Anexos

Apéndice A

Análisis y diseño de la aproximación

Este anexo expone el análisis que se realizó de la aproximación obtenida; se realizo antes de realizar la implementación para comprender la naturaleza del problema y obtener una mejor solución. También se presenta el diseño que finalmente se ha realizado en la implementación.

A lo largo del anexo se mostrará la especificación de requisitos, el diagramas de clases, el diagramas de casos de uso y el diagramas de trazas de eventos, así como se proporcionará una visión general del funcionamiento del sistema.

A.1. Especificación de Requisitos Software

Durante esta sección se presenta la Especificación de Requisitos Software (ERS) según el estándar IEEE 830 obtenida del análisis de la aproximación, esto supone entender la problemática subyacente en cada uno de los aspectos de la aproximación a fin de poder realizar una implementación usable y fiel a la arquitectura propuesta.

A.1.1. Introducción

El sistema desarrollado, aun dentro del ámbito de la investigación, persigue que los usuarios encuentren la manera de realizar búsquedas semánticas sin disponer de más conocimientos que los actualmente disponen.

El actual usuario de servicios de búsqueda está acostumbrado a tratar con motores de búsqueda que funcionan con palabras clave, sin embargo, por el propio paradigma de la Web Semántica y la obtención de los datos estructurados que necesitamos, la naturaleza de nuestras búsquedas requieren de unos conocimientos teóricos y técnicos que no se esperan en los usuarios de este sistema, por lo cual es importante que estos no lleguen a tener contacto con los aspectos más complejos de estas búsquedas y puedan utilizar la solución presentada fácil e intuitivamente.

Definiciones, acrónimos y abreviaturas

Se entiende que algunas de las palabras utilizadas durante todo este documento pueden estar sujetas a confusión, por lo cual a continuación se detalla la definición de cada uno de las partes que componen este PFC.

- El primer punto a destacar del proyecto sería la **aproximación**. Por aproximación se entiende el sistema diseñado para poder realizar las búsquedas semánticas tanto por palabra clave como por refinado. También en ocasiones me refiero a la aproximación como la **solución**.
- Siguiendo a este viene la **implementación de la aproximación** o de la solución. Esto se refiere al código, APIs y herramientas utilizadas para dar forma a la arquitectura que propone la aproximación.
- Por ultimo, tenemos la **utilidad** o **interfaz**, con la que se podrá dar uso a la aproximación. Es la visión gráfica o textual del sistema para acceder a sus funcionalidades.

El presente apéndice no tiene en cuenta la utilidad, solo atiende a la aproximación y su implementación.

Por último, anotar que este PFC surgió con la idea de proporcionar al proyecto de Daniel Martínez Millán una interfaz para poder realizar búsquedas de datos estructurados, tal proyecto pone interfaz a nuestra solución, sin embargo, esta no hace uso de todas las características del sistema, así que fue necesario diseñar una alternativa para poder mostrarlas.

A.1.2. Descripción general

Funciones del sistema

La aproximación propuesta busca realizar búsquedas con contenido semántico. Se proponen dos tipos de búsquedas, por palabras clave y por refinado.

- El primero tipo de búsqueda se basa en encontrar las **palabras clave** introducidas por el usuario en los campos elegidos. La elección de estos campos, que marcarán la amplitud de los resultados e incluso el rendimiento de las consultas, deberá ser realizada por un administrador experto.
- La búsqueda por **refinado** supone obtener, a partir de un resultado previo, datos estructurados que estén relacionados de alguna manera con el resultado elegido. Estas relaciones también deben ser elegidas por un administrador.

Características de los usuarios

Los usuarios de este sistema deberán disponer de los conocimientos y experiencia básicos para el uso de motores de búsqueda convencionales por palabras clave.

Características del administrador

El administrador de este sistema será el encargado de definir la Ontología Dominio, y deberá ser solvente en los siguientes aspectos:

- **Conocimientos básicos sobre la Web Semántica**. Las herramientas actuales de edición de ontologías requieren de un cierto grado de conocimiento de las tecnologías asociadas, normalmente RDF y OWL.
- Opcionalmente soltura con alguna **solución para la edición de ontologías**. Si bien es cierto que se pueden definir ontologías utilizando simplemente un editor de texto, existen diversas APIs y herramientas semánticas que permiten su edición y comprobación de manera mucho más rápida y fiable.
- **Entendimiento del conjunto de datos elegido**. Para poder definir la Ontología Dominio de manera que las posteriores búsquedas por refinado que hacen uso de ella sean útiles y aporten resultados interesantes, es necesario conocer el conjunto de datos sobre los que se va a realizar las consultas, de manera que se puedan conocer las clases y relaciones que conviene o no incluir en la Ontología Dominio.

Restricciones

La Ontología Dominio utilizada en el prototipo se ha definido de acuerdo a los requisitos del proyecto Alfa III GAVIOTA, teniendo como repositorio de datos a la DBPedia. Por este motivo, el uso del prototipo, actualmente, está limitado a este repositorio (pudiendo dar distintas vistas sobre el mismo modificando las categorías participantes en la Ontología Dominio).

Por otro lado, la solución está implementada sobre el lenguaje independiente de la plataforma Java y no dispone de procesos que requieran de realizar tantas operaciones con el procesador como para que el rendimiento de una máquina sin hardware muy potente se vea afectado. Actualmente se están realizando pruebas de rendimiento exhaustivas para verificar el buen comportamiento de nuestro sistema con vistas a presentar un nuevo artículo.

El sistema implementado trabaja sobre el punto de acceso público de la DBPedia. El uso de estos puntos públicos puede estar relacionado con tiempos de respuesta elevados, que dependen a veces incluso del momento del día en que se realice la

consulta. Este hecho puede llegar a reducir la usabilidad del sistema, por lo que es necesario comprender que la solución depende de un repositorio de datos público y está sujeto a su disponibilidad y estado.

Suposiciones y dependencias

Para este PFC reconocemos el repositorio elegido como la DBPedia. Sin embargo, el proyecto de la DBPedia es un proyecto todavía activo, y está sujeto a cambios (incluso los derivados de posibles cambios en las recomendaciones del W3C). Este proyecto no puede suponer que el estado de los datos y las ontologías usadas por la DBPedia se mantengan estables a lo largo del tiempo, por lo que se depende que los datos presentes en el repositorio de la DBPedia estén organizados en un futuro como lo están ahora.

En el ámbito temporal de este proyecto se ha tratado con las versiones 3.6, 3.7 y 3.8 de la DBpedia. Cada nueva versión ha supuesto la realización de cambios en la Ontología Dominio.

A.1.3. Requerimientos específicos

Requerimientos funcionales

RF-1 El sistema deberá disponer de, al menos, dos tipos de búsquedas de contenido semántico a la DBPedia, búsquedas por palabras clave y búsquedas de refinado. La entrada y salida para realizar estas búsquedas deberán ser lo más sencillas posibles y abstraer de las tecnologías semántico-sintácticas usadas.

RF-2 Las búsquedas por palabras clave deberán venir ordenadas. El orden establecido será el número de apariciones de la palabra en relación a lo extenso del texto.

RF-3 Las búsquedas deberán adaptarse a la Ontología Dominio, ontología que marca las clases y relaciones o utilizar durante las consultas.

Interfaces externas

IE-1 El sistema dispondrá de una interfaz con la que comunicarse con otros equipos.

IE-2 La interfaz contendrá, al menos, dos tipos de métodos principales que podrán ser invocados, la búsqueda por palabras clave y la búsqueda por refinado, además de otros métodos auxiliares necesarios para la interfaz gráfica.

Requisitos de rendimiento

Por ser esta una implementación realizada bajo el contexto de la investigación no se plantea la carga que deberá soportar el sistema, sin embargo, al estar especificada la comunicación con otros equipos, deberá ser posible realizar dicha comunicación de manera simultánea sin que el sistema deje de responder a ninguna de ellas y optimizando siempre que sea posible la respuesta del sistema.

Restricciones de diseño

La única restricción es la suposición que el repositorio de datos no está bajo nuestro control en ningún momento. El sistema accederá a un punto de acceso público de la DBPedia para realizar consultas y observar los datos pero no podrá manipularlos.

Atributos del sistema

La fiabilidad esperada del sistema es alta, el estudio de los datos demuestra que es posible realizar búsquedas por palabras clave guiadas y de refinado. Aún y esperando que los resultados sean favorables es de esperar unos tiempos de respuesta del sistema moderadamente elevados, ya que su acceso se realizará a punto de acceso público.

No se requieren medidas de seguridad para el uso del sistema, pues este será anónimo y sin guardado de sesión de ningún tipo.

La mantenimiento que puede ser requerido es desconocido, dependerá de cambios en las recomendaciones del W3C, los repositorios de datos estructurados y el futuro de la Web Semántica.

A.2. Diagrama de clases

En la figura A.1 puede verse el diagrama de clases en alto nivel del sistema. En el diagrama, el módulo interfaz se refiere a la interfaz del servicio web, no a la interfaz gráfica web.

A.3. Casos de uso

En la figura A.2 se exponen los distintos casos de uso que se pueden dar en nuestro sistema.

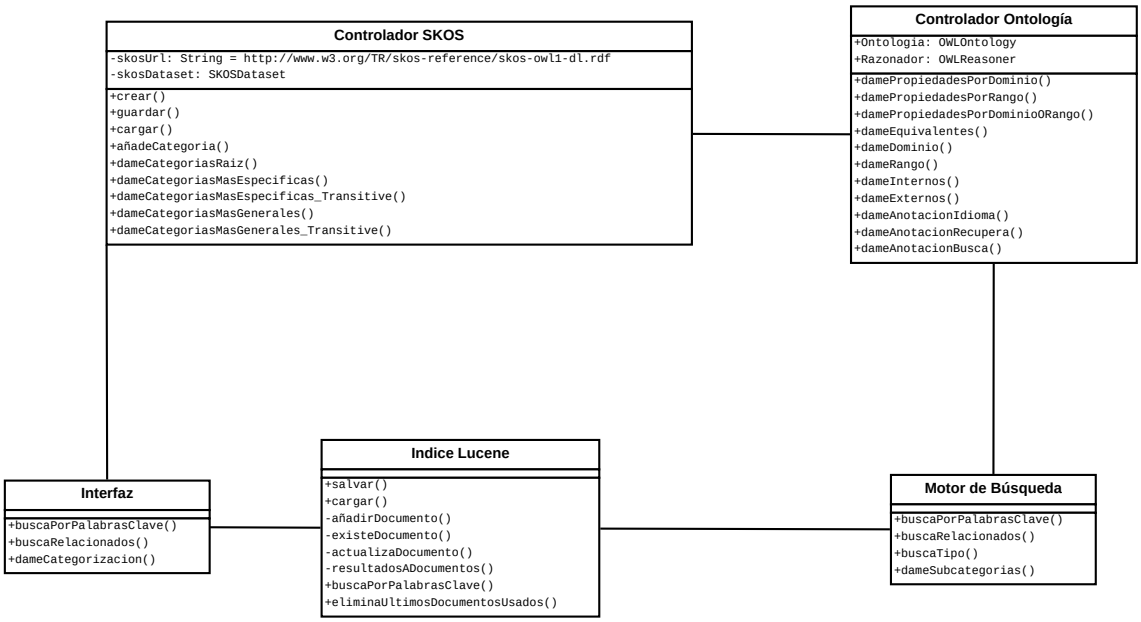

Figura A.1: Diagrama de clases en alto nivel del sistema

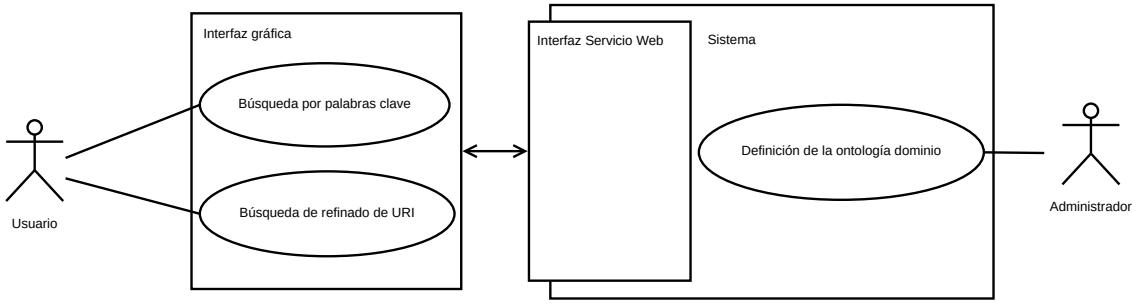

Figura A.2: Diagrama de casos de uso del sistema

Búsqueda por palabras clave En este caso de uso, el usuario, mediante la interfaz gráfica utilizada, definirá una categoría raíz para la búsqueda y las palabras clave que desea buscar en los artículos. La interfaz mandará estos datos al sistema y este devolverá la lista con los resultados. Estos resultados se mostrarán en una tabla con la posibilidad de escoger uno de ellos para pedir más información.

Búsqueda de refinado de URI El usuario elegirá uno de entre los recursos de una búsqueda anterior. La interfaz mandará la URI de este recurso al sistema, el cual buscará información relacionada con él. La interfaz recibirá los datos y los mostrará de manera similar a los de la búsqueda por palabras clave.

Definición de Ontología Dominio El administrados del sistema deberá describir la ontología que definirá el dominio de búsqueda.

A.4. Trazas de eventos

A continuación se exponen las trazas de eventos para los dos posibles casos de uso disponibles para el usuario, en el primer caso de uso, cuya traza de eventos está presente en la figura A.3, se realiza una búsqueda por palabras clave. La segunda, caso de uso de búsqueda por refinado, se encuentra en la figura A.4.

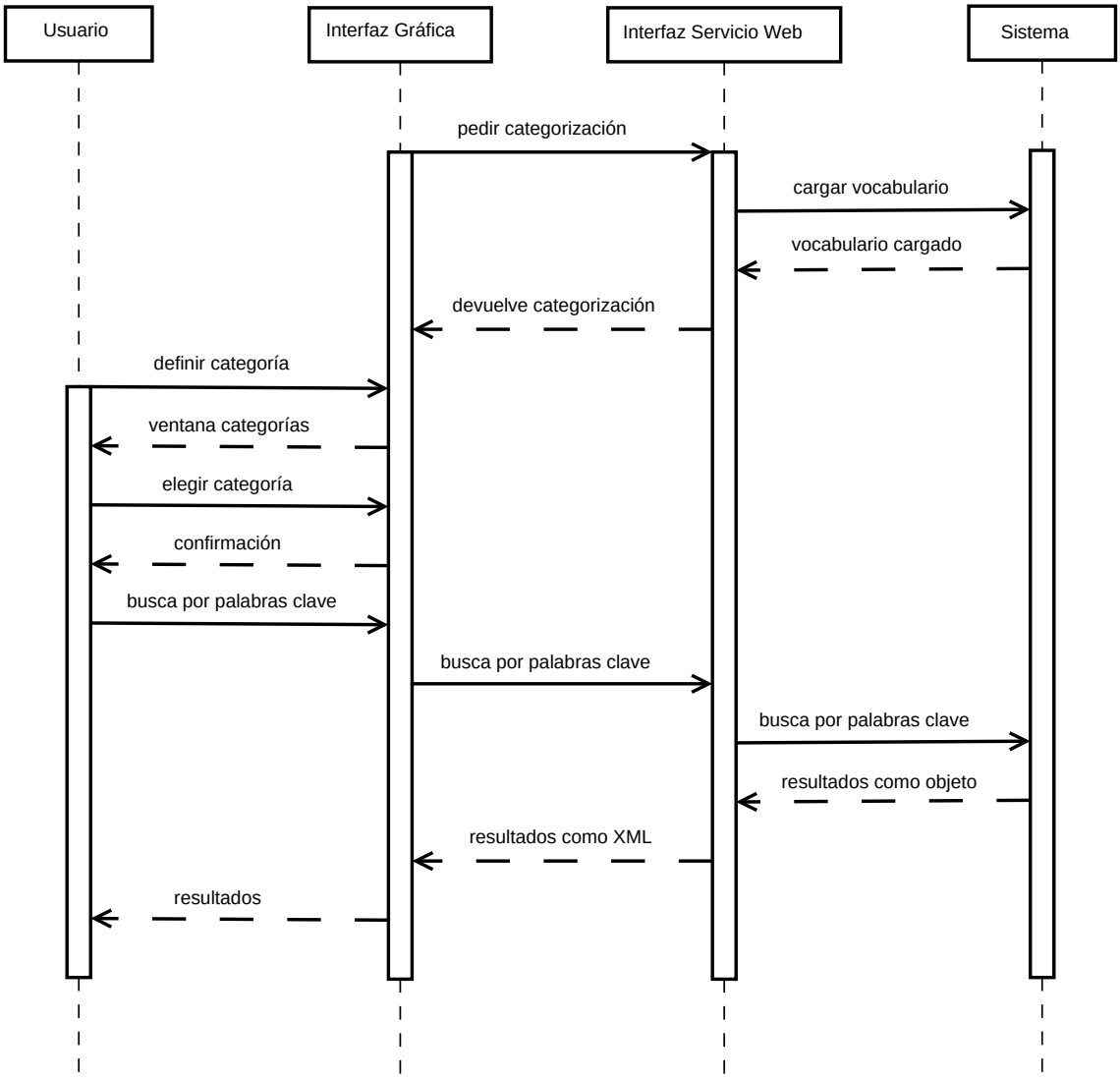

Figura A.3: Traza de eventos del caso de uso de búsqueda por palabras clave

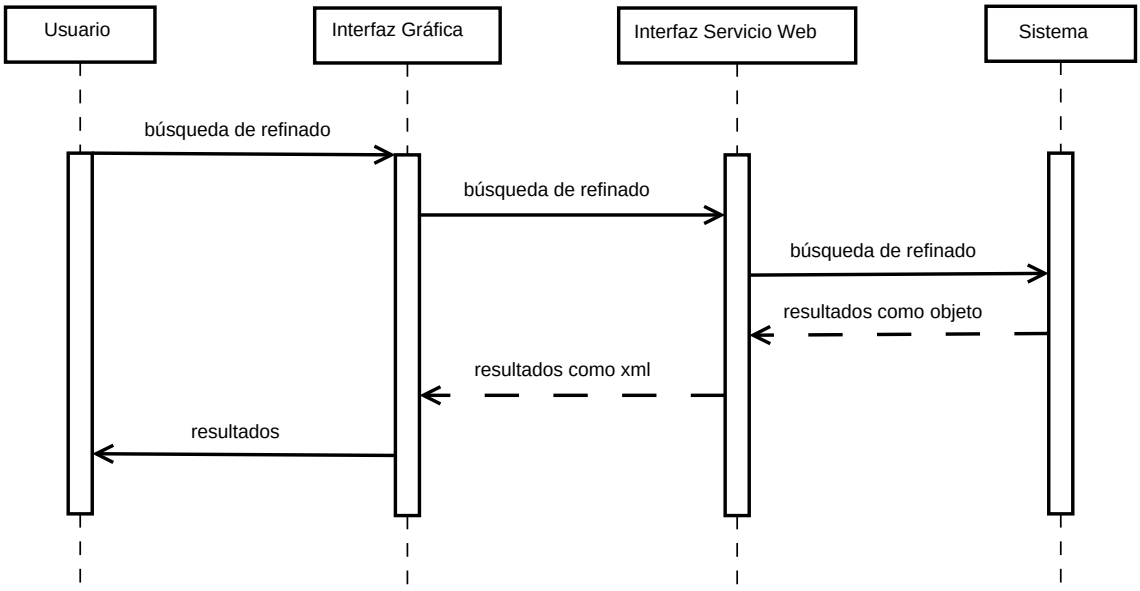

Figura A.4: Traza de eventos del caso de uso de búsqueda de refinado de URI

Apéndice B

Vocabularios, lenguajes y herramientas utilizadas

En este apéndice se describirán detalles específicos sobre OWL y los vocabularios secundarios que utiliza la Ontología Dominio para definir el dominio de búsqueda. También se describirán los lenguajes utilizados y sus principales funcionalidades, en el caso de SPARQL se profundizará un poco más para que puedan comprenderse las consultas que son necesarias para recuperar los datos del repositorio. Finalmente se listarán las herramientas de las que se ha hecho uso para la realización del proyecto.

B.1. Ontologías y vocabularios

B.1.1. OWL

Durante la sección 2.1.2 sobre OWL se mencionan varias de las características más utilizadas del lenguaje sin entrar en mucho detalle conforme a su construcción y forma de uso. Durante el transcurso de este apartado se explicará mediante un ejemplo la construcción de una ontología simple utilizando OWL.

Para exponer el ejemplo se hará uso tanto de las sintaxis RDF/XML [1] y Turtle [2], siendo Turtle una representación más cercana a las tripletas RDF.

Como namespace se hará uso de *rdf*, *rdfs*, *owl*, *foaf*, *skos* y *dcmi*. El namespace de cada documento se define en su cabecera. Siempre que nos encontremos en Turtle con el carácter ":", o en RDF/XML el carácter "#", sin prefijos y seguidos de un nombre, significará que esa entidad se encuentra dentro de nuestra ontología, es decir, que la estamos definiendo nosotros.

[1] RDF/XML, http://www.w3.org/TR/REC-rdf-syntax/
[2] Turtle, http://www.w3.org/TeamSubmission/turtle/

En primer lugar, se pondrán sobre ejemplo los tres elementos básicos de una ontología: las clases, propiedades e instancias (o individuos). En OWL/XML se define una clase de la siguiente forma:

```
<owl:Class rdf:about="#Persona" />
```

Y en Turtle:

```
:Persona rdf:type owl:Class .
```

Con esta cláusula lo único que se indica es que existe una clase con identificada por una URI con nombre Persona y la base del documento donde está definida. Si el documento donde se estuviese esta línea se encontrase alojado en `http://eina.unizar.es/ontology`, la URI de nuestra clase sería `http://eina.unizar.es/ontology#Persona`.

Siguiendo con los elementos básicos, vamos a definir ahora una instancia de la clase Persona. Utilizando la sintaxis RDF/XML:

```
<owl:Thing rdf:about="#Marta">
  <rdf:type rdf:resource="#Persona" />
</owl:Thing>
```

Representado utilizando Turtle como:

```
:Marta rdf:type :Persona .
```

Terminando con los elementos básicos definiremos un par de propiedades. Al estar usando OWL tenemos la posibilidad de especificar entre los dos tipos de propiedades básicas, *ObjectProperty* y *DataProperty*, que relacionan, respectivamente, dos instancias de clases, y una instancia de clase con un valor. En concreto, el dominio de las *ObjectProperty* deben tener como `rdf:type` a `owl:Class`.

```
<owl:ObjectProperty rdf:about="#conoceA">
  <rdfs:domain rdf:resourse="#Persona" />
  <rdfs:range rdf:resource="#Persona" />
</owl:ObjectProperty>

<owl:DataProperty rdf:about="#fechaNacimiento">
  <rdfs:domain rdf:resourse="#Persona" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime" />
</owl:DataProperty>
```

Con estas propiedades se hace uso de las reglas de dominio y rango de RDFS, que definen el tipo de instancias permitidas para usarse con estas propiedades. La definición del rango en la segunda propiedad, al tratarse de una *DataProperty*, tiene que realizarse utilizando algunos de los que define OWL para esta propiedad; estos son, o bien una cadena de caracteres o un tipo de de datos entre los definidos en XML Schema [3]. En el caso de la primera propiedad, de tipo *ObjectProperty*, la única

[3] XML Schema, http://www.w3.org/TR/xmlschema-2/

restricción del dominio y el rango es ser instancia de `owl:Class`, lo que cumple la clase Persona.

En este último ejemplo se pueden observar la relevancia del uso de jerarquías en OWL, pues hay que tener en cuenta las descripciones de las entidades a las que se da uso. Por lo cual no podríamos, por ejemplo, utilizar un tipo de dato XMLS como rango en una *ObjectProperty*.

El anterior ejemplo, expresado en Turtle sería:

```
:conoceA          rdf:type       owl:ObjectProperty .
:conoceA          rdfs:domain    :Persona .
:conoceA          rdfs:range     :Persona .

:fechaNacimiento  rdf:type       owl:DataProperty .
:fechaNacimiento  rdfs:domain    :Persona .
:fechaNacimiento  rdfs:range     <http://www.w3.org/2001/XMLSchema#dateTime> .
```

A continuación, para explicar la jerarquía de clases, se podría definir una clase llama Mujer. Para cualquier lector está claro que existe una relación jerárquica entre Mujer y Persona. Persona es más general que Mujer, siendo cualquier individuo que sea Mujer, también Persona. En OWL, esto puede realizarse utilizando las subclases, en RDF/XML:

```
<owl:Class rdf:about="Mujer">
  <rdfs:subClassOf rdf:resource="Persona"/>
</owl:Class>
```

Y en Turtle como:

```
:Mujer rdfs:subClassOf :Persona .
:Hombre rdfs:subClassOf :Persona .
```

La presencia de este axioma en la ontología permite a un razonador inferir que cualquier individuo que sea instancia de Mujer, también debe serlo de Persona, como refleja el siguiente ejemplo:

```
:Marta rdf:type :Mujer .
```

Este mismo comportamiento puede modelarse con las propiedades. Por ejemplo, la siguiente tripleta en Turtle nos definirá subpropiedad de `conoceA`.

```
:amigoDe rdfs:subPropertyOf :conoceA .
```

Especificando que todo par de individuos relacionados con la propiedad `amigoDe`, también lo estarán con la propiedad `conoceA`. Todas las propiedades definidas como subpropiedades heredan los campos no especificados en su definición de su superpropiedad. Por lo que el dominio y rango de la propiedad `amigoDe` serán instancias de la clase `Persona`. Este comportamiento se da de igual manera con las clases y subclases.

Con OWL, también es posible definir clases equivalentes, siguiendo nuestro ejemplo, en Turtle:

```
:Persona owl:equivalentClass :Humano .
```

De este un razonador podrá inferir que cualquier individuo que sea instancia de Persona también lo es de Humano. Si dos clases que son consideradas equivalente, contienen exactamente los mismos individuos.

Como hemos visto, OWL permite que un individuo sea instancia de varias clases. Sin embargo, la pertenencia a una clase es a veces indicativo de exclusión en otra, para ello OWL incluye la disyunción de clases, en RDF/XML esto se especifica de la siguiente manera:

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
</owl:AllDisjointClasses>
```

Como último punto dentro de OWL se destacará la posibilidad de incluir características lógicas a las *ObjectProperties*. Esto se realiza utilizando un tipo de propiedades especiales, llamadas *SymmetricProperty* y *TransitiveProperty,* subclases de esta primera. En el primer caso, cuando se utiliza *SymmetricProperty,* se define una relación P entre el par (A,B) que es simétrica, pues la misma relación P existe entre el par (B,A). Anteriormente habíamos definido la propiedad `amigoDe`, de tipo *ObjectProperty*, podríamos sin embargo definirla como *SymmetricProperty*:

```
<owl:SymmetricProperty rdf:about="#amigoDe">
  <rdfs:domain rdf:resource="#Persona" />
  <rdfs:range  rdf:resource="#Persona" />
</owl:SymmetricProperty>
```

Utilizando esta propiedad podríamos describir:

```
:Marta :amigoDe :Pablo .
```

Esto implicaría que Marta tiene como amigo a Pablo, y por la característica de esta propiedad, los razonadores podrán inferir que Pablo tiene como amiga a Marta.

La otra subclase de ObjectProperty con características lógicas es TransitiveProperty. Utilizando esta propiedad para definir una relación P entre el par (A,B) y otra relación P entre el par (A,B), se puede inferir que existe la misma relación P entre el par (a,c). Esta propiedad es importante en el sistema desarrollado ya que SKOS, la ontología que utiliza la DBPedia para categorizar sus artículos, hace uso de las propiedades *skos:broader* y *skos:narrower* de pertenecientes a esta. Tanto *broader* como *narrower* son subpropiedades de *broaderTransitive* y *narrowerTransitive*, ambas subpropiedades a su vez de *SymmetricProperty*, y ofrecen a los razonadores la posibilidad de inferir nuevos hechos.

OWL dispone de muchas funcionalidades para facilitar la representación del conocimiento dentro de estas ontologías, si el lector está interesado pueden encontrarse

múltiples documentos referentes a OWL en la web del W3C [4].

Al contener OWL tanta semántica, su visualización es un tema abierto en investigación actualmente. Existe un proyecto del mismo director de este proyecto, Carlos Bobed, del alumno Jorge Bobed, que trabaja en esa línea y está ofreciendo buenos resultados. El proyecto se enmarca como un plugin de visualización de ontologías para la Protégé y utiliza de razonadores para realizar inferencias sobre estas y visualizar las clases, individuos y sus relaciones, tal y como puede observarse en la Figura B.1.

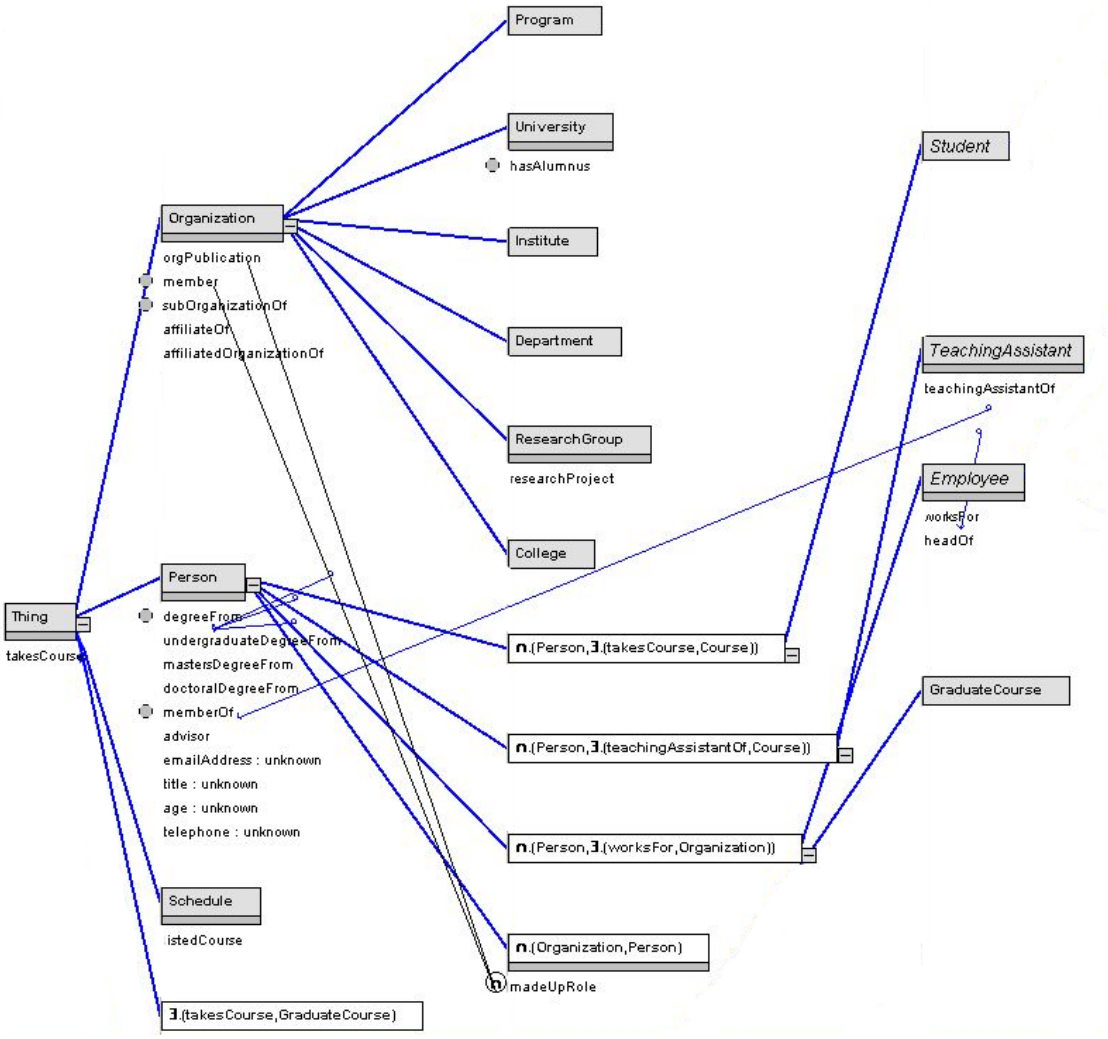

Figura B.1: Visualización de una ontología

[4]Documentación OWL, http://www.w3.org/TR/owl2-overview/#Documentation_Roadmap

B.1.2. SKOS

SKOS (*Simple Knowledge Organization System*) [20] es un vocabulario para RDF que nos permite representar sistemas de organización del conocimiento. Aunque SKOS adopta un modelo formal y objetivos distintos a OWL, puede usarse en conjunción con este.

Con RDF y OWL se puede representar conocimiento complejo: podemos definir infinidad de clases, individuales y sus propiedades y tener representado todo ese conocimiento en OWL para su tratamiento por máquinas. A pesar de la capacidad organizativa de OWL y RDF, tienen limitaciones a la hora de modelar y categorizar: mientras RDF y OWL modelan objetos o conceptos de un dominio, SKOS nos permite definir categorías para cada uno de ellos.

SKOS es un modelo cerrado que hace uso de RDF para definir las clases y propiedades necesarias para categorizar el conocimiento. Con SKOS podremos etiquetar cada recurso para saber a que categoría de conocimiento pertenece. De esta manera se puede determinar con exactitud en que rama de conocimiento nos hallamos, ya que este se encuentra correctamente organizado y etiquetado, pudiendo también utilizar razonadores para inferir nuevos hechos y clasificaciones de nuestros recursos. Entre los elementos que introduce para ello cabe destacar los siguientes:

Concepto El elemento fundamental del vocabulario SKOS es la clase concepto [5]. Un concepto se define como una unidad de pensamiento y es una entidad abstracta independiente del término usado para definirla. Con este elemento se pretende que las ontologías que dan uso a SKOS categoricen los recursos y los agrupen según estos conceptos.

Etiquetas El primer paso para categorizar el conocimiento es aplicar etiquetas a los conceptos. Para poder darles nombre, se les relaciona con un valor en lenguaje natural utilizando alguna de las propiedades de etiquetado que SKOS ofrece.

Relaciones Los conceptos comúnmente presentan relaciones semánticas entre sí, estas relaciones pueden ser jerárquicas o no y son, como las etiquetas, propiedades RDF. En este proyecto, por su aparición en los datos presentes en la DBPedia, se han utilizado exclusivamente las relaciones jerárquicas; estas relaciones son *broader* y *narrower*, atendiendo a conceptos más generales y más específicos respectivamente. Por ejemplo, el concepto **Mecánica** sería *broader than* **Mecánica de Fluidos**, y viceversa, **Mecánica de Fluidos** sería *narrower than* **Mecánica**.

Para el desarrollo de este proyecto se ha utilizado una representación del modelo SKOS en OWL [6], lo que permite extender el razonamiento de OWL al modelo de SKOS.

[5] *Concept*, http://www.w3.org/TR/2009/NOTE-skos-primer-20090818/#secconcept
[6] SKOS en OWL, http://www.w3.org/2004/02/skos/core/owl-dl/skos-core-owl-dl.owl

Para que quede patente la forma en que se hace uso de estos elementos se definirá el siguiente ejemplo. Disponemos de un recurso que hace referencia a un libro, llamado "Java for Dummies", patente en la URI `http://eina.unizar.es/ejemplo#java4dummies`, para clasificar este recurso y saber que esta relacionado con los lenguaje de programación necesitamos crear un nuevo recurso e instanciarlo con la clase Concepto, a este nuevo recurso lo llamamos "Lenguajes de programación", y estará presente en la URI `http://eina.unizar.es/ejemplo#lprog`. Este recurso contendrá el concepto Java que se refiere al lenguaje de programación. A su vez, al tratarse de un libro, el recurso "Java for Dummies" lo instanciaremos a la clase "Libro", identificada por `http://eina.unizar.es/ejemplo#libro`. Exponiendo lo redactado en forma de grado obtenemos la Figura B.2.

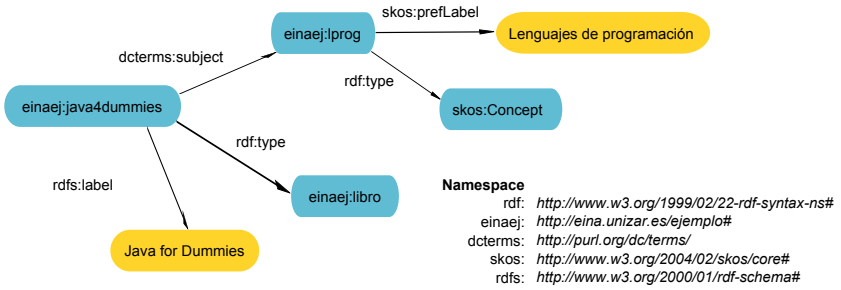

Figura B.2: Categorización de un recurso con SKOS

En el caso de este ejemplo, siguiendo el esquema que usa el repositorio externo de datos estructurados usado durante este proyecto, los recursos enlazan con los conceptos SKOS con la propiedad `dcterms:subject` del DCMI (léase la subsección B.1.3 del apéndice B).

Si se quisiese reducir más el ámbito del recurso, podría definirse por ejemplo un nuevo concepto llamado "Java" con las siguientes tripletas:

- `<einaej:java> <rdf:type> <skos:Concept>`
- `<einaej:java> <skos:prefLabel> "Java"`
- `<einaej:java> <skos:broader> <einaej:lprog>`

Podría eliminarse ahora la propiedad `dcterms:subject` de la Figura B.2 que relaciona el libro "Java for Dummies" con el concepto de "Lenguajes de programación" y enlazarlo sin embargo al nuevo recurso "Java". Este recurso "Java" es más específico que el anterior gracias al uso de la propiedad `skos:broader` de SKOS. De esta manera, aunque el recurso del libro "Java for Dummies" no presente una relación explícita con el concepto de "Lenguajes de programación", esta se podrá inferir de la propiedad `skos:broader` por razonadores DL. El grafo final quedaría de la manera expuesta en la Figura B.3.

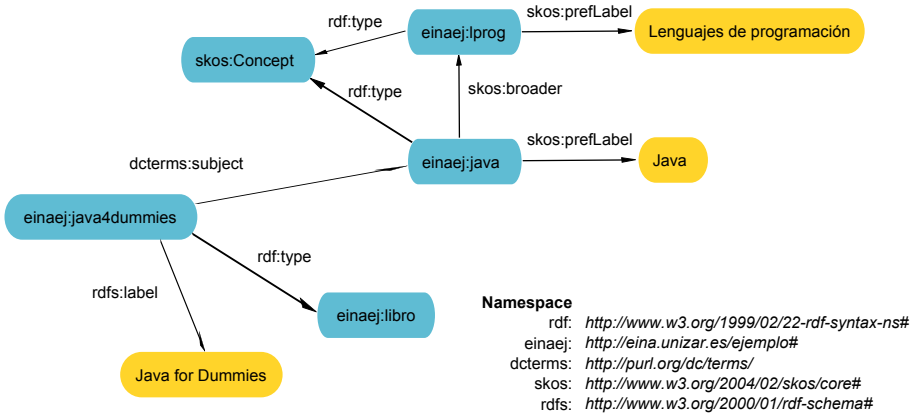

Figura B.3: Categorización de un recurso SKOS con uso de `skos:broader`

SKOS proporciona más vocabulario para organizar el conocimiento a los que no se da uso en este proyecto, para más información se puede consultar la página web de SKOS con aloja el W3C [7].

B.1.3. Dublin Core

Dublin Core es un modelo de metadatos creado por el DCMI [8], organización que fomenta el desarrollo y uso de modelos estándar interoperables de metadatos. La finalidad de este modelo es proporcionar un conjunto de términos con características "base" a todos los recursos, estos elementos son amplios y genéricos, capaces de proporcionar la información semántica básica sobre una gran variedad de recursos.

Este modelo describe 15 propiedades básicas, conocidos como *Dublin Core Metadata Element Set*, entre ellas se destacará por su mayor implicación con este proyecto la propiedad *subject* [9]. Esta propiedad define el tema de un recurso, para especificar a menudo la categoría a la que pertenece o a que ámbito se aplica el recurso.

Esta propiedad *subject* es importante para el proyecto ya que en la recuperación de datos sobre la DBPedia es el enlace entre la categorización de los recursos y sus definiciones.

B.1.4. FOAF

FOAF, acrónimo de *Friend of a Friend* [10], es un proyecto destinado a la creación de datos estructurados y, siguiendo las pautas marcadas por la iniciativa Linked

[7] Documentación SKOS, http://www.w3.org/2004/02/skos/specs
[8] DCMI, *Dublin Core Metadata Initiative*, http://dublincore.org/
[9] *Subject*, http://dublincore.org/documents/dcmi-terms/#terms-subject
[10] FOAF (*Friend of a Friend*), http://www.foaf-project.org/

Data, de enlazarlos entre ellos. Este proyecto está centrado en la información sobre las personas, las relaciones entre ellas y las actividades que estas realizan.

Para tal propósito el proyecto ha creado y mantiene el vocabulario FOAF, conjunto de clases y propiedades para su uso con RDF. Este vocabulario se ha utilizado principalmente para recuperar las URLs de las webs relativas a los recursos, para ello se ha utilizado la propiedad *primaryTopic* [11], que relaciona un documento, en este caso web, con la cosa (*Thing* en OWL) estrechamente relacionada con lo que se puede encontrar en el documento. En nuestro caso, la URI de un recurso con la página web que describe el mismo.

Si bien este vocabulario en el caso concreto de esta implementación solo se le ha dado este uso, es plausible que definiendo un dominio distinto se deban utilizar clases o propiedades pertenecientes a FOAF, pues pueden encontrarse las entidades que este define tanto en la DBPedia como en otros repositorios y webs.

B.2. Java

Para el desarrollo de proyecto el lenguaje de programación elegido ha sido Java. A la hora de elegir este lenguaje, orientado a objetos, se tuvieron en cuenta los siguientes puntos para elegirlo:

- **Experiencia previa**. Durante el trayecto de la carrera se ha hecho uso de este lenguaje de programación para diversas prácticas, lo que fue una buena formación y proporcionó una buena base para que el tiempo dedicado a la familiarización del lenguaje fuera innecesario, solo siendo necesario adaptarse a las especificaciones y usos de las bibliotecas externas utilizadas.
- **Portabilidad**. Gracias a que la ejecución de los programas escritos en Java se ejecuta en máquinas virtuales, este lenguaje se ve independiente de la plataforma que has usado para crear el código, y se comporte de la misma manera sea cual sea el entorno de ejecución.
- **Bibliotecas externas**. El uso extendido de Java ha proporcionado a la comunidad de desarrollo con múltiples bibliotecas externas, APIs, que añaden funcionalidad a las aplicaciones que pueden desarrollarse en este lenguaje sin ser su incorporación y uso en el proyecto un problema, ya que es práctica habitual que vengan acompañadas con documentación y ejemplos de uso para su rápido aprendizaje.
- **Entorno de desarollo**. Java dispone de varios entornos de desarrollo para múltiples plataformas que incorporan varias utilidades para que la escritura de aplicaciones en este lenguaje sea hagan más llevaderas. En este sentido el entorno que se ha utilizado durante el desarrollo del sistema ha sido NetBeans.

[11] *primaryTopic*, *http://xmlns.com/foaf/spec/#term_primaryTopic*

B.3. SPARQL

SPARQL [23] es el acrónimo recursivo de *SPARQL Protocol And RDF Query Language*, y es usado para realizar consultas a uno o varios repositorios RDF. SPARQL es otra recomendación del W3C para la Web Semántica y se le considera el sucesor semántico de SQL. A continuación pasará a describirse el funcionamiento del lenguaje para que puedan entenderse las consultas que se han realizado para extraer los datos de la DBPedia.

SPARQL dispone de 4 maneras básicas de realizar peticiones al punto de acceso RDF o *endpoint*: DESCRIBE, ASK, CONSTRUCT, y SELECT. Las dos primeras son útiles para realizar preguntas descriptivas sobre recursos, de manera que podamos hacernos una idea de cómo son los datos presentes en el repositorio; CONSTRUCT sirve para crear grafos RDF, esto nos puede valer para crear subconjuntos de datos partiendo de la información ya almacenada. Y el tipo más usado durante este proyecto, SELECT, que sigue un formato de consulta parecido al que utiliza SQL, ya que debe formarse con los elementos *SELECT*, *WHERE*, *FROM* y *ORDER BY*. Al igual que cuando se describe un documento RDF, SPARQL ofrece la posibilidad de definir varios *namespaces* para facilitar la redacción y lectura de las consultas.

Para realizar consultas SELECT, primero se definen los datos por los que se está preguntando, esto se realiza en el *SELECT* escribiendo por orden las variables que queremos mostrar como resultado. El elemento *FROM* se utiliza para especificar la URI del grafo al que se quiere acceder, en nuestro caso no se ha tenido que utilizar tal elemento ya que todos los datos estructurados que se buscaron estaban disponibles en grafo por defecto del punto de acceso público de la DBPedia.

Inmediatamente después, en la sección de *WHERE*, se escriben los patrones de tripletas, que no son más que tripletas pero con la posibilidad de reemplazar cualquier parte de estas por variables libres. Todas estas variables van precedidas por el carácter especial "**?**". Para ilustrar cómo se realiza una consulta sencilla se define un ejemplo donde se buscarán los nombres de todas los recursos presentes en el repositorio donde se realice la consulta.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
          ?person foaf:name ?name .
}
```

Cada vez que utilizamos una variable en una tripleta, esta se liga con todas las apariciones del repositorio que concuerden con el resto de la tripleta, en este caso, nuestro patrón nos daría todas las tripletas que tuviesen como predicado a la propiedad identificada por la URI `http://xmlns.com/foaf/0.1/name`.

También es posible realizar emparejamientos con literales, por ejemplo, si el anterior ejemplo lo modificásemos de la siguiente manera:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE {
          ?person foaf:name “Pedro” .
}
```

Con esta consulta obtendríamos las URIs de todas los recursos cuyos nombres fuesen exactamente “Pedro”.

Este mismo resultado se podría realizar usando filtros; estos filtros son muy útiles ya que nos permiten filtrar el número de tripletas resultado de manera más específica. Siguiendo el mismo ejemplo, *foaf:name* es usado habitualmente para referenciar el nombre completo de las personas, si los datos a consultar siguiesen esa conducta nos encontraríamos que la anterior consulta no nos devolvería ningún resultado; para realizar la consulta entonces necesitaremos del elemento *FILTER*, combinado con la función *regex* de la siguiente manera:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE {
          ?person foaf:name ?name .
          FILTER regex(?name, “*pedro*”, “i”)
}
```

El elemento *FILTER* entonces realiza el análisis de cada uno de los literales vinculados a la variable **?name** para ver si responden a la expresión regular “*pedro*”. Dándonos finalmente la consulta como resultado todos las URIs de recursos cuyo nombre contenga la cadena “pedro”.

El carácter final de la función *regex* son los modificadores, en este ejemplo y durante el proyecto el único utilizado ha sido ’**i**’, que marca la no sensibilidad de los caracteres en mayúsculas.

Esta función *regex* es de suma importancia para este proyecto ya que el primer tipo de búsqueda que se realiza es por palabras clave, lo que significa que debemos usar dicha función para descartar recursos que no contengan las palabras buscadas (como se explica en la Sección C.2).

Aunque esta función nos ofrece la ventaja de poder realizar consultas complejas en busca de patrones específicos dentro de los literales enlazados, al combinarse con textos largos, hay que tener presente la cantidad de recursos que puede llegar a consumir, ya que aplicada a un gran conjunto de tripletas puede afectar al rendimiento de nuestra consulta.

Existen varias funciones más que pueden usarse con el elemento *FILTER*, para el caso de este proyecto cabe destacar únicamente la función *langMatches*, que nos filtrará los resultados de literales que estén escritos en un idioma específico:

```
PREFIX foaf: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?label
WHERE {
          <http://dbpedia.org/resource/Australia> foaf:label ?label .
          FILTER langMatches(?name, "JA")
}
```

Esta consulta nos proporcionaría el nombre en japonés del país Australia.

Estas son las bases en las que se basan las consultas SPARQL de nuestro sistema, sin embargo, este lenguaje dispone de más posibilidades para enriquecer las búsquedas, para más información puede consultarse el documento web con la definición del lenguaje y sus características [23].

B.4. Herramientas

B.4.1. Protégé

Para la edición de ontologías se ha hecho uso de Protégé [12], una aplicación libre y de código abierto para la creación y manipulación de ontologías. Esta aplicación, de reconocido uso por la comunidad que trabaja con ontologías, incorpora los lenguajes de modelado RDF y OWL, y permite la incorporación de plugins para implementar nuevas utilidades o incluir nuevos vocabularios. También proporciona la posiblidad de trabajar con razonador e inferir hechos de los axiomas ya presentes en la ontología.

Durante el desarrollo de este proyecto se ha utilizado Protégé con los plugin SKOS Editor [13], que añade el vocabulario de categorización de SKOS y ProSÉ [33], que añade la posibilidad de extracción modular de ontologías ya existentes, lo que ha sido de importancia a la hora de crear la Ontología Dominio, ya que el modulo extrae las recursos elegidos de una o varias ontologías sin perder el contexto al que están vinculados.

B.4.2. OWL y SKOS API

Java dispone de numerosas bibliotecas externas de las que se puede hacer uso para añadir funcionalidades. En este sentido tanto OWL como SKOS disponen de API para Java [21, 22], con todos las clases necesarias para utilizar todo el potencial de OWL y SKOS.

Una vez estudiadas las posibilidades de estas herramientas su uso ha sido de gran ayuda a la hora de implementar la aproximación. OWL API ha servido para poder explorar la ontología a bajo nivel, poder diferenciar fácilmente el dominio

[12] Protégé, http://protege.stanford.edu/
[13] SKOS Editor, http://code.google.com/p/skoseditor/

de búsqueda de la subconjunto extraído de las ontologías del repositorio externo, y realizar inferencias sencillas.

SKOS API hace uso interno de la misma OWL API e incluye el modelo SKOS a sus características, su uso ha servido para crear y almacenar la taxonomía de las categorías de los recursos. Gracias a SKOS API, la recuperación de las categorías de la DBPedia se realiza al inicializar el sistema y queda almacenada en memoria, aliviando al sistema de futuras consultas al repositorio externo para alterar el dominio de búsqueda.

B.4.3. Razonadores DL

Los razonadores son programas informáticos capaces de, partiendo de un conjunto de axiomas o hechos inicialmente presentes, inferir consecuencias lógicas partiendo de un conjunto de reglas. En el caso de este proyecto estas reglas son determinadas por el modelo OWL [13].

Entre las facilidades que ofrece la OWL API se incluye una interfaz para el uso de razonadores DL, lo que ofrece la posibilidad de realizar inferencias en nuestra ontología sin necesidad de implementar un razonador propio dada la presencia de varios de razonadores libres en la Web que implementan esta interfaz.

En el caso de este proyecto el razonador utilizado ha sido Hermit [9]. Ya que ofrece tanto un plugin para el uso con Protégé como implementa la interfaz para el uso de razonadores en la OWL API.

B.4.4. Jena

Jena es una API para escrita para el lenguaje de programación Java que contiene todas las herramientas necesarias para crear y modificar ontologías, y guardarlas y cargarlas en diferentes formatos. Jena incluye también un módulo SPARQL, que ofrece la posibilidad de realizar consultas a grafos RDF tanto internos como externos. También posee un módulo que ofrece una interfaz para utilizar distintas implementaciones de razonadores DL, tal y como ofrece la OWL API.

Aunque comparte diversas funcionalidades con la OWL API, para la creación y modificación de ontologías se ha hecho uso de esta última por la mayor facilidad en su manejo. El principal motivo para el uso de Jena, aparte de para una primera toma de contacto con las herramientas de la Web Semántica, ha sido la disponibilidad del módulo SPARQL, utilizado para realizar todas las consultas al *endpoint* de la DBPedia.

B.4.5. Lucene

Lucene [29] es una librería para la recuperación de información que implementa entre otros un motor de búsqueda de texto. El proyecto Lucene está apoyado por la Apache Software Foundation y dispone de API para varios lenguajes de programación, entre ellos el utilizado durante este proyecto, Java.

Su principal uso durante el proyecto ha sido el de realizar el ordenado por relevancia de los resultados obtenidos de las búsquedas semánticas (explicado con más detalle en la Sección 4.2.3), aunque su uso también ha sido muy útil como caché de búsquedas.

B.4.6. Servicios web

Para servir los resultados que se obtienen con la implementación realizada se ha optado por utilizar un la tecnología de los Servicios Web. Se ha optado por esta solución para que tanto este proyecto como en el paralelo de Daniel Martínez pudiesen trabajar en paralelo, sin que el desarrollo de uno comprometiera la integridad y resultados del otro.

Aunque esto podría haber sido alcanzado también por otros medios, como por ejemplo el de una interfaz para las búsquedas que este proyecto implementase, entre los futuros pasos que se pensaron para este proyecto se incluía el hecho de realizar una copia local del conjunto de datos presentes en la DBPedia, de manera que no se dependiese finalmente de un punto de acceso público para recuperar los datos. Ya que los servicios web ofrecen la posibilidad de realizar peticiones remotas, no será necesario en futuros desarrollos el trasladar, junto a proyectos como el de Daniel que den uso a este sistema, el conjunto de datos estructurados al que acceden.

El servicio web creado ha sido de tipo SOAP[14] con el set de APIs para Java JAX-WS[15].

B.4.7. Applet

Finalmente, para poder mostrar el trabajo realizado y dar uso a la aproximación creada y el servicio web sobre el que implementa, es necesaria la presencia de una interfaz gráfica.

Inicialmente se pensó en diseñar una web para las búsquedas que se comunicase con el servicio web haciendo uso de HTML y PHP. Durante el desarrollo, sin embargo, se optó finalmente por el desarrollo de un Applet. El menor período de aprendizaje que este suponía al tener que usar el lenguaje de programación Java, del

[14] SOAP, *Simple Object Access Protocol*, http://www.w3.org/TR/soap/
[15] JAX-WS, *Java API for XML Web Services*, http://jax-ws.java.net/

cual ya se tenía experiencia previa, y la misma posibilidad que en el caso anterior, de poderse utilizar embebido en una páginas web, siendo además posible su ejecución en local, supuso la elección final del uso de un Applet.

B.4.8. Software de apoyo

Durante el desarrollo del proyecto se utilizaron además las siguientes aplicaciones:

- **NetBeans IDE 7.1.2** como entorno de desarrollo Java.
- **Gantt Project 2.5.3 Praha**, para elaborar el diagrama de Gantt del proyecto.
- **Dia 0.97.2** para la elaboración de diagramas de casos de uso, diagramas de clases y trazas de eventos durante la fase de diseño.
- **Adobe Photoshop CS4** para la generación y edición algunos elementos gráficos de la memoria.
- **Inskape 0.48** para realizar algunas de las figuras que son mostradas a lo largo del documento.
- **LATEX** para la elaboración de la documentación. En conjunción con un entorno gráfico para escribir documentos en LATEX, **LYX 2.0.3**.

Apéndice C

Ontología Dominio y construcción de consultas

En la sección 4.2 se ha explicado el proceso de definición de la Ontología Dominio, así como los dos tipos de búsqueda que la solución ofrece. También se ha expuesto la arquitectura del sistema para podernos hacer una idea de como trabaja el mismo. Sin embargo, el proceso de construcción de consultas utilizando la Ontología Dominio no ha quedado ilustrado con un ejemplo por motivos de espacio.

Para describir adecuadamente como se construyen las consultas estructuradas, el presente apéndice contiene, primero, un ejemplo de Ontología Dominio, con un repaso sobre las propiedades, clases y anotaciones de interés. Y segundo, la metodología que utiliza el sistema para construir consultas partiendo de la ontología, utilizando el anterior ejemplo como referencia.

C.1. Ejemplo de Ontología Dominio

A continuación se describirá una ontología de ejemplo en la Figura C.1. La representación visual de la ontología no corresponde al visualizador nombrado en la Sección B.1.1. En este caso se mezclan varias ontologías y se quiere hacer la distinción entre sus orígenes. Además, se quiere visualizar las anotaciones, normalmente no representados por los visualizadores de ontologías al no influir sobre la semántica de estas.

En esta figura, pueden observarse un conjunto de cajas en tonos grises, cada una de ellas corresponde a una de las clases presentes en la ontología definida. Las clases con relleno gris corresponden a aquellas que se han extraído usando técnicas de extracción de ontologías [14]. Para utilizar este tipo de extracción se ha utilizado un módulo de la utilidad Protégé, ProSÉ [33]. Dicho plugin toma como entrada un conjunto de entidades iniciales, y nos proporciona con una subontología extrayendo

las entidades elegidas junto con aquellas necesarias para que no pierdan el contexto al que están asociadas.

Las cajas con relleno oscuro son las definidas por nosotros, y se corresponden con nuestra vista de los objetos del repositorio. Durante el ejemplo nos referiremos a las clases definidas para nuestra ontología como internas, y aquellas extraídas de otras ontologías como externas.

De entre las clases extraídas nombraremos especialmente a `foaf:Document`, ya que aunque sea una clase, no nos interesa en el mismo aspecto que el resto de clases, pues como puede observarse no posee una clase equivalente interna. Esto es así ya que no estamos buscando los valores de las propiedades que sus instancias puedan poseer, lo único que se busca de los recursos de este tipo es su URI. Para ello se hace uso de la anotación **@dataRetrievable** en la propiedad `foaf:primaryTopicOf`, anotación que se describirá cuando se detalle el uso de las *DataProperties*.

Estas clases se deben relacionar con las presentes en el repositorio para que el razonador que utiliza el Motor de Búsqueda sea capaz de realizar la inferencia de clases. Las flechas negras representan estas relaciones, todas ellas etiquetadas utilizando la propiedad `owl:equivalentClass`. Hay que tener en cuenta también que todas las clases son subclases de `owl:Thing` (diferenciada en la figura por un borde discontinuo), aunque no se hayan dibujado tales relaciones.

Pasando a describir las *ObjectProperties* de la figura, estas están identificadas por flechas, teniendo como dominio la clase presente en la base de la flecha y como rango la encontrada en la punta. Como puede observarse, existen flechas de distintos colores, con relleno y sin relleno. Las flechas con relleno corresponden a aquellas utilizadas en el repositorio externo. Estas fueron las que se pasó junto a las clases como entrada al extractor de módulos de ontologías. Estas propiedades siempre se relacionarán con clases externas.

Las propiedades internas, cada una de ellas identificada por la flecha de un solo color sin relleno, siempre tienen como dominio y rango a clases internas, con la única excepción de `owl:Thing`, la superclase de todas las clases. Cada propiedad interna presenta una relación de `owl:equivalentProperty` con la propiedad externa del mismo color.

Utilizando los axiomas descritos, el razonador puede inferir, por ejemplo, que la propiedad `eina:conocidoPor` tiene como dominio `eina:Persona` y como rango `eina:Instituto` y `eina:Articulo`. Esto es un ejemplo de la vista que tendrá el usuario de los datos, representada en la Figura C.2.

Las *DataProperties* se definen de manera similar a las *ObjectProperties*, cada propiedad interna es equivalente por `owl:equivalentProperty` a la externa extraída de la ontología original. En la Figura C.3 pueden observarse las clases y propiedades internas, así como las propiedades externas utilizadas, y dominios y rangos de estas.

En la ontología definida, las propiedades externas, como puede apreciarse en la figura, son las que nos proporcionan el rango a nuestras propiedades (internas). El

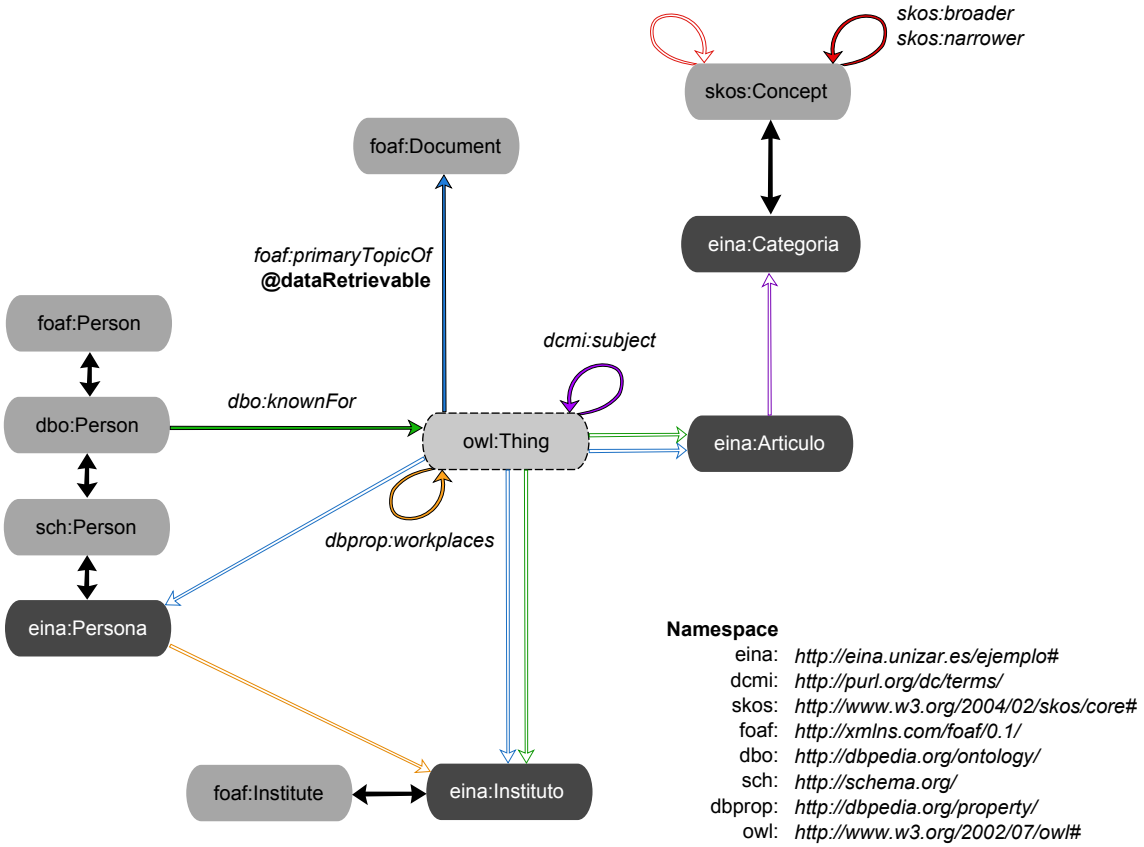

Figura C.1: Clases y *ObjectProperties* de la Ontología Dominio usada en este proyecto

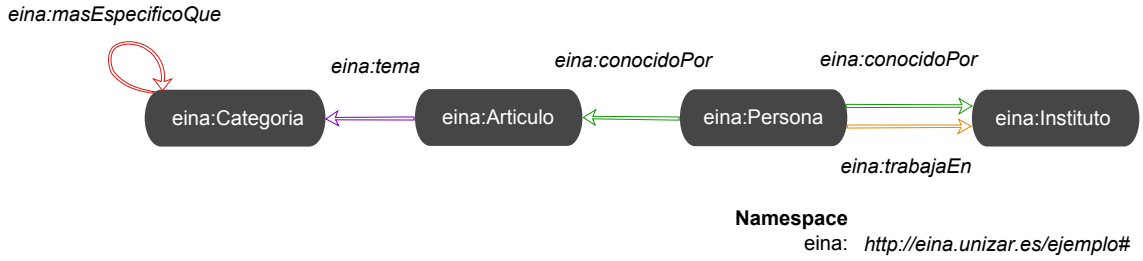

Figura C.2: Vista interna de la ontología

dominio de las propiedades lo definimos nosotros al describir la propiedad, utilizando las clases para las cuales se quiera consultar el valor de la propiedad. Por ejemplo, cuando en una búsqueda nos devuelva un objeto Persona, obtendremos los valores de las tres propiedades de la que es dominio, sin embargo, al devolvernos la búsqueda una Categoría, sólo nos devolverá los valores de las propiedades `dbo:label` y `foaf:primarytopicOf`.

Además, es necesario marcar las propiedades de las que se quiera recuperar su valor con la anotación **@dataRetrievable**, de esta manera se pueden disponer de

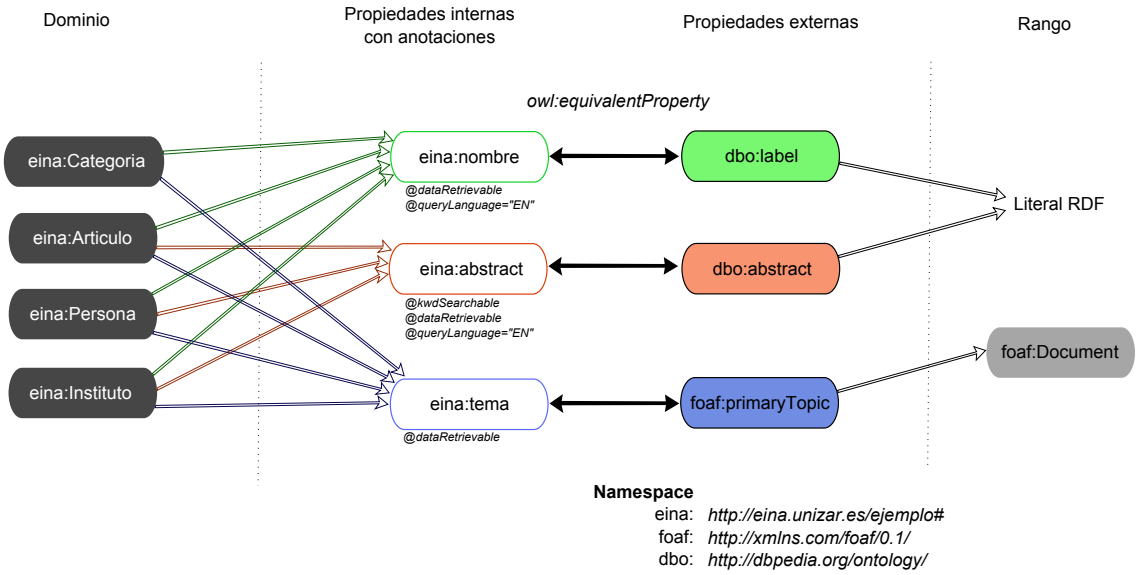

Figura C.3: Propiedades **@dataRetrievable**

propiedades de las cuales no queramos recuperar su valor para una clase en concreto, como ocurre con el resto de *ObjectProperties* de esta ontología.

Existe también otros dos tipos de anotaciones necesarias propias de las *DataProperty*: **@queryLanguage** y **@kwdSearchable**. La primera define el idioma de los datos a recuperar. El repositorio de la DBPedia dispone de numerosos idiomas para millones de recursos, por lo que cambiando el literal que contiene esta anotación podemos recuperar la información en uno u otro idioma. La segunda anotación define sobre qué campo se va a realizar la búsqueda de las palabras clave entrantes. Por lo general esta segunda anotación **@kwdSearchable** es recomendable utilizarla solo sobre la propiedad `dbo:abstract`, puesto que sus valores contienen el texto más amplio definiendo el recurso.

C.2. Construcción de consultas

A continuación se describirán los pasos que sigue el Motor de Búsqueda para construir la consulta SPARQL necesaria para la búsqueda por palabras clave. Para ello, parte de la entrada del usuario y de la Ontología Dominio definida.

En el prototipo realizado para este tipo de búsquedas, donde el repositorio de datos es la DBpedia y la taxonomía una categorización SKOS, el Motor de Búsqueda recibe como entrada un conjunto de palabras clave y una categoría SKOS. Cada palabra clave se analiza por separado y se realiza una consulta independiente. Los resultados recuperados de cada palabra clave son almacenados mediante Lucene y ordenados según el número de apariciones de cada palabras clave.

Para mostrar el funcionamiento del Motor de Búsqueda se utilizará un ejemplo, el cual tomará como entrada la cadena "fish movement" y como categoría "Mechanics". Se utilizará la ontología definida en el apartado anterior como Ontología Dominio.

El primer paso que realiza el motor de búsqueda es la partición de la cadena de entrada "fish movement", en dos cadenas, para cada una de ellas realizará una consulta SPARQL independiente.

Como se ha explicado en apartados anteriores, las búsquedas por palabras clave siempre buscan sobre la clase `eina:Articulo`, que correspondería con los recursos de la DBPedia que no tienen tipo definido. Para construir una consulta para este tipo de búsqueda, en primera instancia, se extraen, mediante el uso de un razonador, todas las propiedades externas que tienen como dominio o rango a esta clase. En el caso de la ontología definida, serían las propiedades `dbo:label`, `dbo:abstract` y `foaf:primaryTopic` [1]. Junto a las propiedades también se debe extraer si estas son dominio o rango de la clase, por que tendremos un par (propiedad,dirección) necesario para empezar a construir la consulta.

Lo primero que se añade a la consulta el la cláusula *SELECT* junto a los identificadores de las variables que queremos recuperar, como ya disponemos de las propiedades que queremos utilizar podemos empezar a construir la consulta de la siguiente manera:

```
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
}
```

Una vez tenemos marcadas las variables de recuperar pasaremos a construir la sección *WHERE*, escrita entre llaves. En primera instancia se determinarán las categorías a analizar de entre todas las presentes en la DBPedia, para ello, se hará uso de la categoría de entrada junto a la profundidad marcada por el sistema. Esta profundidad no se admite en nuestro sistema como variable, ya que al acceder al repositorio externo de la DBPedia, hemos determinado que el mayor rendimiento[2] se obtiene con profundidad 2, y así está configurado el sistema. De esta manera, la consulta utiliza la taxonomía SKOS con la categoría de entrada (*Mechanics*) para formar la siguiente consulta:

[1] En la ontología ejemplo esta propiedad figuraba como `primaryTopicOf`, para este ejemplo se utilizará `primaryTopic`, que es la inversa de la primera.

[2] Cuando aquí se nombra rendimiento nos referimos a la cantidad de artículos que se pueden obtener utilizando una u otra profundidad con respecto al tiempo que luego le puede costar al repositorio realizar dicha búsqueda. Con una profundidad de 1, los resultados recuperados son pocos, mientras que con profundidad 3, el número de tripletas se dispara y el tiempo de consulta se alarga, llegando a veces el servidor a denegar ciertas consultas que sobrepasan un parámetro con el tiempo estimado de consulta.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
   OPTIONAL
   {
      ?subcat1 skos:broader <http://dbpedia.org/resource/Category:Mechanics>
   }
}
```

Esta consulta de por sí misma no nos aporta nada, pues lo único que hemos realizado ha sido un enlace entre la categoría *Mechanics* y sus subcategorías. A continuación añadiremos enlazaremos la variable **?uri** con todas las URIs que cuyo sujeto pertenece a alguna de esas categorías:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
   OPTIONAL
   {
      ?subcat skos:broader <http://dbpedia.org/resource/Category:Mechanics>
   }

   {
      ?uri dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }
   UNION
   {
      ?subcat dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }
}
```

Estas URIs corresponden a los artículos de la Wikipedia en los que vamos a buscar la palabra clave, y de entre los que encontremos la palabra clave recogeremos los atributos marcados en la Ontología Dominio. Primero enlazaremos a todas las propiedades que la ontología marca:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
   OPTIONAL {
      ?subcat skos:broader <http://dbpedia.org/resource/Category:Mechanics>
   }

   {
      ?uri dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }
   UNION
   {
      ?subcat dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }

   ?uri <http://www.w3.org/2000/01/rdf-schema#label> ?label .
   ?primaryTopic <http://xmlns.com/foaf/0.1/primaryTopic> ?uri .
   ?uri <http://dbpedia.org/ontology/abstract> ?abstract
}
```

Estas propiedades se extraen por medio del razonador buscando por las propiedades marcadas como **@dataRetrievable**. A la vez que se extraen se recuperan también las demás anotaciones, que determinarán qué propiedades tienen que venir con un idioma definido o cual propiedad será la utilizada para buscar las palabras clave.

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
   OPTIONAL {
      ?subcat skos:broader <http://dbpedia.org/resource/Category:Mechanics>
   }

   {
      ?uri dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }
   UNION
   {
      ?subcat dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }

   ?uri <http://www.w3.org/2000/01/rdf-schema#label> ?label
   FILTER langMatches(lang(?label), "en")
   ?primaryTopic <http://xmlns.com/foaf/0.1/primaryTopic> ?uri .
   ?uri <http://dbpedia.org/ontology/abstract> ?abstract
   FILTER langMatches(lang(?abstract), "en")
   FILTER regex(?abstract, "[.., )(\\-\"]fish[.., )(\\-\"]", "i")
}
```

La expresión regular para buscar la palabra clave se enriquece, como puede obser-

varse en la consulta, para encontrar aquellas que entre espacios o algunos caracteres especiales elegidos. Finalmente, solo resta añadir una cláusula secundaria más para construir la consulta final que se lanzará al punto de acceso público:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?uri ?label ?abstract ?primaryTopic
{
   OPTIONAL {
      ?subcat skos:broader <http://dbpedia.org/resource/Category:Mechanics>
   }

   {
      ?uri dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }
   UNION
   {
      ?subcat dcterms:subject <http://dbpedia.org/resource/Category:Mechanics>
   }

   ?uri <http://www.w3.org/2000/01/rdf-schema#label> ?label
   FILTER langMatches(lang(?label), "en")
   ?primaryTopic <http://xmlns.com/foaf/0.1/primaryTopic> ?uri .
   ?uri <http://dbpedia.org/ontology/abstract> ?abstract
   FILTER langMatches(lang(?abstract), "en")
   FILTER regex(?abstract, "[.., )(\\-\"]fish[.., )(\\-\"]", "i")
   FILTER regex(?primaryTopic, "en.wikipedia.org", "i")
}
```

Esta última cláusula nos filtrará las URIs de los documentos web de los artículos a solo aquellos que estén en inglés. La DBpedia proporciona enlaces a la web del artículo en varios idioma. Sin embargo, no todos los artículos que tienen web en un idioma en la Wikipedia poseen enlace a su web en la DBpedia. En este caso, por ser el inglés el idioma más extendido en la Wikipedia y al no poder utilizar el filtro de *langMatches*[3], se utiliza un filtro *regex*.

Este tipo de consultas siempre dependerán de como esté definida la ontología dominio: contra más propiedades sean marcadas con la anotación **@dataRetrievable** más campos se recuperarán del repositorio. Sin embargo, tal y como se ha explicado, la implementación de este tipo de búsquedas se planteó para recuperar recursos de tipo Articulo (sin tipar en la DBPedia), por lo cual el índice Lucene solo tiene en cuenta los atributos definidos en la Ontología Dominio ejemplo.

[3]En este caso, *langMatches* no puede utilizarse ya que esta función solo se aplica sobre literales que puedan tener definido un idioma. Siendo que la propiedad `primaryTopic` es de tipo *ObjectProperty* y no puede contener un literal esta función no puede aplicarse.

Apéndice D

Manual de usuario

El presente apéndice tiene como propósito describir las funcionalidades que ofrece la aplicación a través del servicio web y su interfaz web.

D.1. Servicio Web

Para la comunicación con interfaces gráficas, tanto la del proyecto Alfa III GAVIOTA como la realizada en este proyecto, se ha implementado un servicio web. Los métodos de los que dispone el servicio, tal y como se definen en su WSDL [1] son los descritos a continuación:

- *getRootCategories*
  - Entrada: -
  - Salida: XML
  - Descripción: Al llamar a este método se devolverá en formato XML las categorías raíz definidas por el servicio web. Cada elemento, aparte del elemento raíz del XML, contiene una categoría, con dos nodos de texto, uno aportando la URI de la categoría y otro su propiedad *rdfs:label*, con el nombre de la URI para ser leído por humanos.
- *getSubCategories*
  - Entrada: URI
  - Salida: XML

[1] WSDL es el acrónimo de *Web Services Description Language*. Se trata de un formato XML para describir servicios web, especifica la interfaz abstracta a través de la cual un cliente puede acceder al servicio y los detalles de cómo se debe utilizar. http://www.w3.org/TR/wsdl.

- Descripción: Este método devolverá todas las subcategorías de la categoría identificada por la URI de entrada presentes en la taxonomía SKOS del sistema. El formato del XML es el mismo que en el método *getRootCategories*.

- *searchKeywords*
  - Entradas: String (palabras clave separadas por espacios) y URI de la categoría
  - Salida: XML
  - Descripción: El presente método realiza la búsqueda por palabras clave descrita en este documento y devuelve los resultados, una lista de Artículos, en formato XML. Cada elemento del XML contiene un Artículo con la información del recurso extraída del repositorio externo. En el caso concreto del dominio elegido los atributos son: URI, *rdfs:label*, *dbo:abstract*, y *foaf:primaryTopic*. Siendo *dbo:abstract* la descripción larga que contiene la DBPedia del recurso (campo donde se realiza la búsqueda de las palabras clave) y *foaf:primaryTopic* la página web de la Wikipedia con el artículo sobre el recurso.

- *getRelated*
  - Entrada: URI
  - Salida: XML
  - Descripción: Con el uso de la Ontología Dominio, este método recupera recursos relacionados con el de la URI proporcionada como entrada. Los elementos que contiene el XML vienen determinados por las clases y propiedades definidas en la Ontología Dominio. En el caso del dominio definido como ejemplo estos pueden contener la información de Personas, Institutos, Artículos o Categorías, como se define en la sección C.1.

- *getRelatedWithSource*
  - Entrada: URI del recurso y URI de la clase
  - Salida: XML
  - Descripción: Siguiendo lo expuesto en el apartado sobre la búsqueda de refinado, es posible realizar búsquedas forzando a definir la URI entrante del recurso como instancia de una clase determinada. Este método implementa esta posibilidad y realiza una búsqueda de refinado de URI con el mismo comportamiento que la del método *getRelated*, forzando sin embargo a que la URI del recurso se trate como instancia de la clase dada. Los elementos que contiene el XML vienen determinados por las clases y propiedades definidas en la Ontología Dominio. Si la clase dada no existe en la Ontología Dominio el método no devuelve nada.

- *getRelatedWithTarget*
  - Entrada: URI del recurso y URI de la clase
  - Salida: XML
  - Descripción: Este método, aún comportándose de la misma manera que *getRelated*, devuelve solo los recursos que son instancias de la clase dada como URI. El XML devuelto contiene los recursos marcados como relevantes para la clase proporcionada dentro de la Ontología Dominio. Si la clase dada no existe en la ontología el método no devuelve nada.

D.2. Interfaz gráfica

La interfaz gráfica implementada para el sistema presenta el aspecto definido en la figura D.1, a continuación se pasará a describir cada uno de los cuadros marcados en tal figura.

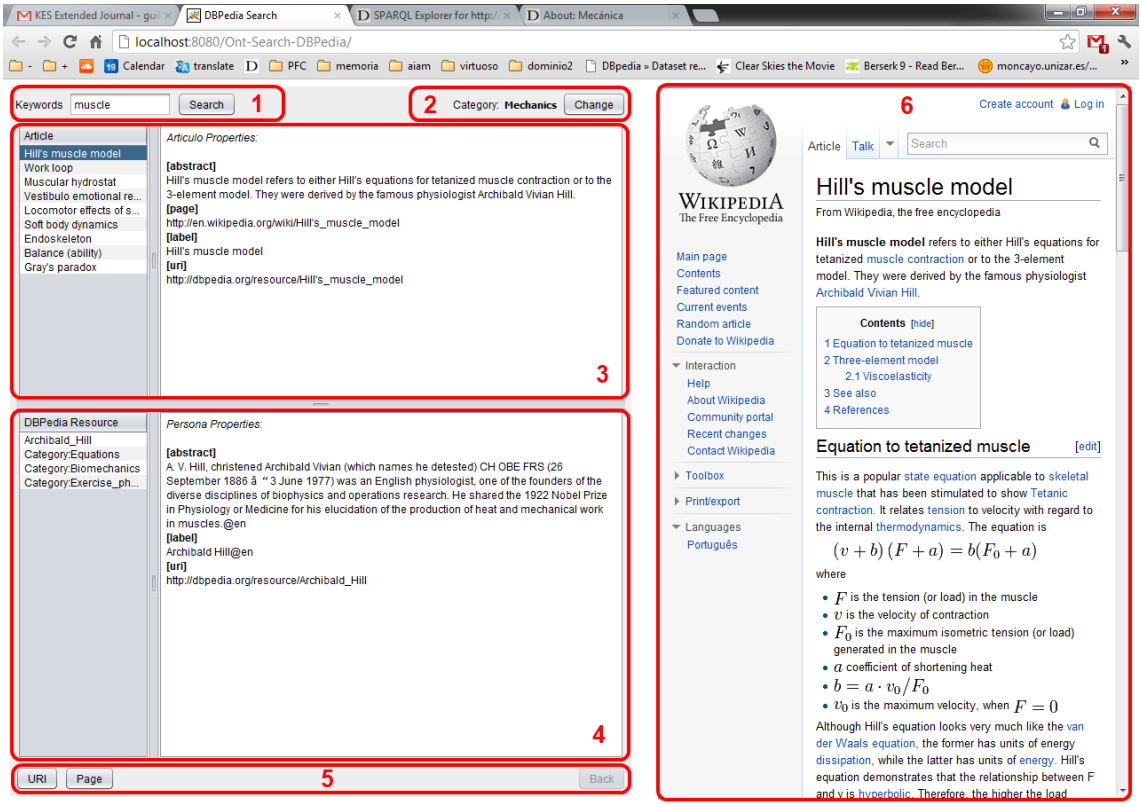

Figura D.1: Elementos de la interfaz gráfica

1 El área número 1 contiene el campo para introducir las palabras clave que se deseen buscar y el botón para empezar la búsqueda de ese tipo. Una vez se

pulsa el botón, la interfaz queda en espera hasta que el servicio nos devuelve los resultados o la búsqueda falle. Si la búsqueda es realizada con éxito, los resultados aparecen en el área 3.

2 Esta parte contiene la actual categoría en activo, es decir, aquella categoría que se tomará como raíz para realizar las búsquedas por palabras clave. Para cambiar tal categoría, es posible pulsar el botón de la parte derecha del área marcada, hecho lo cual aparecerá una ventana con la taxonomía de categorías SKOS almacenada para que el usuario pueda seleccionar una nueva categoría. La composición de la ventana emergente puede verse más adelante en la figura D.2.

3 Este área contiene dos espacios separados verticalmente. El espacio presente en la parte izquierda presenta el listado de artículos que se han recuperado del repositorio tras una búsqueda por palabras clave. Al hacer un clic sobre cualquiera de sus elementos los datos del artículo correspondiente pasan a mostrarse en la parte derecha del área marcada. Los dos contienen, primero el tipo de recurso que se ha recuperado, es decir, su `rdf:type` interno; en este caso será siempre Artículo. A continuación se listarán las propiedades recuperadas junto a sus valores.

4 En esta sección de la interfaz podemos encontrar los resultados de las búsquedas de refinado de URI. Para realizar este tipo de búsquedas es necesario realizar doble clic sobre los elementos de la parte izquierda del área 3 o 4. El comportamiento de este área es el mismo que el de la sección 3, un clic llenará la parte izquierda del cuadro con los datos del recurso. La diferencia significativa es que podremos saber que tipo de recurso hemos recuperado, mientras que en el área 3 los recursos siempre son artículos, en este cuadro podremos encontrarnos con cualquier instancia de clase marcada en la Ontología Dominio, siendo en nuestro caso Persona, Categorías o Institutos.

5 El cuadro 5 presenta 3 botones. El pulsar el primer botón, con etiqueta "URI", el área 6 cargará la web de la DBPedia con el recurso seleccionado en los cuadros 3 o 4 (sólo es posible seleccionar un elemento entre las dos tablas). El segundo botón, etiquetado "Web", hará lo propio, pero esta vez con la web de la Wikipedia sobre el recurso seleccionado. Finalmente, el botón de la derecha, etiquetado "Back", retrocederá el cuadro 4 al estado anterior que tuviese [2].

6 Área para mostrar documentos web, tanto de la DBPedia como de la Wikipedia, sobre los recursos encontrados.

En la Figura D.2 puede observarse la ventana emergente para seleccionar categoría en la que se realizarán las búsquedas por palabras clave. Se trata de un listado

[2]Cuando se realiza un doble clic sobre algún recurso, los nuevos resultados reemplazan a los presentes en el cuadro 4, sin embargo estos son guardados en memoria, de manera que se pueden volver a consultar sin necesidad de repetir la consulta utilizando del botón "Back".

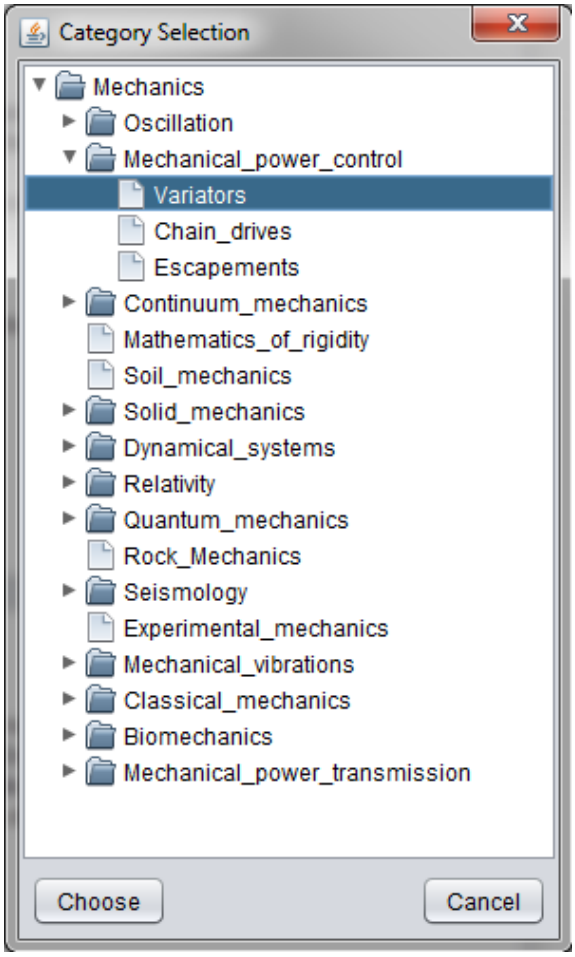

Figura D.2: Ventana emergente de selección de categoría

en árbol de todas las categorías, puede seleccionarse cualquiera de ellas, contenga o no subcategorías. Darle al botón "Choose" seleccionará la categoría elegida, mientras que utilizar el botón "Cancel" o la "X" dejará la categoría previamente seleccionada.

Apéndice E

Artículo KES 2012

En este apéndice se adjunta el artículo publicado en KES 2012 [26], dicho trabajo ha sido realizado por el Director de este PFC Carlos Bobed, el alumno y autor del mismo proyecto Guillermo Esteban, y el Profesor Titular de la Universidad de Zaragoza Eduardo Mena. El artículo [16] ha sido aceptado y será presentado en San Sebastián del 10 al 12 de Setiembre de 2012. Se ha recibido una invitación por parte de los organizadores de dicha conferencia a realizar una versión extendida del artículo publicado para su publicación como *Journal*, trabajo que estamos llevando a cabo incluyendo entre otras algunas de las mejoras mencionadas en la sección sobre trabajo futuro.

Ontology-driven Keyword-based Search on Linked Data

Carlos Bobed, Guillermo Esteban, and Eduardo Mena

IIS Department
University of Zaragoza
50018 Zaragoza, Spain
{cbobed,529679,emena}@unizar.es

Abstract. Nowadays, the Web is experiencing a continuous change that is leading to the realization of the Semantic Web. Initiatives such as Linked Data have made a huge amount of structured information publicly available, encouraging the rest of the Internet community to tag their resources with it. Unfortunately, the amount of interlinked domains and information is so big that handling it efficiently has become really difficult for the final users. DBPedia, one of the biggest and most important Linked Data repositories, is a perfect example of this issue.
In this paper, we propose an approach to provide the users with different domain views on a general data repository, allowing them to perform both keyword and navigational searches. Our system exploits the knowledge stored in ontologies to 1) perform efficient keyword searches over a specified domain, and 2) refine the user's domain searches. We focus on the case of DBPedia, as it mirrors the information stored in the Wikipedia, providing a semantic entry to it.

Keywords: Ontologies, Linked Data, Semantic Web, Keyword Search

1 Introduction

The Web has made a huge and ever-growing amount of information available to its users. The appearance of new interaction paradigms that the so-called Web 2.0 introduced, in which the Web users become part of the information providers, has made this amount even bigger and more difficult to handle. At this point, the Semantic Web [15] has been proposed in order to relieve the user of the burden of processing the available information. By sharing definitions and tagging resources, the Internet is being made understandable to computers, thus, allowing them to process the information on behalf of users.

This progressive structuring is being made using ontologies (which offer a formal, explicit specification of a shared conceptualization [9]) and a set of different technologies built around them, promoted and supported by the W3C [1]. Initiatives such as Linked Data [5] have already made a huge amount of structured information publicly available, providing schemas and data in machine-readable

[1] http://www.w3.org

formats; and, now it is the time to exploit all these information. Unfortunately, it is not realistic to expect that all the resources in the Internet will be perfectly described and annotated, as there are also huge amounts of information that are not semantically annotated as well. So, for the time being, both dimensions (*semantic* and *syntactic* ones) of the Web are condemned to coexist with each other. And so their different techniques and methods are.

In this context, users have become used to keyword-based search interfaces due to their ease of use. However, this ease of use comes from the simplicity of its query model, whose expressivity is low compared with other more complex query models [13]. One possible approach to augment the expressivity while maintaining the ease of use of this query model is the *keyword query interpretation* [8], which is the process to translate them into a structured query. However, the current methods require highly expressive ontologies [7] or building large graphs out of the underlying data [16], which might not be completely available to us (e.g.: we only have access to a single data endpoint to which pose our queries).

In this paper, we present a system that adopts a hybrid search strategy which exploits the knowledge stored in ontologies to focus and enrich the search process on structured data. Our system builds on an external Linked Data repository (which might not be under our control) and takes as input input an ontology which has two roles in the system: 1) to define the taxonomy of the search domain, guiding and narrowing the scope of the keyword-based search; and 2) to define the structure of the objects in the search domain, helping refining and suggesting further search results. With our approach, one can provide different views on a general data repository by just adapting externally the ontology provided. Moreover, our approach can be attached to any public SPARQL endpoint without overloading it (this is important in open scenarios, such as the one depicted by Linked Data). We use the DBPedia [6] as data repository example as it provides us with a semantic entrance to the Wikipedia[2].

The rest of the paper is as follows. In Section 2, we overview the architecture of our search system. The definition of the search domain using an ontology and how we do apply it to the case of DBPedia is explained in Section 3. In Section 4, we focus on how our system uses the ontology to focus the search. Some related work is discussed in Section 5. Finally, the conclusions and future work are drawn in Section 6.

2 Architecture of the system

In this Section, we overview the architecture that allows our system to exploit the information in the external Linked Data repository. As shown in Figure 1, there is a previous offline step that consists of defining the search domain and providing it modeled in an ontology. Our system uses an inner Description Logics reasoner [3] to exploit the information in this ontology. Once it has the domain ontology, our system offers two different but complementary kinds of search depending on the user's input:

[2] http://wikipedia.org

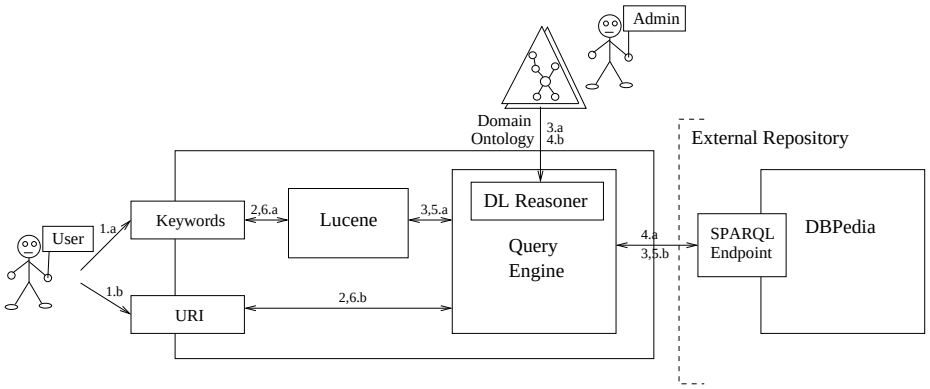

Fig. 1. Our system provides two complementary search services: a) Keyword-based and b) URI refining services.

a) *Keyword-based Search.* This search service takes as input plain keywords (step 1.a) and performs a first search on a Lucene repository (step 2.a). This intermediate storage serves as a cache to alleviate the workload of the external public endpoint (which is not under our control and might have limited availability). If the search has not been performed before, the keyword query is forwarded to our Query Engine (step 3.a) which consults the taxonomy of the ontology to build a focused SPARQL query. This taxonomy includes only the objects we want to be searched, this is, the objects that we define in the search domain. If it is too large, there exists the possibility of specifying a class of the domain ontology to serve as top node of the focused search. When the results are retrieved, they are stored in our Lucene repository to cache them for future searches (step 5.a). The Lucene repository acts a cache for future searches and provides the system with relevance measures and ranking on the results. Finally, the results (a ranked set of URIs) are returned ranked according to their relevance (step 6.a).

b) *URI Refining Search.* The results of the previous search service is a ranked set of URIs, which are presented to the user so s/he can explore them. When the user selects an URI (step 1.b), it is directly forwarded to the Query Engine (step 2.b). The Query Engine consults the data repository to obtain the type of the object behind the URI (step 3.b). Then, it consults the definition of its type (step 4.b) to build a specialized query for that type of object (it consults the relevant properties[3] to retrieve the appropriate data and suggest other related objects) with the help of a DL reasoner. Finally, it forwards the query to the data endpoint (step 5.b) and returns the data (step 6.b). This step is not cached as it is a more specialized query that is not so time consuming as a general search as a keyword query over the whole domain.

[3] They are marked as being relevant during the ontology definition, so the Query Engine can be aware of them.

Notice that the Lucene repository is only used to cache and rank the results obtained from the actual query on the external Linked Data repository. In the following sections, we explain how the search domain has to be defined, how we have applied this architecture to the use case of the DBPedia, and we illustrate and further detail each of the searches.

3 Defining the Search Domain

In this section, we firstly explain how to define the search domain by providing an annotated ontology, and then we present how we have applied it to the use case of the DBPedia.

3.1 Domain Ontology Annotation

First of all, the system has to be provided with an ontology that defines the search domain we want to present to the users. This ontology provides a view on the data and has to be aligned to the ontology that describes the actual data repository. This way our system can consider only part of the data while being able to access it properly. In fact, although it can be built from scratch, we advocate for using ontology extraction techniques [12] to obtain a module and, then, make our system work directly with a subontology of the repository's one. The definition of the search domain is based on three main aspects:

- The *taxonomy* we define in the domain ontology contains the objects that are considered by our system in the searches. The size of this taxonomy is not a problem, as our system can receive an extra parameter to consider any concept as top node and thus focus the search only on its descendants.
- The *object properties* defined in the search domain play a crucial role as they are exploited in the refining step to further propose relevant results. We have to provide the relevant properties (the ones that the Query Engine takes into account) by marking them with an @relevantProperty annotation.
- Finally, the *keyword-searchable* properties of the objects, i.e., the properties that have values that can be processed to perform keyword searches. We annotate them as @kwdSearchField. These properties are the ones that are considered for the keyword search.

The Query Engine exploits this information to build scoped queries on the structured data. Depending on the underlying repository, we can specify via the annotations the different fields which we can perform the searches on and the objects that are relevant to the domain view we want to offer.

3.2 Structure of DBPedia

In this section, we present the inner organization of DBPedia resources to explain how our system handles it. DBPedia [6] is a project that extracts structured data from Wikipedia, and makes this information available on the Web under the principles of Linked Data. This extraction is performed automatically by exploiting the structure of the information stored in Wikipedia. However, the nature of the results of this extraction process differs from the sources in more ways than barely structural and format ones.

When moving from the *article world* of Wikipedia to the *semantic resources* in DBPedia, there are objects that might augment their descriptions as the new semantic model can represent more information about them. The articles extracted from Wikipedia, once in DBPedia, become *resources*. Each resource is represented by an URI and has a direct correspondence to its original Wikipedia's article, inheriting its categorization. The whole taxonomy of article categories of Wikipedia is included as an SKOS[4] ontology in DBPedia; thus, DBPedia provides a first view on the resources according their category.

Depending on the content of its corresponding article, a DBPedia resource might also be representing an object (see Figure 2). The classification of this object dimension of the resources is done via several general domain ontologies, being DBPedia Ontology[5] and YAGO[6] the most important ones. In this way, independently of the article categorization, DBPedia offers a second different view based on the nature of the underlying resources. However, this view does not cover all the DBPedia. There exist resources that, despite being categorized, do not have these descriptions as they are not defined in the used ontologies, as shown in Figure 2.

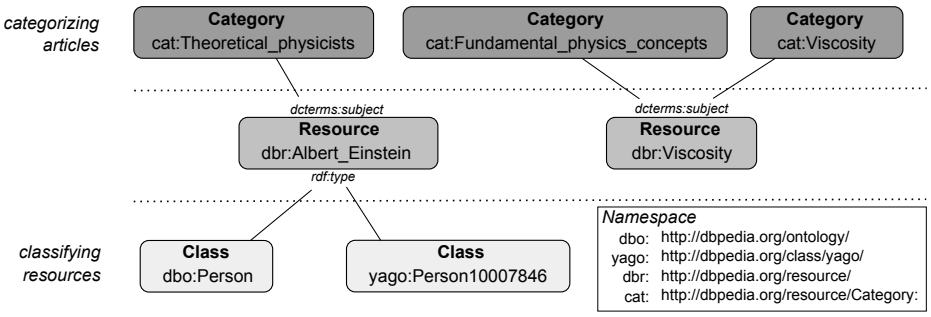

Fig. 2. DBPedia excerpt of the descriptions of *Albert Einstein* and *Viscosity* resources.

Summing up, DBPedia organizes knowledge in two major ways: the SKOS categorization, and an ontological classification. In our system, when working with the DBPedia as underlying data repository, we have considered the SKOS categorization as the general taxonomy to be pruned to focus the keyword search. On the other hand, we obtain the ontological definitions from the DBPedia ontology for the URI refining search. This is done to show the flexibility of our approach as the search domain definition is completely decoupled from the actual process search (the domain ontology in Figure 1 can be separated to focus on different aspects from the object definitions). In particular, in our prototype, we have pruned the SKOS categorization to deal with the categories under

[4] SKOS Simple Knowledge Organization, `http://www.w3.org/2004/02/skos/`
[5] The DBPedia Ontology, `http://wiki.dbpedia.org/Ontology`
[6] YAGO Ontology, `http://www.mpi-inf.mpg.de/yago-naga/yago/`

'Mechanics', and we are only interested, for further refinement, in people, institutions and the different articles that could be related to each resource. The keyword search is performed on the *abstract* property, which gives an excerpt of the Wikipedia entry associated to each resource.

4 Searching on Structured Data

Once we have provided the annotated ontology, our system is able to perform two types of searches on the external Linked Data repostory: a keyword search and a URI refining search. Both searches exploit the ontological definition of the domain we have provided but in different ways, as we detail in the rest of the section.

4.1 Keyword Search

When the user poses a keyword query, the Query Engine consults the domain ontology to obtain information about:

- The properties on which it has to perform the keyword search. Due to the structure of Linked Data (it follows the RDF model based on triplets) the system would have to look for the keywords in every element of the triplets (subject, property, and value) if we do not specify which properties to search on. This would lead to unbearable query processing times (note that our system builds on public endpoints which data are not under our control).
- The taxonomy of the objects that are to be looked up. Exploiting the semantic structure of the Linked Data, the Query Engine is able to focus the search only on the objects that are relevant to the defined domain. Thus, our system pre-filters the triplet values to be checked against the keyword query.

With this information, the Query Engine is able to build a SPARQL query to be posed to the public endpoint. To perform the actual keyword search via SPARQL, the system uses the *regex* function to look for such keywords in the property values. In our example on the DBPedia, the following SPARQL query defines the structure of the queries posed to the DBPedia's SPARQL endpoint:

```
SELECT ?uri ?abstract ?web
WHERE ?uri dbo:abstract ?abstract
      ?uri foaf:page ?web
FILTER regex(?abstract, [keyword])
```

This query firstly matches all the resources that have *abstract* and *page* linked to it, and then, filters the results by looking for the keywords in the annotated property (in this case, *abstract*). A query like that posed to the whole dataset would be too time consuming. Thus, the system uses the taxonomy to limit the possible candidate resources by forcing them to be instances of any of the objects included in it. Querying a smaller set of resources will provide more accurate results according to the selected knowledge context. This limited search space will also impact the performance of the keyword search query, reducing the time required.

The properties that define the taxonomy depends on the modeling language that has been selected to express the ontology. In OWL, the main property is the *is_a* property (subclassOf), while in SKOS the taxonomy is modeled via the *broader* and *narrower* properties. So, depending on the modeling language, our system adopts one or another to build the constraints on the query. In our example, to limit the search scope to a SKOS category, our system adds the following constraint to the query:

```
WHERE ?subCategories skos:broader     [chosenCategory]
      ?uri           dcterms:subject  [chosenCategory]
      ?uri           dcterms:subject  ?subCategories
```

This constraint reduces the set of resources to be checked to only the ones that are in the domain search, which are a strict subset of the whole dataset. Without loss of generality, and if the taxonomy is quite big, the system can accept a category/concept of the domain search to provide a more focused search scope.

The result of the built query is a set of tuples $< URI, \{kwdSearchField_i\} >$ that have to be presented to the user. When the query results are retrieved, they are inserted on a local Lucene repository along with the keyword query that has led to them. We let Lucene index the results according to the retrieved values of the properties we marked in the ontology for the keyword search. Our system benefits from this step in two ways:

- On the one hand, it obtains a ranked set of resources that have been retrieved due to their semantic belonging to the search domain.
- On the other hand, the Lucene repository serves as a cache for further queries, alleviating the dependence on the processing resources of the public endpoints (which availability might be even compromised).

So, in the end, our system provides the user with a ranked set of semantically related resources, taking advantage of both semantic knowledge and information retrieval techniques. For example, for the input '*fish movement*', our system retrieves the following results: 1) tripedalism, 2) fish locomotion, 3) role of skin in locomotion, 4) aquatic locomotion, 5) dynamical system, ...; while performing the same search directly on Wikipedia, it returns: 1) fish migration, 2) lateral line, 3) Murray cod, 4) fishing lure, 5) Bear Island, ... Note the difference: our system focused on the domain '*Mechanics*', as we defined, and only returns resources within that defined domain, while Wikipedia always considers any domain in the search.

4.2 URI Refining Search

The result of the previous keyword search is a set of semantic resources, identified each one by an URI. Once the user selects a resource, the system performs a URI refinement to suggest further related results. This refinement is performed via the properties that have been annotated as relevant in the domain search definition.

Firstly, the Query Engine asks the underlying repository for the concept of the resource to know its defined properties. Then, it consults the ontology to

obtain the relevant properties[7] that are compatible with its definition. Once the system has the definition of the properties, it retrieves their values for the input resource. In the meantime, the system builds property compositions[8] that are suggested as possible queries to retrieve further results (they are not directly posed to the underlying system, but only on user's demand). All the semantic checkings of the domains and ranges of the involved properties are performed with the help of a Description Logics reasoner [3], to detect possible inconsistent queries (according to our domain ontology).

The results of this kind of search lead to more resources that can be navigated again and so on. This way, our system keeps on providing results that are related to the resource without leaving the search domain. Following the previous example, the user could select '*dynamical system*' out of the resources that the keyword search had retrieved. The results of the refinement will depend on how the domain ontology was defined. In this case, we define two properties for articles: *knownFor* and *subject*, that represent people related to the article and the categories where the article is included. Now, the refined results include the following people: Krystyna Kuperberg, Jean-Christophe Yoccoz, Yakov G. Sinai, Denis Blackmore, Bill Parry (mathematician), John Guckenheimer, ...; and categories: Systems, DynamicalSystems, and SystemsTheory.

5 Related Work

When it comes to accessing semantic data from a set of keywords, there are quite a lot different approaches, but most of them start with query building step which translates the input into an structured query [8, 14, 16, 7]. In [14], the input is matched to semantic entities by means of text indexes, and then a set of predefined templates is used to interpret the queries in the language SeRQL. However, in this system, the user has to be aware of the underlying data schema to be able to query as, at least, one of the keywords has to be matched to a class in the ontology that describes the underlying data. Moreover, the search domain and the properties to be used cannot be adapted as it can be done in our system.

Other relevant systems in the area of semantic search are SemSearchPro [16], Q2Semantic [10], SPARK [18], and QUICK [17]. These systems find all the paths that can be derived from a RDF graph, until a predefined depth, to generate the queries. In [8] they propose a similar approach to keyword interpretation but they introduce the context of the user's search (the knowledge about previous queries) to focus the whole search process. However, all of these approaches assume that the data sources are under their control and can pre-calculate complex graphs to perform the query translation and, finally, access the data. Moreover, as they work on the data level, they do not take into account the flexibility that using and adapting the describing ontology provides as we do.

[7] Note that the domain ontology is mapped to the underlying repository, so we assume that we have the information needed to perform the proper translation.

[8] The length of the path is restricted to avoid infinite loops; it has been defined as a system configuration parameter.

There are also some works in the area of databases to provide a keyword-based interface for databases, such as BANKS [1], DISCOVER [11] and DBXplore [2], which translate a set of keywords into SQL queries. However, as emphasized in [4], most of these works only rely on extensional knowledge obtained by applying IR-retrieval techniques, and so they do not consider the intensional knowledge (the structural knowledge). So, again, they have to have the data sources under their control, thus restricting the application in an open scenario such as the Linked Data one.

6 Conclusions and Future Work

In this paper, we have presented a system that enables a hybrid search approach based on keywords and guided by ontologies. Our system combines the ease of use of keyword search with the benefits of exploiting the structure of the underlying data in an efficient way. In particular, our system:

- Provides a keyword-based search guided and focused by the domain ontology, avoiding queries that would be too time-consuming. To do so, it uses the information of the taxonomy of the search domain. This process is highly configurable, as the search can be restricted to a set of specified properties via annotations.
- Exploits the definition of the objects in the domain to suggest further close-related results, refining the search within the domain. This is done with the help of a DL reasoner, that performs all the semantic checkings needed to avoid inconsistent queries.
- Can be built on third parties' Linked Data repositories without overloading them, while allowing to provide the user with different domain views. Our system manages the ontologies as views on the underlying data, decoupling them and processing the results in more flexible ways.

As future work, we are planning to include crossed-domains searches, that is, to include inter-domain relationships in the search, while keeping the adopted hybrid strategy. We also want to perform tests with different kinds of final users to measure the semantic accuracy of our prototype.

Acknowledgments

This work has been supported by the CICYT projects TIN2010-21387-C02-02 and TIN2011-24660. We want to thank Francisco J. Serón for his contributions during the design and development of the system.

References

1. B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China*. Morgan Kauffman, 2002.
2. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of 18th Intl. Conf. on Data Engineering (ICDE'02), USA*. IEEE, 2002.

3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Pastel-Scheneider. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2003.
4. S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo-Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'11), Greece*. ACM, 2011.
5. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
6. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009.
7. C. Bobed, R. Trillo, E. Mena, and S. Ilarri. From keywords to queries: Discovering the user's intended meaning. In *Proc. of 11th Intl. Conf. on Web Information System Engineering (WISE'10), China*. Springer, 2010.
8. H. Fu and K. Anyanwu. Effectively interpreting keyword queries on rdf databases with a rear view. In *Proc. of 10th Intl. Semantic Web Conference (ISWC'11), Germany*. Springer, 2011.
9. T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.
10. Q. L. Haofen Wang, Kang Zhang, D. T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*. Springer, 2008.
11. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China*. Morgan Kauffman, 2002.
12. E. Jimenez, B. Cuenca, U. Sattler, T. Schneider, and R. Berlanga. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*. Springer, 2008.
13. E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):377 – 393, 2010.
14. Y. Lei, V. S. Uren, and E. Motta. SemSearch: A search engine for the semantic web. In *Proc. of 15th Intl. Conf. on Knowledge Engineering and Knowledge Management (EKAW'06), Czech Republic*. Springer, 2006.
15. N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96 –101, jan.-feb. 2006.
16. T. Tran, D. M. Herzig, and G. Ladwig. Semsearchpro: Using semantics throughout the search process. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):349 – 364, 2011.
17. G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries–incremental query construction on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):166–176, 2009.
18. Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. SPARK: Adapting keyword query to semantic search. In *Proc. of 6th Intl. Semantic Web Conference (ISWC'07), South Korea*. Springer, 2007.

Bibliografía

[1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009. 1.1, 2.2.2

[2] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96 –101, jan.-feb. 2006. 1.1

[3] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):377 – 393, 2010. 3

[4] H. Fu and K. Anyanwu. Effectively interpreting keyword queries on rdf databases with a rear view. In *Proc. of 10th Intl. Semantic Web Conference (ISWC'11), Germany*. Springer, 2011. 3

[5] C. Bobed, R. Trillo, E. Mena, and S. Ilarri. From keywords to queries: Discovering the user's intended meaning. In *Proc. of 11th Intl. Conf. on Web Information System Engineering (WISE'10), China*. Springer, 2010. 3

[6] T. Tran, D. M. Herzig, and G. Ladwig. Semsearchpro: Using semantics throughout the search process. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):349 – 364, 2011. 3

[7] IEEE Recommended Practice for Software Requirements Specifications, 4.1. *ANSI/IEEE* Std. 830-1998. 4.1

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Pastel-Scheneider. The Description Logic Handbook. Theory, Implementation and Applications. *Cambridge University Press*, 2003. 2.1.2, 4.2.1

[9] Shearer R, Motik B, Horrocks I. HermiT: a highly-efficient OWL reasoner. *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008): 26-27 October 2008; Karlsruhe, Germany.* 2.3.2, 4.2.1, B.4.3

[10] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Technical Report KSL 92-71. Knowledge Acquisition*, 1993. 2.1.1

[11] MJL Lapuente. Tesis Doctoral. Hipertexto: El nuevo concepto de documento en la cultura de la imagen. Dpto. de Biblioteconomía y Documentación. Universidad Complutense de Madrid. 2.1.1

[12] Resource Description Framework (RDF). http://www.w3.org/RDF/. 2.1.3, 2.1.3

[13] OWL 2 Web Ontology Language Document Overview. http://www.w3.org/TR/owl2-overview/. 2.1.2, 2.3.2, B.4.3

[14] E. Jimenez, B. Cuenca, U. Sattler, T. Schneider, and R. Berlanga. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*. Springer, 2008. 4.2.2, C.1

[15] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009. 2.2.2

[16] C. Bobed, G. Esteban, and E. Mena, Ontology-driven Keyword-based Search on Linked Data, *Proc. of the 16th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 2012), San Sebastián (Spain)*, IOS Press, volume To appear, September 2012. 1, E

[17] R. Studer, V. R. Benjamins, D. Fensel. Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering 25 (1998)*. Pages: 161-197. 2.1.1

[18] RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/. 2.1.2

[19] XML Schema 1.1. http://www.w3.org/TR/xmlschema11-1/. http://www.w3.org/TR/xmlschema11-2/. 2.1.2

[20] SKOS, Simple Knowledge Organization System. http://www.w3.org/2004/02/skos/. 2.1.4, B.1.2

[21] Matthew Horridge, Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web Journal 2(1), Special Issue on Semantic Web Tools and Systems, pp. 11-21*, 2011. 2.3.1, B.4.2

[22] SKOS API, http://skosapi.sourceforge.net. 2.3.1, B.4.2

[23] SPARQL, http://www.w3.org/TR/rdf-sparql-query/. 2.2.1, B.3

[24] Y. Lei, V. S. Uren, and E. Motta. SemSearch: A search engine for the semantic web. In *Proc. of 15th Intl. Conf. on Knowledge Engineering and Knowledge Management (EKAW'06), Czech Republic*. Springer, 2006. 3

[25] Q. L. Haofen Wang, Kang Zhang, D. T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*. Springer, 2008. 3

[26] *Knowledge-based and Intelligent Engineering Systems*, http://kes2012.kesinternational.org/. E

[27] Proyecto Alpha III GAVIOTA, http://alfagaviota.info/. 1

[28] E.Sirin, B. Parsia, B. C. Grau , A. Kalyanpur , Y. Katz . Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web 5(2): 51–53*, Junio 2007. 2.3.2, 4.2.1

[29] Lucene search library, http://lucene.apache.org/core/. 2.3.3, B.4.5

[30] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China.* Morgan Kauffman, 2002. 3

[31] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China.* Morgan Kauffman, 2002. 3

[32] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of 18th Intl. Conf. on Data Engineering (ICDE'02), USA*. IEEE, 2002. 3

[33] ProSÉ, http://krono.act.uji.es/people/Ernesto/safety-ontology-reuse/.

B.4.1, C.1