantalla, ¿por qué no debería alcanzar con un string?
- más o menos es un protocolo estándar (aunque la representación de datos no lo sea): si alguien inventa una base de datos y publica su propio protocolo binario de comunicación, habría muchas más librerías.

Solr está escrito en Java, pero se puede usar con otro lenguaje, simplemente usando las peticiones GET para realizar búsquedas en el índice, y POST para agregar documentos.

Solr se divide en dos partes:

- **Índice:** Sistema de ficheros que almacenan la información. Contiene la configuración de Solr y la definición de la estructura de datos.
- **Servidor:** Proporciona el acceso a los índices y las características adicionales. Admite plugins para añadir funcionalidades.

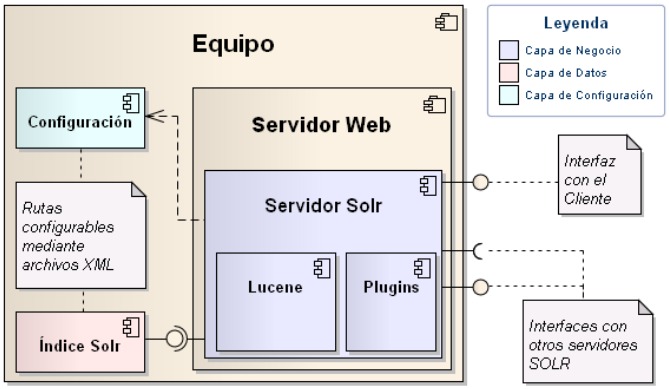

Ilustración 6 Partes en las que se divide de Solr

Solr permite búsquedas distribuidas: Uno de los servidores actúa como maestro, consultando al resto y componiendo la respuesta.

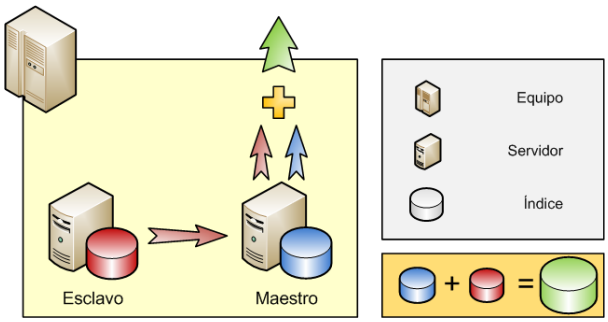

Ilustración 7 Esquema de servidores maestro-esclavo en Solr

2.1.3. SPARQL

Es un lenguaje de consulta para RDF, y es una recomendación del W3C para crear un lenguaje de consulta dentro de la Web semántica que está ya implementada en varios lenguajes y bases de datos. Desde 2005 está en proceso de estandarización por el RDF Data Access Working Group (DAWG) del W3C; y en junio del 2007 se anunció el paso de su especificación a *Candidate Recomendation*.

Actualmente RDF se usa para representar información personal, redes sociales, metadatos de recursos web como video, sonido e imágenes, así como para proporcionar un medio de integración entre fuentes de información heterogéneas, siendo su objetivo principal, facilitar la identificación entre recursos web. Entonces con un lenguaje de consulta RDF, como SPARQL, los desarrolladores y usuarios finales se encontrarán con una gran cantidad de información (recursos identificables) teniendo la facilidad para representar y utilizar los resultados obtenidos de manera más eficiente, sustentándose así su utilidad en la necesidad de recuperar y organizar la información de diferentes fuentes.

SPARQL tiene especificaciones que explican diferentes partes de su funcionalidad; en general, consiste en un lenguaje de consulta, un formato para las respuestas, y un medio para el transporte de consultas y respuestas:

- SPARQL Query Language: Componente principal de SPARQL. Explica la sintaxis para la composición de sentencias y su concordancia.

- SPARQL Protocol for RDF: Formato utilizado para la devolución de los resultados de las búsquedas (queries SELECT o ASK), a partir de un esquema XML.
- SPARQL Query Results XML Format: Describe el acceso remoto de datos y la transmisión de consultas de los clientes a los procesadores. Utiliza WSDL para definir protocolos remotos

El lenguaje de consulta SPARQL está basado en comparación de patrones gráficos. Los patrones gráficos contienen patrones triples. Los patrones triples son como las tripletas RDF, pero con la opción de una variable consulta en lugar de un término RDF en las posiciones del sujeto, predicado u objeto. Combinando los patrones triples obtenemos un patrón grafico básico, donde es necesaria una comparación exacta entre gráficos.

La sintaxis básica de una consulta SPARQL se muestra en la Tabla siguiente.

Prologue (optional)	BASE <iri> PREFIX prefix: <iri> (repeatable)
Query Result forms (required, pick 1)	SELECT (DISTINCT)sequence of ?variable SELECT (DISTINCT)* DESCRIBE sequence of ?variable or <iri> DESCRIBE * CONSTRUCT { graph pattern } ASK
Query Dataset Sources (optional)	Add triples to the background graph (repeatable): FROM <iri> Add a named graph (repeatable): FROM NAMED <iri>
Graph Pattern (optional, required for ASK)	WHERE { graph pattern z}
Query Results Ordering (optional)	ORDER BY ...
Query Results Selection (optional)	LIMIT n, OFFSET m

Tabla 6 Sintaxis básica de una consulta SPARQL

La función de la palabra clave PREFIX es equivalente a la declaración de namespaces en XML, es decir asocia una URI a una etiqueta, que se usará mas adelante para describir el namespace. Pueden incluirse varias de estas etiquetas en una misma consulta.

Todo comienzo de una consulta SPARQL queda marcada por la palabra clave SELECT, similar a su uso en SQL1, sirve para definir los datos que deben ser devueltos en la respuesta. La palabra clave FROM identifica los datos sobre los que se ejecutará la consulta, es necesario indicar que una consulta puede incluir varios FROM. La palabra clave WHERE indica el patrón sobre el que se filtrarán los tripletes del RDF.

SPARQL es un lenguaje de consulta capaz de obtener información desde grafos RDF.

Es la propuesta de estándar del W3C. Al igual que SeRQL, es soportado por Sesame y Jena. Proporciona facilidades para:

- Extraer información en forma de URIs, nodos blancos y literales.

[1] SQL: Structured Query Language

- Extraer subgrafos RDF.
- Construir nuevos grafos RDF basados en los grafos incluidos en la consulta.

Al igual que los lenguajes de acceso a datos, SPARQL es adecuado para el uso en local y remoto. SPARQL se basa en patrones de grafos. Los patrones más simples son las tripletas, los cuales son como las tripletas RDF pero con la posibilidad de tener una variable en el sujeto, predicado u objeto.

Veremos a continuación un ejemplo en el que se muestra una consulta SPARQL, la cual encontrará el título de un libro desde la información dada por un grafo RDF. La consulta consta de dos partes, la cláusula SELECT y la cláusula WHERE. La primera identifica las variables que aparecerán en el resultado de la consulta, y la cláusula WHERE tiene un patrón triple:

Sentencia RDF (en notación N3):
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial"
Consulta:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Resultado:
title
“SPARQL Tutorial”

Las variables en las consultas SPARQL tienen un alcance global. Las variables se indican por “?”. Hay que tener en cuenta que “?” no forma parte de la variable. “$” es una alternativa a “?”. En una consulta que incluya $abc o ¿abc, se trata de la misma variable.

En SPARQL se pueden hacer sintaxis distintas, pero con el mismo resultado. Los tres ejemplos siguientes expresan la misma consulta:

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <http://example.org/book/book1> dc:title ?title }

PREFIX  dc: <http://purl.org/dc/elements/1.1/>
PREFIX  : <http://example.org/book/>
SELECT  $title
WHERE   { :book1  dc:title  $title }

BASE    <http://example.org/book/>
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <book1>  dc:title  ?title }
```

2.2. TRABAJOS RELACIONADOS

En el estado del arte se pueden encontrar aproximaciones y experimentaciones relacionadas con este trabajo. Uno de los trabajos relacionados de forma directa con la búsqueda semántica es el [13]: *Semantically enhanced Information Retrieval: an ontology-based approach* de Miriam Fernández Sánchez y Pablo Castells.

La perspectiva de la recuperación de información en la búsqueda semántica a principios de los 80 comenzó con la elaboración mapas conceptuales y la introducción modelos de recuperación de información:

- Taxonomías (categorías + relaciones de herencia).
- Diccionarios (categorías + relaciones de herencia y asociativas), como WordNet.
- Métodos algebraicos como el *Latent Semantic Analysis*.

En cuanto a la perspectiva de tecnologías basadas en la semántica, más tarde, en los 90, se introdujeron las ontologías de marcos conceptuales (clases + instancias (KBs) + relaciones semánticas arbitrarias + reglas). Pero esto también tenía limitaciones: la búsqueda semántica era entendida como una tarea de recuperación de datos; a veces esto hacía parcial el uso de un poder expresivo de una representación basada en ontologías.

En realidad ¿qué es la búsqueda semántica? Podríamos decir que la búsqueda semántica es el aumento de la representación de los significados de los distintos contenidos a nivel superior de palabras clave, con el fin de mejorar la recuperación de la información actual.

Como las descripciones de contenido actuales y las técnicas de procesamiento de las consultas en la actualidad, están basadas en palabras clave el problema que los autores querían resolver se refería a la capacidad limitada para comprender y explotar las distintas conceptualizaciones que se encuentran en las expresiones de necesidad de información de cada usuario. Esto es:

- Las relaciones entre los términos de búsqueda: “Libros sobre recomendar sistemas” vs “sistemas que recomiendan libros”
- La polisemia: “jaguar” como animal o “jaguar” como coche.
- La sinonimia: “taza” vs “jícara”
- Conclusiones: deportes acuáticos en el Mediterráneo -> windsurf, buceo, etc. en Valencia, Alicante, etc.

La mejor solución al problema de la mala interpretación de las búsquedas basadas en las palabras clave, es la búsqueda semántica. Si las máquinas entendieran el significado de lo que el usuario realmente desea buscar, recibiríamos una información más acertada de lo que necesitamos. Para poder realizar búsquedas actuales en Internet, se debe tener un conocimiento bastante amplio de cómo funcionan los buscadores para poder encontrar la información que necesitamos, ya que, si formulamos la consulta mal, no recibiremos los resultados que queremos.

Luego el principal objetivo es el de reducir la distancia entre la formulación de los sistemas de recuperación de información y la real en la mente del usuario, tanto con respecto a las consultas, como a la comprensión de los distintos significados del contenido de los

documentos a un nivel superior de las palabras clave. En la imagen se observa el diseño de un buscador semántico.

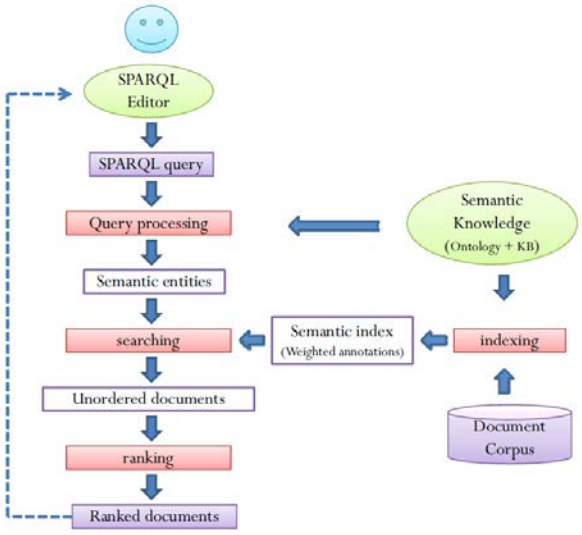

Ilustración 8 Diseño de un buscador semántico

Una adaptación del índice invertido en la búsqueda semántica es asociar conceptos a los documentos, en lugar de palabras clave; y procesar los pesos de utilizando una adaptación del algoritmo TF-IDF.

En el artículo citado se trabajó con:

- Colección de documentos: nuevos artículos de la web de CNN
  - 145,316 documentos (445 MB) de la CNN (NuevoArtículo -> DocumentoTexto)
- Ontología del dominio y KB: KIM con menos extensiones y ajustes
  - 281 clases de dominio, 138 propiedades en muchos dominios.
  - 35,689 instancias, 465,848 frases, (81MB en formato RDF)
- Consultas
  - Un conjunto de veinte consultas que fueron preparadas manualmente. $w_v = 1$ para todas las *v* en las consultas SPARQL.
- Juicios
  - Juicio manual de los documentos de 0 a 5.

Y a continuación se adaptó el modelo de espacio vectorial de recuperación de información como sigue:

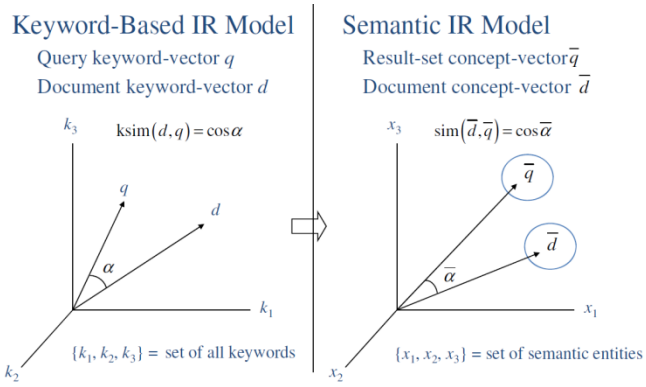

Ilustración 9 Modelo espacio vectorial de recuperación de información

Para crear el vector de consulta $\overline{q}_x$:

- Ejecutamos la consulta (por ejemplo SPARQL) -> Y el conjunto de resultados $R \subset O^{|V|}$
- Los pesos de las variables: para cada variable $v \in V$ en la consulta, $w_v \in [0,1]$
- Para cada $x \in O$,

$$\overline{q}_x = \begin{cases} w_v \text{ if } x \text{ instantiates } v \text{ in some tuple in } R \\ 0 \text{ otherwise} \end{cases}$$

Creando el vector de documento $\overline{d}_x$:

- Mapear conceptos a palabras clave
- Peso por una instancia $x \in O$ que anota un documento *d*: TF-IDF

$$\overline{d}_x = \frac{freq_{x,d}}{\max_{y \in O} freq_{y,d}} \cdot \log \frac{N}{n_x}$$

$freq_{x,d} = n[?][?]mero\ de\ ocurrencias\ de\ palabras\ clave\ de\ x\ en\ d$

$n_x = n[?][?]mero\ de\ documentos\ anotados\ por\ x$

N = número total de documentos

Ejemplo:

- **Consulta SPARQL:**
  - Consulta: “players from USA playing in basketball teams of Catalonia”
  - PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  - PREFIX kb: http://nets.ii.uam.es
  - SELECT ?player ?team
  - WHERE (?player rdf:type kb:SportsPlayer.

?player kb:plays kb:Basketball.

?player kb:nationality kb:USA.

?player kb:playsIn ?team.

?team kb:locatedIn kb:Catalonia.)

- **Resultados**:
  - Player (w=1.0): Aaron Jordan Bramlet, Derrick Alston, Venson Hamilton, Jamie Arnold.
  - Team (w=0.5): Caprabo Lleida, Caprabo Lleida, DKV Joventut, DKV Joventut.
- **Vector de consulta:** (...., 1, 1, 1, 1, 0.5, 0.5, ….)
- **Documentos encontrados**: 66 artículos nuevos clasificados desde 0.1 hasta 0.89, el primer resultado fue:
  - "Johnny Rogers and Berni Tamames went yesterday through the medical revision required at the beginning of each season, which consisted of a thorough exploration and several cardiovascular and stress tests, that their team mates had already passed the day before. Both players passed without major problems the examinations carried through by the medical team of the club, which is now awaiting the arrival of the Northamericans Bramlett and Derrick Alston to conclude the revisioning."
- **Vector de documento**: (..., 1.73 (Bramlett), …, 1.65 (Derrick Alston),...)
- **Valor de ranking semántico:** cos (d, q) = 0.88
- **Valor de ranking de palabra clave:** cos (d, q) = 0.06
- Combinando ambos, el valor es de 0.47.

En los experimentos realizados se demostró que el modelo de búsqueda semántica basado en ontologías obtenía resultados de rendimiento comparables y mejores (en términos de MAP y P@10) que el mejor sistema automático de TREC en 2010. Se concretaron los siguientes aspectos:

- Hay mejor precisión usando consultas semánticas estructuradas (se necesita información más precisa).
  - Ej: a football player **playing in** the Juventus vs. **playing against** the Juventus
- Mejor recall cuando se consultan las instancias por clase (la expansión de consultas)
  - Ej: "News about **companies** quoted on NASDAQ"
- Mejor recall usando conclusiones
  - Ej: "Watersports in Spain"→ScubaDiving, Windsurf, etc. in Cadiz, Valencia, Alicante, etc.
- Mejor precisión usando las variables en las consultas

- Ej: nuevos artículos sobre modelos de coches de este año, donde la fecha de lanzamiento no tiene por qué ser mencionada

- La ambigüedad es más fácil de tratar a nivel de los conceptos
  - Propiedad dominio/rango, clasificación por temática, etc.
- Puede haber condiciones en conceptos y condiciones en documentos.
  - Ej: **film review** published by "Le Monde" within the last 7 days about sci-fi **movie**

Como un efecto secundario de un enfoque basado en ontologías, plantearon el problema de la insuficiencia de conocimientos. Este problema se refiere a la necesidad de recuperar resultados precisos cuando la información semántica no está disponible o es incompleta. Para abordar este problema se propone la combinación de las clasificaciones procedentes de los resultados de búsqueda basados en ontologías y los de la palabra clave. Esta combinación se basa en un algoritmo de normalización de puntuación que deshace los posibles sesgos en la distribución de las puntuaciones.

En este trabajo también se abordó una extensión de recuperación semántica con la necesidad de Identificar entidades ontológicas (clases, propiedades, instancias o literales) dentro de los documentos para generar nuevas anotaciones e indexar con esta información semántica. La anotación semántica seguiría la siguiente ilustración:

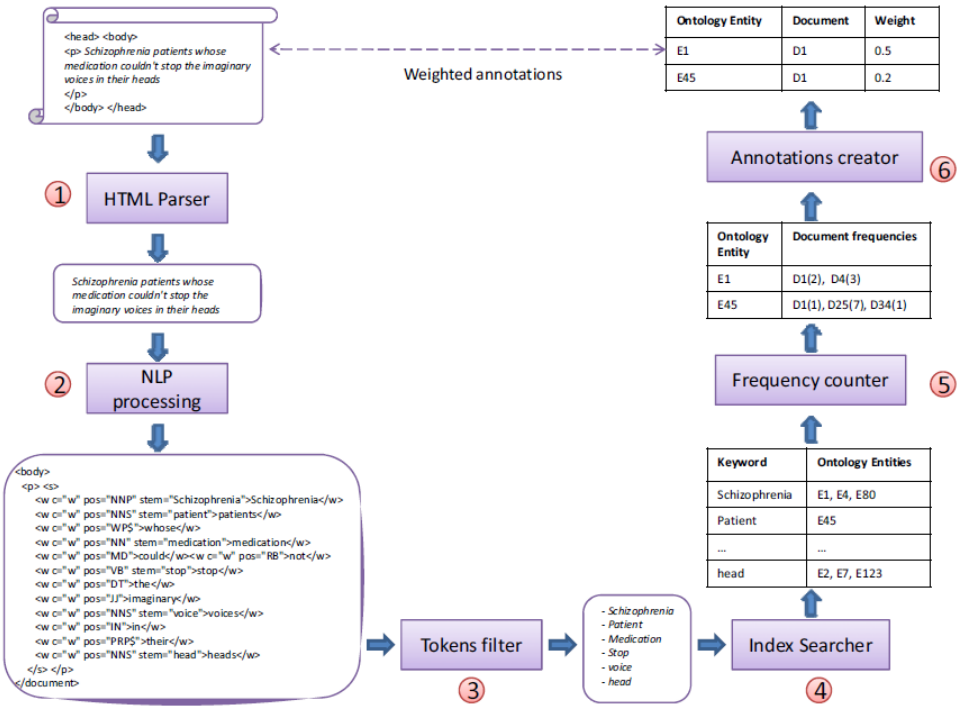

Ilustración 10 Anotación de conceptos semánticos

El tema de la búsqueda semántica es muy amplio y abarca muchos aspectos diferentes que pueden ser abordadas como líneas de investigación futuras. Entre las líneas de futuro más relevantes del trabajo, habría que destacar que:

- El análisis de subgrupos de consultas para las que los algoritmos de búsqueda semántica se desenvuelvan mejor.
- El análisis de la comparación de los resultados obtenidos con ontologías generadas desde bases de conocimiento y los resultados obtenidos con la reutilización de ontologías públicas disponibles en línea.

El estudio y desarrollo de métodos de evaluación novedosos, que tengan en cuenta las medidas de IR no sólo las de rendimiento, tales como la precisión y el recall (o cobertura), sino que también midan las novedades relacionadas con los paradigmas de búsqueda semántica (diferentes formas de interacción, más ricas interfaces de usuario, respuestas estructuradas, etc).

Hay otro trabajo en el estado del arte de cierta influencia en este trabajo en la parte de análisis del catálogo AGS: *"Sistemas de Pregunta-Respuesta"* de los autores *Patricio Martínez-Barco, José Luis Vicedo, Estela Saquete, David Tomás* pertenecientes al *Grupo de Procesamiento del Lenguaje y Sistemas de Información* de la *Universidad de Alicante.*

Recordemos que un sistema de RI tiene por misión devolver, dada una consulta planteada por un usuario (o un perfil de usuario, en los sistemas de filtrado, o encaminado) los documentos más relevantes de acuerdo a la consulta. Por otra parte, la tarea a realizar por los sistemas de Pregunta-Respuesta (sistemas P-R), también conocidos como sistemas de Búsqueda de Respuestas (sistemas BR), y mucho más conocidos por su término inglés *Question-Answering Systems (QA systems)*, es un tipo de recuperación de información en el que se parte de una consulta expresada en lenguaje natural y que debe devolver no ya un documento que sea relevante (es decir que contenga la respuesta) sino la propia respuesta (por ejemplo un hecho).

Si bien los sistemas de RI convencionales utilizaban técnicas básicamente estadísticas, los sistemas de QA modernos, debido a la complejidad de su tarea, necesitan hacer uso de ciertas técnicas de Procesamiento del Lenguaje Natural. Atendiendo al tipo de acceso a la información podemos encontrar sistemas de QA que buscan sobre:

- Datos estructurados (Bases de Datos)
- Datos semi-estructurados (XML, estructuras de texto en Bases de Datos)
- Texto Libre
- Combinación de todos ellos

Uno de los trabajos determinantes en el desarrollo de los actuales sistemas de QA ha sido debido al esfuerzo conjunto de un comité formado por 19 investigadores pertenecientes todos ellos a instituciones diferentes que formaron el *Q&A Roadmap Commitee*. Como consecuencia de este comité se crea "una hoja de ruta", para abordar la investigación de las tareas de procesamiento de las preguntas y extracción de las respuestas y que fue ampliamente debatido en el *Workshop Question Answering: Strategy and Resources* celebrado durante el LREC2002 (Burger et al. 2001).

De este estudio se extrae una serie de expectativas que un usuario real espera del sistema de QA:

- **Respuesta en tiempo razonable**: la respuesta a una pregunta se debe proporcionar en tiempo real, aun cuando múltiples usuarios accedan simultáneamente al sistema. Los nuevos datos deberán incorporarse al sistema tan pronto como se encuentren disponibles incluso en el caso de hechos y eventos muy recientes.
- **Precisión:** Se considera que una respuesta incorrecta es peor que no responder. La investigación en QA debería enfocarse a la manera de evaluar la corrección de las respuestas proporcionadas, incluyendo métodos para detectar que la respuesta no está disponible en la colección de documentos. También las contradicciones en las fuentes de información deberían ser descubiertas, y la información conflictiva debería ser tratada. Para ser más preciso, un sistema de QA debería incorporar conocimiento del mundo y mecanismos que imiten la inferencia del sentido común.
- **Usabilidad**: El prototipado fácil y rápido del conocimiento específico del dominio y su incorporación a las ontologías de dominio abierto debe ser una de las metas a conseguir. A menudo hay que tratar con fuentes de información heterogéneas, por ejemplo textos, bases de datos, video clips o cualquier otro formato multimedia. El sistema de QA debería ser capaz de extraer respuestas sin importar su formato de origen, y proporcionarla al usuario en cualquier formato que desee. Incluso, debería permitir al usuario describir el contexto de la pregunta, pudiendo visualizar y navegar este conocimiento del contexto.
- **Completitud:** Ante una pregunta de usuario se espera una respuesta completa. En algunas ocasiones las preguntas estarán distribuidas a lo largo de un documento, o incluso entre múltiples documentos de la colección. La fusión coherente de la respuesta es también por tanto una de las necesidades del sistema de QA. Un sistema de QA debe incorporar capacidades de razonamiento y usar bases de conocimiento de alto rendimiento. Algunas veces habrá que encontrar analogías con otras preguntas, y su juicio debe realizarse tanto en el contexto definido por el usuario como en el contexto del perfil del usuario. La adquisición automática de perfiles de usuario es un método para permitir sistemas QA colaborativos y para adquirir una retroalimentación desde el usuario muy útil en el proceso de la búsqueda de la respuesta.
- **Relevancia:** La respuesta a una pregunta del usuario debe ser relevante en un contexto específico. A menudo, puede ser necesaria la búsqueda de respuestas interactiva, en la que una secuencia de preguntas ayuda a clarificar la información que se necesita. La complejidad de la pregunta y la taxonomía de preguntas relacionada no puede ser estudiada sin tener en cuenta la representación del contexto, que se convierte en el terreno común existente ente el usuario y el sistema de QA, y tampoco sin tener en cuenta un seguimiento de las preguntas formuladas.

La arquitectura típica de los actuales sistemas de Pregunta-Respuesta tiene en su núcleo más básico un sistema de Recuperación de Información. Las palabras de la pregunta se usan como términos de una consulta y de acuerdo a ella se recuperan los documentos más relevantes. Sin embargo es evidente que hace falta algo más para localizar la respuesta adecuada, ya que:

- No todos los términos de la pregunta son relevantes para buscar la respuesta.

- No basta con encontrar el documento relevante a la pregunta, hay que extraer de él la respuesta.
- Algunos términos de la pregunta pueden ser muy relevantes para saber qué tipo de respuesta se está buscando.

Además, aparte de que el fin del sistema de QA es diferente al del sistema de RI, ocurre que la propia recuperación de la información dentro del sistema de QA está dirigida por intereses distintos a la recuperación de información tratada en general. Ante una consulta como "When did Hitler attack Soviet Union?", un recuperador de información puro actuaría dirigido por la pregunta en sí, es decir, el sistema de RI buscaría documentos que hablen sobre los términos de la pregunta, independientemente de que la respuesta esté o no contenida en ese documento. Sin embargo, la RI que utiliza el sistema de QA debe estar dirigida por la respuesta. No sólo necesitamos buscar documentos que hablen del ataque de Hitler a la Unión Soviética, sino que además en estos documentos debe estar la fecha de ese ataque, debe estar por tanto la respuesta (Magnini, 2005).

Esto hace que la mayoría de los sistemas de QA estén basados en una arquitectura constituida, al menos, por 3 componentes básicos.

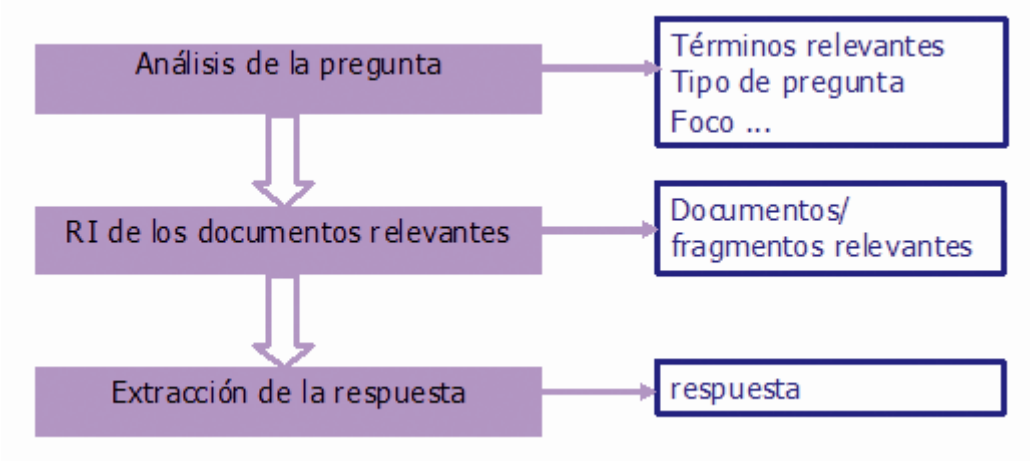

Ilustración 11 Arquitectura básica de un Sistema de Pregunta-Respuesta

Análisis de la pregunta

El módulo de análisis de la pregunta tiene generalmente los siguientes objetivos:

- Determinar la clase de la pregunta (qué, quién, etc.)
- Concretar la clase de la respuesta (objeto, persona, etc.)
- Determinar el foco de la pregunta (el término más importante de la consulta, sobre el que se está preguntando)
- Extraer los términos para efectuar la consulta al sistema de RI
- Concretar el contexto semántico de la respuesta

Para conseguir estos objetivos se emplea diversidad de técnicas. Algunas de ellas son técnicas primitivas con un uso muy limitado del procesamiento del lenguaje natural:

a) **Bag of words**, técnica consistente en considerar toda la consulta como una lista de palabras sueltas que directamente se introducen en el recuperador de información sin contemplar ningún tipo de extensión ni orden de importancia entre ellas.

b) **Stemming**, técnica computacional consistente en reducir una palabra flexionada o derivada a su stem o forma raíz, es decir, la parte de la palabra que es invariante a todas sus formas flexionadas eliminando los sufijos. El stem no tiene porqué coincidir con la raíz morfológica de la palabra que se puede encontrar en un diccionario. Normalmente es suficiente que las diferentes formas de la palabra coincidan en un mismo ítem aunque éste no sea una raíz válida de la palabra según el diccionario. El stemming en el proceso de análisis de la pregunta se aplica a los diferentes términos de la entrada y proporcionará una simplificación en el proceso ya que lo reduce a la búsqueda de la forma básica de la palabra en lugar de su forma derivada. Uno de los algoritmos más usados en Stemming es el famoso algoritmo de Porter (1980), originalmente diseñado para el inglés. Actualmente hay muchas implementaciones del algoritmo de Porter original para numerosas lenguas.

c) **Lematización**, técnica computacional para determinar el lema de una palabra. Este proceso ya implica determinar la categoría gramatical de la palabra por lo que se requiere de una gramática de la lengua y de un diccionario.

d**) Eliminación de palabras de paso o stopwords**. Se llama "palabra de paso" o stopword a aquellas palabras que se eliminan sistemáticamente previo a un proceso de análisis de un texto. La consideración de ser palabra de paso o no viene dada por la utilidad que pueda tener esa palabra dentro de un uso o contexto determinado. En el caso de los sistemas de Pregunta-Respuesta se consideran típicamente palabras de paso los artículos y algunas preposiciones. Sin embargo, algunas palabras que en RI podrían considerarse palabras de paso, en QA sin embargo juegan un papel fundamental en la clasificación correcta de la pregunta. Ciertas preposiciones, partículas interrogativas o pronombres pueden jugar un papel decisivo en la correcta interpretación de la pregunta y la extracción de la respuesta.

Otras técnicas más avanzadas en PLN incluyen:

e) **Análisis sintáctico superficial**. Los sistemas de análisis sintáctico proporcionan datos importantes al análisis de la pregunta. Mediante un análisis sintáctico superficial (o chunking) se pueden detectar constituyentes básicos necesarios para la búsqueda de información como pueden ser los grupos nominales y verbales, aunque no llega a determinar sus constituyentes internos ni el rol que ocupan en la oración. La salida obtenida por este tipo de herramientas sería una lista plana de constituyentes básicos.

f) **Análisis sintáctico de dependencias**. El análisis de dependencias no sólo detecta los constituyentes básicos sino también las dependencias y relaciones entre ellos. Son determinantes para conseguir obtener el rol sintáctico de cada constituyente en la oración. Es especialmente interesante para obtener el rol sintáctico que ocuparía la respuesta esperada en la consulta al ser transformada esta consulta en una afirmación si la respuesta fuera

conocida. La salida de un analizador de dependencias suele quedar representada por un árbol de análisis.

g) **Desambiguación del sentido de las palabras**. La necesidad de la determinación de las características semánticas de algunos términos de la consulta para permitir su correcta clasificación implica el uso de desambiguadores que proporcionan el sentido exacto del término contra una ontología previamente establecida. En caso de tratarse de sistemas de QA de dominio abierto, el recurso más utilizado es Wordnet (Miller, 1990). Sin embargo, los sistemas de QA de dominio restringido suelen definir su propia ontología semántica para términos relacionados con el dominio, cumpliendo así una doble función: reducir enormemente las posibilidades de elección para el desambiguador, y con ello sus errores, y por otra parte descartar aquellos términos que no son relevantes para el dominio. Además una vez determinado el sentido de la palabra es posible añadirle otras propiedades semánticas derivadas de sus relaciones con otras palabras. Tal y como sugieren ciertos trabajos como (Pazienza&Vindigni 2003, Medche&Staab 2001), se pueden encontrar relaciones de equivalencia interesantes a partir de las estructuras IS-A de los conceptos léxicos.

h) **Obtención de representaciones semánticas**. Mediante el uso de estructuras complejas como la formas lógicas se puede llegar a obtener una representación completa sintáctico-semántica que representa un cierto nivel de organización mental de la pregunta (Zajac, 2001). En las formas lógicas la respuesta a obtener quedaría representada por un vacío que tendría asignada una determinada característica sintáctica y semántica proporcionando una clara pista al módulo extractor de respuestas sobre el tipo de respuesta a buscar.

i**) Sistema de clasificación de la pregunta.** Se trata de la clasificación en un conjunto limitado de clases. La forma más básica distingue entre clases de acuerdo con la partícula interrogativa. Clasificaciones más complejas relacionan el tipo de pregunta con el tipo de respuesta esperado creando a su vez subclases de preguntas.

La taxonomía definida en Moldovan (2000) se basa en un estudio de las 200 preguntas de la colección TREC-8. Esta clasificación contiene tres niveles. El primer nivel identifica el tipo básico de la pregunta. En este caso se basa en la partícula interrogativa. El segundo nivel centra más aún el tipo de pregunta añadiendo información sobre el contexto de la pregunta y determinando el tipo de respuesta esperado. Tras alcanzar el segundo nivel ya podemos relacionar el tipo de respuesta esperada con el tipo de objeto a buscar. En WHICH-WHEN estamos esperado un objeto tipo DATE, en WHICH-WHERE esperamos un objeto tipo lugar, y en WHICHWHAT esperamos un objeto organización o nombre propio.

Q-class	Q-subclass	A-type
WHAT	basic what what-who what-when what-where	money | number | definition | title | nnp | undefined person | organization date location
WHO		person | organization
HOW	basic how how-many how-much how-far how-tall how-rich how-large	Maner number money | price distance number undefined number
WHERE		Location
WHEN		Date
WHICH	which-who which-where which-when which-what	Person location date nnp | organization
NAME	name-who name-where name-what	person | organization location title | nnp
WHY		Reason
WHOM		person | organization

Ilustración 12 Taxonomía de preguntas (Moldovan et al., 2000)

La correcta clasificación de una pregunta como “Which city has the oldest relationship as sister-city with Los Angeles?” en el primer nivel de la taxonomía se puede realizar bien mediante concordancia de patrones, o bien mediante técnicas estadísticas. Para el segundo nivel de la taxonomía es necesario realizar un análisis semántico de la pregunta y determinar que “city” implica localización y por tanto se trata del subtipo WHICH-WHERE. Conectar el segundo nivel de la taxonomía con el tercero es ya inmediato. Por tanto sabemos ya que debemos buscar un objeto de tipo lugar. La clasificación de la pregunta puede por tanto necesitar de un cierto nivel de análisis semántico de las palabras.

j) **Obtención del foco de la pregunta**. En algunas ocasiones, el tipo de la pregunta es tan genérico que no aporta nada sobre el tipo de información que se está buscando. Es el caso de las preguntas tipo WHAT básicas: What is the largest city in Germany? El problema se soluciona con la definición de foco de la pregunta, que es una palabra o conjunto de palabras que define la pregunta y la desambigua indicando lo que se está buscando, en este caso, “largest city”. Al conocer el tipo de pregunta y conocer su foco resulta mucho más fácil identificar el tipo de información que se espera como respuesta. Además, el foco es también importante al determinar la lista de términos que se van a recuperar, ya que en muchas ocasiones el foco precisamente se incluye en la pregunta para definir su contexto, pero es muy improbable que se encuentre en la respuesta posible. Es el caso de “In 1990, what day of the week did Christmas fall on?” donde el foco es “day of the week” que siendo el término más

importante de la pregunta sin embargo no debería incluirse en la lista de términos para el recuperador de información ya que es muy poco probable que se encuentre en la respuesta.

k**) Extracción de los términos clave de la pregunta**. Por último, antes de pasar a la fase de recuperación, es necesario establecer cuáles son los términos clave que se van a buscar. Los sistemas más simples se limitan a extraer las palabras de la pregunta eliminando las palabras de paso. Otros enriquecen su lista con una expansión de los términos incluyendo sinonimia y otras relaciones semánticas derivadas de la ontología. En cualquier caso, la determinación de qué términos son relevantes o no para la búsqueda depende mucho del tipo de pregunta. Por ello, algunos sistemas esperan hasta clasificar la pregunta y obtener su foco para decidir, mediante una serie de heurísticas cuál debe ser la lista de palabras clave de búsqueda.

Otros problemas importantes que puede ser necesario resolver a este nivel incluyen:

l) Localización de entidades con nombre (Named Entity Recognition, NER) y su clasificación posterior (NEC)

m) Reconocimiento de términos multipalabra

n) Tratamiento de siglas, abreviaturas, fechas, cantidades, etc.

Además, los sistemas de QA que son sensibles al contexto, es decir, que pueden variar la respuesta ante una misma pregunta dependiendo del contexto en el que se formuló, deberán incluir un mecanismo adicional de detección del contexto que permita representar dicho contexto según el modelo contextual que se haya integrado. Típicamente se trata de incluir información temporal (Pustejovsky, 2001) y espacial, como la fecha y hora de la pregunta, así como el lugar de la misma, e incluso información sobre el perfil del usuario (sus preferencias y datos personales) si se tiene. Al finalizar esta fase del proceso, el sistema de QA dispone de un objeto pregunta, una imagen de la pregunta original a la que se le han marcado todas aquellas características que se necesitarán para la recuperación y la extracción de la respuesta.

La recuperación de información de los documentos relevantes es el segundo gran módulo en el proceso de QA. Se trata de encontrar los documentos relevantes a la pregunta entre los que se espera que se encuentre la respuesta. El proceso de recuperación de información varía mucho dependiendo del tipo de acceso a los datos contenedores de la respuesta:

a) **Recuperación de información sobre datos estructurados**: En este caso la recuperación de información se transforma en un acceso a la base de datos mediante su lenguaje de consulta. Para ello se deberán establecer mecanismos de traducción de la pregunta al lenguaje de consulta (normalmente SQL). Para relacionar los términos de la pregunta con el esquema de la base de datos es necesario establecer un mapeo previo de la base de datos con la ontología. Puramente hablando, el esquema de la base de datos debería constituir en sí la propia ontología del sistema de QA. De esta forma, tras analizar la pregunta se obtendrían todos los parámetros necesarios para construir la consulta a la BD, ya que cada uno de sus términos se corresponderá con un objeto de la BD. En este sentido hay algunas propuestas basadas en la transformación de formas lógicas a SQL.

b) **Recuperación de información sobre datos no estructurados o semiestructurados**: En este caso, al no existir una estructura previa, no se puede realizar un mapeo directo de los términos de la pregunta en la colección documental. La información susceptible de contener la respuesta debe ser analizada y relacionada con la pregunta.

La recuperación de información sobre datos no estructurados estará formada el menos por tres fases:

- Indexación de la colección
- Recuperación de documentos
- Selección de pasajes relevantes

a**) Indexación de la colección**: En las aplicaciones QA de tiempo real, es necesario realizar un preproceso off-line del conjunto de documentos con el fin de garantizar un tiempo de respuesta aceptable. El objetivo básico de este preproceso es conseguir un índice de la documentación de tal manera que sea posible reconocer qué documentos son relevantes a una consulta sin necesidad de procesar en tiempo real cada uno de los documentos. La indexación más básica a aplicar incluye únicamente término, pero este preproceso off-line permite también aplicar ciertos niveles de análisis que generen etiquetados de mayor nivel como el POS o el reconocimiento de entidades con nombre, y que también serían incluidos en los índices. Actualmente, los sistemas de QA están incluyendo los siguientes tipos de indexación:

- Por términos
- Por términos sin stopwords
- Por lemas o raíces
- Por términos multipalabras
- Por entidades con nombre clasificadas
- Por tipo de entidad
- Por conceptos semánticos

b**) Recuperación de documentos**: Los términos clave de la pregunta con su expansión correspondiente se buscan en el índice para seleccionar los documentos más relevantes a la misma. Dichos documentos se devuelven ordenados de acuerdo a un ranking de relevancia. El número de documentos a recuperar aquí suele ser un número fijo: 10, 50, 100, aunque también hay sistemas que se basan en un umbral para aceptar sólo los documentos cuya relevancia supera tal umbral, e incluso, si no existe un número suficiente de documentos, se modifica la consulta (incorporando sinónimos o variantes) para lograr un número mínimo.

c) **Selección de pasajes relevantes**: La presentación de un resultado de Recuperación de Información en forma de documentos podría ser válida cuando se trata de un sistema de Recuperación de Información puro. Sin embargo, en un sistema de QA donde hay que devolver una respuesta concreta, partir directamente de un documento relevante completo dificultaría mucho la extracción correcta de la respuesta. Por este motivo, es necesario pasar por un proceso de refinamiento en el que se seleccionen los pasajes más relevantes a la pregunta dentro del documento. Los pasajes relevantes tendrán las siguientes características:

- Tamaño:
  - Tamaño fijo: 100 palabras, 200 bytes, n líneas
  - Variable: párrafo, oración, pasaje
- Cobertura:
  - Solapados: una oración, la anterior y la siguiente
  - Disjuntos
- Segmentación:
  - Puramente sintáctica: por cambio de oración, párrafo, etc.
  - Por cambio de tema (cambios de tópico).

Una vez obtenida la segmentación de los documentos se puede utilizar de nuevo un algoritmo de Recuperación de Información para obtener los segmentos más relevantes. De nuevo se pueden utilizar sistemas convencionales de RI aunque en este caso se utilizan también técnicas y métricas específicas tanto para la indexación como para la recuperación:

- Se considerará el porcentaje de palabras clave presentes en el pasaje.
- Se tendrá en cuenta si alguna palabra es obligatoria (como por ejemplo, el foco de la pregunta). Los pasajes seleccionados son enviados a la siguiente fase, bien en modo texto directamente, o bien mediante estructuras que permitan la comparación

Los pasajes seleccionados son enviados a la siguiente fase, bien en modo texto directamente, o bien mediante estructuras que permitan la comparación inmediata del pasaje con la representación de la pregunta (como las formas lógicas).

Extracción de la respuesta

El proceso de extracción de la respuesta comprende la localización de la respuesta en los fragmentos relevantes. La dificultad de esta tarea depende del tipo de respuesta esperada. Por ejemplo, la respuesta a un Who ...? puede ser tan simple como encontrar una entidad de tipo persona existente en el fragmento relevante.

En esta fase se concentran los esfuerzos de la mayoría de los sistemas y las técnicas más variadas. El objetivo fundamental es encontrar la forma en que a partir de la información de la pregunta puede inferirse información sobre la respuesta.

La fase de extracción de las respuestas puede subdividirse en las siguientes tareas:

a) Análisis de los pasajes relevantes: Se usan generalmente técnicas de análisis superficial (shallow parsing), con especial énfasis en el reconocimiento de entidades con nombre. En algunas ocasiones se han utilizado gramáticas específicas para cada uno de los tipos de respuesta a buscar. La complejidad del análisis vendrá determinada por el modelo de proyección usado.

b) Extracción de la respuesta: Dependen totalmente del modelo de proyección que se haya elegido, pero en su forma más básica se usan técnicas simples de pattern matching donde se establecen diferentes medidas según el tipo de atributo (coincidencia de términos, densidad de términos relevantes, dispersión, etc.) y que luego se combinan (Ittycheriah et al, 2001).

Who is the author of the "Star Spangled Banner"?

…<PERSON>**Francis Scott Key** </PERSON> wrote the *"Star Spangled Banner" in* <DATE>**1814**</DATE>

Tipo de respuesta esperada = <**PERSON**>
Respuesta candidata = **Francis Scott Key**

c) Validación de la respuesta: En algunos sistemas, una vez localizada la respuesta se intenta a través de algún tipo de razonamiento demostrar que realmente responde a la pregunta. El problema de la validación se puede definir como: "Dada una pregunta Q y un candidato a respuesta A, decidir si A es la respuesta correcta para Q" (Magnini, 2005). La validación automática de la respuesta debe responder a las siguientes premisas:

- Precisión: debe estar a la altura de los juicios humanos
- Eficiencia: a larga escala (Web) y en escenarios de tiempo real
- Simplicidad: debe evitar la complejidad del sistema de QA

Para la validación de las respuestas se han usado tradicionalmente diferentes aproximaciones apoyadas normalmente en la gran colección documental que contiene la Web:

- Basadas en proximidad: que se aprovechan de la redundancia de la Web para comprobar si la respuesta seleccionada aparece en algún lugar cerca de los términos de la pregunta (Magnini et al. 2002).
- Basadas en patrones: donde se construyen patrones de validación en los que la pregunta se reformula como una afirmación que incluye ya la respuesta candidata y se comprueba si aparece en la Web (Subbotin y Subbotin, 2001).
- Estadística: donde se comprueba si la pregunta y la respuesta tienden a aparecer juntos en la web. A diferencia de la primera, ésta no necesita descargar los documentos para comprobar su proximidad. Únicamente cuenta las veces que aparecen juntos (Turney, 2001).

Analicemos la validación con un ejemplo.

Pregunta: ¿Quién inventó el telégrafo?

Respuesta candidata: Samuel Morse.

a) **En una validación basada en proximidad**, tomaríamos los términos relevantes de la consulta junto con la respuesta y pondríamos en marcha la búsqueda en la Web obteniendo pasajes como:

Quien **inventó** el **telégrafo** eléctrico, fue **Samuel Morse**.
Hasta que en 1854, la Corte Suprema de los Estados Unidos, dictaminó, que **Morse** fue quien **inventó** el primer **telégrafo**.
Samuel Finley Breese **Morse** (27 de abril de 1791, Charlestown, Massachusetts - falleció el 2 de abril de 1872, Nueva York), **inventor** del **telégrafo**.
Samuel Morse ha pasado a la Historia por haber **inventado** el **telégrafo** eléctrico y el alfabeto **Morse**.
…

En todos ellos existe una gran proximidad entre los términos: Samuel Morse, inventar, telégrafo. Ello determinaría que la respuesta es correcta.

b) **En una validación basada en patrones**, la pregunta se podría reformular como la siguiente afirmación: "Samuel Morse inventó el telégrafo". Si buscamos exactamente esta afirmación en la Web nos encontraremos con que aparece varias veces (al menos 23 ocurrencias aparecen en Google), con lo que podemos dar por correcta la respuesta.

c) **En la aproximación estadística**, tras lanzar una búsqueda de documentos que contienen los términos Samuel Morse, inventar y telégrafo, Google devuelve hay 9590 documentos que cumplen, con lo que se daría por válida.

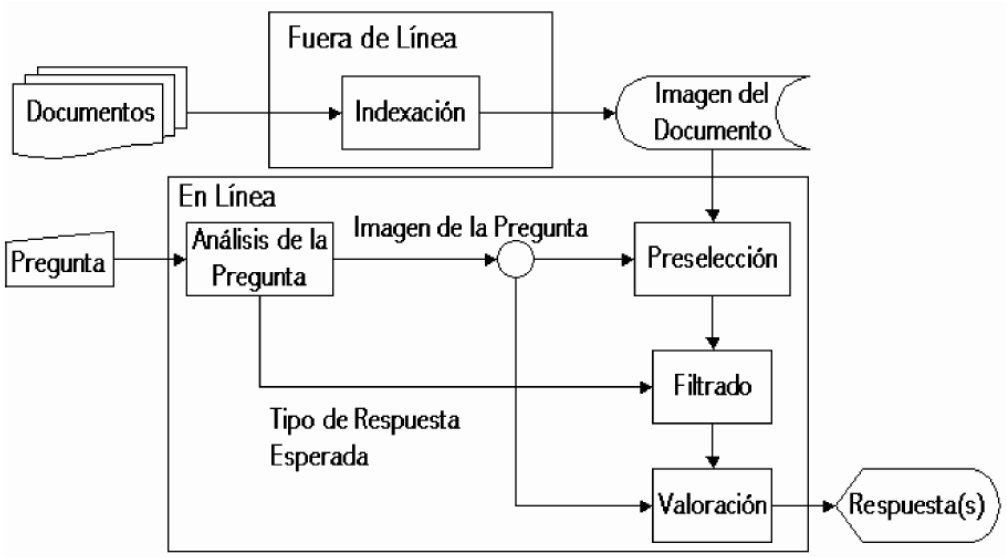

Ilustración 13 Arquitectura general de un sistema de QA (Moya, 2004)

3. ANÁLISIS DEL DOMINIO: CATÁLOGO ON-LINE

En este capítulo se describe con detalle tanto la estructura del catálogo como las fichas del catálogo del Archivo General de Simancas (AGS) sobre Mapas, Planos y Dibujos.

Cada ficha del catálogo AGS on-line contiene una información sobre mapas, planos, ilustraciones y fotos. Entre la información disponible de cada ficha, podemos diferenciar las siguientes facetas o los siguientes campos:

1. Fecha: En este campo se almacena un número de cuatro cifras que indica el año en el que se realizó la obra.
2. Referencias: indica el libro o revista donde podemos encontrar la obra.
3. Creador – Autor: autor de la obra.
4. Tipo : Se distinguen únicamente cinco tipos de obras:
    a. Ilustraciones y fotos
    b. Mapas
    c. Manuscritos
    d. Libros
    e. Otros
5. Idiomas – Lengua: indica la lengua en la que está redactada la obra, de entre todos los idiomas se distinguen los siguientes:
    a. Spa – Español – Castellano
    b. Fre – Francés
    c. Eng – Inglés
    d. Lat – Latín
    e. Ger – Alemán
    f. Ita – Italiano
    g. Por – Portugués
    h. Dut – Holandés, flamenco
    i. Cat - Catalán
6. Temática – Tema: indica la temática a la que pertenecen las obras, los valores son muy diversos y cambiantes en cada obra añadida, algunos de los valores posibles son los siguientes:
    a. Cañones dibujos
    b. Ballenas dibujos
    c. Fauna dibujos
    d. Artillería dibujos
    e. Máquinas dibujos
    f. Barcos dibujos
    g. Uniformes Militares Dibujos
    h. Archivo General de Simancas Planos
    i. Hospitales planos
    j. Faros planos
    k. Barcos planos
    l. Fortificaciones planos

m. Dibujos de arquitectura
n. Municiones

7. Técnica utilizada: nos indica qué técnica se ha utilizado para la realización de la obra, podemos distinguir las siguientes:
    a. Tinta
        i. Puede ser aguada, negra.
    b. Tinta y colores
        i. Los colores pueden ser: negro/a, rojo/a, dorado/a, granate, verde, amarillo/a, gris, verde, sepia.
    c. Lápiz negro
    d. Todas ellas pueden ser:
        i. Con explicación
        ii. Con rotulación
8. Soporte – Impreso en – Escrito en: Este campo nos indica sobre que soporte está realizada la obra, se distinguen los siguientes soportes.
    a. Pergamino
    b. Manuscrito sobre papel
    c. Copia
    d. Grabado (solo ilustraciones y fotos)
    e. Impreso

El primer paso es descargar las fichas en dos formatos distintos:

- En texto plano.
- Y en formato RDF Dublin Core.

Desde la web del AGS, tenemos una herramienta de exportación de fichas que nos permite obtener la información de todas las fichas en un fichero único en cada uno de los dos formatos que necesitamos (texto y RDF DC).

Las fichas se pueden encontrar en la web del Ministerio de Educación, Cultura y Deporte de España:

http://www.mcu.es/ccbae/es/consulta/resultados_busqueda.cmd?tipo_busqueda=mapas_planos_dibujos&posicion=1&id=30485

En este enlace, se accede directamente a una búsqueda general dentro del Catálogo Colectivo de las colecciones de Mapas, Planos y Dibujos del Archivo General de Simancas. El interface inicial muestra información de los distintos objetos existentes en el catálogo, que pueden ser de tres tipos (mapas, planos y dibujos) y en el marco izquierda las materias posibles.

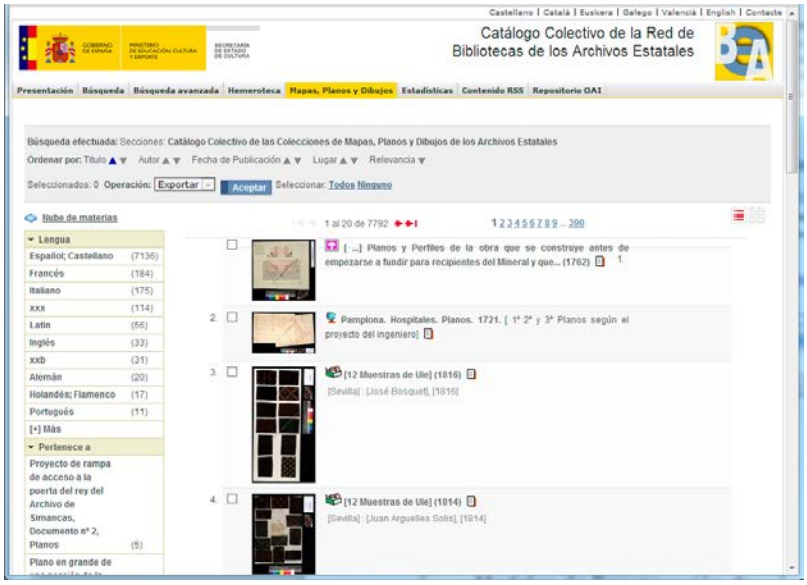

Ilustración 14 Interface inicial de la web del AGS

Si clickamos en cualquiera de los resultados, accederemos a la ficha con el contenido de los distintos elementos, como se puede observar en la siguiente figura:

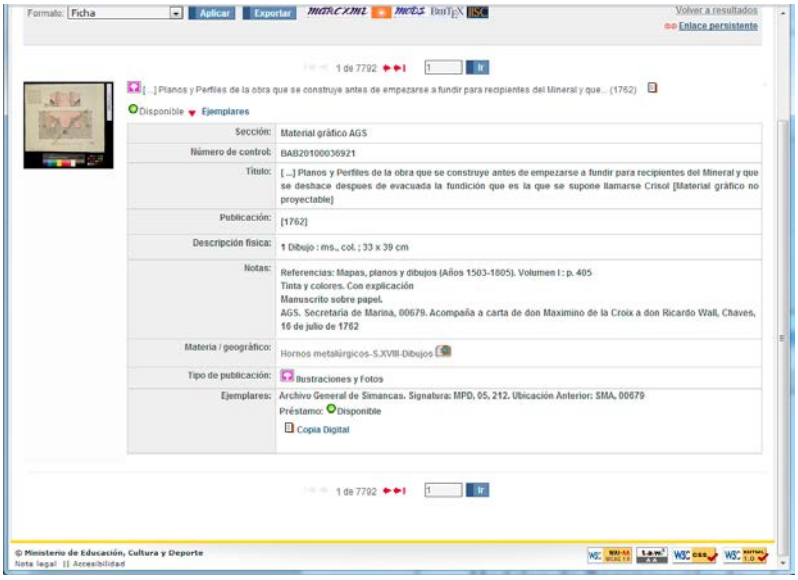

Ilustración 15 Ejemplo de ficha del AGS

Hay que descargar la información de todas las fichas existentes en la web en un formato útil para nuestro objetivo. La plataforma nos ofrece los distintos formatos que se observan en la siguiente figura:

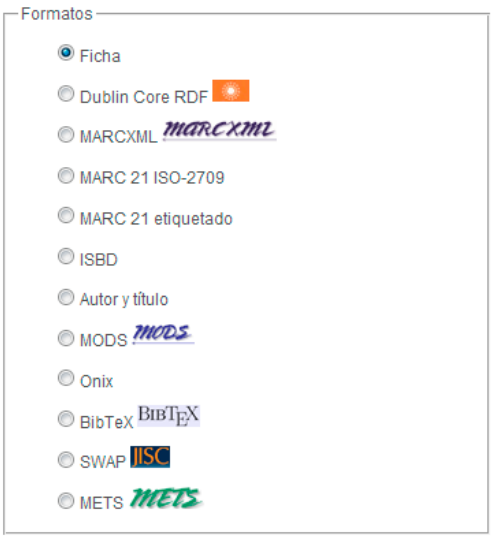

Ilustración 16 Formatos de exportación de fichas del AGS

En nuestro caso se han descargado todas las fichas (7792 registros) en formato Dublin Core RDF, en un fichero comprimido en formato zip que contiene 8 ficheros de tipo .txt con el contenido de todas las fichas encontradas en la web, en formato RDF Dublin Core.

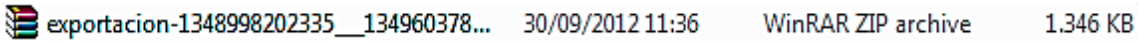

Ilustración 17 Fichero ZIP resultante de la exportación de fichas del AGS

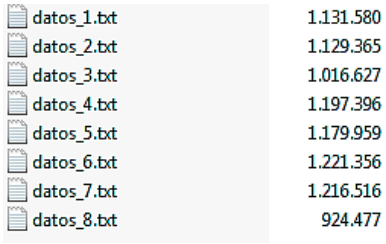

Ilustración 18 Listado de ficheros TXT de la exportación de fichas del AGS

Después de haber almacenado las fichas en cada uno de los formatos, necesitamos contar con una serie de preguntas para poder realizar la comparación correcta de los resultados entre ambos tipos de búsqueda.

PARTE 2: TRABAJO REALIZADO Y EXPERIMENTOS

En el catálogo con que se experimenta en este trabajo hay 7792 fichas obtenidas de la página web del Archivo General de Simancas. Teniendo en cuenta la cantidad de fichas y de información existentes, nos encontramos con la necesidad de realizar un buscador semántico para obtener la información o las propias fichas que necesitemos en momentos determinados.

Para poder realizar este buscador semántico, debemos dividir nuestro trabajo en las siguientes tareas:

1. Desarrollar el entorno para experimentación.
2. Incorporar la información disponible inicialmente (fichas del catálogo o corpus) en el formato estándar decidido.

   Las fichas se pueden descargar en texto plano o en RDF Dublin Core. Pero, aunque para trabajar en Solar, se podía usar cualquiera de estos formatos, para incluir las fichas en una ontología Protégé sólo podían estar en OWL.

   Por ello se comenzó realizando un parser de RDF-DC a OWL. Se encuentra descrito en la sección 4.1 y en el Anexo 1.

3. Se realizaron las tres aproximaciones de almacenamiento del catálogo:
   a. En OWL con Protégé (descrito en la sección 4.2).
   b. En Apache Solr con facetas, las correspondientes en el formato RDF-DC .
   c. En Apache Solr en texto plano.

   Estos dos últimos se describen en la sección 4.3 y en los anexos 2 y 3.

4. En la sección 4.4 se describe el esquema del trabajo planteado y finalmente realizado sobre búsqueda semántica y la justificación de cada etapa y decisión tecnológica tomada.
5. Generar un conjunto preguntas que permitan la comparación entre un buscador ontológico y uno textual. Se describen en la sección 5.1.
6. Desarrollar el buscador textual y el basado en facetas para el AGS (descrito en la sección 5.2).
   a. Para ello se realizó un parser sobre la clasificación de consultas definidas para generar la consulta formal, a partir del análisis para identificar los predicados de nivel 1 y 2.
   b. Y para cada tipo de pregunta se construye el tipo de respuesta esperada.
   c. Además para la búsqueda con Protégeé se construye la consulta correspondiente en SPARQL.
7. Para comparar las aproximaciones desarrolladas, se aprende e instala la herramienta TRECEval, aunque no se puede usar completamente porque finalmente no se dispone de los esperados juicios de relevancia para cada consulta.

4. PROPUESTAS PARA ALMACENAMIENTO DEL CATÁLOGO Y GESTIÓN DE LA BÚSQUEDA

Para poder experimentar con modelos de búsqueda basados en Apache Solr y en Protégé (las dos tecnologías seleccionadas), necesitamos descargar las fichas del catálogo en RDF DC y texto plano. Una vez hemos obtenido estas fichas en ambos formatos, para poder trabajar con Protégé, necesitamos convertir el fichero en formato RDF DC a formato OWL. Para ello, es necesario la creación de un parser que recibirá un fichero de entrada que debe ir en formato **rdf dublin core**, un fichero donde se almacenará la conversión a **OWL** y los campos equivalentes entre ambos formatos.

Una vez realizado este parser, obteniendo nuestro fichero en formato OWL con la información relativa de todas las fichas, realizamos la importación en protègè y ya podremos trabajar con las fichas.

Para poder realizar búsquedas textuales, hemos elegido Apache Solr, basado en el motor de búsqueda de Lucene. Nuestra idea es almacenar la información relativa de todas las fichas de dos formas diferentes: de forma textual sin facetar y de forma facetada.

Por otra parte, con los ficheros planos y en formato RDF DC, instalaremos en un servidor todas las herramientas necesarias para hacer funcionar Apache Solr, crearemos dos directorios, uno para cada tipo de fichas (facetadas y sin facetar), realizaremos la limpieza de los ficheros en cada formato y los indexaremos con un indexador creado en lenguaje bash.

Como se ha almacenado la información de las fichas de estas dos formas, podremos realizar una comparación de búsqueda completa entre las fichas que contienen información ordenada y sin ordenar. Demostraremos que la información obtenida en de las fichas almacenadas por facetas es más rápida, más concreta y más correcta que las recibidas por la búsqueda textual. Además, demostraremos también que a la hora de realizar nuestro buscador semántico y el enriquecimiento semántico de las fichas, será mucho más sencillo de hacer partiendo de este conjunto de fichas facetadas como base.

4.1. CONVERSIÓN DE FORMATO RDF DUBLIN CORE A OWL.

Para poder incorporar la ontología subyacente a las fichas en la herramienta protégé, es necesaria una conversión de formato de RDF Dublin Core a OWL. Véase el contenido de una de las fichas obtenidas en la exportación:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc=
"http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <rdf:Description>
        <dc:relation>Referencias: Mapas, planos y dibujos (Años 1503-1805). Volumen I : p. 405
        </dc:relation>
        <dc:coverage>-S.XVIII</dc:coverage>
        <dc:title>[ ...] Planos y Perfiles de la obra que se construye antes de empezarse a
        fundir para recipientes del Mineral y que se deshace despues de evacuada la fundición
        que es la que se supone llamarse Crisol [Material gráfico no proyectable]</dc:title>
        <dc:description>AGS. Secretaría de Marina, 00679. Acompaña a carta de don Maximino de
        la Croix a don Ricardo Wall, Chaves, 16 de julio de 1762</dc:description>
        <dc:description>Tinta y colores. Con explicación</dc:description>
        <dc:description>Manuscrito sobre papel.</dc:description>
        <dc:type>Ilustraciones y Fotos</dc:type>
        <dc:language>spa</dc:language>
        <dc:date>1762</dc:date>
        <dc:identifier>http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725</dc:identifier>
        <dc:format>image/jpeg</dc:format>
        <dc:subject>Hornos metalúrgicos-Dibujos</dc:subject>
    </rdf:Description>
```

Ilustración 19 Ficha en formato Dublin Core

En la primera línea encontramos la declaración del fichero indicando que se trata de un fichero escrito en **XML**, concretamente en su **versión 1.0**, con codificación **UTF-8** y que además, cumple con las especificaciones de una hoja DTD, con lo cual se trata de un XML **no independiente** (standalone = “no”).

Al pertenecer a un formato determinado que contiene una serie de normas, se indica en la siguiente línea las páginas de referencia que contienen estas normas de estructura del formato RDF Dublin Core.Y a continuación, comienza la instanciación de cada elemento rdf, que en este caso pertenece a cada una de las fichas descargadas. De esta forma, tenemos el contenido de las fichas dentro de las etiquetas **<rdf:Description>** y **</rdf:Description>** . Por cada ficha tenemos un elemento de apertura y otro de cierre, y en su interior, las distintas propiedades de cada ficha.

Para convertir los ficheros al formato OWL, se debe realizar el siguiente proceso:

1) El primer paso es conocer la estructura de un fichero OWL.
   a. OWL es un formato derivado de RDF, con lo cual, la cabecera de todo fichero en formato OWL debe ser de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.jabenitez.com/ontologias/datos_1_prueba.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xml:base="hhttp://www.jabenitez.com/ontologias/datos_1_prueba.owl">
```

Ilustración 20 Cabecera de fichero en formato OWL

   b. Una vez se ha declarado la cabecera, y abierto la etiqueta <rdf:RDF, el siguiente paso para formar nuestro fichero en formato OWL es instanciar la clase ontología, aportando la información necesaria relativa a la misma.

```
<owl:Ontology rdf:about="">
  <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">version 1.3</owl:versionInfo>
  <rdfs:comment xml:lang="es">Ontologia de los elementos cartograficos de mi tesis</rdfs:comment>
  <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
  <protege:defaultLanguage rdf:datatype="http://www.w3.org/2001/XMLSchema#string">es</protege:defaultLanguage>
</owl:Ontology>
```

Ilustración 21 Campo Ontology en fichero OWL

De esta manera, declaramos nuestra ontología con las etiquetas **<owl:Ontology>** para comenzar y **</owl:Ontology>** , para finalizar. Indicamos la versión de la ontología, un pequeño comentario acerca de la ontología que necesitamos crear y también le indicamos en que idioma se encontrará nuestra ontología.

   c. El siguiente paso es el de crear la clase principal, con la que luego instanciar los distintos objetos de nuestra ontología. En nuestro caso, los objetos que necesitaremos instanciar son **Fichas**, con lo cual, la clase que debemos crear en este fichero, se denomina Ficha.

```
<owl:Class rdf:ID="Ficha">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
```

Para declarar una clase en formato OWL, se hace con las etiquetas **<owl:Class>** en su apertura y **</owl:Class>** en el cierre. Para indicar el nombre de la clase, agregamos la propiedad **rdf:ID="Ficha"** en nuestra etiqueta de clase, generando así una clase de tipo Ficha, con la que luego podremos crear objetos de este tipo.

2) Después de conocer la estructura de los ficheros OWL y habiendo declarado ya la clase principal de la ontología (Ficha), el siguiente paso es comparar las clases que existen en el formato actual RDF Dublin Core y crear las propiedades pertinentes para nuestra Ficha en el nuevo formato. De la siguiente forma, trasladamos una a una las propiedades que contiene el formato RDF Dublin Core a la nueva ontología personalizada:

ETIQUETA EN FORMATO DC	PROPIEDAD CORRESPONDIENTE EN NUESTRA NUEVA ONTOLOGÍA
DC. Title	Propiedad **Titulo** de nuestro objeto **Ficha**.
DC. Creator	Propiedad **Creador** de nuestro objeto **Ficha**.
DC. Subject	Propiedad **Tematica** de nuestro objeto **Ficha**.
DC. Description	Propiedad **Notas** de nuestro objeto **Ficha**.
DC. Publisher	Propiedad **Publicador** de nuestro objeto **Ficha**.
DC. Date	Propiedad **Fecha** de nuestro objeto **Ficha**.
DC. Type	Propiedad **Tipo** de nuestro objeto **Ficha**.
DC. Format	Propiedad **Formato** de nuestro objeto **Ficha**.
DC. Identifier	Este será el identificador de cada una de las **Fichas** que nosotros creemos en nuestros ficheros OWL.
DC. Language	Propiedad **Idioma** de nuestro objeto **Ficha**.
DC. Relation	Propiedad **Referencias** de nuestro objeto **Ficha**.
DC. Coverage	Propiedad **Materia** de nuestro objeto **Ficha**.

Tabla 7 Tabla de correspondencia de campos en formato DC a formato OWL

3) Conociendo la tabla anterior, se crea en formato OWL cada una de estas propiedades, de la siguiente forma:

```
<owl:DatatypeProperty rdf:ID="Referencias">
    <rdfs:domain rdf:resource="#Ficha"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Ilustración 22 Ejemplo de propiedad en OWL

Se instancia un objeto **owl:DatatypeProperty** dándole un nombre con la propiedad **rdf:ID** y a continuación en su interior, indicamos el objeto al que pertenece esa propiedad (en este caso **Ficha**) y de qué tipo de dato se trata (en este caso, al ser cadenas de texto, indicamos que es un String). Realizando estos pasos con cada uno de los datos de la tabla, obtenemos las siguientes propiedades en nuestro fichero OWL:

```
<owl:DatatypeProperty rdf:ID="Titulo">
    <rdfs:domain rdf:resource="#Ficha"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Creador">
    <rdfs:domain rdf:resource="#Ficha"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="Tematica">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Notas">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:ID="Publicador">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Fecha">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Tipo">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Formato">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Idioma">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Referencias">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="Materia">
     <rdfs:domain rdf:resource="#Ficha"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Una vez iniciado el fichero OWL con la declaración de nuestro objeto principal **Ficha** y sus respectivas propiedades:

- En el fichero en formato RDF Dublin Core, cada objeto está instanciado de la forma: <rdf:Description> CONTENIDO DE LA FICHA </rdf:Description>
- Nosotros necesitamos que cada una de las fichas, esté declarada de la forma: <Ficha rdf:about="**Identificador único de cada ficha">** CONTENIDO </Ficha>

Por esta razón, se crea un parser que convierta de forma automática el fichero en formato RDF DC a OWL, realizando las siguientes tareas principales:

1. Comenzar a leer fichero inicial en RDF DC.
2. Encontrar declaración de un objeto rdf:Description.
3. Realizar la búsqueda del identificador (dc:identifier)
4. Obtener el contenido de esa propiedad.
5. Una vez obtenido el identificador, abrimos nuestro nuevo fichero .owl
6. Comenzamos la creación de un objeto <Ficha> con el identificador obtenido anteriormente, indicándolo con la propiedad rdf:about : <Ficha rdf:about="Identificador encontrado">
7. Una vez hecho esto, se lee el contenido de <rdf:Description>, buscando la propiedad que se está leyendo en ese momento, recopilando su contenido y transformándolo a la nueva ontología creada, de la siguiente manera:
    a. Supongamos que tenemos el siguiente objeto dentro de un fichero **datos.txt**:

```
<rdf:Description>
                    <dc:relation>Referencias: Mapas, planos y dibujos (Años 1503-1805). Volumen I : p.
405</dc:relation>
                    <dc:coverage>-S.XVIII</dc:coverage>
                    <dc:title>[ ...] Planos y Perfiles de la obra que se construye antes de empezarse a fundir para
recipientes del Mineral y que se deshace despues de evacuada la fundición que es la que se supone llamarse
Crisol [Material gráfico no proyectable]</dc:title>
                    <dc:description>AGS. Secretaría de Marina, 00679. Acompaña a carta de don Maximino de la
Croix a don Ricardo Wall, Chaves, 16 de julio de 1762</dc:description>
                    <dc:description>Tinta y colores. Con explicación</dc:description>
                    <dc:description>Manuscrito sobre papel.</dc:description>
                    <dc:type>Ilustraciones y Fotos</dc:type>
                    <dc:language>spa</dc:language>
                    <dc:date>1762</dc:date>

          <dc:identifier>http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725</dc:identifier>
                    <dc:format>image/jpeg</dc:format>
                    <dc:subject>Hornos metalúrgicos-Dibujos</dc:subject>
</rdf:Description>
```

El algoritmo primero encontraría la etiqueta **<rdf:Description>** , anotaría la línea en la que se encuentra esta etiqueta, y seguiría recorriendo su contenido hasta encontrar la etiqueta **<dc:identifier>**.

Una vez encontrada esta etiqueta, se guardaría una cadena con el contenido, en este caso *http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725* y crearíamos un fichero **datos.owl** con la línea

<Ficha rdf:about=" http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725">

Creada esta línea, con otro lector distinto, nos situamos en la línea siguiente a <rdf:Description> y mediante un bucle, buscamos la equivalencia de la propiedad actual en nuestro nuevo formato, es decir:

- La línea a la que accedemos es <dc:coverage>-S.XVII</dc:coverage>
- Nuestro algoritmo recogería que dc:coverage, en nuestra nueva ontología se llama <Materia>.
- Almacenaría el contenido *–S.XVII* en una cadena de texto
- Y escribiría esta propiedad en fichero **datos.owl** de la siguiente forma:

- <Materia>-S.XVII</Materia>

Este proceso de búsqueda de etiquetas y gestión de contenido, lo realizaría con cada uno de los objetos rdf:Description, de forma que en el ejemplo anterior, se crearía un objeto Ficha en nuestro fichero **datos.owl** de la siguiente forma:

```
<Ficha rdf:about="http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725">
        <Titulo> Planos y Perfiles de la obra que se construye antes de empezarse a fundir para
recipientes del Mineral y que se deshace despues de evacuada la fundición que es la que se supone
llamarse Crisol [Material gráfico no proyectable]</Titulo>
        <Referencias>Referencias: Mapas, planos y dibujos (Años 1503-1805). Volumen I : p.
        405</Referencias>
        <Materia>-S.XVIII</Materia>
        <Notas>AGS. Secretaría de Marina, 00679. Acompaña a carta de don Maximino de la Croix a
        don Ricardo Wall, Chaves, 16 de julio de 1762</Notas>
        <Notas>Tinta y colores. Con explicación</Notas>
        <Notas>Manuscrito sobre papel.</Notas>
        <Tipo>Ilustraciones y Fotos</Tipo>
        <Idioma>spa</Idioma>
        <Publicacion>1762</Publicacion>
        <Formato>image/jpeg</Formato>
        <Tematica>Hornos metalúrgicos-Dibujos</Tematica>
</Ficha>
```

La realización de la conversión del fichero en formato RDF DC al formato OWL, se podría hacer de forma manual. Teniendo en cuenta que son más de siete mil fichas las que hay que convertir, en el marco de este trabajo se ha creado una herramienta en lenguaje **JAVA** que solicita al usuario el fichero de origen en formato RDF Dublin Core, nombre de la clase principal del nuevo fichero OWL y la equivalencia de las distintas propiedades entre ambos formatos.

El parser creado tiene la siguiente apariencia:

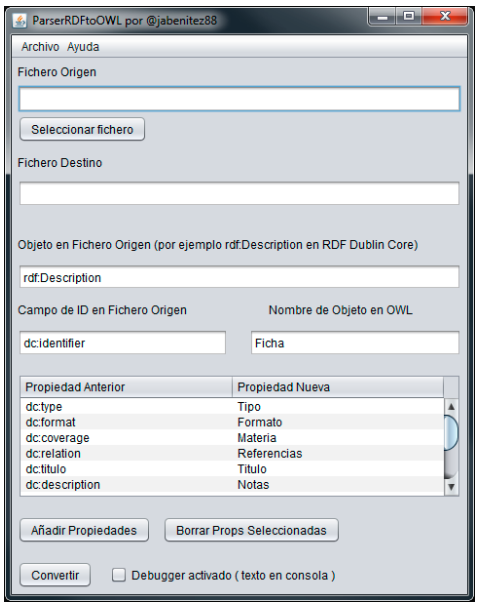

Ilustración 23 Parser RDF to OWL: Pantalla de inicio

Para poder desarrollar esta herramienta, he elegido el lenguaje **JAVA** y el IDE Gráfico de **NetBeans** por el amplio conocimiento que tengo del lenguaje y la gran facilidad de uso de este IDE. Se describe con detalle en el Anexo 1 de esta memoria.

Al finalizar la conversión, si lo ha hecho correctamente, se muestra un cuadro de diálogo indicando que se ha realizado correctamente:

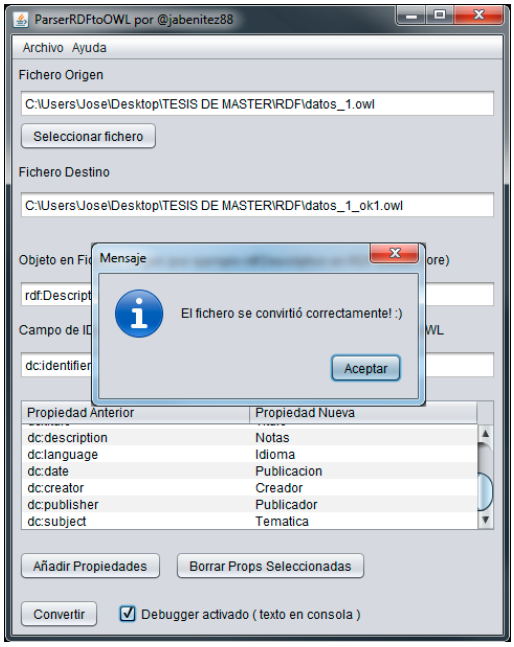

Ilustración 24 Parser RDF to OWL: Diálogo de conversión correcta

Y si nos dirigimos a nuestro directorio, podremos ver que ha creado correctamente el fichero en formato OWL.

Ilustración 25 Parser RDF to OWL: Fichero OWL creado

Si por algún caso, el fichero que intentamos leer fuera ilegible, o tuviera un formato que no puede reconocer, nuestra aplicación mostraría un cuadro de diálogo con un mensaje de error como el mostrado en la siguiente figura:

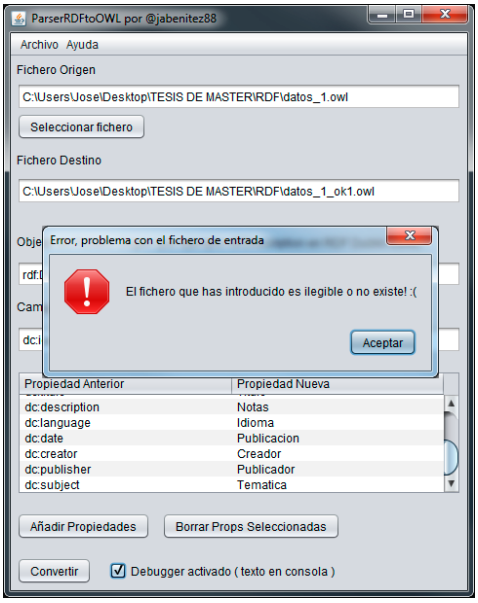

Ilustración 26 Parser RDF to OWL: Mensaje de error

4.2. MODELO ONTOLÓGICO CON PROTÉGÉ.

Una vez convertidos los ficheros con formato rdf:dc en ficheros con formato .owl, se puede trabajar con protégé. La versión utilizada para este proyecto ha sido la **Protégé v4.2.0 beta.** Cabe destacar también, que para conseguir el total funcionamiento de dicho programa, tuve que instalar el **Graph Viz** en su versión 2.28.0.

Se ejecuta el programa y se carga el fichero .owl generado (**datos_total.owl**). Para ello hay que ir al menú **File > Open** y seleccionar en la ventana de diálogo, la ubicación del, tal y como se indica en las imágenes siguientes:

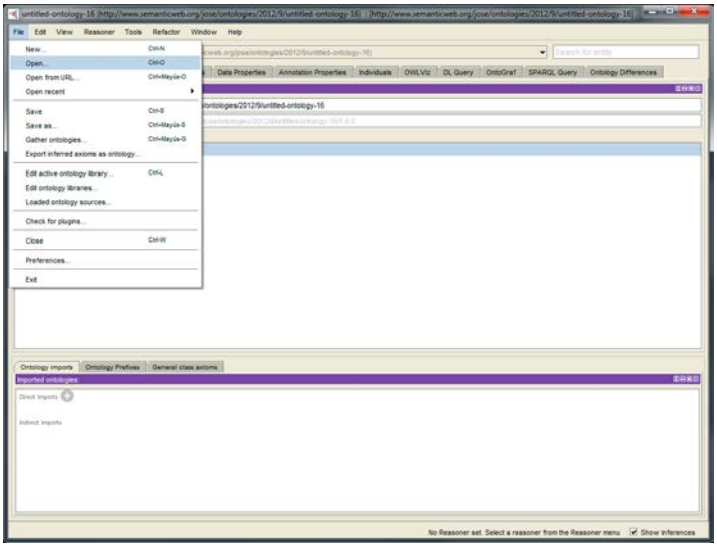

Ilustración 27 Protègè: Paso 1 - File > Open

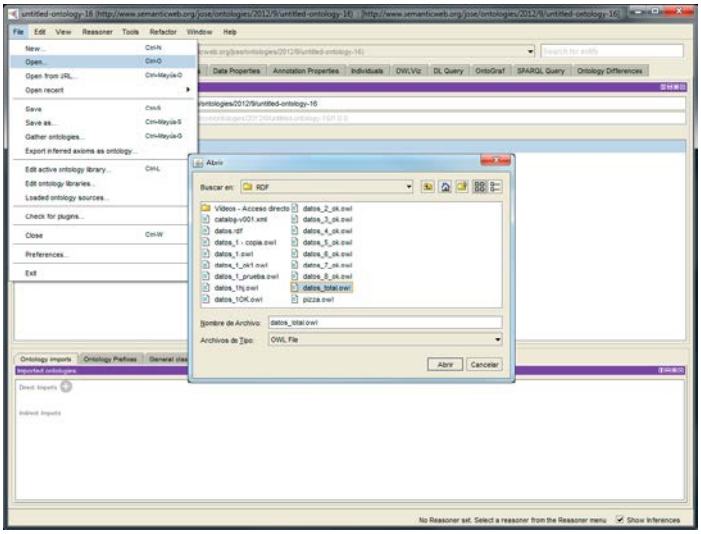

Ilustración 28 Protègè: Paso 2 - Seleccionar fichero

Después de haber abierto el fichero, obtenemos una información inicial en la pestaña “Active ontology”, en la cual podemos observar los comentarios previamente escritos en la ontología, el idioma por defecto, y la versión de la ontología.

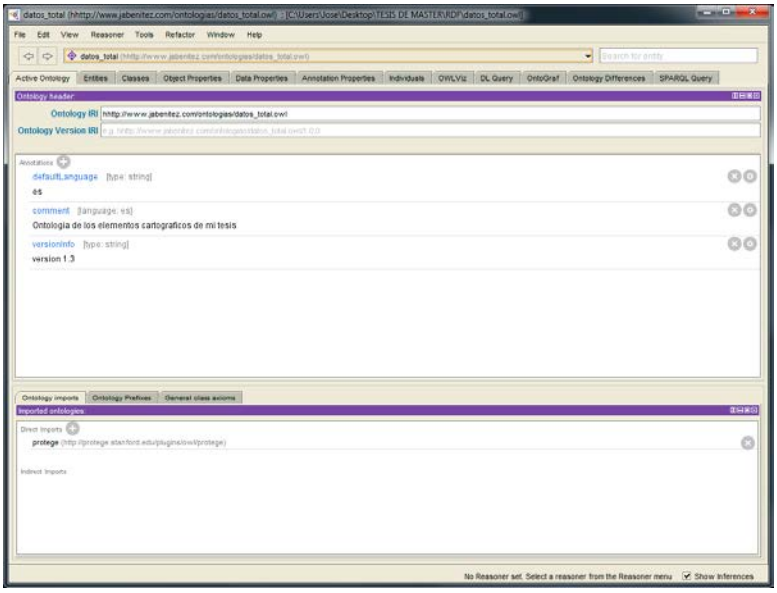

Ilustración 29 Protègè: Comentarios sobre la ontología, idioma y versión.

Si se clicka en la pestaña **Entities**, podremos ver las entidades que contiene la ontología pertenecientes a distintas clases. Observamos que hay 7792 entidades de la clase **Ficha** creadas, pertenecientes a cada una de las fichas encontradas en la web.

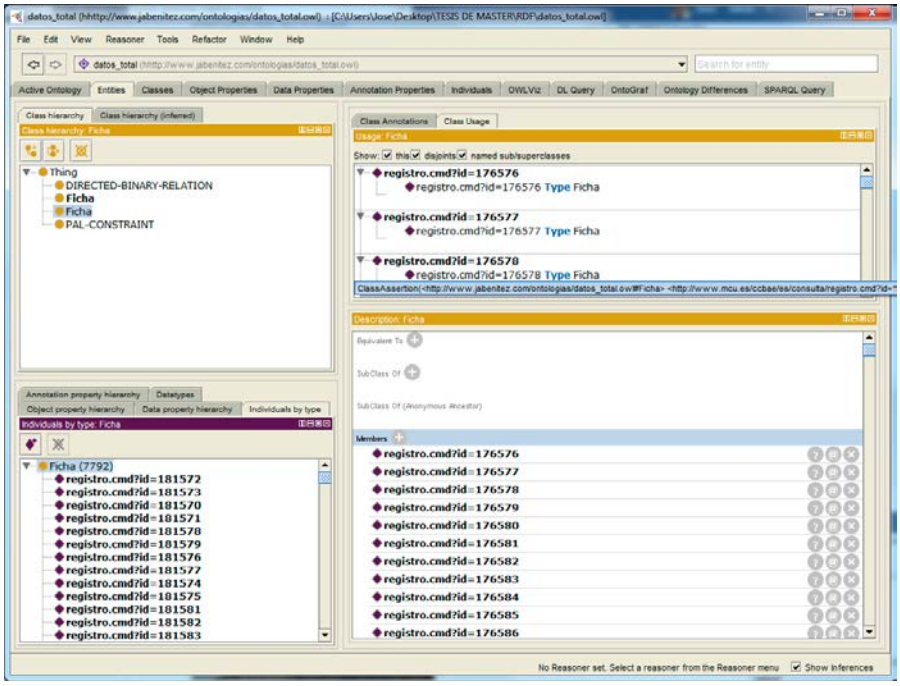

Ilustración 30 Protègè: Entidades

Si se clicka en cualquiera de los registros individuales, se observa que el programa nos muestra la información de cada una de las propiedades de esa entidad creada. Desde protégé podemos realizar búsquedas en todos los registros que hemos cargado con gran facilidad.

Protégé tiene numerosas pestañas que despliegan distinta información sobre la ontología que se haya cargado previamente. En ocasiones, varias pestañas pueden mostrar la misma información pero de diferente forma, es decir, podemos ver en otra pestaña las propiedades de cada objeto creado en nuestra ontología, profundizando en el tipo de datos que puede recibir, rango, etc.

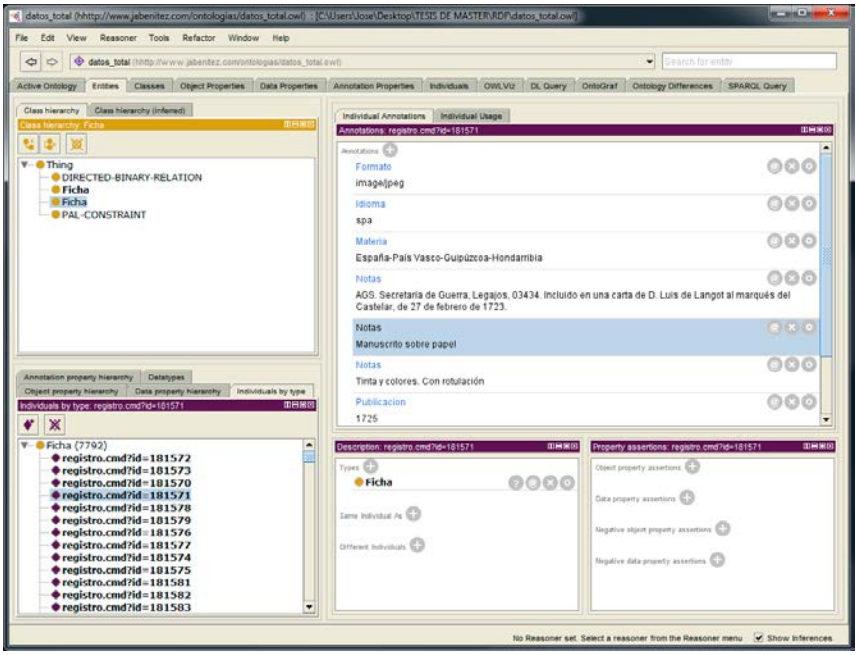

Ilustración 31 Protègè: Información de las entidades

En el caso de cada objeto creado de la clase Ficha, podemos ver que desde esta plataforma tenemos la capacidad de editar, borrar o crear propiedades de estos objetos de forma gráfica, con los 3 botones que aparecen en la parte derecha de cada propiedad.

Ilustración 32 Protègè: Propiedades de una entidad

Y en la siguiente figura se ve de forma gráfica la jerarquía de la ontología incluyendo las distintas entidades creadas en nuestro proyecto.

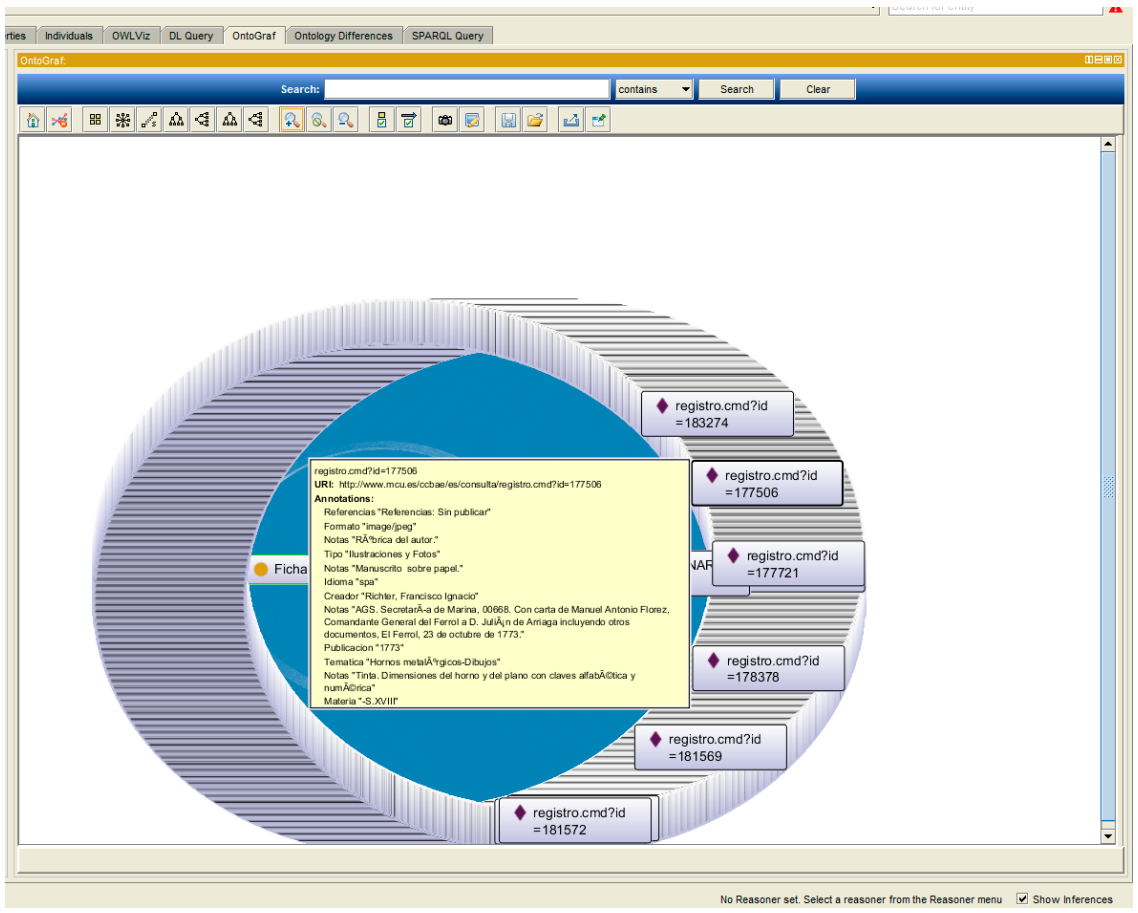

Ilustración 33 Protègè: Jerarquía de la ontología de forma gráfica

4.3. MODELO TEXTUAL CON SOLR

En el Anexo 3 se describe con detalle la forma de construir el modelo textual sobre la base de un índice (o varios) a partir de un conjunto de información estructurada y en concreto de las fichas descargadas desde el catálogo de mapas, planos y dibujos del archivo general de Simancas.

Una vez instalado Apache Solr, el siguiente paso es crear los directorios donde se almacenan las fichas categorizadas por sus características y sin categorizar, para poder hacer una comparación en la búsqueda de los resultados en ambos casos.

En el caso de las fichas facetadas, debemos especificar qué tipos de datos se almacenan para realizar las posteriores búsquedas teniendo en cuenta las distintas características. Una vez declarados los campos para almacenar la información de cada ficha hay que crear un fichero .xml con el que podamos insertar la información de las **7792 fichas** de forma rápida, quedando así indexadas en SOLR.

Antes de realizar la indexación, se debe eliminar del fichero los caracteres de control que puedan provocar un fallo por parte del indexador, lo que se soluciona ejecutando el siguiente comando:

```
tr -d [:cntrl:] < datos_facetados.xml > datos_facetados_limpio.xml
```

Teniendo en cuenta que se necesitará indexar ficheros a menudo, para facilitar se ha realizado un script en **bash** que ayudará a realizar esta tarea. Después se procederá a insertar las fichas facetadas y sin facetar en sus respectivos núcleos:

```
sh indexador.sh datos_facteados_limpio.xml fichasFacetadas
sh indexador.sh datos_liso_limpio.xml fichas
```

4.4. PROPUESTA DE BÚSQUEDA SEMÁNTICA

Se han planteado y realizado las siguientes fases en este trabajo de:

- **Análisis del problema principal a resolver**

En esta primera fase, hemos observado y analizado de manera exhaustiva el problema particular al que nos enfrentamos. La conclusión a la que hemos llegado es la siguiente: partimos de una base de datos de **7792** fichas pertenecientes al **Archivo General de Simancas** que contienen una serie de datos mediante los cuales se pueden filtrar búsquedas por diferentes campos.

Actualmente esta base de datos cuenta con un buscador bastante limitado y lento para el usuario debido a sus características, ya que, carece de analizador semántico y es complejo encontrar la información de una obra determinada introduciendo en el buscador una consulta semántica.

Concluimos entonces que el problema principal que debemos resolver es el siguiente: disponemos de una base de datos de distintas obras que contienen una gran variedad de características comunes, con valores diferentes, a la cual se tiene un acceso limitado en el momento de realizar una consulta y obtener información sobre una o varias obras determinadas por un criterio.

- **Estudio de las posibles soluciones existentes para resolver el problema de búsqueda**

Entendiendo nuestro problema principal, se plantean una serie de soluciones para resolver el problema. Realizando un análisis no muy exhaustivo, se entiende que la necesidad principal es la de obtener un modo de búsqueda semántica de la base de datos. Se pretende la obtención de la información de las obras que realmente necesite el usuario, sabiendo interpretar los criterios de búsqueda utilizados por el mismo, sin necesidad de que este conozca la estructura de del catálogo.

- **Selección de la solución que vamos a experimentar en este trabajo de investigación**

Para poder resolver el problema, se ha visto necesario la aplicación de la siguiente solución:

- Analizar la ontología en el domino particular del **Archivo General de Simancas (AGS)**.
- Crear un conjunto de consultas jerarquizadas y categorizadas de nuestro dominio particular, siguiendo las pautas que se siguen en un sistema de pregunta–respuesta, sobre la base a la información contenida dentro de la base de datos.
- Crear un parser que analice semánticamente una consulta real insertada por cualquier usuario de forma sencilla bajo una interfaz gráfica accesible, usable y funcional.
- Crear un sistema capaz de nutrir automáticamente la base de datos categorizada y que aumente el entendimiento semántico del parser a medida que la base de datos crece.

- **Elección de las herramientas necesarias para poder aplicar la solución elegida**
  Para poder aplicar esta solución, se ha llegado a la conclusión de que debemos trabajar con las siguientes herramientas y lenguajes:
  - En el campo de las ontologías, los formatos principales de los datos son los basados en **RDF**, **OWL** y en el lenguaje de consultas **SPARQL**.
  - Para poder trabajar de forma sencilla con estos estándares, existen multitud de herramientas, en este trabajo de investigación, tras una búsqueda intensa de información al respecto, se ha llegado a la conclusión de que **Protègè** y **Apache SOLR** son las herramientas necesarias para poder aplicar nuestra solución al problema particular con nuestra ontología a la hora de trabajar con la base de datos y de poder realizar las consultas pertinentes con un buen resultado de respuesta.
  - Por otra parte, después de haber realizado la investigación en lo referente al dominio particular de nuestra ontología, el siguiente paso fue elegir en qué lenguaje realizar el parser semántico. Teniendo en cuenta las posibilidades que nos proporciona Apache SOLR y los conocimientos que tiene el autor de este trabajo, se decidió crear una herramienta de búsqueda bajo la unión de los lenguajes **PHP, JavaScript, AJAX, HTML5** y **CSS3.** Concretamente, la elección de JavaScript como lenguaje principal de creación del parser, fue como consecuencia del tipo de datos que nos proporciona Apache SOLR, ya que, es capaz de enviarnos los resultados en formato **JSON**, de esta forma trabajar con JavaScript se convierte en una tarea más sencilla.

- **Desarrollo de las herramientas necesarias para abarcar el problema**
  Después de haber analizado el problema que nos encontramos, tras la posterior elección de la solución a aplicar y después de investigar cuáles eran las herramientas necesarias para poder aplicar nuestra solución, una de las fases intermedias en nuestra investigación ha sido la de realizar las aplicaciones necesarias para obtener los ficheros en formatos determinados que puedan ser utilizados por las herramientas seleccionadas.

  En la web del AGS disponemos de una herramienta de exportación de fichas, la cual ofrece la descarga de todas en ellas en diferentes formatos. Debido a las necesidades particulares de nuestro proyecto, hemos elegido como formato de descarga **RDF:DC (RDF Dublin Core)**, entendiendo que esta era la mejor opción.

  A medida que hemos avanzado con la investigación, debido a los formatos de ficheros que son comprensibles para la herramienta Protègè, se ha visto la necesidad de crear una aplicación en lenguaje **JAVA** que convierte un fichero de entrada en formato RDF:DC en un fichero de salida en formato **OWL** adecuado a un dominio particular.

  Una vez realizada esta aplicación, se realizó la conversión de las fichas obtenidas en formato RDF:DC a formato OWL y así ya pudimos trabajar con ellas desde nuestra herramienta Protègè.

Además de este programa de conversión, otra herramienta necesaria para la realización de este proyecto, es la comentada en el punto anterior, una **interfaz gráfica de usuario** mediante la cual se puedan realizar búsquedas semánticas de manera sencilla y sin tener ningún conocimiento previo de informática ni de consultas estructuradas en un lenguaje determinado.

- **Fase de aplicación de la solución, realizando la experimentación y analizando los resultados obtenidos tras la prueba.**
  Realizadas las fases anteriormente descritas de este trabajo de investigación, el siguiente paso es aplicar la solución propuesta, realizar las pruebas de experimentación necesarias y realizar un análisis de los resultados obtenidos.

  Para ello, en esta fase se han tenido que realizar las siguientes tareas:

  1. Clasificación de consultas

  La tarea de clasificación de consultas en el dominio particular de nuestra ontología, ha consistido en la realización de un análisis exhaustivo del contenido de nuestras fichas para la obtención de consultas determinadas en base a las características descritas en cada ficha. Entendiendo así, que ha sido necesario evaluar cada tipo de dato contenido en las fichas y cada categoría de dato existente para poder obtener las preguntas generales posibles que puede realizar un usuario contra la base de datos.

  La tabla de preguntas-respuestas obtenida cuenta con 5 columnas, cada una de ellas representa:

  i. **Pregunta principal**

     Hay 5 tipos de preguntas principales que puede realizar cualquier usuario, en base a ellas, existen las distintas variantes. Son indispensables para poder conocer el dato sobre el que se realiza la pregunta y el tipo de respuesta que desea obtener el usuario.

  ii. **Sujeto de nivel 1 / Predicado de nivel 1**
     Una vez tenemos la pregunta principal, el siguiente paso es analizar sobre qué predicado o sobre qué sujeto se realiza la pregunta, de esta manera se consigue concretar más el tipo de respuesta obtenido y el sujeto sobre el que se va a realizar la consulta.

  iii. **Sujeto de nivel 2 / Predicado de nivel 2**
     La tercera columna muestra un nivel superior, cerrando más los tipos de respuesta posibles a la pregunta inicial.

  iv. **Sentencia extendida**

Llegados a este nivel, disponiendo de la pregunta principal y los dos sujetos / predicados de nivel 1 y nivel 2, se escriben las distintas frases completas que pueden simular una consulta real por parte del usuario.

v. **Tipo de respuesta**
Y por último, esta columna muestra el tipo de respuesta que debe devolver nuestro sistema, basándose en las categorías de datos existentes en nuestra ontología.

2. Realización de búsquedas de elementos de forma facetada y de forma textual

Habiendo completado la tabla de preguntas-respuestas posibles a realizar por parte de un usuario a nuestra base de datos, el siguiente paso fue generar 6 consultas siguiendo la tipología creada en dicha tabla.

Después de obtener las seis consultas y comprobar de manera manual cuáles serían los resultados reales que debería devolvernos la base de datos en base a las preguntas realizadas, se procedió a la conversión de dichas consultas en formato **Apache SOLR** de dos formas distintas:

- Realizando la consulta sobre la base de datos que poseía categorías, es decir, base de datos facetada.
- Realizando la consulta sobre la base de datos textual, con los datos insertados sin facetar.

Posteriormente, se realizó el proceso de conversión de consultas en lenguaje natural, a lenguaje **SPARQL**, para que fuera posible la comparación de resultados también en el programa Protègè.

En esta fase se procedió al descarte del uso del programa Protègè en la fase final de proyecto debido a la gran versatilidad ofrecida por el sistema Apache SOLR a la hora de crear aplicaciones en las cuales fuera posible su utilización. Habiendo llegado a la conclusión de que los resultados obtenidos en Apache SOLR y las posibilidades que ofrece este sistema, son mucho mejores y más funcionales que las obtenidas con Protègè.

3. Comparación de los resultados obtenidos

Habiendo realizado las tareas anteriores, la última fase ha consistido en la realización de una comparación de resultados obtenidos en la base de datos que estaba semánticamente estructurada (es decir, facetada) y la base de datos sin facetar, que poseía un único campo de valor en el cual estaban todos los datos de las fichas.

Para la realización de esta comparativa, se crearon unos ficheros de forma manual en los cuales se indicaron, en cada una de las seis consultas, los diez primeros resultados más acertados para cada búsqueda. Posteriormente se formaron los ficheros con las respuestas a las distintas consultas en cada uno de los dos entornos y mediante la herramienta **TREC_EVAL,** se procedió a la lectura de los resultados comparativos obtenidos en ambos entornos.

Los resultados fueron positivos para la búsqueda semántica en todos los sentidos:

- La velocidad de carga de las búsquedas fue mayor, debido a que, al poseer campos categorizados, la búsqueda entre los mismos era siempre más rápida que la búsqueda global en un único campo.
- Los resultados fueron más acertados, al poseer categorías, las consultas buscaban los términos en los campos correctos, obteniendo así los resultados esperados.

El entorno facetado obtuvo una mejor valoración, teniendo en cuenta también que esta es la mejor vía para un procesado posterior de enriquecimiento semántico semiautomático realizado en un trabajo futuro sobre este proyecto de investigación.

5. EXPERIMENTACIÓN

En este trabajo de han realizado tres fases para la experimentación:

- Clasificación de consultas: En esta fase, hemos generado una tabla de preguntas de tipo Q-A, especificando los tipos de pregunta que el usuario puede plantear a la base de datos, junto con los objetos posibles mediante los cuales podría realizar la pregunta, destacando qué resultado deberíamos obtener.
- Realización de búsquedas sobre la base de datos facetada y sin facetar: Habiendo realizado una buena clasificación de consultas, el siguiente paso ha sido obtener y almacenar los distintos resultados obtenidos de dichas consultas en nuestra base de datos con datos sin facetar y con datos facetados, para así, en una última fase, poder valorar la diferencia de tiempo / calidad de respuesta en cada uno de los casos.
- Comparación de los resultados obtenidos en ambos casos: En esta fase, habiendo realizado un estudio manual de las respuestas correctas a las consultas formuladas, procedemos al análisis y evaluación de los resultados obtenidos en las consultas realizadas sobre la base de datos de fichas facetadas y sin facetar.

5.1. CLASIFICACIÓN DE CONSULTAS

La propuesta de preguntas a realizar dentro de nuestra ontología para la posterior comparación y corroboración de los diferentes resultados de las búsquedas en los términos de tiempo, recall, cobertura y precisión, son las siguientes:

Q - Clase				Respuesta
QUÉ	ficha mapa plano dibujo	Fue realizado/a hecho/a	mediante la técnica de **$TÉCNICA** con los colores **$COLOR** entre las fechas **$FECHA** y **$FECHA** en el idioma **$IDIOMA** en los idiomas **$IDIOMA** y **$IDIOMA** en la época **$ÉPOCA** con la temática **$TEMÁTICA** sobre el soporte **$SOPORTE** sobre la ciudad **$CIUDAD** sobre el país de **$PAÍS** por el autor **$AUTOR**	[Título]
	autor	realizó hizo	el mapa/plano/dibujo algún mapa/plano/dibujo • con título **$TÍTULO** • en el idioma **$IDIOMA** • mediante la técnica de **$TÉCNICA** • con los colores **$COLOR** • entre las fechas **$FECHA y $FECHA** • en el idioma **$IDIOMA** • en los idiomas **$IDIOMA y $IDIOMA** • en la época **$ÉPOCA**	[Autor]
	técnica	fue utilizada para se utilizó para		[Técnica]

			• con la temática **$TEMÁTICA** • sobre el soporte **$SOPORTE** • sobre la ciudad **$CIUDAD** • sobre el país de **$PAÍS** • por el autor **$AUTOR**	
QUIÉN	hizo / realizó / utilizó / dibujó / fabricó / publicó	El/la/los/las Algún/ alguno/ algunos/ algunas un/una/ unos/unas *planos/ dibujos mapas*	mediante la técnica de **$TÉCNICA** con los colores **$COLOR** entre las fechas **$FECHA** y **$FECHA** en el idioma **$IDIOMA** en los idiomas **$IDIOMA** y **$IDIOMA** en la época **$ÉPOCA** con la temática **$TEMÁTICA** sobre el soporte **$SOPORTE** sobre la ciudad **$CIUDAD** sobre el país de **$PAÍS**	[Autor] [Publicador] [Contribuyente]
CÓMO	se hizo / se realizó / se hicieron / se realizaron	El/la/los/las Algún/ alguno/ algunos/ algunas un/una/ unos/unas *planos/ dibujos mapas*	por el autor **$AUTOR** con el título **$TÍTULO** con los colores **$COLOR** entre las fechas **$FECHA** y **$FECHA** en el idioma **$IDIOMA** en los idiomas **$IDIOMA** y **$IDIOMA** en la época **$ÉPOCA** con la temática **$TEMÁTICA** sobre el soporte **$SOPORTE** sobre la ciudad **$CIUDAD** sobre el país de **$PAÍS**	[Técnica] [Soporte]
DÓNDE	se realizó / se utilizó / se realizó	El/la/los/las Algún/ alguno/ algunos/ algunas un/una/ unos/unas *planos/ dibujos mapas*	por el autor **$AUTOR** con el título **$TÍTULO** con los colores **$COLOR** entre las fechas **$FECHA** y **$FECHA** en el idioma **$IDIOMA** en los idiomas **$IDIOMA** y **$IDIOMA** en la época **$ÉPOCA** con la temática **$TEMÁTICA** sobre el soporte **$SOPORTE**	[Ciudad] [País] [Continente]
CUÁNDO	se realizó / se utilizó / se realizó	El/la/los/las Algún/ alguno/ algunos/ algunas un/una/	por el autor **$AUTOR** con el título **$TÍTULO** con los colores **$COLOR** entre las fechas **$FECHA** y **$FECHA** en el idioma **$IDIOMA** en los idiomas **$IDIOMA** y **$IDIOMA** en la época **$ÉPOCA** con la temática **$TEMÁTICA** sobre el soporte **$SOPORTE**	[Época] [Año]

		unos/unas *planos/ dibujos mapas*	sobre la ciudad **$CIUDAD** sobre el país de **$PAÍS**	

Tabla 8 Tipo de preguntas Q-A en nuestro dominio particular

Relación de facetas que se deben consultar en las consultas en función de lo marcado en la tabla anterior:

Variable	Campo a consultar en DC	Campo en OWL
$TÍTULO	dc:title	Título
$FECHA	dc:date	Fecha
$AUTOR	dc:autor	Autor
$CREADOR	Dc:creator	Creador
$TEMÁTICA	Dc:subject	Temática
$SOPORTE	Dc:description	Notas
$PUBLICADOR	Dc:publisher	Publicador
$FECHA **$ÉPOCA**	Dc:date, DC:description	Fecha
$TIPO	Dc:type	Tipo
$IDIOMA	Dc:language	Idioma
$REFERENCIA	Dc:relation	relacionados
$CIUDAD **$PAÍS** **$CONTINENTE**	Dc:coverage	materia

Tabla 9 Equivalencia Variable - Consulta en DC - Campo en OWL

1. Qué mapas/planos/dibujos se hicieron entre los años XX y XX

    1.a en la época XX
    1.b en el siglo XX

Relacionado principalmente con: dc:title,dc:date.

Consulta en Solr para Qué fichas datan sobre una obra realizada en 1950:

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=titulo%3A*1950*+or+materia%3A*1950*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=name%2C+titulo [EN FORMATO JSON]
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*1950*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

2.a Qué mapas/planos/dibujos están hechos en PERGAMINO/MANUSCRITO SOBRE PAPEL/ COPIA/GRABADO/IMPRESO

2.b están fabricados en ....
2.c están impresos en
2.d tienen soporte en.....

Relacionado principalmente con: dc:description (notas).

Consulta en Solr para Qué fichas están fabricadas en pergamino:

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=notas%3A*pergamino*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=name%2C+titulo [EN FORMATO JSON]
- http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=titulo%3A*1950*&fq=&start=0&rows=10&fl=titulo%2Cscore&wt=&explainOther=&hl=on&hl.fl=titulo
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*pergamino*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

3.a Qué mapas/planos/dibujos utilizan la técnica de TINTA (aguada o negra)/ tinta y colores (negro, rojo, dorado.....)/ lapiz negro --- CON O SIN ROTULACIÓN/EXPLICACIÓN

3.b están hechos con la técnica de
3.c fueron realizados con .....
3.d se crearon mediante la técnica de..

Relacionado principalmente con: dc:description.

Consulta en Solr para Qué fichas utilizan la técnica de tinta con rotulaciones

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=notas%3A*Tinta*+AND+notas%3A*rotulacion*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl= [EN FORMATO JSON]
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*Tinta*+AND+fichas%3A*rotulacion*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

4.a Qué mapas/planos/dibujos fueron creados por ...

4.b fueron hechos por ...
4.c tienen como autor a ...
4.d fueron realizados por

4.e han sido realizados por

Relacionado principalmente con: dc:creator, dc:autor (creador, contribuyente).

Consulta en Solr para Qué fichas fueron creadas por Juan Baptista

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=creador%3A*Juan*+AND+creador%3A*Baptista*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl= [EN FORMATO JSON]
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*Juan*+AND+fichas%3A*Baptista*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

5.a Qué mapas/planos/dibujos están escritos en lenguaje SPA/FRE/ENG/GER/LAT/POR/ITA/DUT/CAT

5.b tienen como idioma
5.c fueron escritos en ....

Relacionado principalmente con: dc:language (idioma).

Consulta en Solr para Qué fichas están escritas en español

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=idioma%3A*spa*+&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl= [EN FORMATO JSON]
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*spa*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

6.a Qué mapas/planos/dibujos tienen como temática (Cañones, ballenas, faunas.....)

6.b pertenecen a la temática....
6.c poseen como tema principal ....
6.d tratan sobre.....
6.e constan de la temática

Relacionado principalmente con: dc:subject (temática).

Consulta en Solr para Qué fichas poseen como temática ballenas

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select/?q=tematica%3A*Ballenas*+OR+tematica%3A*ballenas*&version=2.2&start=0&rows=10&indent=on&wt=json [EN FORMATO JSON]
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3

A*Ballenas*+OR+fichas%3A*ballenas*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

5.2. BÚSQUEDAS FACETADA Y TEXTUAL (SIN FACETAR)

5.2.1. FUNCIONAMIENTO DE CONSULTAS EN SOLR

Solr ofrece un interfaz de administración que permite hacer consultas contra los índices. A continuación se muestra cómo indicar los campos sobre los que buscar, condiciones, filtros, ordenación, depuración, y cómo cambiar el tipo de respuesta y procesarla.

Entrando en la pantalla principal del administrador de Solr, podemos cambiar de *core*, revisar su configuración y el esquema; pero sobre todo podemos hacer consultas y analizarlas. La pantalla principal incorpora un formulario rápido de consulta, pero es más útil el formulario avanzado.

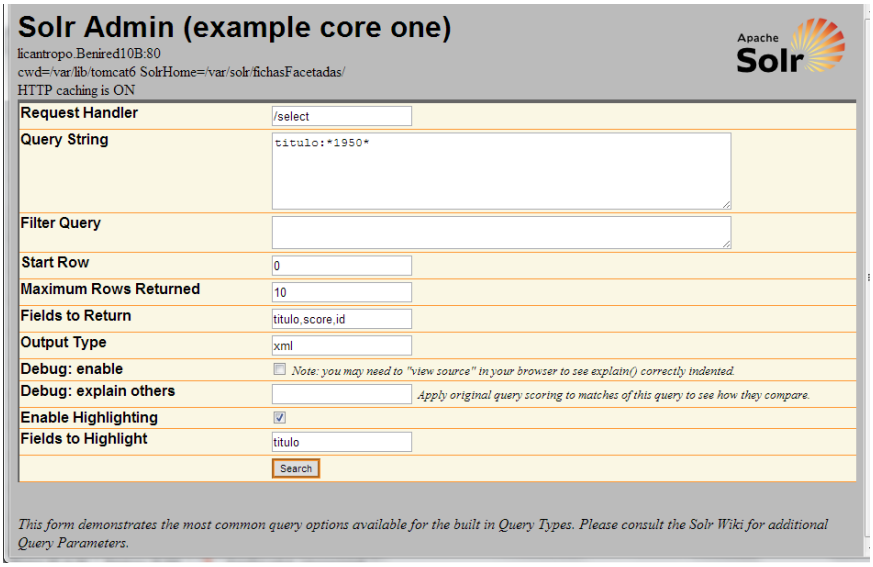

Ilustración 34 Pantalal inicial de consulta en Solr

Todas las opciones que se pueden utilizar desde la pantalla de consultas se traducen en parámetros en la url de consulta.

http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&**q=titulo%3A*1952***&fq=&start=0&rows=10&**fl=titulo%2Cscore**&wt=&explainOther=&hl=on&hl.fl=titulo

Esta consulta indica que buscamos todos los resultados que contentan en el título, la cifra 1942: **q=titulo%3ª*1942** (%3ª es el carácter : en ascii). Y además, que los campos que deseamos recibir son el campo título y la puntuación: **fl=titulo%2Cscore** (%2C es el valor del símbolo de la puntuación, en ascii)

Por defecto Solr devuelve los resultados en forma de XML, pero permite seleccionar diferentes tipos usando el parámetro **wt**.

Como ejemplo, veamos los resultados devueltos para los siguientes valores de wt:xml, json, php y ruby:

XML

```
<response>
<lst name="responseHeader">
<int name="status">0</int>
<int name="QTime">112</int>
</lst>
<result name="response" numFound="1" start="0" maxScore="1.0">
<doc>
<float name="score">1.0</float>
<str name="id">
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845
</str>
<arr name="titulo">
<str>Valladolid. 1942. Archivos. Planos</str>
<str>
Archivo General de Simancas, Nuevo edificio en Valladolid [Material
cartográfico]
</str> </arr>
</doc>
</result>
<lst name="highlighting">
<lst name="http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845">
<arr name="titulo">
<str><em>Valladolid. 1942. Archivos. Planos</em></str>
</arr>
</lst>
</lst>
</response>
```

JSON

```
{
  "responseHeader":{
    "status":0,
    "QTime":117},
  "response":{"numFound":1,"start":0,"maxScore":1.0,"docs":[
      {
        "id":"http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845",
        "titulo":["Valladolid. 1942. Archivos. Planos","Archivo General de
Simancas, Nuevo edificio en Valladolid [Material cartográfico]"],
        "score":1.0}]
  },
  "highlighting":{
    "http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845":{
      "titulo":["<em>Valladolid. 1942. Archivos. Planos</em>"]}}}
```

PHP

```
array(
  'responseHeader'=>array(
    'status'=>0,
    'QTime'=>147),
  'response'=>array('numFound'=>1,'start'=>0,'maxScore'=>1.0,'docs'=>array(
      array(
        'id'=>'http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845',
        'titulo'=>array('Valladolid. 1942. Archivos. Planos','Archivo General
de Simancas, Nuevo edificio en Valladolid [Material cartográfico]'),
        'score'=>1.0))
  ),
  'highlighting'=>array(
    'http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845'=>array(
      'titulo'=>array('<em>Valladolid. 1942. Archivos. Planos</em>'))))
```

RUBY ON RAILS

```
{
  'responseHeader'=>{
    'status'=>0,
    'QTime'=>136},
  'response'=>{'numFound'=>1,'start'=>0,'maxScore'=>1.0,'docs'=>[
      {
        'id'=>'http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845',
        'titulo'=>['Valladolid. 1942. Archivos. Planos','Archivo General de
Simancas, Nuevo edificio en Valladolid [Material cartográfico]'],
        'score'=>1.0}]
  },
  'highlighting'=>{
    'http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845'=>{
      'titulo'=>['<em>Valladolid. 1942. Archivos. Planos</em>']}}}
```

En todos los casos la estructura es la misma, una sección ***header*** en la que nos devuelve los parámetros de configuración que le hemos pasado y ***response*** con los documentos. A estas estructuras básicas se le van añadiendo otras colecciones como resultado de operaciones de facetación, highlighting y otros.

Se pueden encontrar librerías para facilitar la tarea de consulta y procesado de la respuesta desde distintos lenguajes de programación: java, php, .net, ruby, python, perl, etc. Antes de decantarse por la adopción de una librería, es importante comprobar que es capaz de usar todas las funcionalidades de Solr, y de que realmente ayuda; ya que a veces sólo introducen una capa más que aprender, debes aprender a usar Solr y además a hacer que la librería transmita al buscador lo que quieres. También podemos encontrar componentes ya creados para integrar Solr con aplicaciones conocidas.

Los parámetros básicos para la búsqueda en Solr son los siguientes:

q (query) Especifica la consulta de búsqueda.

fq (filter query) Permite filtrar los resultados de la búsqueda según criterios.

sort (ordenación) Ordena de forma ascendente o descendente.

fl (fields) Permite especificar los campos que va a devolver Solr. Por defecto *, score

wt (writer type) Indica cuál es el procesador de salida que componga la cadena de respuesta. por defecto wt=xml.

Start Indica la primera fila a devolver del conjunto de elementos resultantes (comienzo de página).

Rows Número máximo de elementos a devolver del resultado (elementos por página).

omitHeader Permite obviar el elemento header en la respuesta.

Los resultados de una búsqueda dependen, tanto de la consulta, como de la forma en que se han indexado los datos. Por ejemplo, cuando buscamos sobre un campo de tipo string o uno numérico, no se hace ningún tipo de búsqueda por aproximación. En el caso de un número parece obvio, pero en los campos de texto el resultado depende mucho de cómo se esté analizando el texto en el momento de la indexación.

Los campos String sólo admiten búsquedas literales, no harán búsquedas parciales ni cambiarán el "case" de un texto para encontrarlo; por otro lado un campo de texto que esté tokenizando el contenido, no servirá para hacer filtros ya que aplicará búsquedas por aproximación. Esto no debe ser un problema, podemos usar ***copyfields*** y mantener el contenido en campos con diferentes tipos de indexación.

Solr tiene dos tipos de sintaxis de consulta que se indican el parámetro defType. El tipo por defecto es LuceneQParser (Búsqueda de Lucene).

Las búsquedas se definen con los parámetros ***q*** y ***fq***, veamos unos ejemplos:

q=titulo:*1950	q= titulo:*1950	q=*.*
fq=tipo:Ilustracion	fq=-tipo:Ilustracion	fq=tipo:Mapa
Documentos que contengan "1950" en el campo título y que sean de tipo Ilustración.	Documentos que contengan "1950" en el campo título y que NO sean de tipo Ilustración.	Todos los documentos que sean de tipo Mapa

Aquellas consultas que incluyan cualquiera de los caracteres reservados de Lucene: + - && || ! ( ) { } [ ] ^ " ~ * ? : \ deberán escaparse utilizando o entrecomillando la consulta.

q=\[opcional\]
q="Atencion!"

Cuando se hace una consulta al índice, si no se dice lo contrario, se utilizarán los valores por defecto definidos en el fichero schema.xml en los campos defaultSearchField y defaultOperator.

Podemos indicar varios campos sobre los que hacer la búsqueda, e indicar el operador lógico si es distinto al configurado por defecto en el esquema. Por ejemplo:

q=título:*1950* OR descripción:*1950* OR fecha:*1950* fq=idioma:spa
Documentos escritos en español que contengan "1950" en cualquiera de los campos: título, descripción, fecha.

Es importante saber cómo evalúa Solr las consultas para entender los resultados. SE activa la depuración de consulta añadiendo el parámetro debugQuery=on; nos devuelve una estructura como la que sigue, al final de la respuesta normal de la consulta:


```
{
  "responseHeader":{
    "status":0,
    "QTime":175},
  "response":{"numFound":1,"start":0,"maxScore":1.0,"docs":[
      {
        "id":"http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845",
        "titulo":["Valladolid. 1942. Archivos. Planos","Archivo General de
Simancas, Nuevo edificio en Valladolid [Material cartográfico]"],
        "score":1.0}]
  },
  "highlighting":{
    "http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845":{
      "titulo":["<em>Valladolid. 1942. Archivos. Planos</em>"]}},
  "debug":{
    "rawquerystring":"titulo:*1942*",
    "querystring":"titulo:*1942*",
    "parsedquery":"titulo:*1942*",
    "parsedquery_toString":"titulo:*1942*",
    "explain":{
      "http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183845":"\n1.0 =
(MATCH) ConstantScore(titulo:Valladolid. 1942. Archivos. Planos), product of:\n
1.0 = boost\n  1.0 = queryNorm\n"},
    "QParser":"LuceneQParser",
    "timing":{
      "time":175.0,
      "prepare":{
        "time":1.0,
         "org.apache.solr.handler.component.QueryComponent":{
           "time":1.0},
```

```
        "org.apache.solr.handler.component.QueryComponent":{
         "time":1.0},
        "org.apache.solr.handler.component.FacetComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.MoreLikeThisComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.HighlightComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.StatsComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.DebugComponent":{
         "time":0.0}},
      "process":{
        "time":174.0,
        "org.apache.solr.handler.component.QueryComponent":{
         "time":85.0},
        "org.apache.solr.handler.component.FacetComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.MoreLikeThisComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.HighlightComponent":{
         "time":44.0},
        "org.apache.solr.handler.component.StatsComponent":{
         "time":0.0},
        "org.apache.solr.handler.component.DebugComponent":{
         "time":45.0}}}}}
```


La ordenación por defecto de Solr es por score en descendente; es decir que muestra primero los elementos que considera más relevantes. Podemos ordenar por otros campos y esto no afecta al subconjunto de datos devueltos por la query (completo, no sólo la página actual); sólo al orden de presentación.

Se puede indicar indicar uno o varios criterios separados por comas:

&sort=date asc

La ordenación dependerá del modo en que esté indexando el dato. Un String distingue entre mayúsculas y minúsculas y las trata como caracteres distintos, si queremos una ordenación que encaje con las expectativas del usuario, deberemos hacerlo con un campo de texto que incorpore un ***LowerCaseFilterFactory*** al indexar el contenido.

5.2.2. PARSER SEMÁNTICO QUE TRADUCE CONSULTAS A LENGUAJE SOLR

La interfaz de búsqueda está realizada con los lenguajes de programación **PHP, HTML5, CSS3, JAVASCRIPT y AJAX** (realizando consultas de forma directa a nuestro servidor de Apache Solr recibiendo los datos con la estructura **JSON** para poder trabajar con los datos en JavaScript). En el ANEXO 4 se encuentra los detalles de la implementación.

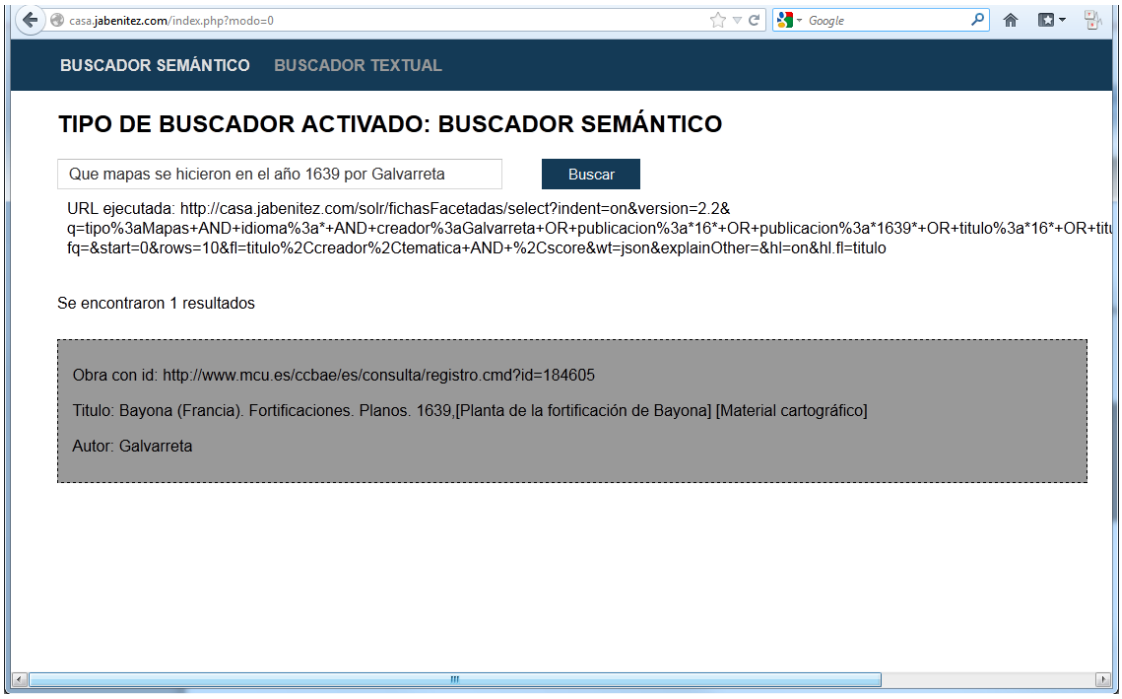

Ilustración 35 Pantalla inicial del parser Semántico propio

Han sido creados dos tipos de buscador: un buscador textual, en el cual, se recogen los datos introducidos por el usuario en el cuadro de búsqueda y se realiza una consulta a nuestra base de datos almacenada en Solr sin facetar ; y un buscador semántico, mediante el cual, con un parser realizado en JavaScript, PHP y AJAX, se recibe la cadena de texto introducida por el usuario en el campo de texto y se interpreta mediante una serie de funciones la semántica de la frase, pudiendo así responder de forma natural a las necesidades que realmente tiene el usuario.

buscadorTextual.js

El buscador textual tiene un funcionamiento sencillo, el proceso que se sigue para realizar la consulta es el siguiente:

- En primer lugar creamos un objeto de tipo **Ajax** mediante el cual realizaremos una consulta a nuestro Solr, recibiendo la respuesta de Solr en formato JSON para poder mostrarlo en pantalla desde nuestro script realizado en JavaScript.
- Una vez hemos creado el objeto, recibimos y almacenamos en una variable llamada **datos_formulario** la cadena de texto insertada por el usuario en el cuadro de búsqueda.

- Obtenida la cadena, asignamos a la propiedad **url** de nuestro objeto en AJAX, el siguiente valor:

  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q='+**datos_formulario**+'&fq=&start=0&rows=7500&fl=id%2Cscore&wt=json&explainOther=&hl=on&hl.fl=titulo

- Esta cadena realiza la consulta de forma directa a nuestra base de datos almacenada en Solr, enviándole como consulta la cadena de texto insertada por el usuario, buscando así, la cadena de texto introducida en toda nuestra base de datos del AGS.
- De esta manera, recibiremos una respuesta de las fichas que contienen el texto insertado el usuario.
- Al ser una búsqueda de tipo textual, este buscador no cumple con los requisitos de búsqueda semántica, lo que dificulta su uso para una persona que desconoce el verdadero contenido de la base de datos.
- Un ejemplo práctico: Supongamos que el usuario desea tener la siguiente información: **¿Qué mapas fueron realizados en el año 1950 por el autor Juan Baptista?**
  En el caso de nuestro buscador semántico, recogerá la cadena de texto introducida por el usuario y nos devolverá los resultados de buscar esa cadena completa en nuestra base de datos, no sabiendo interpretar qué datos realmente necesita conocer el usuario ni de dónde debe obtenerlos.

buscadorSemantico.js

En oposición a nuestro buscadorTextual.js, con el buscador semántico pretendemos facilitar al usuario de a pie la realización de búsquedas, realizando una evaluación semántica de la frase que recibamos del cuadro de búsqueda.

El funcionamiento de este buscador tiene distintas fases, posee una complejidad bastante más elevada que el buscadorTextual.

Su funcionamiento se divide en las siguientes fases:

1) Teniendo en cuenta el sistema QA descrito en el punto 5.1. de este trabajo, tenemos constancia de que el usuario puede realizar 5 tipos de consulta diferenciadas por la pregunta que realice: qué, quién, cómo, dónde y cuándo. Cada pregunta va a devolver un tipo de dato distinto almacenado en nuestra base de datos, con lo cual, sabemos lo siguiente:
   a. *Qué*: si el usuario realiza una consulta con este término, los datos que deberemos devolver serán los almacenados como tipo de dato **título, autor** o **técnica**.
   b. *Quién:* devolverá datos pertenecientes a los campos **autor, contribuyente o publicador.**
   c. *Cómo:* recibiremos valores de los campos **técnica** o **soporte**, que responderían a “cómo se realizó/hizo una obra/mapa/revista.”

d. *Dónde:* en consultas con este término, el sistema devolverá una respuesta obtenida de los campos **ciudad, país** o **continente.**
e. *Cuándo:* si realizamos una consulta con este término, recibiremos una respuesta contenida en los campos **época** o **año**.

Para conseguir esto, en primer lugar hemos creado una constante que almacena estos 5 tipos de pregunta con sus posibles variantes, de manera que, una vez recibida la cadena de texto insertada por el usuario, en una primera evaluación de la frase, buscaremos estos términos para poder generar una consulta que nos devuelva los campos que realmente necesitamos. De esta forma, si el usuario inserta una consulta como la siguiente: **¿Qué mapas se realizaron en el año 1950?**, nuestro parser en una primera fase leerá la cadena de texto, detectará la palabra "Qué" y almacenará en otra variable los posibles valores a mostrar (en este caso, **título, autor o año**).

2) En una segunda fase, una vez hemos evaluado el tipo de pregunta que realiza el usuario nuestro parser realiza un análisis del objeto al que hace referencia nuestra pregunta, detectando si se trata de un **autor,** un **mapa,** una **obra,** un **dibujo** o cualquiera de los campos descritos en la segunda columna de la tabla indicada en este trabajo en el punto 5.1.
Una vez se ha detectado cualquiera de estas palabras, nuestro parser sabra de qué **tipo** es objeto que debe devolver. En el caso del ejemplo propuesto, nuestro parser ya ha podido descartar un gran número de elementos en su búsqueda, limitando la consulta a devolver resultados que pertenecen al tipo **mapas** de nuestra base de datos solar y además ha podido concluir que el dato principal solicitado, es el título en este caso.

3) En una tercera fase de la consulta, nuestro parser detecta el verbo de la frase y el sujeto al que se hace referencia, pudiendo así detectar nuevos campos en los que deberá realizar su consulta. En el caso de nuestro ejemplo particular, analizaría la cadena de texto **"se realizaron en el año 1950"**, siendo capaz de detectar que la consulta debe realizarla en el campo **fecha** de nuestra base de datos Solr, con el dato **1950**.
Este proceso funciona de la siguiente manera:
   - Al detectar la frase "se realizaron en el año", nuestro parser llama a una función que realiza una consulta a la base de datos Solr, almacenando en un array todos los valores de tipo **fecha** que existen actualmente en la base de datos.
   - Una vez generado ese array, nuestro parser realiza un segundo análisis, en el que compara la cadena de texto **1950** con los valores obtenidos en ese array.
   - Ese array almacenaba las diferentes IDs de las obras/mapas/revistas que contenían la fecha 1950, con lo cual, después de realizar esta comparativa, almacenamos en un nuevo array los valores de los campos consultados que poseen como fecha de creación **1950**

4) Finalmente, habiendo realizado las fases anteriores, el objeto que habíamos creado en AJAX, el cual realiza las distintas consultas a nuestra base de datos Solr mediante JSON, muestra los datos almacenados en el array de resultados, con una apariencia específica y amigable para el usuario.

5.3 COMPARACIÓN DE LOS RESULTADOS OBTENIDOS EN BÚSQUEDAS TEXTUALES Y FACETADAS

Antes de realizar la comparativa de las distintas consultas en las dos bases de datos, a continuación explicaré las medidas de evaluación que se utilizan para valorar la eficiencia de las búsquedas con la herramienta **TREC_EVAL.**

5.3.1. MEDIDAS DE EVALUACIÓN

Debido a que generalmente se trabaja con colecciones muy grandes de información, es necesario determinar unos parámetros de evaluación que podrán variar en función del objetivo con el que se desarrolle el sistema de recuperación.

Para obtener estos parámetros en una evaluación concreta, se utilizan estándares, como por ejemplo el trec_eval (http://trec.nist.gov/trec_eval/index.html). Esta herramienta es la utilizada por la comunidad TREC para evaluar herramientas de recuperación a partir de un fichero de resultados con un formato determinado y de un conjunto de resultados prejuzgados.

En la siguiente figura se puede ver un ejemplo del resultado que se obtiene al ejecutar esta herramienta para una consulta cuyo ID es 1.

num_ret	1	200	
num_rel	1	35	
num_rel_ret	1	34	
map	1	0.3123	
Rprec	1	0.4571	
bpref		1	0.9714
recip_rank	1	0.5000	
iprec_at_recall_0.00		1	0.5000
iprec_at_recall_0.10		1	0.5000
iprec_at_recall_0.20		1	0.5000
iprec_at_recall_0.30		1	0.5000
iprec_at_recall_0.40		1	0.5000
iprec_at_recall_0.50		1	0.3846
iprec_at_recall_0.60		1	0.2500
iprec_at_recall_0.70		1	0.2393
iprec_at_recall_0.80		1	0.2393
iprec_at_recall_0.90		1	0.2286
iprec_at_recall_1.00		1	0.0000
P_5		1	0.4000
P_10	1	0.2000	
P_15	1	0.2000	
P_20	1	0.3500	
P_30	1	0.4667	
P_100		1	0.2300
P_200		1	0.1700
P_500	1	0.0680	

P_1000	1	0.0340

Tabla 10 Resultado de trec_eval

Las medidas que puede calcular trec_eval son:

- **num_ret:** número total de documentos recuperados.
- **num_rel:** número total de documentos relevantes.
- **num_rel_ret:** número total de documentos relevantes recuperados.
- **map:** promedio de precisión media (*Mean Average Precision*, MAP). Con esta medida se puede determinar la robustez de un sistema de recuperación.
- **gmap:** precisión media. La precisión se calcula después de que cada documento relevante es recuperado. Si no se recupera un documento relevante, la precisión es 0.0. Finalmente, se hace una media de todos los valores de precisión calculados para obtener un único valor.
- **Rprec:** *R-Precision*. Se trata de la precisión después de R (número de documentos relevantes) documentos recuperados.
- **bpref:** preferencia binaria, se trata de los primeros R documentos juzgados como no relevantes.
- **recip_rank:** ranking recíproco de los primeros documentos relevantes.
- **ircl_prn.X:** *recall* interpolado. Son medidas de la precisión (porcentaje de documentos recuperados que son relevantes) en varios niveles de *recall* (después de un cierto porcentaje de que todos los documentos relevantes para una consulta hayan sido recuperados). Interpolado significa que, por ejemplo, precisión en el *recall* 0.10 (después de que un 10% de los documentos relevantes hayan sido recuperados) se toma para ser el máximo de precisión en todos los puntos de *recall* mayor de .010. Estos valores se utilizan para los gráficos de *Recall-Precision*. Se proporcionan valores para X=0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90 y 1.00.
- **PX:** precisión (porcentaje de documentos recuperados que son relevantes) después de que X documentos (tanto relevantes como no relevantes) hayan sido recuperados. Si Y documentos no se recuperan para una consulta, entonces todos los documentos perdidos se consideran no relevantes. Se proporcionan valores para X=5, 10, 15, 20, 30, 100, 200, 500 y 1000.

Estas medidas se pueden calcular al mismo tiempo para un conjunto de consultas, y la herramienta trec_eval puede devolver los valores para cada consulta de forma individual junto con una media del total y también para una media del conjunto de consultas únicamente.

Otra medida de evaluación es el *subtopic recall* (Zhai, Cohen, & Lafferty, 2003), también conocida como **S-recall**. Esta medida se crea para evaluar la cantidad de documentos recuperados de cada uno de los subconjuntos que abarca una determinada consulta, es decir, la diversidad de los resultados recuperados. Esto significa que la utilidad de un documento en

un ranking depende de la utilidad del resto de documentos. Esta medida se puede calcular utilizando la herramienta ClusterEval (http://imageclef.org/ClusterEval).

5.3.2. RESULTADOS Y SU COMPARACIÓN

Teniendo en cuenta los diferentes buscadores que hemos obtenido tras realizar un análisis específico de nuestro caso particular, a continuación detallaré en una tabla los distintos resultados obtenidos de las consultas realizadas en la base de datos de las fichas sin facetar y facetadas.

Para ello hemos necesitado obtener, de cada consulta dos ficheros:

- Qrel.txt: Contiene una estructura del tipo

query_id	iteración	docid	rd
001	0	REGISTRO1	1

  Query_id: Representa el identificador de la consulta que realizamos.
  Iteración : Si la consulta se ha repetido varias veces, indica el número de iteración en el que se obtuvieron esos resultados.
  Docid: Muestra el identificador del documento recuperado.
  Rd: Es un booleano que representa si ese documento se encontró en esa búsqueda o no (en nuestro caso siempre tendrá valor 1).
- Data.txt: Posee una estructura del tipo:

Query_id	iteración	docid	rank	score	run_id
001	Q0	REGISTRO1	1	0.5	0

  Rank: representa la posición del documento en la lista de resultados tras la búsqueda realizada.
  Score: puntuación que valora la relevancia del documento con la búsqueda realizada.
  Run_id: nombre del identificador de la ejecución, no utilizado en las comparaciones que queremos realizar.

A continuación se muestra para cada consulta los resultados obtenidos.

1. Relacionado principalmente con: dc:title,dc:date.

Consulta en Solr para ¿**Qué fichas datan sobre una obra realizada en 1950?**:

- fichasFacetadas
  http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=titulo%3A*1950*+or+materia%3A*1950*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=name%2C+titulo
- fichasSinFacetar:
  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*1950*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

Búsqueda facetada			Busqueda sin facetar		
num_ret	001	4	num_ret	001	9
num_rel	001	4	num_rel	001	9
num_rel_ret	001	4	num_rel_ret	001	9
map	001	1.0000	map	001	1.0000
Rprec	001	1.0000	Rprec	001	1.0000
bpref	001	1.0000	bpref	001	1.0000
recip_rank	001	1.0000	recip_rank	001	1.0000
iprec_at_recall_0.00	001	1.0000	iprec_at_recall_0.00	001	1.0000
iprec_at_recall_0.10	001	1.0000	iprec_at_recall_0.10	001	1.0000
iprec_at_recall_0.20	001	1.0000	iprec_at_recall_0.20	001	1.0000
iprec_at_recall_0.30	001	1.0000	iprec_at_recall_0.30	001	1.0000
iprec_at_recall_0.40	001	1.0000	iprec_at_recall_0.40	001	1.0000
iprec_at_recall_0.50	001	1.0000	iprec_at_recall_0.50	001	1.0000
iprec_at_recall_0.60	001	1.0000	iprec_at_recall_0.60	001	1.0000
iprec_at_recall_0.70	001	1.0000	iprec_at_recall_0.70	001	1.0000
iprec_at_recall_0.80	001	1.0000	iprec_at_recall_0.80	001	1.0000
iprec_at_recall_0.90	001	1.0000	iprec_at_recall_0.90	001	1.0000
iprec_at_recall_1.00	001	1.0000	iprec_at_recall_1.00	001	1.0000
P_5	001	0.8000	P_5	001	1.0000
P_10	001	0.4000	P_10	001	0.9000
P_15	001	0.2667	P_15	001	0.6000
P_20	001	0.2000	P_20	001	0.4500
P_30	001	0.1333	P_30	001	0.3000
P_100	001	0.0400	P_100	001	0.0900
P_200	001	0.0200	P_200	001	0.0450
P_500	001	0.0080	P_500	001	0.0180
P_1000	001	0.0040	P_1000	001	0.0090

Tabla 11 Resultados Trec Eval - Consulta 1

2. Relacionado principalmente con: dc:description (notas).

Consulta en Solr para ¿**Qué fichas están fabricadas en pergamino**?:

- fichasFacetadas
  http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=notas%3A*pergamino*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=name%2C+titulo
- fichasSinFacetar:
  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*pergamino*&fq=&start=0&rows=30&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

Búsqueda facetada			Busqueda sin facetar		
num_ret	002	21	num_ret	002	21
num_rel	002	21	num_rel	002	21
num_rel_ret	002	21	num_rel_ret	002	21
map	002	1.0000	map	002	1.0000
Rprec	002	1.0000	Rprec	002	1.0000
bpref	002	1.0000	bpref	002	1.0000
recip_rank	002	1.0000	recip_rank	002	1.0000
iprec_at_recall_0.00	002	1.0000	iprec_at_recall_0.00	002	1.0000
iprec_at_recall_0.10	002	1.0000	iprec_at_recall_0.10	002	1.0000
iprec_at_recall_0.20	002	1.0000	iprec_at_recall_0.20	002	1.0000
iprec_at_recall_0.30	002	1.0000	iprec_at_recall_0.30	002	1.0000
iprec_at_recall_0.40	002	1.0000	iprec_at_recall_0.40	002	1.0000
iprec_at_recall_0.50	002	1.0000	iprec_at_recall_0.50	002	1.0000
iprec_at_recall_0.60	002	1.0000	iprec_at_recall_0.60	002	1.0000
iprec_at_recall_0.70	002	1.0000	iprec_at_recall_0.70	002	1.0000
iprec_at_recall_0.80	002	1.0000	iprec_at_recall_0.80	002	1.0000
iprec_at_recall_0.90	002	1.0000	iprec_at_recall_0.90	002	1.0000
iprec_at_recall_1.00	002	1.0000	iprec_at_recall_1.00	002	1.0000
P_5	002	1.0000	P_5	002	1.0000
P_10	002	1.0000	P_10	002	1.0000
P_15	002	1.0000	P_15	002	1.0000
P_20	002	1.0000	P_20	002	1.0000
P_30	002	0.7000	P_30	002	0.7000
P_100	002	0.2100	P_100	002	0.2100
P_200	002	0.1050	P_200	002	0.1050
P_500	002	0.0420	P_500	002	0.0420
P_1000	002	0.0210	P_1000	002	0.0210

Tabla 12 Resultados Trec Eval - Consulta 2

3. Relacionado principalmente con: dc:description.

Consulta en Solr para **¿Qué fichas utilizan la técnica de tinta con rotulaciones?**

- fichasFacetadas
  http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=notas%3A*Tinta*+AND+notas%3A*rotulacion*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=
- fichasSinFacetar:
  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*Tinta*+AND+fichas%3A*rotulacion*&fq=&start=0&rows=140&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=fichas

Búsqueda facetada			Busqueda sin facetar		
num_ret	003	139	num_ret	003	139
num_rel	003	139	num_rel	003	140
num_rel_ret	003	139	num_rel_ret	003	139
map	003	1.0000	map	003	0.9929
Rprec	003	1.0000	Rprec	003	0.9929
bpref	003	1.0000	bpref	003	0.9929
recip_rank	003	1.0000	recip_rank	003	1.0000
iprec_at_recall_0.00	003	1.0000	iprec_at_recall_0.00	003	1.0000
iprec_at_recall_0.10	003	1.0000	iprec_at_recall_0.10	003	1.0000
iprec_at_recall_0.20	003	1.0000	iprec_at_recall_0.20	003	1.0000
iprec_at_recall_0.30	003	1.0000	iprec_at_recall_0.30	003	1.0000
iprec_at_recall_0.40	003	1.0000	iprec_at_recall_0.40	003	1.0000
iprec_at_recall_0.50	003	1.0000	iprec_at_recall_0.50	003	1.0000
iprec_at_recall_0.60	003	1.0000	iprec_at_recall_0.60	003	1.0000
iprec_at_recall_0.70	003	1.0000	iprec_at_recall_0.70	003	1.0000
iprec_at_recall_0.80	003	1.0000	iprec_at_recall_0.80	003	1.0000
iprec_at_recall_0.90	003	1.0000	iprec_at_recall_0.90	003	1.0000
iprec_at_recall_1.00	003	1.0000	iprec_at_recall_1.00	003	0.0000
P_5	003	1.0000	P_5	003	1.0000
P_10	003	1.0000	P_10	003	1.0000
P_15	003	1.0000	P_15	003	1.0000
P_20	003	1.0000	P_20	003	1.0000
P_30	003	1.0000	P_30	003	1.0000
P_100	003	1.0000	P_100	003	1.0000
P_200	003	0.6950	P_200	003	0.6950
P_500	003	0.2780	P_500	003	0.2780
P_1000	003	0.1390	P_1000	003	0.1390

Tabla 13 Resultados Trec Eval - Consulta 3

4. Relacionado principalmente con: dc:creator, dc:autor (creador, contribuyente).

Consulta en Solr para **¿Qué fichas fueron creadas por Juan Baptista?**

- fichasFacetadas
  http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=creador%3A*Juan*+AND+creador%3A*Baptista*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=
- fichasSinFacetar:
  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*Juan*+AND+fichas%3A*Baptista*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

Búsqueda facetada			Busqueda sin facetar		
num_ret	004	1	num_ret	004	3
num_rel	004	1	num_rel	004	3
num_rel_ret	004	1	num_rel_ret	004	3
map	004	1.0000	map	004	1.0000
Rprec	004	1.0000	Rprec	004	1.0000
bpref	004	1.0000	bpref	004	1.0000
recip_rank	004	1.0000	recip_rank	004	1.0000
iprec_at_recall_0.00	004	1.0000	iprec_at_recall_0.00	004	1.0000
iprec_at_recall_0.10	004	1.0000	iprec_at_recall_0.10	004	1.0000
iprec_at_recall_0.20	004	1.0000	iprec_at_recall_0.20	004	1.0000
iprec_at_recall_0.30	004	1.0000	iprec_at_recall_0.30	004	1.0000
iprec_at_recall_0.40	004	1.0000	iprec_at_recall_0.40	004	1.0000
iprec_at_recall_0.50	004	1.0000	iprec_at_recall_0.50	004	1.0000
iprec_at_recall_0.60	004	1.0000	iprec_at_recall_0.60	004	1.0000
iprec_at_recall_0.70	004	1.0000	iprec_at_recall_0.70	004	1.0000
iprec_at_recall_0.80	004	1.0000	iprec_at_recall_0.80	004	1.0000
iprec_at_recall_0.90	004	1.0000	iprec_at_recall_0.90	004	1.0000
iprec_at_recall_1.00	004	1.0000	iprec_at_recall_1.00	004	1.0000
P_5	004	0.2000	P_5	004	0.6000
P_10	004	0.1000	P_10	004	0.3000
P_15	004	0.0667	P_15	004	0.2000
P_20	004	0.0500	P_20	004	0.1500
P_30	004	0.0333	P_30	004	0.1000
P_100	004	0.0100	P_100	004	0.0300
P_200	004	0.0050	P_200	004	0.0150
P_500	004	0.0020	P_500	004	0.0060
P_1000	004	0.0010	P_1000	004	0.0030

Tabla 14 Resultados Trec Eval - Consulta 4

5. Relacionado principalmente con: dc:language (idioma).

Consulta en Solr para **¿Qué fichas están escritas en español?**

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&version=2.2&q=idioma%3A*spa*+&fq=&start=0&rows=7200&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl=
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*spa*&fq=&start=0&rows=7200&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

Búsqueda facetada			Busqueda sin facetar		
num_ret	005	7145	num_ret	005	5603
num_rel	005	7145	num_rel	005	5603
num_rel_ret	005	7145	num_rel_ret	005	5603
map	005	1.0000	map	005	1.0000
Rprec	005	1.0000	Rprec	005	1.0000
bpref	005	1.0000	bpref	005	1.0000
recip_rank	005	1.0000	recip_rank	005	1.0000
iprec_at_recall_0.00	005	1.0000	iprec_at_recall_0.00	005	1.0000
iprec_at_recall_0.10	005	1.0000	iprec_at_recall_0.10	005	1.0000
iprec_at_recall_0.20	005	1.0000	iprec_at_recall_0.20	005	1.0000
iprec_at_recall_0.30	005	1.0000	iprec_at_recall_0.30	005	1.0000
iprec_at_recall_0.40	005	1.0000	iprec_at_recall_0.40	005	1.0000
iprec_at_recall_0.50	005	1.0000	iprec_at_recall_0.50	005	1.0000
iprec_at_recall_0.60	005	1.0000	iprec_at_recall_0.60	005	1.0000
iprec_at_recall_0.70	005	1.0000	iprec_at_recall_0.70	005	1.0000
iprec_at_recall_0.80	005	1.0000	iprec_at_recall_0.80	005	1.0000
iprec_at_recall_0.90	005	1.0000	iprec_at_recall_0.90	005	1.0000
iprec_at_recall_1.00	005	1.0000	iprec_at_recall_1.00	005	1.0000
P_5	005	1.0000	P_5	005	1.0000
P_10	005	1.0000	P_10	005	1.0000
P_15	005	1.0000	P_15	005	1.0000
P_20	005	1.0000	P_20	005	1.0000
P_30	005	1.0000	P_30	005	1.0000
P_100	005	1.0000	P_100	005	1.0000
P_200	005	1.0000	P_200	005	1.0000
P_500	005	1.0000	P_500	005	1.0000
P_1000	005	1.0000	P_1000	005	1.0000

Tabla 15 Resultados Trec Eval - Consulta 5

6. Relacionado principalmente con: dc:subject (temática).

Consulta en Solr para **¿Qué fichas poseen como temática ballenas?**

- fichasFacetadas http://casa.jabenitez.com/solr/fichasFacetadas/select/?q=tematica%3A*Ballenas*+OR+tematica%3A*ballenas*&version=2.2&start=0&rows=10&indent=on&wt=json&fl=*%2Cscore
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*Ballenas*+OR+fichas%3A*ballenas*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

Búsqueda facetada			Busqueda sin facetar		
num_ret	006	1	num_ret	006	1
num_rel	006	1	num_rel	006	1
num_rel_ret	006	1	num_rel_ret	006	1
map	006	1.0000	map	006	1.0000
Rprec	006	1.0000	Rprec	006	1.0000
bpref	006	1.0000	bpref	006	1.0000
recip_rank	006	1.0000	recip_rank	006	1.0000
iprec_at_recall_0.00	006	1.0000	iprec_at_recall_0.00	006	1.0000
iprec_at_recall_0.10	006	1.0000	iprec_at_recall_0.10	006	1.0000
iprec_at_recall_0.20	006	1.0000	iprec_at_recall_0.20	006	1.0000
iprec_at_recall_0.30	006	1.0000	iprec_at_recall_0.30	006	1.0000
iprec_at_recall_0.40	006	1.0000	iprec_at_recall_0.40	006	1.0000
iprec_at_recall_0.50	006	1.0000	iprec_at_recall_0.50	006	1.0000
iprec_at_recall_0.60	006	1.0000	iprec_at_recall_0.60	006	1.0000
iprec_at_recall_0.70	006	1.0000	iprec_at_recall_0.70	006	1.0000
iprec_at_recall_0.80	006	1.0000	iprec_at_recall_0.80	006	1.0000
iprec_at_recall_0.90	006	1.0000	iprec_at_recall_0.90	006	1.0000
iprec_at_recall_1.00	006	1.0000	iprec_at_recall_1.00	006	1.0000
P_5	006	0.2000	P_5	006	0.2000
P_10	006	0.1000	P_10	006	0.1000
P_15	006	0.0667	P_15	006	0.0667
P_20	006	0.0500	P_20	006	0.0500
P_30	006	0.0333	P_30	006	0.0333
P_100	006	0.0100	P_100	006	0.0100
P_200	006	0.0050	P_200	006	0.0050
P_500	006	0.0020	P_500	006	0.0020
P_1000	006	0.0010	P_1000	006	0.0010

Tabla 16 Resultados Trec Eval - Consulta 6

7. Relacionado principalmente con: dc:description.

Consulta en Solr para **¿Qué obras tienen las medidas en varas?.**

- fichasFacetadas
  http://casa.jabenitez.com/solr/fichasFacetadas/select/?q=%20notas%3A*varas*&version=2.2&start=0&rows=470&indent=on&wt=json&fl=*%2Cscore
- fichasSinFacetar:
  http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*varas*&fq=&start=0&rows=1700&fl=*%2Cscore&wt=json&explainOther=&hl.fl

Búsqueda facetada			Busqueda sin facetar		
num_ret	007	467	num_ret	007	1683
num_rel	007	467	num_rel	007	1683
num_rel_ret	007	467	num_rel_ret	007	1683
map	007	1.0000	map	007	1.0000
Rprec	007	1.0000	Rprec	007	1.0000
bpref	007	1.0000	bpref	007	1.0000
recip_rank	007	1.0000	recip_rank	007	1.0000
iprec_at_recall_0.00	007	1.0000	iprec_at_recall_0.00	007	1.0000
iprec_at_recall_0.10	007	1.0000	iprec_at_recall_0.10	007	1.0000
iprec_at_recall_0.20	007	1.0000	iprec_at_recall_0.20	007	1.0000
iprec_at_recall_0.30	007	1.0000	iprec_at_recall_0.30	007	1.0000
iprec_at_recall_0.40	007	1.0000	iprec_at_recall_0.40	007	1.0000
iprec_at_recall_0.50	007	1.0000	iprec_at_recall_0.50	007	1.0000
iprec_at_recall_0.60	007	1.0000	iprec_at_recall_0.60	007	1.0000
iprec_at_recall_0.70	007	1.0000	iprec_at_recall_0.70	007	1.0000
iprec_at_recall_0.80	007	1.0000	iprec_at_recall_0.80	007	1.0000
iprec_at_recall_0.90	007	1.0000	iprec_at_recall_0.90	007	1.0000
iprec_at_recall_1.00	007	1.0000	iprec_at_recall_1.00	007	1.0000
P_5	007	1.0000	P_5	007	1.0000
P_10	007	1.0000	P_10	007	1.0000
P_15	007	1.0000	P_15	007	1.0000
P_20	007	1.0000	P_20	007	1.0000
P_30	007	1.0000	P_30	007	1.0000
P_100	007	1.0000	P_100	007	1.0000
P_200	007	1.0000	P_200	007	1.0000
P_500	007	0.9340	P_500	007	1.0000
P_1000	007	0.4670	P_1000	007	1.0000

Tabla 17 Resultados Trec Eval - Consulta 7

8. Consulta en Solr para **¿Qué obras representan un monumento de ceuta?.**

- fichasFacetadas: http://casa.jabenitez.com/solr/fichasFacetadas/select/?q=materia%3A*Ceuta*&version=2.2&start=0&rows=250&indent=on&wt=json&fl=*%2Cscore
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*ceuta*&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

Búsqueda facetada	Busqueda sin facetar
num_ret 008 257	num_ret 008 3
num_rel 008 257	num_rel 008 3
num_rel_ret 008 257	num_rel_ret 008 3
map 008 1.0000	map 008 1.0000
Rprec 008 1.0000	Rprec 008 1.0000
bpref 008 1.0000	bpref 008 1.0000
recip_rank 008 1.0000	recip_rank 008 1.0000
iprec_at_recall_0.00 008 1.0000	iprec_at_recall_0.00 008 1.0000
iprec_at_recall_0.10 008 1.0000	iprec_at_recall_0.10 008 1.0000
iprec_at_recall_0.20 008 1.0000	iprec_at_recall_0.20 008 1.0000
iprec_at_recall_0.30 008 1.0000	iprec_at_recall_0.30 008 1.0000
iprec_at_recall_0.40 008 1.0000	iprec_at_recall_0.40 008 1.0000
iprec_at_recall_0.50 008 1.0000	iprec_at_recall_0.50 008 1.0000
iprec_at_recall_0.60 008 1.0000	iprec_at_recall_0.60 008 1.0000
iprec_at_recall_0.70 008 1.0000	iprec_at_recall_0.70 008 1.0000
iprec_at_recall_0.80 008 1.0000	iprec_at_recall_0.80 008 1.0000
iprec_at_recall_0.90 008 1.0000	iprec_at_recall_0.90 008 1.0000
iprec_at_recall_1.00 008 1.0000	iprec_at_recall_1.00 008 1.0000
P_5 008 1.0000	P_5 008 0.6000
P_10 008 1.0000	P_10 008 0.3000
P_15 008 1.0000	P_15 008 0.2000
P_20 008 1.0000	P_20 008 0.1500
P_30 008 1.0000	P_30 008 0.1000
P_100 008 1.0000	P_100 008 0.0300
P_200 008 1.0000	P_200 008 0.0150
P_500 008 0.5140	P_500 008 0.0060
P_1000 008 0.2570	P_1000 008 0.0030

Tabla 18 Resultados Trec Eval - Consulta 8

9. Relacionado principalmente con: dc:title.

Consulta en Solr para **¿Qué obras son Material cartográfico?**

- fichasFacetadas: http://casa.jabenitez.com/solr/fichasFacetadas/select/?q=titulo%3A*cartogr*fico*&version=2.2&start=0&rows=5500&indent=on&wt=json&fl=*%2Cscore
- fichasSinFacetar: http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&q=fichas%3A*cartogr*fico*&fq=&start=0&rows=5500&fl=*%2Cscore&wt=json&explainOther=&hl.fl=

Búsqueda facetada	Busqueda sin facetar
num_ret 009 5404	num_ret 009 5436
num_rel 009 5404	num_rel 009 5436
num_rel_ret 009 5404	num_rel_ret 009 5436
map 009 1.0000	map 009 1.0000
Rprec 009 1.0000	Rprec 009 1.0000
bpref 009 1.0000	bpref 009 1.0000
recip_rank 009 1.0000	recip_rank 009 1.0000
iprec_at_recall_0.00 009 1.0000	iprec_at_recall_0.00 009 1.0000
iprec_at_recall_0.10 009 1.0000	iprec_at_recall_0.10 009 1.0000
iprec_at_recall_0.20 009 1.0000	iprec_at_recall_0.20 009 1.0000
iprec_at_recall_0.30 009 1.0000	iprec_at_recall_0.30 009 1.0000
iprec_at_recall_0.40 009 1.0000	iprec_at_recall_0.40 009 1.0000
iprec_at_recall_0.50 009 1.0000	iprec_at_recall_0.50 009 1.0000
iprec_at_recall_0.60 009 1.0000	iprec_at_recall_0.60 009 1.0000
iprec_at_recall_0.70 009 1.0000	iprec_at_recall_0.70 009 1.0000
iprec_at_recall_0.80 009 1.0000	iprec_at_recall_0.80 009 1.0000
iprec_at_recall_0.90 009 1.0000	iprec_at_recall_0.90 009 1.0000
iprec_at_recall_1.00 009 1.0000	iprec_at_recall_1.00 009 1.0000
P_5 009 1.0000	P_5 009 1.0000
P_10 009 1.0000	P_10 009 1.0000
P_15 009 1.0000	P_15 009 1.0000
P_20 009 1.0000	P_20 009 1.0000
P_30 009 1.0000	P_30 009 1.0000
P_100 009 1.0000	P_100 009 1.0000
P_200 009 1.0000	P_200 009 1.0000
P_500 009 1.0000	P_500 009 1.0000
P_1000 009 1.0000	P_1000 009 1.0000

Tabla 19 Resultados Trec Eval - Consulta 9

5.3.3. CONSULTAS SPARQL EN PROTÈGÈ

Para poder realizar las consultas en Protègè, ha sido necesario realizar un estudio del funcionamiento del lenguaje de consultas SPARQL, explicado en el punto 2.1.3. de este trabajo.

Consulta en Protègè para **Qué fichas datan sobre una obra realizada en 1950:**

```
SELECT distinct ?s ?p ?o
WHERE {
        ?s ?p ?o .
        ?s a :Ficha .
        FILTER (regex(?o, "^1950"))}
```

Resultados obtenidos 4:

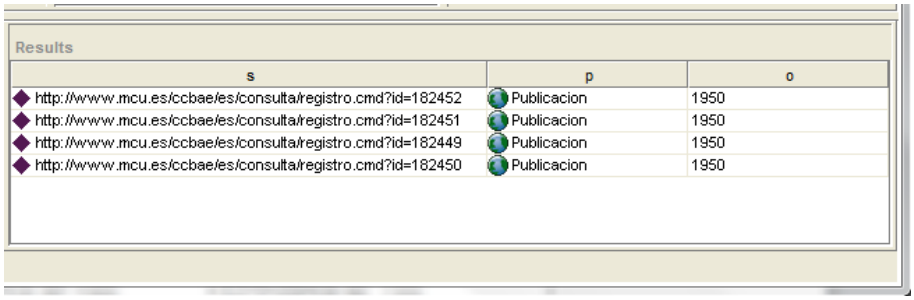

Ilustración 36 Resultados Protègè - Consulta 1

Consulta en Protègè para **Qué fichas están fabricadas en pergamino**:

```
SELECT distinct ?id ?campo ?valor
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "pergamino"))
}
```

Se han obtenido un total de **21 resultados**, los diez primeros son los siguientes:

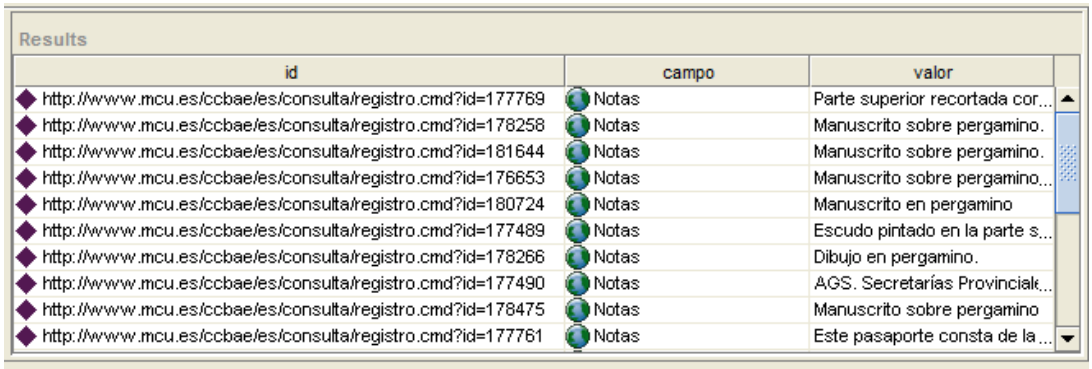

Ilustración 37 Resultados Protègè - Consulta 2

Consulta en Protègè para **Qué fichas utilizan la técnica de tinta con rotulaciones**

```
SELECT distinct ?id ?campo ?valor
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "rotulacion"))
}
```

Se han obtenido un total de **169 resultados**, los diez primeros son los siguientes:

Results

id	campo	valor
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177013	Notas	Tinta y colores. Con explicación y rotulacic...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=176862	Notas	Tinta y colores verde y amarillo para las zc...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177843	Notas	Tinta y colores a la aguada amarillo y gris....
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=176895	Notas	Explicación y rotulaciones. Rosa de los vie...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177853	Notas	Tinta. Con rotulaciones
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177180	Notas	Tinta y colores a la aguada amarillo, verde...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178105	Notas	Con explicación y rotulaciones
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177066	Notas	Tinta y colores a la aguada encarnado y o...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=176639	Notas	Aguada en verde y rojo. Con rotulaciones ...
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=177862	Notas	Tinta y colores a la aguada encarnado y v...

Ilustración 38 Resultados Protègè - Consulta 3

Consulta en Protègè para **Qué fichas fueron creadas por Juan Baptista**

```
SELECT DISTINCT ?id
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "Juan")) .
        FILTER (regex(?valor, "Baptista"))
}
```

Se han obtenido un total de **3 resultados**:

Results

id
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=180501
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178297
http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=176716

Ilustración 39 Resultados Protègè - Consulta 4

Consulta en Protègè para **Qué fichas están escritas en español**

```
SELECT  (COUNT(DISTINCT ?s) AS ?count)
WHERE {
        ?s ?p ?o .
        ?s a :Ficha .
        FILTER (regex(?o, "spa"))
}
```

Se han obtenido un total de **7380 resultados,** los diez primeros son los siguientes:

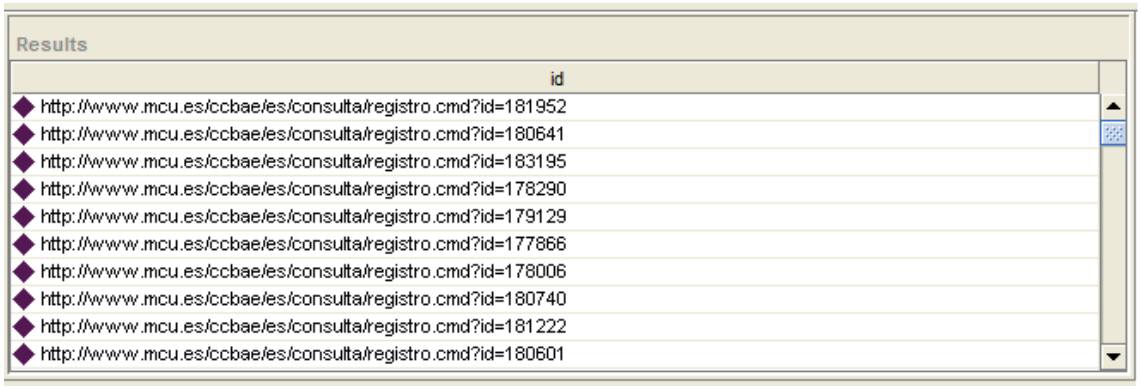

Ilustración 40 Resultados Protègè - Consulta 5

Consulta en Protègè para **Qué fichas poseen como temática ballenas**

```
SELECT ?id ?campo ?valor
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor,  "Ballenas"))
}
```

Se han obtenido un total de **1 resultado**:

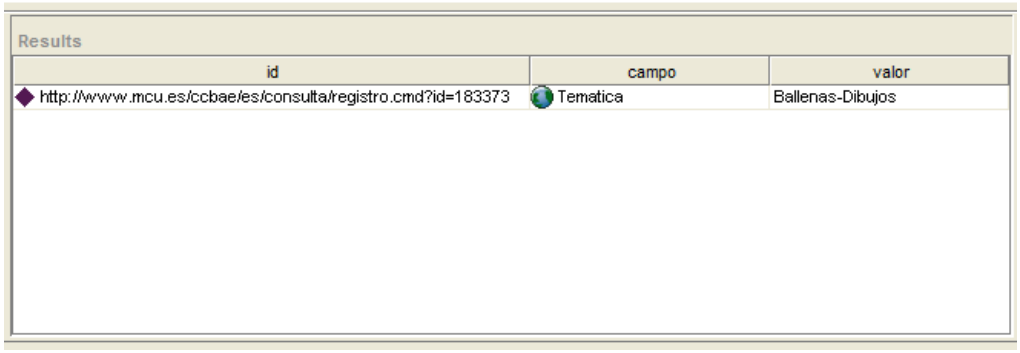

Ilustración 41 Resultados Protègè - Consulta 6

Consulta en Protègè para **Qué obras tienen las medidas en varas.**

```
SELECT  DISTINCT ?id
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "varas"))
}
```

Se han obtenido un total de **495 resultados,** los diez primeros son los siguientes:

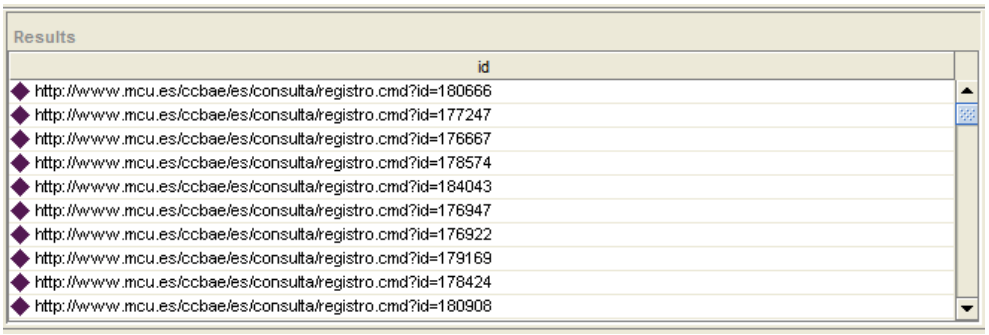

Ilustración 42 Resultados Protègè - Consulta 7

Consulta en Protègè para **Qué obras representan un monumento de ceuta.**

```
SELECT  DISTINCT ?id
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "Ceuta"))
}
```

Se han obtenido un total de **273 resultados,** los diez primeros son los siguientes:

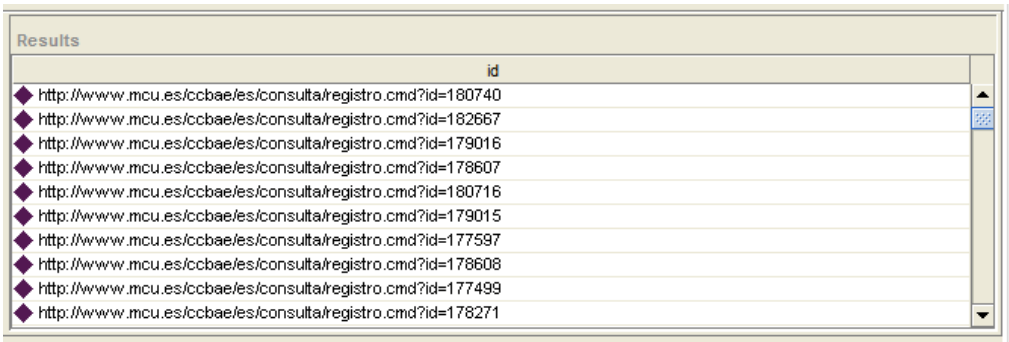

Ilustración 43 Resultados Protègè - Consulta 8

Consulta en Protègè para **Qué obras son Material cartográfico**

```
SELECT  DISTINCT ?id
WHERE {
        ?id ?campo ?valor .
        ?id a :Ficha .
        FILTER (regex(?valor, "Material cartografico")).
}
```

Se han obtenido un total de **5404 resultados,** los diez primeros son los siguientes:

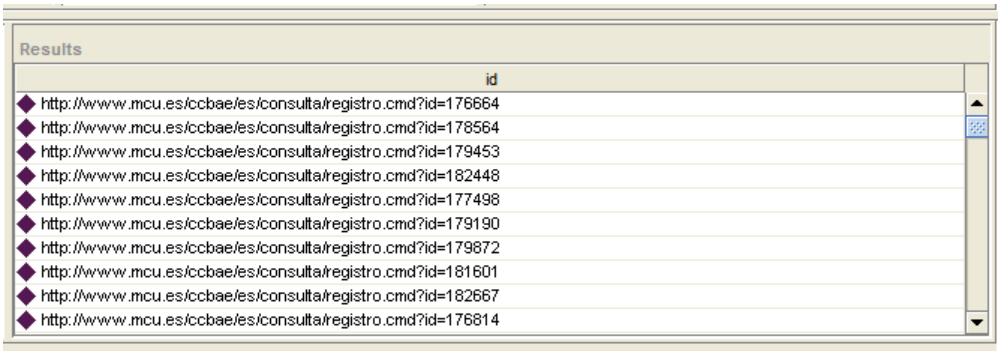

Ilustración 44 Resultados Protègè - Consulta 9

6. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO.

Tras el desarrollo de las funcionalidades básicas para la experimentación en el corpus de prueba, el catálogo de mapas, planos y dibujos del archivo general de Simancas, y tras la reflexión sobre el panorama tecnológico y el estado del arte y la selección de tareas a experimentar, podemos destacar que este Trabajo de Fin de Máster, afronta desde diversas perspectivas, el problema de la búsqueda semántica a partir de texto en lenguaje natural.

La comparación entre el modelo de búsqueda textual no facetado, facetado y ontológico (o clasificación con la ontología) nos permite observar que para alcanzar conclusiones relevantes es necesario seguir investigando en modelos y técnicas semánticas.

6.1 FUTURAS LÍNEAS DE TRABAJO

La creación de ontologías y su mantenimiento de la forma más automatizada posible es uno de los principales retos a los que se ha enfrentado desde su nacimiento la web semántica. El enriquecimiento semántico automatizado de las ontologías englobará la creación (semi-) automática de conceptos, relaciones, instancias y atributos ontológicos. Por su parte la población o instanciación (semi-)automática se centra en enriquecer, mediante instancias de las clases y/o de las relaciones, una ontología ya existente.

La importancia de la población de ontologías reside fundamentalmente en el hecho de que (1) las ontologías existentes necesitan de actualizaciones periódicas (2) el tejido ontológico es lo suficientemente amplio como para dar cobertura a numerosos dominios especializados y, la instanciación de las ontologías que lo conforman supone un salto cualitativo en tareas como por ejemplo, la recuperación de información. Las metodologías de instanciación de ontologías que encontramos en la literatura combinan técnicas de PLN, tales como extracción y reconocimiento de patrones lingüísticos, POS-tagger y análisis sintáctico, junto con técnicas de aprendizaje automático.

Se proponen dos metodologías para el enriquecimiento semi-automático de las ontologías:

- Metodología basada en la distancia contextual y la ganancia de conocimiento.
- Metodología basada en roles semánticos.

Las dos metodologías propuestas abordarían la instanciación automática de ontologías desde un punto de vista que combina el análisis lingüístico tradicional y tecnologías para la extracción de conocimiento textual.

La lingüística como ciencia del lenguaje se ha desarrollado durante siglos. Ciertamente, los métodos estadísticos y computacionales son necesarios para el desarrollo de herramientas de PLN, pero estos métodos serán más eficaces si consideran las características lingüísticas de los textos objeto del procesamiento, siendo ésta una de las principales contribuciones de la tesis.

Las metodologías propuestas son modulares, es decir, están divididas en fases y en ellas se reutilizan algunos recursos ya desarrollados y disponibles gratuitamente, de amplia difusión, lo que facilita su adaptación a diversos escenarios de aplicación.

PROPUESTA 1: METODOLOGÍA BASADA EN LA DISTANCIA COTEXTUAL Y LA GANANCIA DE CONOCIMIENTO

La primera metodología se basa en la distancia contextual, como elemento lingüístico, y en la ganancia de conocimiento, como elemento ontológico.

La distancia contextual se refiere a la distancia física que existe entre dos unidades lingüísticas en el texto. En cuanto a la ganancia de conocimiento es una medida que hace referencia al conocimiento cuantitativo adquirido por el sistema y susceptible de ser incluido en a ontología, es decir, a mayor cantidad de conocimiento adquirido mayor ganancia de conocimiento.

El punto de partida de esta metodología es nuestro corpora relativo al dominio particular del Archivo General de Simancas, que se ha analizado desde un punto de vista discursivo siguiendo los parámetros propuestos por Bhatia (1993). Los datos obtenidos se han interpretado, de modo que la información que arroja el análisis se ha utilizado para (1) determinar cuáles son las instancias relevantes del dominio (2) determinar cómo dichas instancias aparecen representadas textualmente (3) desarrollar los componentes lingüísticos que permitan la extracción de las mismas y (4) ajustar los parámetros cotextuales de la metodología de instanciación.

Además, se propone como trabajo futuro, el desarrollo de una ontología de AGS en la que se insertan las instancias. La metodología de instanciación se dividiría en cuatro fases secuenciales:

1. Fase de Procesamiento de Lenguaje Natural y de Procesamiento del Corpus.
2. Fase de Reconocimiento e identificación de las Entidades Nombradas.
3. Fase de Población de la Ontología.
4. Verificación de la consistencia de la ontología.

En la primera fase obtendríamos la estructura morfosintáctica de cada oración en el corpus con el objetivo de extraer la información lingüística necesaria para las siguientes fases. Para ello, necesitaríamos utilizar el framework GATE 1 (http://gate.ac.uk/).

Durante la segunda fase se extraerían un conjunto de menciones de entidades nombradas. Con este propósito, sería necesario desarrollar unas listas de gazetteers de carácter general, como por ejemplo título, época de las obras u otros datos, así como de carácter específico, como tipos de obra. Estas listas, combinadas con reglas basadas en expresiones regulares, permitirían la anotación de entidades nombradas. Cuanto mayor sea el número de anotaciones en esta fase, mayor será el número potencial de instancias de la ontología, ya que, en la siguiente fase, dichas menciones serán candidatas a instancias o a valores de propiedades de la ontología.

Durante la fase de instanciación de la ontología, el sistema recibiría como entrada el conjunto de anotaciones de la fase anterior, convirtiéndose éstas en candidatas a instancias. Sin embargo, durante el proceso de anotación pueden surgir ambigüedades, como por ejemplo que una anotación o grupo de anotaciones esté relacionado con más de una entidad nombrada o que una entidad esté relacionada con varios recursos de la ontología. En estos casos el sistema procederá a la desambiguación. Para ello, se basa en la distancia que las separa en el texto (distancia contextual), por un lado, y en la cantidad de conocimiento que aportan a la ontología, por otro.

La ganancia de conocimiento se fundamenta en el hecho de que una instancia no aparece aislada en el discurso, sino que en el texto circundante, es decir, en el contexto, se pueden localizar otros elementos que aporten información sobre la misma. Esta información es susceptible tanto de ser incluida en la ontología como de ser utilizada para la desambiguación de dicha instancia. La desambiguación se lleva a cabo creando un árbol combinatorio con todas las clases y propiedades que podrían ser instanciadas en función de la instancia ambigua y aquellas más cercanas en el texto con las que se podría relacionar. Finalmente se elegiría la combinación que aporta mayor conocimiento a la ontología.

Una vez desambiguadas las anotaciones se insertan en la ontología como un individuo de una o más clases. Así mismo, se identificarían los valores de sus atributos y relaciones. Si una instancia no se insertara previamente, entonces el sistema instanciaría la ontología con la información extraída, creando así un nuevo individuo. Si por el contrario, una instancia ya existía ésta se enriquece con nuevas relaciones y atributos, en el caso de que hayan sido identificados. Finalmente, en la última fase, se comprobaría la consistencia de la ontología mediante un razonador OWL-DL para detectar individuos erróneos y que no cumplan las restricciones de la ontología.

La evaluación se debería llevar a cabo considerando por un lado las entidades anotadas por el sistema y por otro, la cantidad de instancias que se incluyen en la ontología. En este trabajo, vamos a realizar una experimentación con la herramienta TREC EVAL, teniendo unos resultados previos en función de las consultas que queremos realizar, analizando si los documentos recibidos en las consultas semánticas son los correctos, basándonos en una evaluación semi-automática que necesita la supervisión de una persona.

PROPUESTA 2: METODOLOGÍA BASADA EN ROLES SEMÁNTICOS

La integración de diversos recursos ontológicos y lingüísticos es la base de esta metodología en la que se combinarían ontologías de alto nivel del dominio del AGS con frames semánticos. Dada la baja disponibilidad de recursos para nuestro dominio específico, a pesar de la existencia de diversas herramientas para el PLN, se propone un marco de trabajo que integra ambos recursos. Estos recursos son por un lado, la ontología del AGS de la que disponemos a raíz de los análisis realizados hasta el momento y los frames de FrameNet (Baker et al., 1998).

Si dispusiéramos de una ontología de alto nivel relacionada con nuestro problema en particular en las que se expresen relaciones genéricas de nuestro dominio, podríamos

seleccionar algunas de las relaciones de las ontologías y definir un conjunto de axiomas. La ontología resultante podría mapearse con frames o marcos semánticos extraídos de FrameNet mediante la asociación de uno o más frames a cada relación. Los frames son representaciones esquematizadas de situaciones del mundo real en base a las cuales se organiza la información. Las relaciones ontológicas y los frames tienen en común que son generalizaciones de situaciones discursivas cuya expresión lingüística consiste fundamentalmente en una forma verbal o su nominalización. En consecuencia, aquellas relaciones ontológicas que tienen asociado un frame, contienen tanto expresiones lingüísticas, que representan dichas relaciones a un nivel textual, como unos roles semánticos que describen cada relación. El modelo ontológico resultante sería nuestro AGS Enriquecido.

El proceso de instanciación debería llevarse a cabo, por un lado, mediante la identificación de entidades nombradas con el reconocedor de entidades nombradas de AGS, convirtiéndose las mismas en candidatas a instancias de la ontología, y por otro lado, mediante la identificación de relaciones entre ellas para lo que utilizaríamos nuestra ontología AGS enriquecida. Cuando dos entidades nombradas se relacionan a través de las unidades léxicas incluidas en AGS Enriquecido, dicha relación se incluiría provisionalmente como instancia de una ObjectProperty, y las instancias implicadas en la misma se convierten en candidatas a instancias de la ontología.

Un frame conectaría las dos entidades más cercanas en el texto siempre y cuando cumplan con ciertos requisitos, como por ejemplo, la distancia que existe entre ellas.

Finalmente un razonador comprobaría la consistencia de la ontología, de modo que si la ontología es consistente, las clases y propiedades correspondientes serían instanciadas.

Además mediante el razonador se podrían obtener nuevas relaciones en base a los axiomas previamente definidos en el modelo ontológico. La validación del sistema se llevaría a cabo mapeando el modelo ontológico con una ontología de dominio, de manera que tanto los marcos semánticos como los axiomas definidos en el modelo pasan a formar parte de la ontología de dominio. De nuevo aquí se deberían evaluar dos aspectos del proceso de instanciación, por un lado las entidades nombradas identificadas y por otro la cantidad de relaciones ontológicas que se han establecido. Las medidas utilizadas serían la exhaustividad, precisión y medida de F.

OTRAS LÍNEAS DE TRABAJO FUTURO

Dentro de la dinámica de este trabajo y orientado más hacia el desarrollo de herramientas que automaticen distintas funciones, otras líneas de trabajo futuro serían las siguientes:

Importancia en el tiempo de respuesta y los resultados obtenidos en las consultas realizadas a las fichas facetadas.

Una vez realizada la obtención de la información de las diferentes fichas del AGS en distintos formatos y una vez almacenadas en nuestro servidor Apache Solr, cabe destacar que las consultas realizadas sobre la base de datos de las fichas facetadas fue mucho más rápida y mucho más certera que la realizada sobre la base de datos de las fichas sin facetar. Esto nos

demuestra pues, que a mayor conocimiento que tenga nuestro sistema sobre cualquier tipo de elementos, mejores serán los resultados de nuestra búsqueda y más fácil nos será encontrar la información que verdaderamente necesitemos.

En la época en la que nos encontramos tecnológicamente hablando y más concretamente en el ámbito de la web semántica, cada día que pasa vemos la necesidad de seguir mejorando nuestro sistema de búsqueda para que todo el mundo pueda tener acceso a la información que necesite sin tener conocimiento alguno de cómo funcionan realmente los algoritmos de búsqueda para obtener unos buenos resultados, creo que este es el fundamento principal de la web semántica y hacia donde nos dirigimos.

Buscador semántico vs buscador textual

En relación a una de las partes mayoritariamente prácticas de este trabajo, cabe destacar la gran dificultad a la hora de la realización de un buscador semántico que comprenda de manera absoluta la información que realmente necesita el usuario y la búsqueda que está realizando, de qué términos se trata.

El buscador textual simplemente recibe la información que el usuario desea buscar, realiza una consulta general a la base de datos sin buscar ningún campo en concreto y devuelve unos resultados de búsqueda en los cuales ha encontrado la cadena que ha buscado el usuario, pudiendo obtener de esta manera resultados que no le interesan o que no reflejan realmente la información que necesita, tal y como funcionaban los buscadores hasta hace muy poco tiempo.

Sin embargo y por el contrario, el buscador semántico es capaz de analizar en una cadena de texto insertada por el usuario, qué tipo de información desea obtener, qué campos concretos de la base de datos desea leer y sobre qué campos está realizando la búsqueda. Es por ello que, crear un analizador semántico posee una dificultad en cuanto a la realización de los distintos algoritmos bastante alta si deseamos obtener un buscador rápido y eficiente al mismo tiempo. Nuestro buscador es capaz de leer una cadena y basándose en una serie de patrones, stopwords, stopwords de dominio y comparando la propia cadena con distintos resultados de diferentes campos, mostrará la información que cree que le interesa al usuario.

Este buscador semántico actualmente es bastante limitado y no comprende todas las posibilidades que puede consultar un usuario normal de a pie.

Por todo ello, el buscador semántico tiene una gran línea de trabajo futuro en el que, mejorando el sistema y los algoritmos utilizados, podríamos obtener mejores resultados que los encontrados hasta el momento. Algunos ejemplos de mejoras posibles serían los siguientes:

- Automatización de eliminación de stopwords obteniendo estas de un listado externo actualizado por una comunidad. Actualmente el buscador cuenta con una constante declarada en el fichero JavaScript que contiene un número de stopwords. A medida que probamos nuestro buscador, nos damos cuenta de que existen muchas más que podrían ir añadiéndose, de esta forma mejorarían notablemente los resultados obtenidos y la velocidad de búsqueda.
- Mejora en la comprensión de la información que necesitamos: nuestro buscador actualmente interpreta de una manera bastante acertada la información que necesita

el usuario si se trata de una búsqueda no muy compleja. Por ejemplo: si el usuario desea obtener los resultados de una búsqueda como la siguiente:

Me gustaría saber qué mapas ha hecho Galvarreta en el año 1639 en español

En esta búsqueda tenemos un tipo de objeto a recibir en los resultados de búsqueda especificado por la expresión *"qué mapas"* , conocemos que uno de los campos que se debe consultar es el de autor, ya que *"Galvarreta"* es un autor y nuestro sistema es capaz de detectarlo correctamente, además sabemos que la obra debe haber sido hecha en 1639, con lo cual, tendremos que consultar también el campo *fecha* y por último el campo *idioma* buscando todos los mapas que estén hechos en castellano.

Esta búsqueda contiene únicamente un dato por cada campo, pero si por el contrario nosotros quisiéramos obtener los resultados de la siguiente consulta:

Me gustaría saber qué mapas han sido realizados por Galvarreta y por Juan Baptista entre los años 1639 y 1670 en los distintos idiomas.

Por las características de nuestro buscador actual, actualmente sí es capaz de detectar que el usuario desea obtener los mapas realizados por dos autores distintos, pero no tiene la capacidad de detectar que debe comprenderse en un rango de fechas determinado por el usuario. Realizar esto no es una tarea muy compleja, pero sí que requiere un estudio y un análisis mayor de la consulta que nos obligaría a mejorar nuestro algoritmo actual de análisis de la misma.

- Enriquecedor semántico automatizado: actualmente hemos obtenido una serie de información de la cual ya conocíamos su ontología y los campos que realmente eran importantes para poder realizar las búsquedas, pero todo esto ha tenido que ser analizado previamente de forma manual. Se podría crear un analizador en el propio buscador que, en función de las búsquedas que realizan los usuarios, almacene la información que más se solicita, y si esta no está actualmente categorizada, que automáticamente el sistema cree esa nueva faceta y la almacene para unas futuras búsquedas más rápidas. Pongamos un ejemplo: nos damos cuenta que muchos usuarios realizan búsquedas de mapas por coordenadas geográficas, pero estas coordenadas no tienen un campo concreto como tal, actualmente se encuentran en nuestro campo **descripción**. Nuestra propuesta es, detectar que muchos usuarios realizan búsquedas por este tipo de dato, crear en nuestro Apache Solr ese nuevo campo, añadir los valores del campo en las fichas actuales y conseguir que nuestro buscador se nutra de esto, de forma que, sea capaz de saber que existe un nuevo campo en nuestra base de datos al que realizar consultas.

7. REFERENCIAS

1. **Natalya F. Noy and Deborah L. McGuinness**. Desarrollo de Ontologías-101: Guía Para crear Tu Primera Ontología. 2005

2. **Martínez**, reTaskXML: Especificación de modelos de tareas a partir de especificaciones de interfaces de usuario, M.Sc. thesis, Univ. Castilla-La Mancha, AlbaceteDic. 2007.

3. **Giraldo Pasmín** , **Olga Ximena** Manejo del conocimiento en los cuadernos de laboratorio Universidad Nacional de Colombia, Facultad de Ciencias Agropecuarias. Dpto. de Fitomejoramiento Palmira. 2011.

4. **Maldonado Ibáñez , Ana**. Desarrollo de un modelo espacio-temporal para la simulación del movimiento espontáneo de las personas mediante la creación de superficies de movimiento. 2010.

5. **Haide Pérez**, **Laura.** Extensión de Protégé para la Integración de Ontologías. 2008

6. **López Herrera, Antonio Gabriel**. Modelos de Sistemas de Recuperación de Información Documental Basados en Información Lingüística Difusa. 2006.

7. **Gitte Kristiansen**. Referencia exofórica y estereotipos lingüísticos: una aproximación sociocognitiva a la variación alofónica libre en el Lenguaje Natural. Madrid, 2003.

8. **Pradeep K. Atrey , M. Anwar Hossain, Abdulmotaleb El Saddik, Mohan S. Kankanhalli.** *Multimodal fusion for multimedia analysis: a survey*. 2010.

9. **Berners-Lee, T., Hendler, J. and Lassila, O.** *The Semantic Web*. 2001.

10. **Wiratna S. Wiguna, Juan J. Fernández-Tébar, and Ana García-Serrano.** *Using Fuzzy Model for Combining and Reranking Search Result from Different Information Sources to Build Metasearch Engine.*

11. **Grubinger , M. , Clough P., Müller, H. and Deselaers, T.** *The IAPR TC-12 Benchmark: A New Evaluation Resource for Visual Information Systems*. 2006

12. **Maria Ruiz-Casado, Enrique Alfonseca y Pablo Castells** *Automatic extraction of semantic relationships for Wordnet by means of pattern learning from Wikipedia*. 2011

13. **Fernández Sánchez, M., Castells, P.** *Semantically enhanced Information Retrieval: an ontology-based approach.* 2009

14. ***Patricio Martínez-Barco, José Luis Vicedo, Estela Saquete, David Tomás*** *Grupo de Procesamiento del Lenguaje y Sistemas de Información* de la *Universidad de Alicante, "Sistemas de Pregunta-Respuesta"* 2009.

15. Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Girju, R., Goodrum, R., Rus, V. (2000). The Structure and Performance of an Open-Domain Question Answering System. In Proceedings of the Conference of the Association for Computational Linguistics (ACL-2000), 563--570.

16. **Hyvönen, E., Styrman, A. and Saarela, S.** *Ontology-Based Image Retrieval*. 2003

17. **Qin, Jian**. *Desarrollo de Ontologías para la Representación y el Acceso a la Información.* Madrid, 2008.

18. **Latorres Enrique.** *Reuso de Reglas de Negocio: Una experiencia de reuso de ontologías en un dominio restringido.* 2002.

19. **Marquez Solis, Santiago.** *Web semántica y servicios web semánticos.* 2007.

20. **López, D. y Caldón, E.** *Enriquecimiento Semántico de Contenidos en Redes Socialesbasado en Ontologías y Vocabularios de Domini.* 2009

21. **Murua, Idoia**. *Utilidad de las herramientas de anotación para el desarrollo dela web semántica. Revista / Vigilancia Tecnológica.*2006

22. **Blog Cassora.** http://blog.classora.com/2012/08/29/motores-de-enriquecimiento-semantico-de-contenidos/ . *Motores de Enriquecimiento Semántico*

23. **Ferrete Benítez, José Manuel.** *Conectar Apache con Tomcat.* 2011

24. **Sánchez Suárez, José Manuel**. Test de integración con Solr y el soporte de EmbeddedSolrServer. 2012

25. **Pérez, Alberto**. *Indexar en Solr.* 2012.

26. **Vilches, Alberto.** *Debian, Apache 2 y Tomcat 6 usando múltiples dominios.*2009.

27. **Burges, C.J.C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G**. *Learning to rank using gradient descent,* Proceedings of ICML 2005.

28. **Nie, L., Davison, B. D., Qi, X.**, *Topical link analysis for web search, Proc. of SIGIR* 2006.

29. **Shakery, A., Zhai, C. X**. *Relevance Propagation for Topic Distillation UIUC TREC 2003 Web Track Experiments*, Proceedings of TREC 2003.

30. **Xue, G. R., Yang, Q., Zeng, H. J., Yu, Y., and Chen, Z.** *Exploiting the hierarchical structure for link analysis.* Proceedings of SIGIR 2005.

31. **Qin, T., Liu, T. and Li, H.** *The TREC Datasets in LETOR*. 2007

32. **Eckard, Emmanuel and Chappelier, Jean-Cédric.** *Free Software for research in Information Retrieval and Textual Clustering*. 2007.

33. **Fernández Gavilanes, Milagros**. *Adquisición y representación del conocimiento mediante procesamiento del lenguaje natural.* 2012.

34. **Rodil Garrido, Antonio.** *Estudio de los lenguajes de consulta para documentos RDF*. Enero 2006.

35. **Rivero Osuna, Carlos Rafael.** *Exchanging data amongst semantic-web ontologies*. 2012.

36. **Durán, E., Álvarez, M., Unzaga, S.** *Sistemas de información web personalizados, basados en ontologías, para soporte al aprendizaje ubicu*. 2012

37. **Fernández Verdejo, S.** *XML y web semántica diseño y población semiautomático de ontologías.* 2012.

38. **Moreno Agudelo, Carlos Arturo Sánchez Reyes, Yakeline.** *Prototipo de buscador semántico aplicado a la búsqueda de libros de ingeniería de sistemas y computación en la biblioteca Jorge Roa Martínez de la Universidad Tecnológica de Pereira.* 2012

39. **Ruíz Martínez, J.M.** *Metodología para la población automática de ontologías: aplicación en los dominios de medicina y turismo*. 2012.

40. **CHAVARRO PORRAS, Julio César**. *Marco de referencia para la gestión del cambio en ontologías basados en modelos conceptuales. Doctorado en Ingeniería Énfasis en Ciencias de la Computación. Cali: Universidad del Valle. Escuela de Ingeniería de Sistemas y Computación. 2010.*
41. **CORBERA DELGADO, Susana**. *Estudio sobre Sistemas de Gestión de Conocimiento para la Web Semántica. Ingeniera Superior Informática. Madrid: Universidad Carlos III de Madrid. Ingeniería Superior Informática. 2007.*
42. **DÍEZ CARRERA, Carmen**. *Las industrias de la lengua: panorámica para los gestores de la información. Fesabid, 1994. ISBN 84-88699-09-3.*
43. **FONER, L.N.** *¿Qué es un Agente de todos modos? Un caso de estudio sociológico - Memo Agente 93-01, agentes del Grupo, del MIT Media Lab (1993).*
44. **GARCÍA, Gerardo y IBÁÑEZ, Patricia**. *Informática Computer Scienc. Segunda edición. México D.F.: Cengage Learning Editores S.A., 2009. ISBN 10: 607-481-091-5. 1 v.*
45. **GRUBER, Thomas**. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing. En: International Workshop on Formal Ontology, 23 de agosto de 1993.*
46. **JIMÉNEZ RUIZ, Ernesto**. *Desarrollo de sistemas multi-agente con jade: aplicación de las ontologías. Castellón: Universidad Jaume I de Castellón. Curso de Doctorado: Robótica Avanzada: Percepción y Manipulación.*
47. **JORQUERA, Marcos.** *Administración de servicios de Internet, publicaciones Universidad deAlicante. Alicante, Publicaciones Univ. de Alicante, 2008. ISBN: 978-84-7908-989-4.*
48. **LAVÍN VILLA, Moisés.** *Construcción automática de diccionarios semánticos usando la similitud distribucional. Trabajo de grado Maestro en Ciencias de la Computación. México, D.F. Instituto Politécnico Nacional, 2010. 35, 37 p.63*
49. **MCBRIDE, Brian**. *Jena: A Semantic Web Toolkit. Hewlett-Packard Laboratories, Bristol, UK. 2002.*

50. **MCGUINNESS, Deborah L. y NOY, Natalya F..** *Desarrollo de Ontologías-101: Guía Para Crear Tu Primera Ontología. Traducido por: Erick Antezana. Stanford University, 2005.*
51. **MIZOGUCHI, Riichiro**. *Tutorial on ontological engineering. The Institute of Scientific and Industrial Research, Osaka University.*
52. **ROMANA GARCÍA, María Luisa; SÁEZ, Juan Manuel y ÚCAR VENTURA, María Pilar**. *Traducción e interpretación: estudios, perspectivas y enseñanzas, 2011.*
53. **SERRANO, Sebastia**. *La semiótica: una introducción a la teoría de los signos. España. Editorial Montesinos, 1998. ISBN:84-85859-32-4.*
54. **STAAB, Steffen y STUDER, Rudi**. *Handbook on Ontologies. Berlín: Editorial Springer. Segunda edición, 2009. ISBN: 978-3-540-70999-2.*

55. **Sobre RDF**
    - *http://www.xml.com/pub/a/98/06/rdf.html (BRAY, Tim. RDF and Metadata)*
    - *http://www.openrdf.org/ (openRDF.org)*
    - *http://planet.rdfhack.com/ (Planet RDF)*
    - *http://www.w3.org/PICS/ (W3C. Platform for Internet Content Selection)*
    - *http://www.w3.org/TR/rdf-pics (W3C. PICS Rating Vocabularies in XML/RDF)*
    - *http://www.w3.org/TR/rdf-dawg-uc/ (W3C. RDF Data Access Use Cases and*
    - *http://www.w3.org/RDF/ (W3C. Resource Description Framework (RDF))*
    - *http://www.w3.org/2001/11/IsaViz/ (W3C. IsaViz: A Visual Authoring Tool for RDF)*
    - *http://www.w3schools.com/rdf/ (W3School. RDF Tutorial)*

56. **Sobre OWL**
    - *http://www.w3.org/2004/OWL/*
    - *http://www.hipertexto.info/documentos/owl.htm*
    - *http://www.w3.org/TR/owl-guide/ (W3C. OWL Web Ontology Language Guide)*
    - *http://www.w3.org/TR/owl-ref/ (W3C. OWL Web Ontology Language. Reference)*
    - *http://www.w3.org/TR/owl-test/ (W3C. OWL Web Ontology Language Test Cases)*

57. **Sobre Web Semántica**
    - *http://www.w3.org/2000/10/swap/Primer (Conociendo RDF y la Web Semántica con N3)*
    - *http://www.w3.org/DesignIssues/Semantic (Hoja de Ruta de la Web Semántica)*
    - *http://purl.org/swag/whatIsSW (Que es la Web Semántica)*
    - *http://uwimp.com/eo.htm (Semantic Web Primer)*
    - *http://logicerror.com/semanticWeb-long (Introducción a la Web Semántica - Extenso)*
    - *http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html (SciAm: The Semantic Web)*
    - *http://www.xml.com/pub/a/2001/03/07/buildingsw.html (Construyendo la Web Semántica)*
    - *http://infomesh.net/2001/06/swform/ (The Semantic Web, Taking Form)*
    - *http://www.w3.org/2001/sw/Activity (SW Activity Statement)*
    - *http://www.w3.org/2000/01/sw/ (SWAD)*

PARTE III: ANEXOS

ANEXO 1: CONVERSOR RDF to OWL

1. PROYECTO

El proyecto creado en NetBeans se compone de los siguientes ficheros:

- **ParserRDFtoOWL.java**
- **VentanaPrincipalParser.java**

1.1. *ParserRDFtoOWL.java*

Este fichero contiene el algoritmo importante que realiza las tareas principales de conversión del fichero en formato RDF al nuevo fichero en formato OWL.

La clase posee el constructor por defecto de JAVA, ya que, no he implementado ninguno en particular, y además posee un método, que es el que realiza todo el algoritmo, y se llama **Parseador**.

```
public void Parseador(String fichOrigen, String fichDestino, int numDePropiedades, String cBuscar, String  cIden, String nIden, String[] props, String[] propNuevas, JFrame frame, boolean debugger){
```

- String fichOrigen : Cadena con el nombre del fichero inicial que vamos a convertir (ruta completa del fichero)
- String fichDestino : Cadena con el nombre y la ruta completa del fichero de destino (fichero convertido)
- int numDePropiedades : Número entero que contiene el número total de propiedades de nuestro objeto principal (del cual crearemos sus instancias , por ejemplo el objeto Ficha que contiene propiedades como Tipo, Nombre, Formato.... etc)
- String cBuscar : Cadena que delimita cada objeto instanciado en nuestro fichero de Origen, me explico :
  - En RDF Dublin core, la cadena a buscar sería “rdf:Description” , ya que los objetos instanciados se encuentran delimitados por <rdf:Description> y </rdf:Description>
- String cIden : Cadena de la que vamos a obtener los identificadores de nuestros objetos instanciados :
  - En el caso de nuestro fichero en formato RDF Dublin Core, el texto que se encuentra entre las etiquetas <dc:identifier> y </dc:identifier> es lo que usaremos como identificador con lo cual cIden debería ser “dc:identifier” sin comillas.
- String nIden : Cadena que delimita nuestros nuevos objetos creados, en nuestro caso, queremos objetos de tipo Ficha, con lo cual la cadena a enviar sería “Ficha” sin comillas. Y los objetos que se van a crear. Son del tipo <Ficha about=”Identificador único de cada objeto”><Propiedades></Propiedades></Ficha>
- String[] props : Array de cadenas que almacena cada propiedad actual en el fichero Origen:
  - En nuestro ejemplo, dc:type, dc:format, dc:publisher, dc:relation, dc:date son algunas de las propiedades de cada objeto rdf:Description.

- String[] propsNuevas: Array de cadenas que almacena cada propiedad nueva del fichero de destino :
    - En nuestro caso, de dc:type la propiedad nueva es Tipo, de dc:format la propiedad nueva es Formato...
- JFrame frame : Simplemente le pasamos la ventana de nuestra aplicación para que pueda crear diálogos de información o de error.
- boolean debugger : Es un booleano que maneja los mensajes en consola, si es true, mostrará mensajes en consola, si es false, no.

Código comentado:

```
public void Parseador(String fichOrigen, String fichDestino, int numDePropiedades,
            String cBuscar, String  cIden, String nIden, String[] props, String[] propNuevas,
            JFrame frame, boolean debugger){
        // TODO code application logic here

        // Instanciamos e inicializamos nuestro fichero de lectura, que será
        // nuestro fichero a convertir, con sus respectivos lectores.
        // Tenemos dos lectores porque necesitaremos acceder a distintas zonas del fichero
        // en un mismo recorrido, por ello declaramos dos lectores distintos, lector1 y lector2.
        File archivo = null;
        FileReader lector1 = null;
        FileReader lector2 = null;
        BufferedReader br = null;

        // Instanciamos e inicializamos nuestro fichero de escritura, que será
        // nuestro fichero de destino, donde se guardará nuestro XML con nuevo
        // formato.
        FileWriter fichero = null;
        PrintWriter pw = null;


        // Instanciamos las cadenas necesarias para poder realizar la conversión.
        String cadenaBuscamos = "<" + cBuscar + ">";
        String cadenaBuscamos2 = "</" + cBuscar + ">";
        String cadenaIdentificador = "<" + cIden + ">";
        String nuevoIdentificador = nIden;

        String[] propiedades = new String[50];
        String[] propiedadesNuevas = new String[50];

         // Rellenamos nuestras propiedades con las que obtenemos como parámetro.
        for(int k=0;k<numDePropiedades;k++) {
            propiedades[k] = props[k];
            propiedadesNuevas[k] = propNuevas[k];
            if(debugger) System.out.println("Propiedad Vieja " + propiedades[k]);
            if(debugger) System.out.println("Propiedad Nueva " + propiedadesNuevas[k]);
        }

        try {
            // Apertura del fichero y creacion de BufferedReader para poder
            // hacer una lectura comoda.
            archivo = new File(fichOrigen);
            lector1 = new FileReader(archivo);

            int lineaDef = 0;
            int lineaDef2 = 0;
```

```
LineNumberReader lr1 = new LineNumberReader(lector1);
String s1 = null;

fichero = new FileWriter(fichDestino);
pw = new PrintWriter(fichero);

// Comenzamos la lectura del fichero origen con el lector1
while ((s1 = lr1.readLine()) != null) {

    /*
     * Si encontramos la cadena que crea cada objeto, en nuestro
     * ejemplo <rdf:Description>, guardamos en una variable
     * el número de línea en el que nos encontramos, para poder
     * comenzar una lectura con nuestro lector2 más adelante en esa
     * línea.
     *
     */
    if (s1.indexOf(cadenaBuscamos) != -1) {
        lineaDef = lr1.getLineNumber();
        if(debugger) System.out.println("Linea # " + lineaDef + " con valor " + s1);
    }

     /*
     * Después de encontrar <rdf:Description>, el siguiente paso es
     * encontrar la cadena que identifica cada objeto, que en nuestro
     * caso particular es dc:identifier. En caso de encontrarla
     * el programa entra dentro de este control if, almacena
     * lo que se encuentra dentro de <dc:identifier></dc:identifier>
     * y comienza a funcionar el segundo lector, lector2.
     *
     */
    if (s1.indexOf(cadenaIdentificador) != -1) {

        String s2 = null;
        int contador = 0;
        lector2 = new FileReader(archivo);
        LineNumberReader lr2 = new LineNumberReader(lector2);

        lr2.setLineNumber(lineaDef);


       /* Para obtener lo que tenemos dentro de <dc:identifier>
       * y </dc:identifier> utilizamos la cadena contenidoEtiqueta1
       * y sacamos una subcadena de la línea que estamos leyendo,
       * de forma que, con esto que hemos escrito conseguimos quitar
       * de la línea leída las etiquetas, es decir:
       *
       * - Si tenemos una línea con el siguiente contenido:
       *   <dc:identifier> Identificador </dc:identifier>
       * Este pequeño método, obtiene la palabra "Identificador"
       * eliminando así la etiqueta de apertura y de cierre.
       */
       String contenidoEtiqueta1 = null;
       int tamPropiedad1 = cadenaIdentificador.length();
       int tamS1 = s1.length();
       contenidoEtiqueta1 = s1.substring(tamPropiedad1, tamS1 - tamPropiedad1 - 1);

       /* Para situarnos con el lector2, en la línea que hemos
        * encontrado nuestro rdf:Description, utilizamos un bucle
        * while, en el que le ponemos como freno lineaDef-1,
        * ayudándonos de contador.
        *
        */
       while (((s2 = lr2.readLine()) != null) && (contador < lineaDef-1)){contador++;}

       if(debugger) System.out.println("<" + nuevoIdentificador + " rdf:about=\"" +
       contenidoEtiqueta1  + "\">");
       pw.println("<" + nuevoIdentificador + " rdf:about=\"" + contenidoEtiqueta1  +
       "\">");
```

```
if(debugger) System.out.println("LINEA ACTUAL: "+lineaDef);

/*
 * Una vez hemos situado el cursor en la parte del documento
 * que nosotros queríamos ( justo en la siguiente línea de
 * <rdf:Description> ), comenzamos a buscar las etiquetas
 * con las propiedades del fichero de Origen, y las convertimos
 * en las nuevas etiquetas, es decir:
 * Si tenemos un ejemplo como el siguiente:
 * <rdf:Description>
    <dc:title>Ibiza (Baleares). Fortificaciones. Planos. 1597</dc:title>
    <dc:relation>Referencias: Mapas, planos y dibujos (Años 1503-1805). Volumen
    I : p. 590</dc:relation>
    <dc:coverage>España-Baleares (Comunidad Autónoma)-Ibiza</dc:coverage>
    <dc:title>Diseño de una parte del Castillo de Ibiza [Material
    cartográfico]</dc:title>
    <dc:description>AGS. Guerra y Marina, Legajos, 00481. Incluido en carta de
    don Alonso de Zanoguera al rey, de 22 de enero de 1597.</dc:description>
    <dc:description>Tinta negra. Con rotulación</dc:description>
    <dc:description>Manuscrito sobre papel.</dc:description>
    <dc:type>Mapas</dc:type>
    <dc:language>spa</dc:language>
    <dc:date>1597</dc:date>
    <dc:creator>Saura, Antonio (m. 1634?)</dc:creator>

    <dc:identifier>http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=183413</dc:
    identifier>
    <dc:format>image/jpeg</dc:format>
    </rdf:Description>
 *
 * El cursor comenzaría en <dc:title>, encontraría esa etiqueta,
 * obtendría el contenido de la etiqueta ( en este caso
 * "Ibiza (Baleares). Fortificaciones. Planos. 1597" ) y
 * crearía la nueva etiqueta <Titulo> en el fichero de
 * destino, con el contenido, de la siguiente forma:
 * <Titulo>Ibiza (Baleares). Fortificaciones. Planos. 1597</Titulo>
 *
 * y así sucesivamente con todas las propiedades.
 *
 * Para poder hacer esto, con cada línea que leemos, recorremos
 * con un bucle for el array de propiedades, buscando a qué
 * propiedad antigua hace alusión cada línea y cuál es la nueva.
 *

 */
while (((s2 = lr2.readLine()) != null) && (s2.indexOf(cadenaBuscamos2) == -1)) {
    if(debugger) System.out.println("CONTENIDO S2: "+s2);

    // Bucle que busca las propiedades antiguas y las sustituye por las nuevas,
    en cada línea que se lee
    for (int i = 0; i < numDePropiedades; i++) {

        // Cuando encuentra de qué propiedad se trata, obtiene el contenido
        // del fichero antiguo, y lo agrega con el nuevo formato.
        if (s2.indexOf(propiedades[i]) != -1) {
            String contenidoEtiqueta = null;
            int tamPropiedad = propiedades[i].length();
            int tamS2 = s2.length();
            contenidoEtiqueta = s2.substring(tamPropiedad + 2, tamS2 -
            tamPropiedad - 3);


            if(debugger) System.out.println("<" + propiedadesNuevas[i] + ">" +
            contenidoEtiqueta + "</" + propiedadesNuevas[i] + ">");
            pw.println("<" + propiedadesNuevas[i] + ">" + contenidoEtiqueta +
            "</" + propiedadesNuevas[i] + ">");



        }
    }
}
```

```
                if(debugger) System.out.println("</" + nuevoIdentificador + ">");
                // Al finalizar con el contenido del objeto, cierra la instanciación del objeto
                de esa clase que hemos creado.
                pw.println("</" + nuevoIdentificador + ">");
            }

        }
        // Cuando finaliza con la lectura , cierra el fichero de lectura y de escritura.
        lr1.close();
        fichero.close();
        JOptionPane.showMessageDialog(frame,"El fichero se convirtió correctamente! :)");
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(frame,"El fichero que has introducido es ilegible o no
        existe! :(","Error, problema con el fichero de entrada",    JOptionPane.ERROR_MESSAGE);
    } finally {
        // En el finally cerramos el fichero, para asegurarnos
        // que se cierra tanto si todo va bien como si salta
        // una excepcion.
        try {
            if (null != lector1) {
                lector1.close();
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }

}
```

Ilustración 45 Código Fuente ParserRDFtoOWL.java

1.2. *VentanaPrincipalParser.java*

Contiene la interfaz gráfica del programa creada en con la biblioteca **Swing** de **JAVA**. El código no lo adjunto ya que es bastante comprensible y está anexado a este proyecto.

1.3. Funcionamiento

Para poder ejecutar esta aplicación, es necesario tener instalada la máquina virtual de JAVA. JAVA es un lenguaje que destaca, entre otras cosas, por ser multiplataforma y esto lo consigue gracias a su sistema de ejecución de ficheros a través de su máquina virtual.

Gracias a ello, tan solo debemos instalar la máquina virtual de JAVA en cualquiera de nuestras plataformas (ya sea Windows, MacOS o sistemas UNIX) y ya podremos hacer funcionar cualquiera de nuestras aplicaciones creadas en JAVA. La web para poder descargar la JVM es la siguiente: http://www.oracle.com/technetwork/java/javase/downloads/index.html . Necesitamos instalar el JRE, y en caso de querer desarrollar en JAVA, instalaremos el JDK que incluye el JRE.

En este caso, el ordenador utilizado para probar este programa, cuenta con el sistema operativo Windows 7 Ultimate en su versión de 64bits. Si tenemos bien configurada nuestra máquina virtual, con tan solo hacer “doble click” sobre el fichero generado en el proyecto (ParserRDFtoOWL.jar) se abrirá la aplicación.

Para poder mostrar todas las opciones que tiene este pequeño programa, voy a mostrar la ejecución del mismo desde la consola de comandos de Windows (Inicio > Ejecutar > CMD). Nos dirigimos al directorio que contenga nuestro fichero .jar y ejecutamos la siguiente línea:

java -jar ParserRDFtoOWL.jar

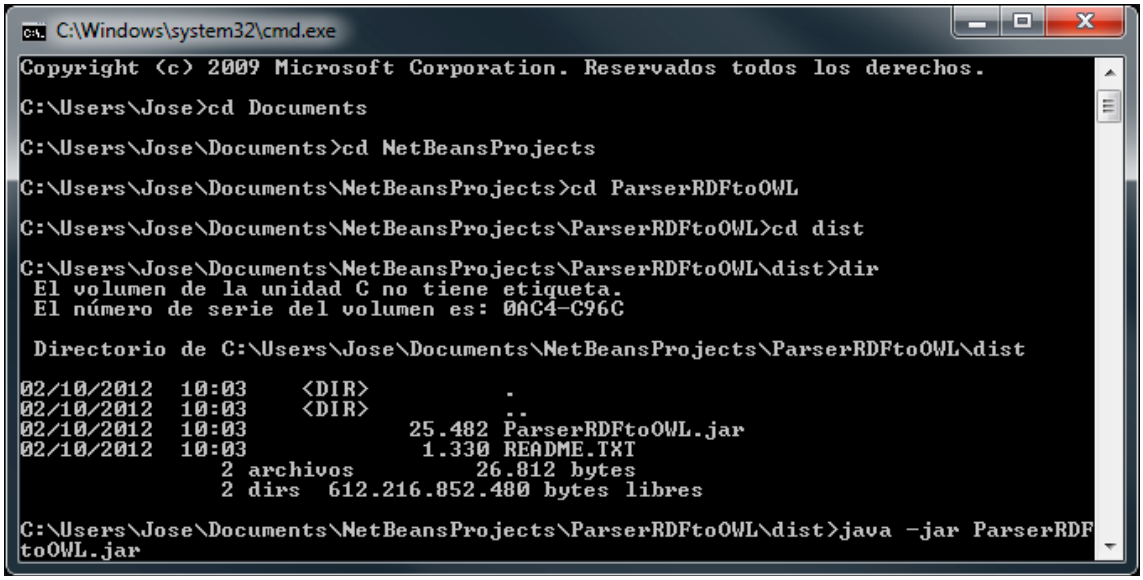

Ilustración 46 Compilación de ParserRDFtoOWL.java

Realizando esta ejecución, se mostrará la ventana de nuestro programa principal con la que podremos interactuar y comenzar a trabajar.

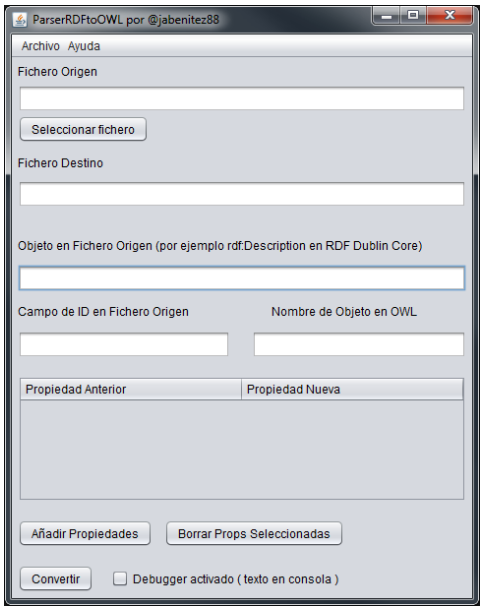

Ilustración 47 Interfaz gráfico del parser desarrollado

Nos encontramos una ventana que contiene los siguientes elementos:

- **Zona de Fichero Origen**: Contiene una etiqueta JLabel con el título “Fichero Origen”, un campo de tipo JTextField, no editable, en el que veremos la ruta y nombre de nuestro fichero de origen a convertir y un botón de tipo JButton para Seleccionar el fichero, este abrirá una ventana de dialogo para poder seleccionar nuestro fichero de origen entre las diferentes carpetas de nuestro ordenador.
- **Zona de Fichero Destino**: Está compuesto por una etiqueta JLabel con título Fichero Destino y un campo JTextField en el que introduciremos la ruta de nuestro fichero de

destino y el nombre. Para facilitar al usuario esta tarea, el programa escribe de forma automática en este recuadro el nombre del fichero a convertir añadiéndole al final "_ok.owl", de manera que, si tenemos un fichero origen como el siguiente:

- C:\ficherosOntologia\ficheroOrigen.txt
- El programa escribirá en el recuadro de fichero de destino C:\ficherosOntologia\ficheroOrigen_ok.owl de forma automática, pudiendo luego ser editado por el usuario.

- **Zona de Objeto en Fichero Origen:** Compuesta por una etiqueta JLabel y un campo JTextField para escribir nuestro objeto a convertir. Aquí lo que debemos convertir es la clase principal de nuestro fichero origen. En nuestro caso, al querer pasar de formato RDF DC a otro formato, nuestro objeto principal se encuentra dentro de las etiquetas **<rdf:Description>** y **</rdf:Description>.** Siendo así, lo que debemos escribir en este recuadro es rdf:Description .
- **Zona de identificador en el origen y nueva clase principal del nuevo formato**: Está compuesta por 2 etiquetas JLabel y dos campos de texto JTextField. Lo que debemos escribir aquí es lo siguiente:
  - Comenté anteriormente que nuestro fichero en RDF DC poseía una propiedad interna que contenía un identificador del objeto. Esta propiedad se llama **<dc:identifier>**, es por ello, que en nuestro caso particular, lo que debemos escribir en el primer JTextField es **dc:identifier** , para que nuestro programa realice la posterior búsqueda.
  - En relación a la clase principal, al tratarse de Fichas de distintos elementos, nuestra clase principal se llamará **Ficha** , y esta es la palabra que escribiremos en el segundo campo de texto.
- **Zona de propiedades en fichero origen y en fichero de destino:** Esta zona se compone de una tabla JTable, con un modelo de tabla DefaultTableModel, en el cual añadiremos filas compuestas por dos columnas. En la primera columna se inserta el nombre de la etiqueta en el formato del fichero origen, y en la segunda columna se inserta el nombre de la etiqueta en el formato de destino. Por ejemplo: en el fichero en formato RDF DC, tenemos una propiedad que se llama **<dc:type>** y en nuestro nuevo formato, esa propiedad pasará al llamarse **<Tipo>**, pues deberemos crear una fila, clickando sobre el botón "Añadir propiedades" y escribiremos en la fila creada **dc:type** y **Tipo** respectivamente.

  Si en algún momento tuviésemos algún error, o añadiésemos más campos de la cuenta, podemos solucionarlo seleccionando las filas que nos necesitemos y pulsando sobre el botón "Borrar Props seleccionadas"
- **Zona de conversión y debugger:** En la zona inferior de nuestra aplicación, tenemos un botón Convertir y un CheckBox para activar o desactivar un depurador. Para realizar el programa y comprobar que todo funciona correctamente, inserté unas impresiones en consola a medida que se ejecutaban distintas partes del programa.

  Una vez finalizado y viendo que el programa funcionaba correctamente, decidí crear una variable booleana que permitiera al usuario elegir si quería leer estos mensajes en consola, o no.

Rellenando nuestro programa con los datos de nuestro caso particular, obtendríamos una ventana como la siguiente:

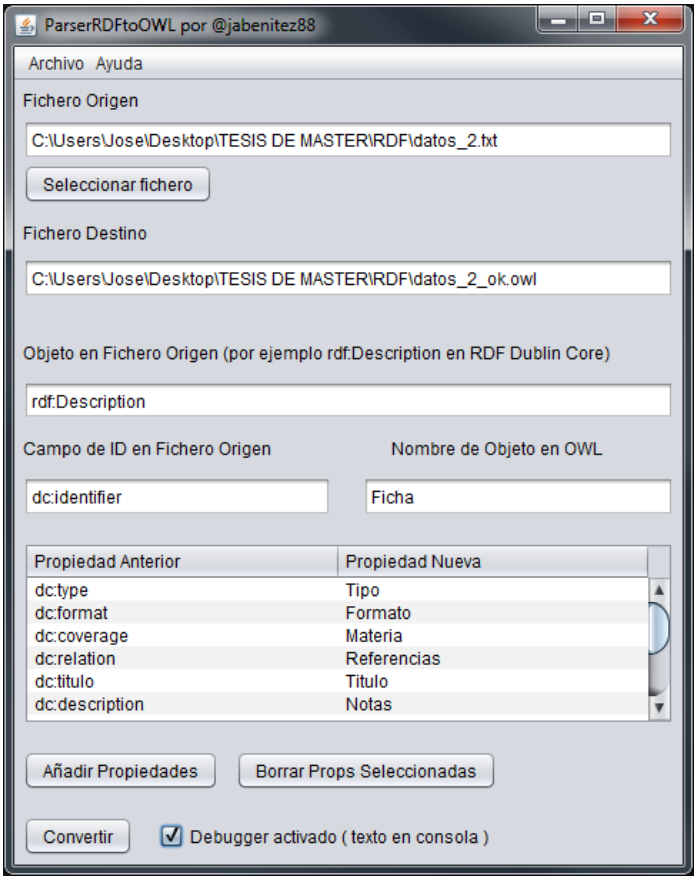

Ilustración 48 Parser con datos insertados

Al clickar en convertir, teniendo el debugger activo, se pueden ver mensajes en la consola de MSDOS como se muestra en la siguiente figura:

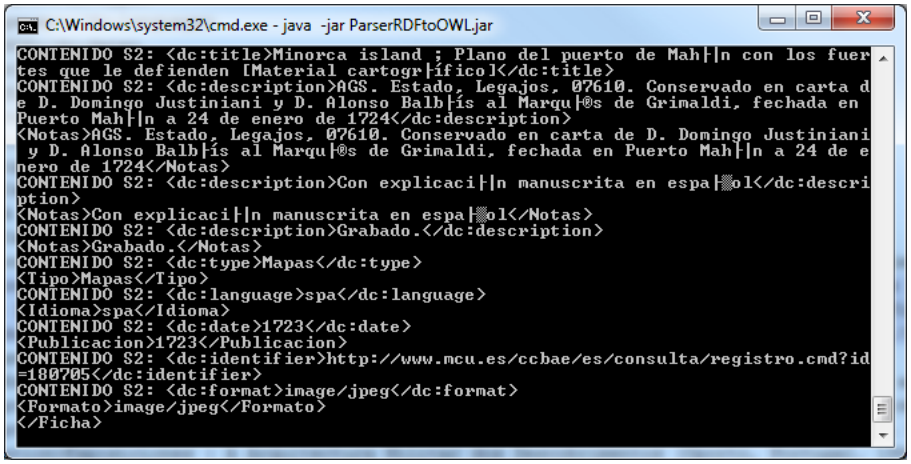

Ilustración 49 Consola con modo debug activado

ANEXO 2: INSTALACIÓN DE LUCENE Y SOLR

1. LUCENE

En la librería Lucene, una vez se tiene el listado de stopwords que se desea utilizar, se puede crear una clase que heredará de **org.apache.lucene.analysis.Analyzer** con el siguiente código:

```
03. import java.io.File;
04. import java.io.IOException;
05. import java.io.Reader;
06. import java.util.HashSet;
07. import java.util.Set;
08. 
09. import org.apache.lucene.analysis.Analyzer;
10. import org.apache.lucene.analysis.LowerCaseFilter;
11. import org.apache.lucene.analysis.StopFilter;
12. import org.apache.lucene.analysis.TokenStream;
13. import org.apache.lucene.analysis.WordlistLoader;
14. import org.apache.lucene.analysis.standard.StandardFilter;
15. import org.apache.lucene.analysis.standard.StandardTokenizer;
16. 
17. 
18. /** Filters {@link StandardTokenizer} with {@link StandardFilter}, {@link
19.  * LowerCaseFilter}, {@link StopFilter} and {@link SpanishStemFilter}. */
20. 
21. /**
22.  * Analyzer for Spanish using the SNOWBALL stemmer. Supports an external list of stopwords
23.  * (words that will not be indexed at all).
24.  * A default set of stopwords is used unless an alternative list is specified, the
25.  * exclusion list is empty by default.
26.  *
27.  * @author jose
28.  */
29. 
30. public class SpanishAnalyzer extends Analyzer {
31. 
32.     /** An array containing some common Spanish words that are usually not
33.      * useful for searching. Imported from http://www.unine.ch/info/clef/.
34.      */
35.     // TODO: no pego en el tutorial el listado de stopWords utilizado para
36.     // no sobredimensionarlo, son 351 términos.
37.     public static final String[] SPANISH_STOP_WORDS = { "" };
38. 
39.     /**
40.      * Contains the stopwords used with the StopFilter.
41.      */
42.     private Set<Object> stopTable = new HashSet<Object>();
43. 
44.     /**
45.      * Contains words that should be indexed but not stemmed.
46.      */
47.     private Set<Object> exclTable = new HashSet<Object>();
48. 
49.     /**
50.      * Builds an analyzer with the default stop words.
51.      */
52.     public SpanishAnalyzer() {
53.         stopTable = StopFilter.makeStopSet(SPANISH_STOP_WORDS);
54.     }
55. 
56.     /** Builds an analyzer with the given stop words. */
57.     public SpanishAnalyzer(String[] stopWords) {
58.         stopTable = StopFilter.makeStopSet(stopWords);
59.     }
60. 
61.     /**
62.      * Builds an analyzer with the given stop words from file.
63.      * @throws IOException
64.      */
65.     public SpanishAnalyzer(File stopWords) throws IOException {
66.         stopTable = new HashSet(WordlistLoader.getWordSet(stopWords));
67.     }
68. 
69.     /** Constructs a {@link StandardTokenizer} filtered by a {@link
70.      * StandardFilter}, a {@link LowerCaseFilter}, a {@link StopFilter}
71.      * and a {@link SpanishStemFilter}. */
72.     public final TokenStream tokenStream(String fieldName, Reader reader) {
73.         TokenStream result = new StandardTokenizer(reader);
74.         result = new StandardFilter(result);
75.         result = new LowerCaseFilter(result);
76.         result = new StopFilter(result, stopTable);
77.         result = new SpanishStemFilter(result);
78.         return result;
79.     }
80. }
```

Ilustración 50 Código Fuente de Detector de Stopwords en Lucene

En la constante **SPANISH_STOP_WORDS** incluiremos el listado por defecto de stopWords en castellano que tendrá el analizador. Lo ideal sería que el listado final de stopWords se

obtuviese de un recurso externo parametrizable (un fichero de propiedades, base de datos...). En el método **tokenStream** es donde se filtra el contenido, para ello hemos incluido varios filters:

- StandardFilter: básicamente, elimina los signos de puntuación,
- LowerCaseFilter: convierte el contenido a minúsculas,
- StopFilter: filtrará el contenido con el listado de stopWords,
- SpanishStemFilter: clase propia que explico a continuación

SpanishStemFilter

Nuestro filtro de stemming heredará de **org.apache.lucene.analysis.TokenFilter** y tendrá el siguiente código fuente:

```
02.
03.  import java.io.IOException;
04.
05.  import net.sf.snowball.ext.SpanishStemmer;
06.
07.  import org.apache.lucene.analysis.Token;
08.  import org.apache.lucene.analysis.TokenFilter;
09.  import org.apache.lucene.analysis.TokenStream;
10.
11.  /**
12.   * Spanish stemming algorithm.
13.   */
14.  public final class SpanishStemFilter extends TokenFilter {
15.
16.      private SpanishStemmer stemmer;
17.      private Token token = null;
18.
19.      public SpanishStemFilter(TokenStream in) {
20.          super(in);
21.          stemmer = new SpanishStemmer();
22.      }
23.
24.      /** Returns the next input Token, after being stemmed */
25.      public final Token next() throws IOException {
26.          if ((token = input.next()) == null) {
27.              return null;
28.          }
29.          else {
30.              stemmer.setCurrent(token.termText());
31.              stemmer.stem();
32.              String s = stemmer.getCurrent();
33.              if ( !s.equals( token.termText() ) ) {
34.                  return new Token( s, token.startOffset(),
35.                  token.endOffset(), token.type() );
36.              }
37.              return token;
38.          }
39.      }
40.
41.      /**
42.       * Set a alternative/custom Stemmer for this filter.
43.       */
44.      public void setStemmer(SpanishStemmer stemmer) {
45.          if ( stemmer != null ) {
46.              this.stemmer = stemmer;
47.          }
48.      }
49.  }
```

Ilustración 51 Código fuente de SpanishStemmer

2. SOLR

Los pasos a realizar para la utilización de Apache Solr son los siguientes:

- Copia del índice.
- Despliegue de servidor sobre el servidor de aplicaciones.
- Configuración de enlace entre servidor e índice. (Modificación del fichero web.xml)

Para poder realizar la indexación de contenido, se debe definir el esquema de datos teniendo en cuenta las siguientes indicaciones:

- El fichero schema.xml es un archivo XML que define la estructura de datos a indexar.
- Estructura de campo:

```
<field name=“nombre de campo" type=“tipo de dato" />
```

- Tipos de Datos definidos por clases java.
- Parámetros opcionales:
    - default: Valor a usar si no se recibe ninguno
    - required: Define si un campo es obligatorio.
    - indexed: Determina si un campo es buscable u ordenable.
    - stored: Determina si un campo se puede recuperar en una consulta.
    - multiValued: El campo contiene más de un valor.
- Canales para el envío de documentos a la hora de indexarlo:
    - Petición HTTP: Se puede realizar el envío de una instrucción y de los datos asociados vía HTTP POST.
    - Cliente Solrj: Otra posibilidad es utilizar un cliente realizado en JAVA llamado Solrj. Este cliente nos permite realizar las diferentes operaciones sobre el índice y enviar la información en diferentes formatos.
- Fuentes de datos para la indexación:
    - XML: Coherente con la estructura de datos definida.
    - Objetos Java: Representación binaria del documento XML.
    - CSV: Documento de texto con valores separados.
    - Documentos enriquecidos: PDF, XLS, DOC, PPT, …
    - Base de Datos: Adaptador intermedio (DataImportHandler).

Para realizar las búsquedas con Solr, una vez indexado el contenido, hayque tener en cuenta lo siguiente:

1. Hay tres canales de búsqueda:
    - Petición HTTP: Envío de instrucción de búsqueda y parámetros mediante HTTP GET.
    - Administrador de Solr: proporciona un recubrimiento para simplificar la petición HTTP.

- Cliente Solrj: Posee también los métodos necesarios para realizar búsquedas sobre los índices.

2. Respuesta como estructura XML.

Los distintos parámetros para poder realizar la búsqueda son los siguientes:

- **q:** Petición con formato "campo:valor"
- **start:** Documento inicial a partir del cual se van a mostrar los resultados.
- **rows:** Indica el número máximo de resultados a mostrar.
- **facets:** Indica si se desean mostrar facetas. Parámetros adicionales para indicar el campo por el que realizarlas, límite, ordenación, etc.
- **sort:** Define la ordenación de los resultados. Ordenaciones combinadas. Formato de ordenaciones: "precio desc, nombre asc"
- **fl:** Campos que se devuelven en la respuesta
- **fq:** Mismo formato que "q". Limita la query (actúa como filtro). Los resultados se cachean.
- Otros …

Para la búsqueda avanzada, Solr cuenta con **wildcards** y con **operadores lógicos**:

- **Wildcards:** Solr no permite wildcards iniciales
  - word*: * sustituye a cualquier número de caracteres.
  - w?rd: ? sustituye a un único carácter.
  - w?*d: Pueden componerse ambos comodines.
- Operadores Lógicos
  - AND: word1 AND word2 = word1 && word2 = +word1 +word2
  - OR: word1 OR word2 = word1 || word2 = word1 word2
  - NOT: word1 NOT word2 = word1 -word2

Apache Solr cuenta además con una serie de analizadores de texto que ayudan a conseguir nuestro objetivo a la hora de realizar un estudio comparativo. No obstante, si los analizadores que contiene Solr no sirven, se puede crear uno con la estructura necesaria e implementarlo dentro de Solr.

Estos analizadores son configurables por XML (schema.xml), aplicables a campos específicos, aplicables en tiempo de indexación, y durante la búsqueda o en ambos. Algunos de los que proporciona Solr son :

- Tokenización
- Stemming
- Sinónimos
- Stop Words
- N-Gramas

Brevemente consisten en:

- **Tokenización:** División de texto mediante diferentes expresiones (Espacios en blanco, etiquetas html, signos de puntuación, expresiones regulares…)

```
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
```

- **Stemming:** Reducción de términos derivados a su forma raíz.
- **Sinónimos:** Transformación de texto mediante definición explícita de

```
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
```

```
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
```

- **Stop words:** Eliminación de palabras no significativas para el proceso de búsqueda.
- **N-Gramas:** Separación de texto en los diferentes grupos de caracteres que lo

```
<filter class="solr.StopFilterFactory" words="stopwords.txt" ignoreCase="true"/>
```

```
<filter class="solr.NGramFilterFactory" minGramSize="2" maxGramSize="15"/>
```

Solr permite añadir una serie de componentes de búsqueda que no vienen por defecto. Los componentes de búsqueda proporcionan funciones adicionales a la recuperación de resultados. Además, son configurables en tiempo de consulta (algunos pueden necesitar configuración adicional en los archivos xml). Algunos de los componentes de búsqueda que proporciona Solr son:

- Highlighting: Resaltado de términos.
- Spell Checker: Corrección ortográfica.
- More Like This: Resultados similares.

Highlighting: Ofrece el resaltado de términos buscados dentro de una cadena de texto.

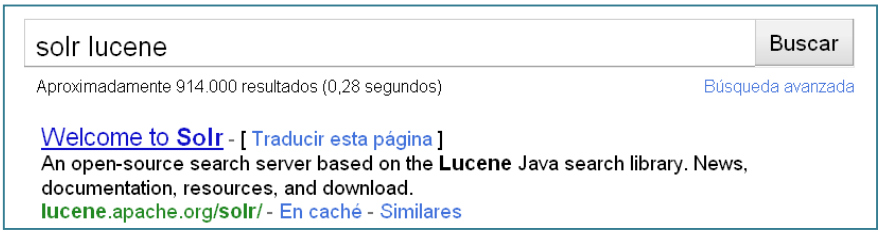

Ilustración 52 Resultados de búsqueda de Lucene

Algunos parámetros de configuración:

- hl=true: Activa el resaltado de términos
- hl.fl=[fieldnames]: Campo o campos sobre los que se realizará el resaltado.
- hl.simple.pre / hl.simple.post=[etiqueta]: Etiqueta que se añadirá antes y después del término resaltado (Ej. término)

Spell Checker: Proporciona sugerencias para errores ortográficos basadas en el contenido indexado o en un diccionario. Para sugerencias por contenido indexado es recomendable crear un campo para corrección ortográfica (De tipo "textSpell" o "textSpellPhrase").

Configuración:

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <str name="queryAnalyzerFieldType">textSpell</str>
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="classname">solr.IndexBasedSpellChecker</str>
      <str name="field">corrector</str>
    <str name="spellcheckIndexDir">./spellchecker</str>
  </lst>
</searchComponent>
```

More Like This: Componente de sugerencia de documentos similares. Se basa en los términos que aparecen en el documento recuperado y la frecuencia de los mismos.

Configuración por parámetros de búsqueda:

- mlt=true: Activa la utilidad de resultados similares.
- mlt.fl=[fieldnames]: Campo o campos analizados.
- mlt.count=[número]: Número de resultados devueltos.
- mlt.qf=[field1^2.0 field2^5.0]: Configuración de ponderación sobre diferentes campos para el cálculo de similitud.
- Otros …

3. INSTALACIÓN Y PUESTA EN MARCHA DE APACHE SOLR

Para realizar la instalación de Apache Solr, he utilizado un ordenador que tengo actuando de servidor con la distribución **Ubuntu 10.04.2 LTS Server**.

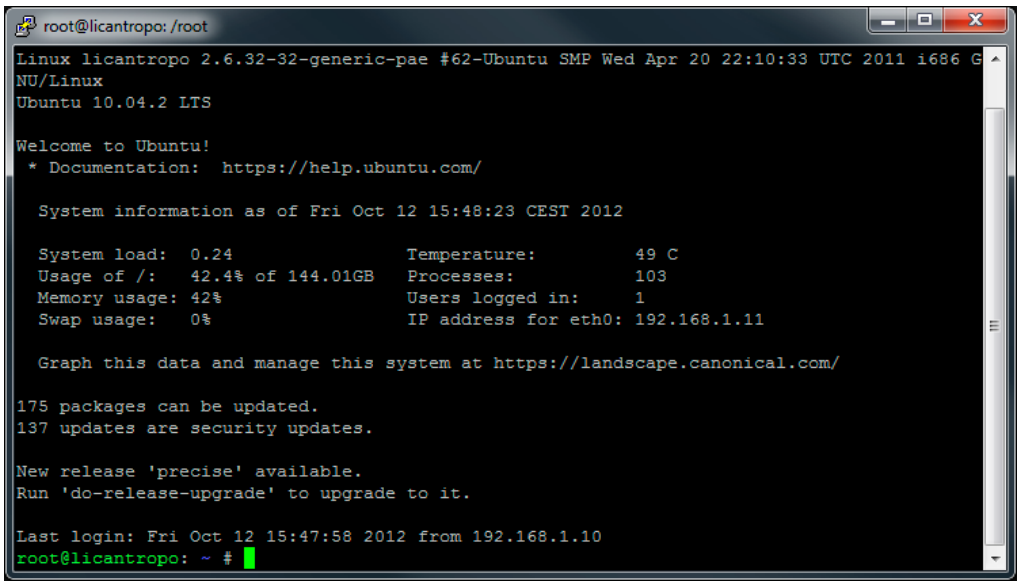

Ilustración 53 Datos de la máquina donde instalamos Solr

Para poder trabajar con facilidad desde la máquina donde estoy realizando este trabajo, accedo al servidor donde voy a instalar Apache Solr mediante el programa **Putty** , mediante el cual me conecto al servidor vía **SSH** accediendo por la IP local.

Una vez dentro, al disponer de esta distribución de Linux, procedente de debían, lo primero que necesito instalar para poder hacer funcionar Apache Solr es el servidor Tomcat. Tomcat es un servidor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Para instalar tomcat, utilizo el gestor de paquetes de Ubuntu **aptitude**, indicando que deseo instalar tomcat tecleando **sudo apt-get install tomcat6**

Ilustración 54 Instalación de tomcat6 con el gestor de paquetes apt-get

Instalado este servidor, el siguiente paso es descargar el paquete de Apache Solr, de la web de apache y proceder a su instalación, realizando los siguientes pasos:

```
mkdir -p ~/tmp/solr/
cd ~/tmp/solr/
wget http://apache.ziply.com/lucene/solr/3.6.1/apache-solr-3.6.1.tgz
tar xzvf apache-solr-3.6.1.tgz
```

Como todos los índices y núcleos de Solr van a ir al directorio **/var/solr** creamos este directorio:

```
sudo mkdir -p /var/solr
```

Una vez hecho esto, el siguiente paso es copiar la aplicación web Solr junto con sus ejemplos y ficheros de configuración al directorio que hemos creado en el paso anterior, para ello ejecutamos en la línea de comando las siguientes instrucciones:

```
sudo cp apache-solr-3.6.1/dist/apache-solr-3.6.1.war /var/solr/solr.war
sudo cp -R apache-solr-3.6.1/example/multicore/* /var/solr/
sudo chown -R tomcat6 /var/solr/
```

Apuntamos Catalina en Solr de la siguiente manera:

```
echo -e '<Context docBase="/var/solr/solr.war" debug="0" privileged="true"
allowLinking="true" crossContext="true">\n<Environment name="solr/home"
type="java.lang.String" value="/var/solr" override="true" />\n</Context>' | sudo tee -a
/etc/tomcat6/Catalina/localhost/solr.xml

echo 'TOMCAT6_SECURITY=no' | sudo tee -a /etc/default/tomcat6
```

Seguidamente, editamos el fichero de iniciación de tomcat6 para asignar la solr.home en el directorio /var/solr:

```
JAVA_OPTS="$JAVA_OPTS -Dsolr.home=/var/solr"
```

Reiniciamos nuestro servidor tomcat6 para que surtan efecto estos cambios:

```
sudo /etc/init.d/tomcat6 restart
```

Y ya tengo acceso a Solr desde mi navegador, tecleando la ip de mi servidor, o su nombre, de la siguiente forma http://licantropo:8080/solr/:

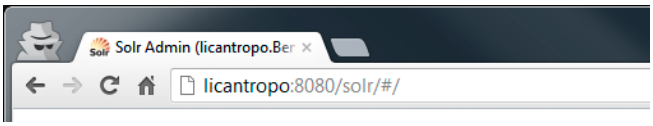

Ilustración 55 Dirección de servidor local haciendo funcionar Solr

De esta manera, accedemos a la interfaz de Solr como se muestra en la siguiente figura:

Ilustración 56 Pantalla inicial de Solr

ANEXO 3: CREACIÓN DEL ÍNDICE DEL CATÁLOGO EN APACHE SOLR

Una vez instalado Apache Solr, el siguiente paso es crear los directorios donde se almacenan las fichas categorizadas por sus características y sin categorizar, para poder hacer una comparación en la búsqueda de los resultados en ambos casos.

Para poder conseguir esto, lo primero que debemos ejecutar es el siguiente comando de Linux, que realiza una copia del directorio **core0** que trae solr por defecto:

```
sudo cp -Rp /var/solr/core0 /var/solr/fichasFacetadas
sudo cp -Rp /var/solr/core0 /var/solr/fichas
```

nuevos directorios.

Para ellos hemos ejecutado: **nano /var/solr/solr.xml** y hemos insertado dos nuevos <cores>

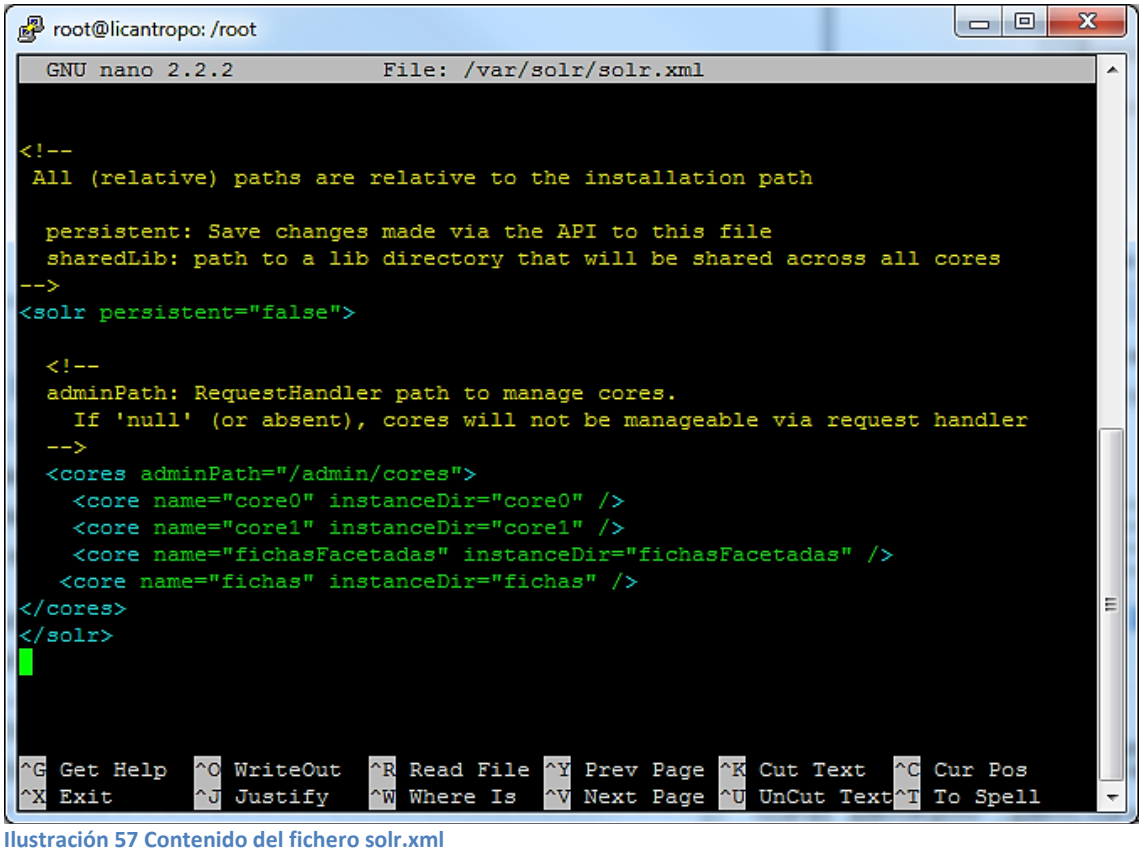

Ilustración 57 Contenido del fichero solr.xml

En el caso de las fichas facetadas, debemos especificar qué tipos de datos queremos almacenar para realizar las posteriores búsquedas teniendo en cuenta distintas características, para ello editamos el fichero de configuración del núcleo de **fichasFacetadas**:

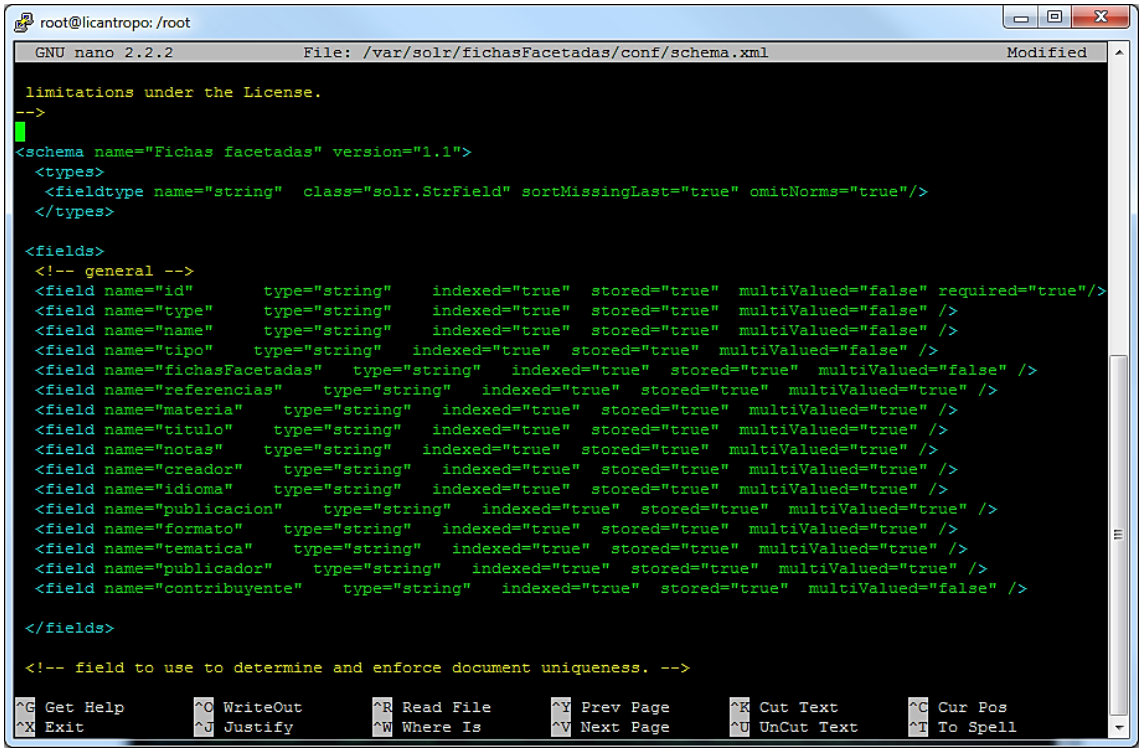

Ilustración 58 Fichero de configuración schema.xml de las fichasFacetadas

Hemos declarado los campos que habíamos tratado en los puntos anteriores, para poder almacenar la información de cada ficha con sus respectivas características almacenadas en los distintos campos que hemos creado, que son los que se ven en la figura anterior:

```
<fields>
 <!-- general -->
 <field name="id"       type="string"    indexed="true"  stored="true"  multiValued="false" required="true"/>
 <field name="type"     type="string"    indexed="true"  stored="true"  multiValued="false" />
 <field name="name"     type="string"    indexed="true"  stored="true"  multiValued="false" />
 <field name="tipo"     type="string"    indexed="true"  stored="true"  multiValued="false" />
 <field name="fichasFacetadas"   type="string"   indexed="true"  stored="true"  multiValued="false" />
 <field name="referencias"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="materia"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="titulo"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="notas"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="creador"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="idioma"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="publicacion"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="formato"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="tematica"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="publicador"    type="string"   indexed="true"  stored="true"  multiValued="true" />
 <field name="contribuyente"    type="string"   indexed="true"  stored="true"  multiValued="false" />

</fields>
```

Ilustración 59 Fichero schema.xml de las fichas sin facetar

El siguiente paso a realizar es la creación de un fichero .xml con el que podamos insertar la información de las **7792 fichas** de forma rápida, quedando así indexadas en nuestro sistema SOLR.

La estructura del fichero sería como la siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<add>
    <doc>
        <field name="referencias">Referencias: Mapas, planos y dibujos (Años 1503-1805).
        Volumen I : p. 405</field>
        <field name="materia">-S.XVIII</field>
        <field name="titulo">[ ...] Planos y Perfiles de la obra que se construye antes de
        empezarse a fundir para recipientes del Mineral y que se deshace despues de evacuada la
        fundición que es la que se supone llamarse Crisol [Material gráfico no proyectable]
        </field>
        <field name="notas">AGS. Secretaría de Marina, 00679. Acompaña a carta de don Maximino
        de la Croix a don Ricardo Wall, Chaves, 16 de julio de 1762</field>
        <field name="notas">Tinta y colores. Con explicación</field>
        <field name="notas">Manuscrito sobre papel.</field>
        <field name="tipo">Ilustraciones y Fotos</field>
        <field name="idioma">spa</field>
        <field name="publicacion">1762</field>
        <field name="id">http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=178725</field>
        <field name="formato">image/jpeg</field>
        <field name="tematica">Hornos metalúrgicos-Dibujos</field>
    </doc>
    <doc>
        <field name="titulo">Pamplona. Hospitales. Planos. 1721</field>
        <field name="referencias">Referencias: Mapas, planos y dibujos (Años 1503-1805).
        Volumen I : p. 799</field>
        <field name="materia">España-Navarra (Comunidad Autónoma)-Pamplona</field>
        <field name="titulo">[ 1° 2° y 3° Planos según el proyecto del ingeniero] [Material
        cartográfico]</field>
        <field name="notas">Contiene : Primer plano que contiene 68 camas en cada suelo ;
        Segundo Plano q[ue] contiene 168 Camas en cad suelo ; Tercer Plano primera Idea
        contiene 161 camas en cada suelo ; Segunda Idea del tercer Plano 180 camas en cada suelo
        </field>
        <field name="notas">AGS. Secretaría de Guerra, Legajos, 02452. Acompaña a la exposición
        y detalle del costo e incidencias del proyecto del hospital que hace Félix Ponsich al
        Sr. Marqués de Castelar, Pamplona, 29 de enero de 1721</field>
        <field name="notas">Tinta y colores a la aguada. Con rotulación</field>
        <field name="notas">Manuscrito sobre papel. Forma irregular.</field>
        <field name="tipo">Mapas</field>
        <field name="idioma">spa</field>
        <field name="publicacion">1721</field>
        <field name="creador">Mauleón, Francisco</field>
        <field name="id">http://www.mcu.es/ccbae/es/consulta/registro.cmd?id=179879</field>
        <field name="formato">image/jpeg</field>
    </doc>
</add>
```

Ilustración 60 Fichas en formato XML aceptado por Solr

Antes de realizar la indexación, debemos eliminar del fichero los caracteres de control que puedan provocar un fallo por parte del indexador, esto se soluciona ejecutando el siguiente comando:

```
tr -d [:cntrl:] < datos_facetados.xml > datos_facetados_limpio.xml
```

Teniendo en cuenta que necesitaremos indexar ficheros a menudo, para facilitar el trabajo hemos realizado un script en **bash** que nos ayudará a realizar esta tarea:

```
jabenitez@licantropo: ~/TFM $ nano indexador.sh
```

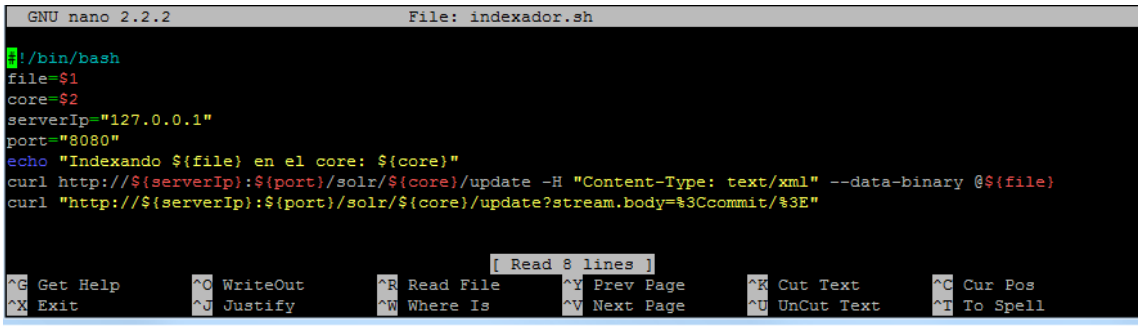

Ilustración 61 Contenido de indexador.sh

Después de haber guardado nuestro script, procedemos a insertar las fichas facetadas y sin facetar en sus respectivos núcleos:

```
sh indexador.sh datos_facteados_limpio.xml fichasFacetadas
sh indexador.sh datos_liso_limpio.xml fichas
```

Obtenemos el siguiente mensaje que nos confirma que se ha indexado todo correctamente:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">0</int><int name="QTime">199</int></lst>
</response>
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">0</int><int name="QTime">78</int></lst>
</response>
```

ANEXO 4: DESARROLLO DE UN INTERFAZ GRÁFICO PARA BÚSQUEDA

La interfaz de búsqueda está realizada con los lenguajes de programación **PHP, HTML5, CSS3, JAVASCRIPT y AJAX** (realizando consultas de forma directa a nuestro servidor de Apache Solr recibiendo los datos con la estructura **JSON** para poder trabajar con los datos en JavaScript).

Los directorios y ficheros de los que se compone nuestra interfaz son los siguientes:

- css
  - estilo.css : este fichero es la hoja de estilos del sitio web, en el cual hemos definido la estructura de cada elemento de nuestra web, las fuentes, los colores, los fondos y los efectos del sitio.
- include
  - head.php : contiene el título del sitio web, las metaetiquetas y la declaración de los ficheros javascript que van a ser utilizados, dependiendo del buscador en el que nos encontremos. Si elegimos el buscador semántico, se realizará la llamada al fichero **js/buscador.js** , si por el contrario hemos elegido el buscador textual, se realiza la llamada al fichero **js/buscadorTextual.js**.

```
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
   <title>Buscador textual y ontológico de mapas, planos y dibujos del
AGS</title>
   <link rel="stylesheet" href="css/estilo.css" media="screen" />
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"
></script>
   <?php if($modo == 0){ ?>
        <script type="text/javascript" src="js/buscador.js"></script>
   <?php }else if($modo == 1){ ?>
   <script type="text/javascript"
src="js/buscadorTextual.js"></script>
   <?php } ?>
```

- js
  - buscadorTextual.js

```
var respuestasBusqueda1 = Array("Los mapas que fueron realizados","Los planos
que fueron realizados","Los dibujos que fueron realizados","Los autores que
realizaron las obras","Las técnicas utilizadas para las obras","El lugar donde
se realizaron las obras","La técnica utilizada para realizar las obras","Las
fechas en las que se realizaron las obras");
var respuestasBusqueda2 = Array("con título","en el idioma","en los idiomas",
"con los colores","entre las fechas","en la época","con la temática","sobre el
soporte","por el autor","sobre la ciudad","sobre el país","sobre el continente"
,"mediante la técnica");

// Listado de stopwords que utilizaremos para que sean elimanadas de la
consulta realizada por el usuario
// gracias a una función auxiliar que recorrerá el array buscando cada una de
las palabras
// y borrándolas de la cadena insertada por el usuario.

var stopwords = Array("un "," una "," unas "," unos "," uno "," sobre "," todo
"," también "," tras "," otro "," algún "," alguno "," alguna "," algunos ","
algunas "," ser "," es "," soy "," eres "," somos "," sois "," estoy "," esta "
," estamos "," estais "," estan "," en "," para "," atras "," porque "," por
qué "," estado "," estaba "," ante "," antes "," siendo "," ambos "," pero ","
poder "," puede "," puedo "," podemos "," podeis "," pueden "," fui "," fue ",
" fuimos "," fueron "," hacer "," hago "," hace "," hacemos "," haceis ","
hacen "," cada "," fin "," incluso "," primero "," conseguir "," consigo ","
consigue "," consigues "," conseguimos "," consiguen "," ir "," voy "," va ","
vamos "," vais "," van "," vaya "," gueno "," ha "," tener "," tengo "," tiene
"," tenemos "," teneis "," tienen "," el "," la "," lo "," las "," los "," su "
," aqui "," mio "," tuyo "," ellos "," ellas "," nos "," nosotros "," vosotros
"," vosotras "," si "," dentro "," solo "," solamente "," saber "," sabes ","
sabe "," sabemos "," sabeis "," saben "," ultimo "," largo "," bastante ","
haces "," muchos "," aquellos "," aquellas "," sus "," entonces "," tiempo ","
verdad "," VERDADERO "," verdadera "," ciertos "," cierta "," ciertas ","
intentar "," intento "," intenta "," intentas "," intentamos "," intentais ","
intentan "," dos "," bajo "," arriba "," encima "," usar "," uso "," usas ","
usa "," usamos "," usais "," usan "," emplear "," empleo "," empleas ","
emplean "," ampleamos "," empleais "," valor "," muy "," era "," eras ","
eramos "," eran "," modo "," bien "," cual "," mientras "," con "," entre ","
sin "," trabajo "," trabajar "," trabajas "," trabaja "," trabajamos ","
trabajais "," trabajan "," podria "," podrias "," podriamos "," podrian ","
podriais "," yo "," aquel ","textoBusqueda="," de ", " a "," ante ", " bajo ",
" cabe ", " con ");

/*
    function eliminarDuplicados(arr)

    Función que recibe un array y elimina los elementos que sean exactamente
    iguales dentro del array.
    De esta forma, si insertamos en un array de cadenas de texto la misma
    cadena varias veces, gracias
    a esta función se eliminaran las cadenas repetidas.
*/
function eliminarDuplicados(arr) {
 var i,
     len=arr.length,
     out=[],
     obj={};

 for (i=0;i<len;i++) {
    obj[arr[i]]=0;
 }
 for (i in obj) {
    out.push(i);
 }
 return out;
}
```

```
/*
    remover_acentos(str)

    Función que recibe una cadena de texto y elimina los acentos.
    Utiliza un array (map) que contiene los elementos con tilde y su equivalente sin tilde.
    Va a ser muy útil para convertir cadenas en cadenas sin tilde.
*/
function remover_acentos(str) {

    var map={
    'À':'A','Á':'A','Â':'A','Ã':'A','Ä':'A','Å':'A','Æ':'AE','Ç':'C','È':'E',
    'É':'E','Ê':'E','Ë':'E','Ì':'I','Í':'I','Î':'I','Ï':'I','Ð':'D','Ñ':'N','Ò'
    :'O','Ó':'O','Ô':'O','Õ':'O','Ö':'O','Ø':'O','Ù':'U','Ú':'U','Û':'U','Ü':
    'U','Ý':'Y','ß':'s','à':'a','á':'a','â':'a','ã':'a','ä':'a','å':'a','æ':
    'ae','ç':'c','è':'e','é':'e','ê':'e','ë':'e','ì':'i','í':'i','î':'i','ï':
    'i','ñ':'n','ò':'o','ó':'o','ô':'o','õ':'o','ö':'o','ø':'o','ù':'u','ú':'u'
    ,'û':'u','ü':'u','ý':'y','ÿ':'y','A':'A','a':'a','A':'A','a':'a','A':'A',
    'a':'a','C':'C','c':'c','C':'C','c':'c','C':'C','c':'c','C':'C','c':'c','D'
    :'D','d':'d','Đ':'D','d':'d','E':'E','e':'e','E':'E','e':'e','E':'E','e':
    'e','E':'E','e':'e','E':'E','e':'e','G':'G','g':'g','G':'G','g':'g','G':'G'
    ,'g':'g','G':'G','g':'g','H':'H','h':'h','H':'H','h':'h','I':'I','i':'i',
    'I':'I','i':'i','I':'I','i':'i','I':'I','i':'i','I':'I','i':'i','?':'IJ',
    '?':'ij','J':'J','j':'j','K':'K','k':'k','L':'L','l':'l','L':'L','l':'l',
    'L':'L','l':'l','?':'L','?':'l','L':'L','l':'l','N':'N','n':'n','N':'N','n'
    :'n','N':'N','n':'n','?':'n','O':'O','o':'o','O':'O','o':'o','O':'O','o':
    'o','Œ':'OE','œ':'oe','R':'R','r':'r','R':'R','r':'r','R':'R','r':'r','S':
    'S','s':'s','S':'S','s':'s','S':'S','s':'s','Š':'S','š':'s','T':'T','t':'t'
    ,'T':'T','t':'t','T':'T','t':'t','U':'U','u':'u','U':'U','u':'u','U':'U',
    'u':'u','U':'U','u':'u','U':'U','u':'u','U':'U','u':'u','W':'W','w':'w','Y'
    :'Y','y':'y','Ÿ':'Y','Z':'Z','z':'z','Z':'Z','z':'z','Ž':'Z','ž':'z','?':
    's','ƒ':'f','O':'O','o':'o','U':'U','u':'u','A':'A','a':'a','I':'I','i':'i'
    ,'O':'O','o':'o','U':'U','u':'u','U':'U','u':'u','U':'U','u':'u','U':'U',
    'u':'u','U':'U','u':'u','?':'A','?':'a','?':'AE','?':'ae','?':'O','?':'o'
    };
        var res=''; //Está variable almacenará el valor de str, pero sin acentos y tildes
        for (var i=0;i<str.length;i++)
        {
            c=str.charAt(i);res+=map[c]||c;
        }
    return res;
}

/*
    eliminaSignos(frase)

    Función que recibe una cadena de texto y elimina los signos.
*/
function eliminaSignos(frase){
        frase =frase.toLowerCase();
        frase.replace("+", " ");
        frase.replace(".", " ");
        frase.replace(/,/g, "");
        return frase;
}
/*
    eliminaStopWords(palabra)

    Función que recibe una cadena de texto y convierte todas las palabras en minúsculas.
    Una vez convertida, busca las coincidencias entre la cadena y las palabras stopwords
    que habíamos definido anteriormente. En el caso de encontrar alguna, la elimina de la
    cadena inicial insertada y devuelve finalmente la cadena sin esas stopwords.
*/
```

```
74      function eliminaStopWords(palabra){
75              //palabra = palabra.toLowerCase();
76              for(var i=0;i<stopwords.length;i++) palabra = palabra.replace(stopwords
                [i], "");
77              palabra.replace("+", "");
78              return palabra;
79      }
80
81
82
83      $(document).ready(function(){
84
85          // En primer lugar mantenemos escondido el div que contendrá la respuesta.
86          $('#respuesta').hide();
87
88          // Asignamos al botón con la id tuto-ajax-form
89          $('#buscador').submit(function(evento){
90              $('#respuesta').hide();
91              evento.preventDefault();
92
93              // Recogemos los datos escritos en el formulario y los almacenamos en
                la variable
94              // datos_formulario
95              var datos_formulario = $(this).serialize();
96
97              // Eliminamos las palabras stopwords de la búsqueda.
98              var datos_buscar = eliminaStopWords(datos_formulario);
99              var datos_buscar2 = datos_buscar.split(" ");
100
101             var busqueda = "";
102
103             // Generamos la búsqueda específica para ejecutar en Solr desde Ajax.
104             for(var i=0;i<datos_buscar2.length-1;i++){
105                 if(i==0) busqueda = "fichas%3a*" + datos_buscar2[i]+ "*";
106                 else busqueda = busqueda + "fichas%3a*" + datos_buscar2[i]+ "*+OR+";
107             }
108             busqueda = busqueda + "fichas%3a*" + datos_buscar2[datos_buscar2.length
                -1]+ "*";
109
110
111             // Creamos un objeto de tipo Ajax de jQuery en el que le indicamos la
                url
112             // que debe ejecutar para poder recibir los datos que necesitamos desde
113             // nuestro servidor de Apache Solr.
114             //
115             // En este caso, al ser una búsqueda textual, todos los elementos
                tienen un
116             // único campo llamado "fichas". Es aquí donde realizará la búsqueda
                de lo que
117             // escriba el usuario en el recuadro.
118             $.ajax({
119                 url:
                    'http://casa.jabenitez.com/solr/fichas/select?indent=on&version=2.2&
                    q='+busqueda+
                    '&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&hl.fl
                    =titulo',
120                 data: datos_formulario,
121                 type: 'GET',
122                 dataType: 'json',
123                 success: function(datos){
124                     var resultadosParaMostrar = "<p>Se encontraron "+datos.response
                        .numFound+" resultados</p>";
125
126                     for(var i=0;i<datos.response.docs.length;i++){
127                         resultadosParaMostrar += "<div id='obra-"+i+"'
                            class='nuevaObra-"+(i%2)+"'>";
```

```
128
129                                 resultadosParaMostrar += "<p class='fichas'>
                                    Información de la obra:<br/> " +   datos.response.docs[
                                    i].fichas+"</p>";
130
131
132                                 resultadosParaMostrar += "</div>";
133                                 //resultadosParaMostrar += "<p>" +
                                    datos.response.docs[i].creador+"</p>";
134                             }
135                             $('#resultados').html(resultadosParaMostrar).fadeIn('slow');
136                             $('#respuesta').html("URL ejecutada: "+
                                'http://casa.jabenitez.com/solr/fichas/select?indent=on&version=
                                2.2&q='+busqueda+
                                '&fq=&start=0&rows=10&fl=*%2Cscore&wt=json&explainOther=&hl=on&h
                                l.fl=titulo').fadeIn('slow');
137                         }
138                     });
139
140
141                 });
142             });
```

Ilustración 62 Código fuente buscadorTextual.js

- buscador.js : Fichero que contiene la funcionalidad del buscador semántico, capaz de detectar, gracias a todas las funciones que se encuentran en el mismo, la búsqueda de lo que necesita el usuario entendiendo perfectamente lo que él realmente desea.

```
80  /*
81      detectaTipo(palabra,claveBusqueda,claveSolr,separador)
82
83      Función que recibe una palabra, un array bidimensional que contiene claves
84      de búsqueda, un array con claves para apache solr y una cadena de sparación.
85
86      Esta función es uno de los pilares del buscador, gracias a ella, enviando
87      por parámetro las variables correspondientes, el buscador será capaz de
        analizar
88      qué datos necesita realmente obtener el usuario y qué datos son los que
        envía.
89
90      Si el usuario escribe una consulta como la siguiente:
91
92      Qué mapas se hicieron en el año 1639 por Galvarreta
93
94      Mediante esta función, analizará la frase y será capaz de llegar a la
        conclusión
95      de que el usuario necesita obtener la información de las obras de tipo
        "mapas"
96      que se hicieron en el año 1639 y que fueron hechas por el autor Galvarreta.
97
98      Todo ello es posible mediante diferentes llamadas a esta función con
        diversos parámetros
99      que están definidos en este código más adelante.
100
101 */
102
103 function detectaTipo(palabra,claveBusqueda,claveSolr,separador){
104         var primera = 0;
105         var abuscar = "";
106         palabra = palabra.toLowerCase();
107         for(var i=0;i<claveBusqueda.length;i++) {
108             for(var j=0;j<claveBusqueda[i].length;j++){
109                 var n = palabra.match(claveBusqueda[i][j]);
110                 if(n != null) {
111                     if(primera != 0) abuscar = abuscar + separador;
112                     abuscar = abuscar +  claveSolr[i];
113                     primera++;
114                     break;
115                 }
116             }
117         }
118         if(primera == 0) abuscar = claveSolr[claveSolr.length-1];
119
120         abuscar = abuscar + "+AND+";
121
122         //document.write(abuscar);
123         return abuscar;
124 }
125
126 /*
127     detectaTipo2(palabra,claveBusqueda,separador,campo,usaAux,rLike)
128
129     Función auxiliar que tiene 2 parámetros más que la anterior, pero que nos
        servirá
130     para detectar qué campos de búsqueda debemos utilizar en nuestra búsqueda
        semántica.
131
132 */
133
```

```
133
134     function detectaTipo2(palabra,claveBusqueda,separador,campo,usaAux,rLike){
135
136             var primera = 0;
137             var abuscar = "";
138             var auxiliar = "";
139             var cadenaCortada = "";
140
141             palabra = palabra.toLowerCase();
142             simbolo1 = "%3a";
143             simbolo2 = "";
144
145
146             if(rLike){
147                 simbolo1 = "%3a*";
148                 simbolo2 = "*";
149             }
150
151             //document.write(claveBusqueda[1]);
152             for(var i=0;i<claveBusqueda.length;i++) {
153
154                     auxiliar = claveBusqueda[i].replace(/[^a-zA-Z 0-9.]+/g,'');
155
156                     auxiliar = auxiliar.toLowerCase();
157
158
159                     /*
160                     auxiliar = eliminaStopWords(auxiliar);
161
162                     cadenaCortada = auxiliar.split(" ");
163
164                     for(var j=0;j<cadenaCortada.length;j++) {
165                         auxiliar = cadenaCortada[j].replace(/[^a-zA-Z 0-9.]+/g,'');
166                         auxiliar = auxiliar.toLowerCase();
167
168                         var n = palabra.match(auxiliar);
169                         if(n != null) {
170
171                             if(primera == 0) abuscar = "+AND+";
172                             if(primera != 0) abuscar = abuscar + separador;
173
174
175
176                             if(usaAux) abuscar = abuscar + campo + simbolo1 +
                                auxiliar + simbolo2;
177                             else abuscar = abuscar + campo + simbolo1 +
                                cadenaCortada[j] + simbolo2;
                                cadenaCortada[j] + simbolo2;
178
179                             console.log("\n\n Palabra = "+palabra+" \n Aux =
                                "+auxiliar);
180
181                             primera++;
182
183                         }
184                     }
185                     */
186
187
188                     var n = palabra.match(auxiliar);
189                     if(n != null) {
190
191                         //if(primera == 0) abuscar = "+AND+";
192                         if(primera != 0) abuscar = abuscar + separador;
193
194                         if(usaAux) abuscar = abuscar + campo + simbolo1 + auxiliar
                            + simbolo2;
195                         else abuscar = abuscar + campo + simbolo1 + claveBusqueda[i
                            ] + simbolo2;
```

```
                        ] + simbolo2;
196
197
198
199                     primera++;
200
201                 }
202
203             }
204
205             //document.write(abuscar);
206             return abuscar;
207     }
208     /*
209         function dameLoQueQuiero(datosQueNecesito,indiceNecesidad)
210         Función que recibe una palabra que indica qué campo de nuestra base de
            datos necesitamos
211         y su índice de necesidad correspondiente.
212
213         Esta función la utilizaremos a lo largo de nuestra aplicación para poder
            obtener y
214         almacenar en un array los distintos valores de un campo determinado de la
            misma.
215
216         Es decir, si necesitamos obtener todos los autores que están almacenados
            actualmente
217         en las distintas obras en Solr, gracias a esta función obtenemos un array
            con
218         todos los autores.
219     */
220
221     function dameLoQueQuiero(datosQueNecesito,indiceNecesidad)
222     {
223         // En función de la cadena que recibamos como parámetro en datosQueNecesito
224         // y en relación al indiceNecesidad que pidamos, realizaremos una consulta
225         // a nuestro servidor de Apache Solr, almacenando en un array todos los
            valores
226         // únicos de un campo concreto de cada una de las obras.
227
228         // Índices de campos:
229         // 1 - id
230         // 2 - titulo
231         // 3 - creador
232         // 4 - tematica
233         // 5 - notas
234         // 6 - materia
235         // 7 - referencias
236         // 8 - idioma
237         // 9 - publicador
238         // 10 - publicacion
239         var datosAgrupadosSinTilde2 = "";
240         var datosAgrupados2 = new Array(10000);
241         $.ajax({
242                 url:
                    'http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&ver
                    sion=2.2&q=*%3A*&fq=&start=0&rows=10000&fl='+datosQueNecesito+
                    '&wt=json&explainOther=&hl=on&hl.fl=titulo',
243                 type: 'GET',
244                 dataType: 'json',
245                 async: false,
246                 success: function(datos){
247                     //$('#resultados').html(JSON.stringify(datos, null,
                        4)).fadeIn('slow');
248
249                     var datosAgrupados = new Array(datos.response.docs.length);
250                     var datosAgrupadosSinTilde = "";
251
252
253
```

```
254                     switch(indiceNecesidad)
255                     {
256                         case 1:
257                             for(var i=0;i<datos.response.docs.length;i++){
258                                 datosAgrupados[i] = datos.response.docs[i].
                                    id;
259                             }
260                         break;
261
262                         case 2:
263                             for(var i=0;i<datos.response.docs.length;i++){
264                                 datosAgrupados[i] = datos.response.docs[i].
                                    titulo;
265                             }
266                         break;
267
268                         case 3:
269                             for(var i=0;i<datos.response.docs.length;i++){
270                                 datosAgrupados[i] = datos.response.docs[i].
                                    creador;
271                             }
272                         break;
273
274                         case 4:
275                             for(var i=0;i<datos.response.docs.length;i++){
276                                 datosAgrupados[i] = datos.response.docs[i].
                                    tematica;
277                             }
278                         break;
279
280                         case 5:
281                             for(var i=0;i<datos.response.docs.length;i++){
282                                 datosAgrupados[i] = datos.response.docs[i].
                                    notas;
283                             }
284                         break;
285
286                         case 6:
287                             for(var i=0;i<datos.response.docs.length;i++){
288                                 datosAgrupados[i] = datos.response.docs[i].
                                    materia;
289                             }
290                         break;
291
292                         case 7:
293                             for(var i=0;i<datos.response.docs.length;i++){
294                                 datosAgrupados[i] = datos.response.docs[i].
                                    referencias;
295                             }
296                         break;
297
298                         case 8:
299                             for(var i=0;i<datos.response.docs.length;i++){
300                                 datosAgrupados[i] = datos.response.docs[i].
                                    idioma;
301                             }
302                         break;
303
304                         case 9:
305                             for(var i=0;i<datos.response.docs.length;i++){
306                                 datosAgrupados[i] = datos.response.docs[i].
                                    publicador;
307                             }
308                         break;
309
310                         case 10:
311                             for(var i=0;i<datos.response.docs.length;i++){
312                                 datosAgrupados[i] = datos.response.docs[i].
                                    publicacion;
313                             }
```

```
                                    break;

                                    default:
                                }

                        datosAgrupados = eliminarDuplicados(datosAgrupados);
                        for(var i=0;i<datosAgrupados.length;i++){
                                datosAgrupadosSinTilde += "<p>" + datosAgrupados[i] +
                                "</p>";

                        }

                        datosAgrupadosSinTilde = remover_acentos(datosAgrupadosSinTilde
                        );
                        datosAgrupadosSinTilde = eliminaSignos(datosAgrupadosSinTilde);

                        datosAgrupadosSinTilde2 = datosAgrupadosSinTilde;
                        datosAgrupados2 = datosAgrupados;
                        //$('#respuesta').html(datosAgrupadosSinTilde).fadeIn('slow');
                        //return datosAgrupados;
                        //console.log(datosAgrupadosSinTilde);
                        //return datosAgrupadosSinTilde;
                    }
                });
                return datosAgrupados2;
                //return datosAgrupadosSinTilde2;

        }


        /*************************************************************/
        /*************************************************************/
        /*                  TIPOS DE PREGUNTAS                       */
        /*************************************************************/
        /*************************************************************/
        /*
            En función de si el usuario escribe "qué, quién, cómo, dónde o cuándo"
            nosotros deberemos proporcionarle la información contenida en uno u otro
            campo.

            Si la pregunta es "Qué autor realizó alguna obra en el año 1639"
            Nuestra aplicación mostrará entre uno de los campos el autor que realizó
            una obra en ese año.

        */
        var preguntasClave = Array("titulo%2Ccreador%2Ctematica",
        "creador%2Cpublicador%2C","notas","materia","fecha%2Cnotas");
        var datosAMostrarQue = Array("que","qué","qu\u00e9","quÃ©");
        var datosAMostrarQuien = Array("quien");
        var datosAMostrarComo = Array("como");
        var datosAMostrarDonde = Array("donde");
        var datosAMostrarCuando = Array("cuando");
        var datosAMostrarTotal = Array(datosAMostrarQue,datosAMostrarQuien,
        datosAMostrarComo,datosAMostrarDonde,datosAMostrarCuando);

        var sujetoClave = Array("mapa","plano","dibujo","autor","tecnica","epoca",
        "tematica","idioma","fecha","tematica","soporte","ciudad","pais","continente");

        var tecnicasClave1 = Array("tinta","tinta negra","tinta aguada","lapiz negro",
        "lápiz negro","rotulación","rotulaciones","rotulacion");
        var tecnicasClave2 = Array();
        var tecnicasClave3 = Array("a pluma");
```

```
370
371     /*************************************************************/
372     /*************************************************************/
373     /*                  TIPOS DE OBRAS                           */
374     /*************************************************************/
375     /*************************************************************/
376     /*
377         Si el usuario escribe la palabra "obras" los resultados a obtener serán
378         todas las obras, ya sean de tipo ilustración, manuscrito, mapa, libro u
            otros.
379
380         Pero si el usuario únicamente desea ver obras que son de tipo "mapas" la
            aplicación
381         gracias a estas variables, haciendo uso de una de las funciones anteriores,
382         lograremos obtener las obras que pertenecen al tipo Mapas.
383
384     */
385
386     var tiposClaveA = Array("tipo%3aIlust*","tipo%3aMapas","tipo%3aManuscritos",
        "tipo%3aLibros","tipo%3aOtros","tipo%3a*Rev*","tipo%3a*");
387     var tiposClaveA_Ilustraciones = Array("ilustraciones","ilustracion",
        "ilustración");
388     var tiposClaveA_Mapas = Array("mapa","mapas","plano","planos");
389     var tiposClaveA_Manuscritos = Array("manuscrito","manuscritos");
390     var tiposClaveA_Libros = Array("libro","libros");
391     var tiposClaveA_Otros = Array("otro","otros");
392     var tiposClaveA_Periodicos = Array("periódicos","periódico","periodico",
        "periodicos","revista","revistas");
393     var tiposClaveA_Obras = Array("obra","obras");
394     var tiposClaveASolr = new Array(tiposClaveA_Ilustraciones,tiposClaveA_Mapas,
        tiposClaveA_Manuscritos,tiposClaveA_Libros,tiposClaveA_Otros,
        tiposClaveA_Periodicos,tiposClaveA_Obras);
395
396     /*************************************************************/
397     /*************************************************************/
398     /*                          IDIOMAS                          */
399     /*************************************************************/
400     /*************************************************************/
401     /*
402         En función del idioma que inserte el usuario en el cuadro de búsqueda
403         nuestro buscador será capaz de detectar el idioma y buscar las obras
404         relacionadas con dicho lenguaje.
405
406     */
407     var idiomasClaveA = Array("idioma%3aspa","idioma%3afra","idioma%3aeng",
        "idioma%3alat","idioma%3ager","idioma%3aita","idioma%3aport","idioma%3adut",
        "idioma%3acat","idioma%3a*");
408     var idiomasClaveA_1 = Array("español","castellano","espanol");
409     var idiomasClaveA_2 = Array("frances","francés");
410     var idiomasClaveA_3 = Array("ingles","inglés");
411     var idiomasClaveA_4 = Array("latin","latín");
412     var idiomasClaveA_5 = Array("alemán","aleman");
413     var idiomasClaveA_6 = Array("italiano");
414     var idiomasClaveA_7 = Array("portugues","portugués");
415     var idiomasClaveA_8 = Array("holandes","holandés");
416     var idiomasClaveA_9 = Array("catalan","catalán");
417
418     var idiomasClaveASolr = new Array(idiomasClaveA_1,idiomasClaveA_2,
        idiomasClaveA_3,idiomasClaveA_4,idiomasClaveA_5,idiomasClaveA_6,idiomasClaveA_7
        ,idiomasClaveA_8,idiomasClaveA_9);
419
```

```
419
420
421
422     /*
423         Haciendo uso de la anterior función explicada, almacenamos en cuatro
            variables los autores, las temáticas, los lugares y las fechas existentes
            en nuestra base  de datos actual.
424     */
425     var autoresExistentes = dameLoQueQuiero("creador",3);
426     var tematicasExistentes = dameLoQueQuiero("tematica",4);
427     var lugaresExistentes = dameLoQueQuiero("materia",6);
428     var fechasExistentes = dameLoQueQuiero("publicacion",10);
429
430
431
432     /* Funciones de ayuda para la codificación */
433     function encode_utf8( s )
434     {
435       return unescape( encodeURIComponent( s ) );
436     }
437
438     function decode_utf8( s )
439     {
440       return decodeURIComponent( escape( s ) );
441     }
442
443     $(document).ready(function(){
444
445
446
447         $('#respuesta').hide();
448         $('#buscador').submit(function(evento){
449             $('#respuesta').hide();
450             evento.preventDefault();
451
452             var datos_formulario = $(this).serialize();
453                 datos_formulario = encode_utf8(datos_formulario);
454             var datos_buscar = eliminaStopWords(datos_formulario);
455             var busqueda  = detectaTipo(datos_buscar,tiposClaveASolr,tiposClaveA,
                "+OR+",true);
456
457                 busqueda  += detectaTipo(datos_buscar,idiomasClaveASolr,
                    idiomasClaveA,"+OR+",true);
458
459                 // Comprobamos qué datos inserta, si desea buscar indicando una
                    temática
460                 // una fecha, un autor.... etc, debemos ir uno por uno.
461                 // Índices de campos:
462                 // 1 - id
463                 // 2 - titulo
464                 // 3 - creador
465                 // 4 - tematica
466                 // 5 - notas
467                 // 6 - materia
468                 // 7 - referencias
469                 // 8 - idioma
470                 // 9 - publicador
471                 // 10 - publicacion (es el campo date o fecha)
472
473                 busqueda += detectaTipo2(datos_buscar,autoresExistentes,"+OR+",
                    "creador",false,false);
474                 if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                    busqueda = busqueda + "+OR+";
475
476                 busqueda += detectaTipo2(datos_buscar,fechasExistentes,"+OR+",
                    "publicacion",true,true);
477                 if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                    busqueda = busqueda + "+OR+";
```

```
478
479             busqueda += detectaTipo2(datos_buscar,fechasExistentes,"+OR+",
                "titulo",true,true);
480             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
481
482             busqueda += detectaTipo2(datos_buscar,tematicasExistentes,"+OR+",
                "tematica",true,true);
483             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
484
485             busqueda += detectaTipo2(datos_buscar,tematicasExistentes,"+OR+",
                "tematica",false,true);
486             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
487
488             busqueda += detectaTipo2(datos_buscar,lugaresExistentes,"+OR+",
                "titulo",true,true);
489             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
490
491             busqueda += detectaTipo2(datos_buscar,lugaresExistentes,"+OR+",
                "materia",true,true);
492             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
493
494             busqueda += detectaTipo2(datos_buscar,lugaresExistentes,"+OR+",
                "materia",false,true);
495             if((busqueda != null) && (busqueda[busqueda.length-1] != "+"))
                busqueda = busqueda + "+OR+";
496
497             if(busqueda[busqueda.length-1] == '+') busqueda = busqueda +
                "tipo%3aM";
498
499
500             console.log(busqueda);
501
502
503         //document.write(datos_buscar);
504         var datos_mostrar = eliminaStopWords(datos_formulario);
505             datos_mostrar = detectaTipo(datos_mostrar,datosAMostrarTotal,
                preguntasClave,",");
506             //document.write(datos_mostrar);
507         //document.write(remover_acentos(datos_formulario));
508
509
510
509
510
511
512
513         $.ajax({
514             url:
                'http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on&ver
                sion=2.2&q='+busqueda+'&fq=&start=0&rows=10&fl='+datos_mostrar+
                '%2Cscore%2Cid&wt=json&explainOther=&hl=on&hl.fl=titulo',
515             data: "",
516             type: 'POST',
517             dataType: 'json',
518             success: function(datos){
519
520
521                 var idsAgrupadas = "<p>Se encontraron "+datos.response.numFound
                    +" resultados</p>";
522
```

```
523                     for(var i=0;i<datos.response.docs.length;i++){
524                         idsAgrupadas += "<div id='obra-"+i+"' class='nuevaObra-"+(i
                            %2)+"'>";
525                         if(datos_formulario.match("autor")){
526                             idsAgrupadas += "<p>" + datos.response.docs[i].creador+
                                "</p>";
527                         }else{
528                             idsAgrupadas += "<p class='id'> Obra con id: " +
                                datos.response.docs[i].id+"</p>";
529                             idsAgrupadas += "<p class='id'> Titulo: " + datos.
                                response.docs[i].titulo+"</p>";
530                             idsAgrupadas += "<p class='id'> Autor: " +  datos.
                                response.docs[i].creador+"</p>";
531                         }
532                         idsAgrupadas += "</div>";
533
534                     }
535                     $('#resultados').html(idsAgrupadas).fadeIn('slow');
536                     $('#respuesta').html("URL ejecutada: "+
                        'http://casa.jabenitez.com/solr/fichasFacetadas/select?indent=on
                        &version=2.2&q='+busqueda+'&fq=&start=0&rows=10&fl='+
                        datos_mostrar+
                        '%2Cscore&wt=json&explainOther=&hl=on&hl.fl=titulo').fadeIn(
                        'slow');
537                 }
538             });
539
540
541         });
542     });
```

Ilustración 63 Código fuente buscador.js

- index.php

```
<!DOCTYPE html>
<html lang="es">
<?php

    $modo = $_GET["modo"];
    if($modo == NULL) $modo = 0;
    if($modo == '') $modo = 0;

?>
<?php include_once("include/head.php");?>
<?php

    $textoBuscador = array("SEMÁNTICO","TEXTUAL");

?>

<body>

<div class="item100">
    <nav id="selectorBuscador">
        <ul>
            <li><a href="index.php?modo=0" <?php if($modo==0) echo "class='activo'"
            ;?>>BUSCADOR SEMÁNTICO</a></li>
            <li><a href="index.php?modo=1" <?php if($modo==1) echo "class='activo'"
            ;?>>BUSCADOR TEXTUAL</a></li>
        </ul>
    </nav>
</div>

<div class="item100">

    <section id="contenedor-formulario">
    <h1>TIPO DE BUSCADOR ACTIVADO: BUSCADOR <?php echo $textoBuscador[$modo];?></h1>
        <form id="buscador" accept-charset="UTF-8">
            <input type="text" id="textoBusqueda" name="textoBusqueda" placeholder=
            "Búsqueda" />

            <button type="submit" id="submit">Enviar</button>
        </form>
        <section id="respuesta"></section>
    </section>

</div>
<div class="item100">

    <section id="resultados">

    </section>
</div>

</body>

</html>

```

Ilustración 64 Código fuente index.php