CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Implementación de un recolector de RDF

Autor: Álvaro Moreno García

Director: Raúl García Castro

MADRID, JULIO 2015

Contenido

RESUMEN

La web semántica aporta un mayor conocimiento a los datos para que estos puedan ser procesados por las máquinas. Esto es posible gracias a estándares como por ejemplo *Resource Framework Description* (RDF). Éste, aporta un marco para que la información pueda ser representada de una manera más comprensible para las maquinas.

Muchas veces la información no se encuentra codificada en RDF pero igualmente es interesante aprovecharse de sus características. Es por ello que surge la necesidad de crear una herramienta que permita consultas entre distintas fuentes de datos apoyándose en el estándar RDF independientemente del formato de origen de los datos.

De esta manera se conseguirá realizar consultas entre las diversas fuentes, las cuales, sin la unificación en un estándar semántico, serían mucho más difíciles de conseguir.

ABSTRACT

The Semantic Web provides a new knowledge framework to data, therefore computers would become capable of analyzing the data. Standards, as Resource Framework Description (RDF), help to achieve it. RDF promotes the easier way for computers on how to describe data.

Sometimes data are coded in a different way from RDF, nevertheless it would also be interesting to examine it. Accordingly, the need to create new software emerges. The software, based on RDF, would be able to combine information from different sources regardless of its format.

Consequently, several sources, whatever their original formats were, could be queried on an easier way since a common semantic standard is available.

1 INTRODUCCIÓN Y OBJETIVOS

La información como fuente de conocimiento puede provocar mejoras en las tomas de decisión y de esta manera proporcionar una perspectiva más compleja de la realidad. En la actualidad se maneja gran cantidad de datos; muchas veces éstos no pueden ser fácilmente interpretados por los sistemas automáticamente. Por lo tanto el conocimiento se diluye y no permite obtener el máximo aprovechamiento.

Si se analizase la web y sus lenguajes de estructuración, lo primero que pudiera venir a la mente sería el estándar HTML. Sin embargo, se puede observar cómo HTML establece una clara distinción entre el contenido y la representación del mismo. La representación es la parte de HTML que se centra para que las máquinas entiendan cómo estructurar la información al usuario. Todos los creadores de contenidos se preocupan de cómo los diferentes sistemas van a mostrar su contenido e incluso lenguajes específicos como CSS están muy extendidos.

Cuando se escruta lo que respecta al contenido no existe la misma preocupación y solo se enfoca principalmente hacia su consumo por el ser humano, es el llamado texto plano. En el contenido se concentra la información más valiosa y una correcta etiquetación del mismo permitiría que este pudiera ser mejor interpretado por las máquinas.

Para una mejor interpretación de la información por parte de los sistemas informáticos se desarrolló lo que se conoce como la web semántica. Esto se explica por una mayor preocupación por la estructuración del contenido. Desde el consorcio de la *World Wide Web* (W3C), y con el apoyo del creador de la Web y percusor de la web semántica, *Tim Berners-Lee*, se han asentado las bases y de esa estandarización nace el marco de descripción de recursos, más conocido por su denominación anglosajona, *Resource Description Framework* (RDF). Este lenguaje define como debe ser definida y agrupada la información en la web para una mejor interpretación.

La web semántica tiene más lenguajes y estándares para apoyar el uso de RDF como el lenguaje de ontologías para la web, *Web Ontology Language* (OWL) que permite establecer un conocimiento adicional a los datos aportados en documentos RDF. O el lenguaje de consultas SPARQL que permite extraer la información de los modelos RDF.

Todos estos esfuerzos no han sido suficientes para que la mayoría de los creadores codifiquen la información de una manera unificada y comprensible. Si bien es cierto que en los últimos tiempos el acceso a los datos por medio de distintas interfaces de programación de aplicaciones (API) ha provocado que el acceso a la información sea más sencillo y comprensible. Sin embargo el uso de RDF como estándar no está tan extendido como para poder hablar de una web semántica potente.

Para intentar unificar la información de diversas fuentes, este trabajo de fin de grado (TFG) desarrollara una herramienta software que se basara en los principios del estándar RDF y todo el entorno que establece el W3C y los diversos colaboradores de la web semántica.

El software deberá ser lo suficientemente flexible y modular para recoger información desde diversas fuentes y formatos de datos y unificarlas a formato RDF. También se contempla la opción de que ya esté en formato RDF.

El software se asemejará a los denominados Web *crawlers*. Esto quiere decir que recorra una fuente de información, como es el caso de la Web. Deberá diseñarse con las estructuras necesarias para que se pueda ampliar lo más rápida y sencillamente posible. Esto quiere decir que debe presentar una configuración sencilla para extraer la información de diversas fuentes haciendo los menores cambios en el programa. Estas fuentes pueden estar disponibles tanto online como offline.

Como prueba de la eficacia de realizar búsquedas en varias fuentes de información distribuidas, se procederá a su análisis en el que se arrojaran los resultados de las consultas realizadas por el usuario sobre la información disponible que de otra manera no hubiera sido tan fácil de procesar.

2 ESTADO DE LA CUESTIÓN

2.1 Introducción

Este apartado tiene como propósito principal dar a conocer las principales características de los conceptos que se van a tratar en el TFG. Aunque no es el objetivo principal del trabajo el hacer una profunda revisión de la web semántica. Para una mejor comprensión del funcionamiento de la herramienta se describirán los principales estándares recogidos por el W3C. Algunos de los principales usados por el software son OWL, SPARQL y en particular el modelo RDF.

2.2 Resource Description Framework (RDF)

RDF es una recomendación del W3C que establece una serie de normas para representar la información dentro de un documento estableciendo un conjunto de etiquetas, propiedades y relaciones entre la información que permite que esta sea comprendida por los sistemas de información [RDF].

Los datos se modelan estableciendo la instantánea de un determinado momento. Hay información que puede variar a lo largo de tiempo por lo que un correcto modelado puede solucionar los problemas que puedan aparecer para resolver inconsistencias temporales.

El modelado de datos en RDF es una actividad compleja, se pretende clasificar en pequeñas piezas de información los conocimientos humanos. Hay que realizar un esfuerzo en el diseño previo para obtener los beneficios posteriores de los datos estructurados.

Una de sus principales características viene definida por la manera en la que está definido, que permite la posibilidad de comparar y realizar búsquedas entre distintos modelos distribuidos que en un principio no fueron diseñados para estar conectados.

2.2.1 Nodos RDF

La propiedad principal de RDF es el modelado de datos en forma de tripletas de información. Estas contienen la información en sus elementos o nodos. El predicado establece la relación existente entre los nodos sujeto y objeto. Este modelo puede verse como un grafo dirigido etiquetado.

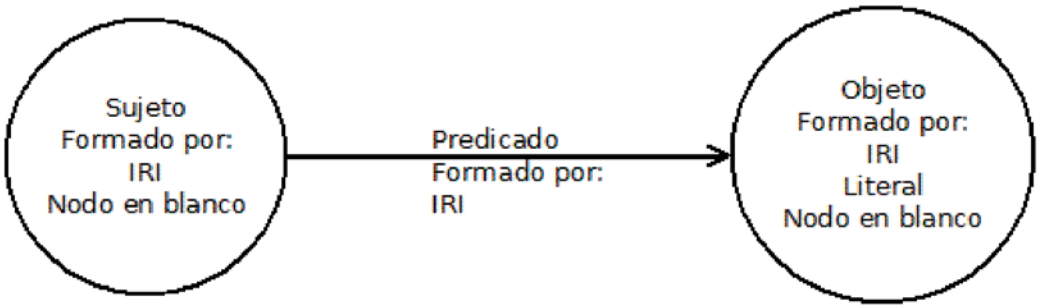

Ilustración 1: Tripleta de información en RDF

Estos nodos pueden contener la información de diferentes formas:

- Identificador de recursos internacionales, *internationalized resource identifier* (IRI).
- Literales.
- Nodos en blanco.

Lo que hace interesante al modelo, por su propiedad de identificador único y evitar confusión, son las IRI. Una IRI puede ser descrita como algo del mundo, es decir, un recurso o entidad. Esta puede contener cosas físicas, documentos, conceptos abstractos, números o cadenas de texto.

Una IRI debe ser diseñada con un alcance global y único. Por convención social el dueño es el que la define y mantiene. Se puede establecer una especificación en un documento. Es importante señalar que las IRI no tienen por qué tener una relación directa con una página web aunque se representen aparentemente en el mismo formato. Las IRI pueden ser abreviadas en un documento RDF usando un espacio de nombres, o *namespace*, aunque hay que tener presente que esto solo se hace con carácter local dentro del documento RDF, esto también es lo que se conoce como vocabularios.

Un literal puede contener números, texto y fechas. Además pueden estar asociados con una IRI que representa un tipo de dato o metadato, por ejemplo XSD *datatypes* [RDF]. Esto se conoce como etiquetado del literal. Este etiquetado puede ser por ejemplo el idioma o el tipo de datos. El asociarla con un tipo permite que la información contenida en un literal pueda ser definida con una mayor riqueza semántica. De tal modo que podemos comparar dos literales siempre y cuando estén etiquetados bajo el mismo tipo de dato.

Algunos nodos no identifican a ningún recurso o literal. Estos son conocidos como nodos en blanco. Por ejemplo, si de un libro no se conoce el autor, el nodo sujeto podría ser un nodo en blanco en la tripleta autor escribe libro. En algunos modelos interesará sustituir

estos nodos en blanco por IRI para normalizar el modelo. Estas IRI denotaran el recurso nodo en blanco pero de esta manera todos nuestros nodos serán IRI.

La relación que establece el modelo determina que el predicado, la propiedad entre el sujeto y el objeto, siempre va a estar representado por una IRI. Esto es debido a que la propiedad debe estar definida a un nivel superior, no puede ser un simple literal. De esta manera se puede uniformar las relaciones de todos los nodos y evitar la confusión de tener dos propiedades iguales identificadas de manera diferente.

2.2.2 Vocabulario

Los vocabularios permiten acceder a una colección de diversos recursos acortando las IRI dentro de un mismo espacio de nombres (*namespace*). Los vocabularios pueden ser bien propios del usuario o públicos para el uso por parte de la comunidad.

Uno de los principales usos de los vocabularios públicos son las definiciones de propiedades que se encuentran para establecer en los predicados y los nodos. Estas pueden resultar útiles para utilizarlas dentro de nuestro modelo. El reutilizar propiedades está recomendado. Provoca que distintos documentos RDF que compartan las mismas propiedades, estas puedan ser analizadas entre ellas. También permite rebajar la carga a la hora de diseñar el conjunto de propiedades RDF, pues ya han sido diseñadas por otros. Por el contrario pueden existir propiedades definidas muy similares y habrá que tomar la decisión de cual usar, si bien no es un problema, se pueden establecer que dos propiedades son similares.

Los vocabularios no solo definen propiedades, también establecen los tipos de datos que estamos usando, por ejemplo el lenguaje que estamos usando en un literal. Debemos ver los vocabularios como una herramienta para completar nuestros modelos con el conocimiento que ya ha sido desarrollado por la comunidad.

Algunos de los vocabularios más importantes son los que aparecen en la Tabla 1 [DCAT]. El uso de prefijos permite que sean más fáciles de manejar por los desarrolladores que el espacio de nombres completo.

Prefijo	Namespace
Dcat	http://www.w3.org/ns/dcat#
Dct	http://purl.org/dc/terms/
dctype	http://purl.org/dc/dcmitype/
Foaf	http://xmlns.com/foaf/0.1/
Rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
Rdfs	http://www.w3.org/2000/01/rdf-schema#
Skos	http://www.w3.org/2004/02/skos/core#
Vcard	http://www.w3.org/2006/vcard/ns#
Xsd	http://www.w3.org/2001/XMLSchema#

Tabla 1 Ejemplo de distintos vocabularios

Es importante señalar que esto solo es un ejemplo de vocabularios generales. También existen vocabularios definidos para usos muy específicos que no son objeto de este TFG. Una de las fuentes principales de referencia donde poder encontrar vocabularios puede ser la DBpedia.

2.2.3 Representación plana de un grafo RDF

Aunque la representación de forma gráfica del modelo RDF es única, en forma de tripletas, cuando el grafo es codificado en un documento puede presentar una serie de diferencias a nivel sintáctico y en cuanto a su etiquetado. Los modelos de representación se definen intentando tener distintas propiedades, como el ser más fácil de parsear por la máquina, entendibles por los desarrolladores, convertibles desde otros formatos. Los distintos lenguajes de representación pueden ser transformados a otro, aunque algunos lenguajes más ricos semánticamente pueden perder información al convertirlos a otros más sencillos. Esto sería el caso de *TriG* que extiende a *Turtle* [TRIG].

En la mayoría de formatos de representación se definen al principio del documento una serie de prefijos para acortar las direcciones de los espacios de nombres. La diferencia radica principalmente en la sintaxis utilizada. Posteriormente se representa el conjunto de relaciones entre la IRI sujeto y todas las IRI de los predicados y objetos que contiene el sujeto [TURTLE]. Esto es el cuerpo del modelo RDF y es donde podemos encontrar mayor diferencia entre los diversos estándares. En *TriG*, por ejemplo, permite darle un nombre simbólico a cada gráfico. Esto tiene como objetivo primordial ayudar a su visualización y también permite realizar otros tipos de búsquedas.

Otro modelo es JSON-LD, de su nombre se puede deducir que se basa en el formato JSON, este no tiene un origen en el modelo RDF. Para la primer parte, el definir los prefijos

de los vocabularios, se adhiere una sintaxis extra apoyándose en la ya existente en el estándar JSON. Se añade al principio una etiqueta @context donde además de definirse los prefijos se añade la definición de toda la lógica de predicado objeto que acompaña al modelo RDF utilizando las etiquetas que tendría un documento JSON [JSON-LD].

A continuación se muestra un ejemplo de como un modelo RDF puede ser representado en diferentes formatos. El ejemplo extraído es una simplificación de un recurso de la página de la DBpedia. Primero se muestra la representación en forma gráfica, posteriormente en formato JSON-LD y *Turtle*.

Vocabulario:

dbpedia: http://dbpedia.org/resource/

dbpedia-owl: http://dbpedia.org/ontology/

xsd: http://www.w3.org/2001/XMLSchema#/

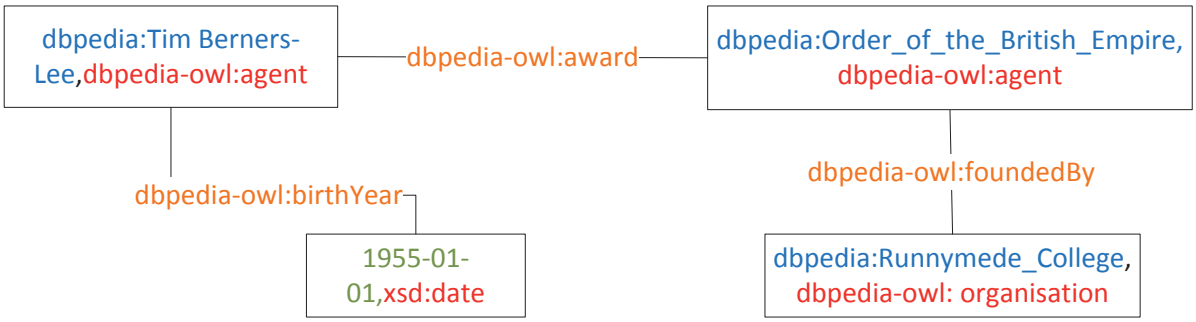

Ilustración 2: Ejemplo gráfico de RDF

```
{
“@Context”:
 {
 “dbepedia”:                <http://dbpedia.org/resource/>,
 “dbpedia-owl:              < http://dbpedia.org/ontology/>,
 “xsd”:                     <http://www.w3.org/2001/XMLSchema#>,
 “dbpedia-owl:birthYear”:
    {
       “@id”:               < http://dbpedia.org/ontology/birthYear >,
       “@type”:             “xsd:date”
    },
  },
“@id”:                      “dbpedia: Tim Berners-Lee”,
“@type”:                    “dbpedia-owl:agent”,
“dbpedia-owl:birthYear “:   " 1955-01-01",
“dbpedia-owl:award “ : {
    “@id”:                  “dbpedia:Order_of_the_British_Empire”,
    “@type”:                “dbpedia-owl:agent”
    “dbpedia-owl:foundedBy”: {
                            “@id”:          “dbepedia:Runnymede_College”,
                            “@type”:        “dbpedia-owl:organization”
                              },
                        },
}
```

Tabla 2 Ejemplo JSON-LD

```
@base                       <http://dbpedia.org/resource/>.
@prefix dbpedia-owl:        < http://dbpedia.org/ontology/>.
@prefix xsd:                <http://www.w3.org/2001/XMLSchema#>.

: Tim Berners-Lee           A                   dbpedia-owl:agent;
                            dbpedia-owl:birthYear   " 1955-01-01"^^xsd:date;
                            dbpedia-owl:award   :Order_of_the_British_Empire.

:Order_of_the_British_Empire; A                 dbpedia-owl:agent;
                            dbpedia-owl:foundedBy   :Runnymede_College.

:Runnymede_College;         A                   dbpedia-owl:organization.
```

Tabla 3 Ejemplo Turtle

Los modelos de representación anteriormente mostrados pretenden solamente dar a conocer cómo un mismo grafo puede ser codificado de distintas maneras. Un mismo formato de representación no tiene una manera única de escritura, se podían haber representado utilizando la misma notación de manera equivalente. Ambos modelos tienen mayor sintaxis que la mostrada. Los colores mostrados no tienen ningún significado, simplemente sirven para identificar distintos conceptos.

Esta memoria no pretende dar a conocer todos los distintos formatos de representación existentes. La mayoría de los estándares están publicados con todos los detalles en la página web del W3C.

2.3 Ontology Web Language (OWL)

OWL surgió de la necesidad de enriquecer semánticamente los modelos RDF. Aunque pueda parecer que simplemente extiende a RDF Schema, primer enfoque para definir ontologías en RDF y con raíces a su vez en XML Schema, estos tienen funciones más limitadas y diferentes [RDFS]. A diferencia de los denominados *schemas*, OWL no tiene como objetivo validar los tipos de datos de un modelo RDF sino aportar por medio de un lenguaje más elaborado la posibilidad de expresar las propiedades semánticas de un modelo [OWL].

El lenguaje declarativo para representar ontologías, que sirve de apoyo a RDF, es OWL. Aporta más consistencia a la información representada para que pueda ser procesada por una aplicación. Las ontologías son dependientes del domino, establecen un conocimiento más profundo de las cosas y sus relaciones. Recogen pequeñas piezas de conocimiento que en conjunto forman la ontología. La manera de representar el conocimiento es más difícil que sea entendida por un humano que por una máquina. Esto es debido a que mucho del conocimiento que hay que definir está interiorizado por el desarrollador.

Los elementos básicos que forman el lenguaje OWL son los axiomas, las entidades del mundo real y las expresiones que combinan estas.

La ontología permite definir distintas propiedades de clase. Se deben entender las clases como una categorización. De esta manera podemos afirmar desde nivel de recurso que María es de clase mujer o la jerarquía entre clases, mujer es persona. Una clase se puede entender como una entidad que engloba a recursos concretos, con los anteriores axiomas podemos deducir que María es de clase persona. También podemos establecer clases equivalentes. Aunque muchas veces se puedan entender que las clases son por

defecto disjuntas es una propiedad que puede resultar útil puesto que los sistemas no tienen ese conocimiento.

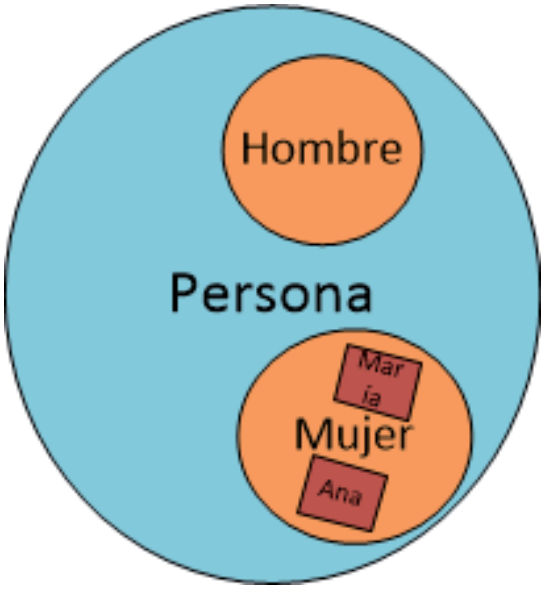

Ilustración 3: Ontología de clases

Otra de las propiedades que permite definir OWL son las relativas a los denominados objetos, estas describen individuales, por ejemplo relaciones en una familia, como es el hecho de que una persona determinada tenga una mujer determinada. El orden es importante, no es lo mismo que la mujer de Juan es María, que al revés. Las propiedades deben ser definidas intentando evitar la confusión.

Las jerarquías de objetos tienen diversas propiedades. Establecen al igual que con las clases diversos rangos y además pueden proporcionar restricciones de rango y dominio. Por ejemplo que una propiedad es sub-propiedad de otra.

Si semánticamente no hay diferencia entre dos individuos (ejemplo empresas del mismo sector) puede interesar definir que son diferentes. Sin embargo otras veces interesa de manera explícita decir que dos objetos son iguales.

El tipo de datos también puede ser definido, así se puede establecer que la edad sea un entero, rangos de enteros o crear nuestras propias definición de datos o que una propiedad solo es aplicable al dominio de una clase.

Hay que tener cuidado con los dominios que se definen porque si algo que no es persona le definimos una edad se convierte en persona para el modelo si se ha definido

edad como propiedad de persona. También es interesante establecer relaciones avanzadas entre clases como sus interacciones y las diferentes o restricciones de las propiedades.

OWL también permite establecer otros tipos de restricciones más avanzadas como que todos los individuos deben tener una clave que puede ser el número de DNI.

La diferente manera de combinar propiedades puede confundir a un principiante. El estándar permite una potente sintaxis para poder establecer todas las diferentes combinaciones e incluso combinar distintas ontologías. Para conocer la sintaxis en detalle lo mejor es consultar la documentación que proporciona el W3C.

2.4 SPARQL

Para extraer información de un modelo RDF podemos apoyarnos en un lenguaje de consultas como es SPARQL. Este lenguaje, también promovido por el W3C, tiene como principal característica el permite combinar diversas fuentes de datos permitiendo búsquedas más complejas.

SPARQL, al igual que otros lenguajes estructurados de consultas, permite por medio de una serie de palabras clave y condiciones hacer búsquedas de información. Algunas de estas sentencias son: FROM permite seleccionar los modelos RDF desde donde se desea extraer la información; SELECT identifica la información que se desea buscar; WHERE establece el patrón de búsqueda; por último, los modificadores para los resultados de la salida FILTER, ORDER BY, LIMIT [SPARQL].

Este estándar también permite hacer consultas directamente contra *endpoints*. Los *endpoints* son una manera de acceder a RDF definidos que resultan en consultas a través de HTTP devolviendo los resultados en diferentes formatos. Esto permite una mayor accesibilidad y abstracción a las fuentes de datos y establece un entorno de información más distribuida.

En la caso de la DBpedia, existe un *endpoint*, implementado con el sistema Virtuoso, disponible públicamente. Se pueden encontrar multitud de herramientas que permiten realizar consultas sobre la DBpedia usando SPARQL. A continuación se muestra un sencillo ejemplo para resumir los conceptos más importantes.

La siguiente búsqueda ha sido realizada en http://dbpedia.org/sparql. En lenguaje natural la búsqueda se traduce como aquellas personas premiadas por la *Royal Society* nacidas a partir de 1955. Solo mostrara 5 resultados ordenados ascendentemente [SPARQL-EX].

```
PREFIX    dbpedia-owl:<http://dbpedia.org/ontology/>
PREFIX    dbo:<http://dbpedia.org/resource/>
SELECT    ?person WHERE {
                    ?person dbpedia-owl:award dbo:Royal_Society;
                    dbpedia-owl:birthYear ?birthdate
          FILTER (?birthdate > "1955-01-01"^^xsd:date)
          }
          ORDER BY ASC(?person)
          LIMIT 5
```

Tabla 4 Búsqueda en SPARQL

person
http://dbpedia.org/resource/Andre_Geim
http://dbpedia.org/resource/Ara_Darzi,_Baron_Darzi_of_Denham
http://dbpedia.org/resource/Ben_Green_(mathematician)
http://dbpedia.org/resource/Bonnie_Bassler
http://dbpedia.org/resource/Christopher_Dye

Tabla 5 Resultados de la búsqueda en SPARQL

2.5 Fuentes de datos

Cuando se buscan fuentes de información varios problemas surgen: la codificación de los datos y las licencias de uso que tienen los mismos. Como en cierta parte pueda parecer lógico muchos datos se distribuyen bajo licencias restrictivas que no permiten su uso, esto dificulta el conexionado de datos sobre todo en herramientas con enfoque comercial.

En lo que respecta a como están codificada la información encontramos que muy pocas permiten acceder a los datos en formato RDF, como fuente principal se puede citar a DBpedia que es accesible por medio de un *endpoint* por SPARQL. Existe un listado con algunos de los *endpoints* activos; estos pueden ser consultados en la wiki del W3C.

Muchas otras fuentes sí permiten acceder sus datos de manera codificada en formatos como XML o JSON, el principal problema es que estos no están codificados en formato RDF y se tienen que hacer la conversión. Por último se encuentran las fuentes que no establecen ninguna facilidad en sus datos por lo que hay que diseñar desde el sistema de extracción a su conversión a RDF, esto produce un mayor esfuerzo y una fuente de errores mayor pues pequeños cambios en el texto plano pueden provocar el cambio en la manera que se transforma.

2.5.1 API Flickr

El servicio online de alojamiento de fotos www.flickr.com tiene una serie de información que puede ser interesante para su procesamiento semántico. Algunos de los datos más interesantes que se recogen en el servicio pueden verse en la Ilustración 4.

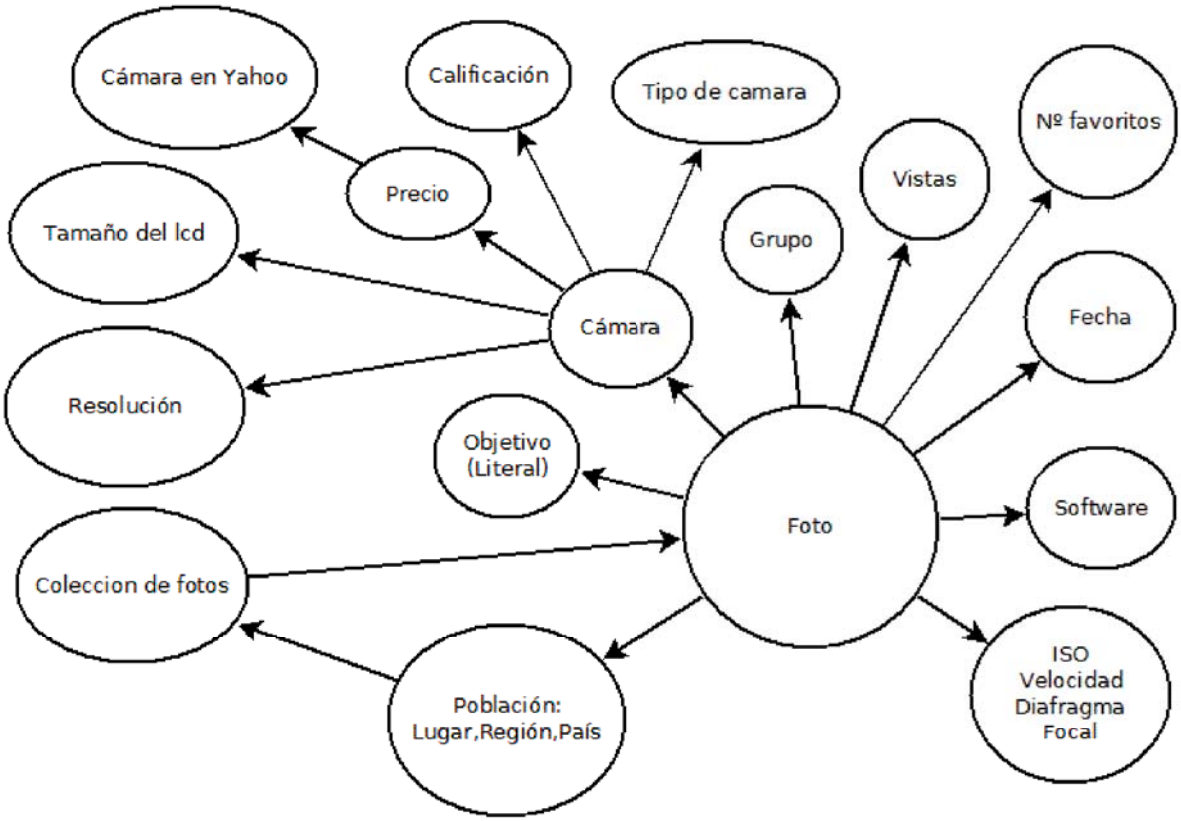

Ilustración 4 Conocimiento en flickr.com

Para acceder a estos datos se proporciona una API REST para los desarrolladores de aplicaciones con la posibilidad de invocar una serie de métodos. Estos son tanto para subir nuevos datos al servicio como para consultar los datos alojados.

En el caso de este TFG los más interesantes son los que permiten obtener información, en particular las relativas a las fotos y su geolocalización. Es por ello que a continuación se muestra un resumen de los métodos que pueden ser más interesantes para la implementación de un servicio que los consultase. Las tablas muestran un resumen de los parámetros que pueden ser pasados; si se quiere tener un mayor detalle Flickr proporciona una documentación pública que puede ser consultada. Los resultados se pueden subir u

obtener en diversos formatos como XML o JSON. Estos se indican en la URL que invoca al método requerido.

URI	**https://api.flickr.com/services/rest/?method=flickr.photos.getInfo**		
Método	GET		
PRE	El usuario debe tener permisos para consultar la photo_id. No requiere autenticación		
Cadena de Consulta	Obligatorio	api_key=	La clave para la API de la aplicación
		photo_id=	El id de la foto a consultar
	Opcional	secret=	Seguridad para firmar
		format=	Formato de los resultados de los devueltos
		Nojsoncallback=	Json sin procesar si valor es 1
Devuelve	200 OK + (respuesta según format, XML por defecto) Principales errores: 1: Foto no encontrada 100: API Key invalida 105: Servicio no disponible 111: Formato no encontrado 112: Method no encontrado 116: URL incorrecta		

Tabla 6 Servicio getInfo

URI	**https://api.flickr.com/services/rest/?method=flickr.photos.getExif**		
Método	GET		
PRE	El usuario debe tener permisos para consultar la photo_id. No requiere autenticación		
Cadena de Consulta	Obligatorio	api_key=	La clave para la API de la aplicación
		photo_id=	El id de la foto a consultar
	Opcional	secret=	Seguridad para firmar
Devuelve	200 OK + (respuesta según format, XML por defecto) Principales errores: 1: Foto no encontrada 100: API Key no válida 105: Servicio no disponible 111: Formato no encontrado 112: Method no encontrado 116: URL incorrecta		

Tabla 7 Servicio getExif

URI	**https://api.flickr.com/services/rest/?method=flickr.photos.search**		
Método	GET		
PRE	El usuario debe estar autorizado o autenticado para ver las fotos según el nivel de permisos.		
Cadena de Consulta	Obligatorio	api_key=	La clave para la API de la aplicación
	Opcional	place_id=	Un place_id válido de la API.
Devuelve	200 OK + (respuesta según format, XML por defecto) Principales errores: 1: Demasiadas etiquetas en la consulta 3: Búsqueda sin parámetros. 10: La función de buscar no está disponible momentáneamente. 18: Argumentos no lógicos 100: API Key no válida 105: Servicio no disponible 111: Formato no encontrado 112: Método no encontrado 116: URL incorrecta		

Tabla 8 Servicio photos.search

URI	**https://api.flickr.com/services/rest/?method=flickr.places.findByLatLon**		
Método	GET		
Cadena de Consulta	Obligatorio	api_key=	La clave para la API de la aplicación
		lat=	Latitud
		lon=	Longitud
	Opcional	accuracy =	Precisión. Rango 1-16. Por defecto 16
Devuelve	200 OK + (respuesta según format, XML por defecto)		
	Principales errores: 1: Falta argumento obligatorio 2: Latitud no válida 3: Longitud no válida 4: Precisión no válida. 100: API Key no válida 105: Servicio no disponible 111: Formato no encontrado 112: Method no encontrado 116: URL incorrecta		

Tabla 9 Servicio findByLatLon

2.6 Librerías

Para facilitar el desarrollo del software, se pueden usar algunas herramientas de apoyo que ya han sido desarrolladas por la comunidad y que tienen las licencias adecuadas para poder ser usadas en el TFG. Las herramientas han sido elegidas por el conocimiento previo o reputación No se han seguido ningún criterio de rendimiento ni seguridad para ser elegidas.

2.6.1 Jena

Jena es un *framework* para el lenguaje Java desarrollado por HP, actualmente mantenida por la fundación Apache. Proporciona un marco de apoyo para distintos estándares definidos por el W3C como RDF, OWL y SPARQL, todos ellos usados en este TFG. La librería se encuentra bajo licencia Apache en su versión 2.0.

En este apartado se desarrollarán las principales funcionalidades que se usan en la herramienta. No pretende ser un repaso en profundidad de la API y los métodos disponibles.

2.6.2 Jena RDF

El marco de apoyo al estándar RDF es aportado a través de lo que denomina modelos. Para crear un modelo, *model*, existe la posibilidad de utilizar una factoría, *ModelFactory*. De esta manera se puede crear un modelo, por defecto, utilizando el método *createDefaultModel* de la citada factoría.

Algunos de los elementos principales que gestiona Jena son recursos, propiedades y literales se proporcionan los tipos *Resource*, *Property* y *Literal*. La creación de estos es gestionada a partir de un modelo, es decir, para crear un recurso se debe invocar al método *createResource* de un objeto *model* con la URI, *string* y/o un tipo de recurso asociado. Al igual para crear un literal, *createliteral*, asociando una cadena y/u opcionalmente un idioma o para una propiedad, *createproperty*, con la URI de ésta [JENA].

Las propiedades se gestionan como se ha explicado anteriormente pero es importante señalar que Jena incorpora en su API algunos de los vocabularios más utilizados por defecto. Algunos de estos incluyen propiedades que pueden ser usadas en los modelos que se están creando. Esta manera de tratar a las propiedades se puede tomar como referencia para crear un conjunto de propiedades propias del modelo, y puede ayudar a un mejor diseño.

Una de las maneras de añadir las tripletas de información al modelo es mediante el uso del método *add*. De este modo relacionamos los recursos que hemos creado por medio de propiedades, bien con otros recursos o con literales.

Jena permite representar los modelos en diferentes estándares de representación; tan solo hay que indicárselo al método que realiza la escritura de los datos. Hay que tener en cuenta que algunos estándares como *Trig* necesitan información adicional que habría que incorporar al modelo.

2.6.3 Jena SPARQL

Para realizar consultas a los modelos a través de la API Jena se puede utilizar el estándar SPARQL, explicado en el apartado estado de la cuestión. Por lo tanto lo primero que hay que crear es una cadena de caracteres con los parámetros de la consulta. Los modelos en los que se pueden hacer las búsquedas pueden ser varios. En este TFG nos centramos en los denominados *endpoints*, *service* según la denominación de la API, y los modelos propios gestionados con la propia API Jena.

Para invocar una consulta se debe proporcionar por lo tanto el modelo y la cadena de parámetros de búsqueda. Invocando a la *QueryExecutionFactory* se puede obtener el conjunto de resultados que después se puede iterar para tratar los recursos obtenidos individualmente.

2.6.4 GSON

GSON proporciona una API para Java. Fue creado por Google y liberado a la comunidad bajo licencia Apache en su versión 2.0. Proporciona los métodos necesarios para la conversión de documentos JSON en objetos Java.

La primera consideración para usar GSON es que se debe construir las clases que contienen a los objetos que se considera en el documento JSON. Para la transformación no es necesario reconstruir todos los objetos que contiene un documento.

Posteriormente la API se crea por medio del constructor de GSON, diferentes métodos pueden ser invocados. La conversión es bidireccional. Por ejemplo, en el caso de JSON a objetos, funciona creando la instancia del objeto y pasándolo como atributo al método convertidor de GSON [GSON].

3 REQUISITOS

3.1 Introducción

El presente capitulo tiene el objetivo de servir de guía para la construcción del software recolector de RDF. Se especificaran los requisitos principales del software de manera informal, teniendo en cuenta que se contempla una amplia visión de la herramienta. Es el punto de partida del software que se desarrollará.

El software parte de la necesidad de crear un software lo suficientemente flexible y modular para recoger información de diversas fuentes de datos y convertirla al formato RDF para su posterior análisis en el que se arrojaran los resultados de las consultas realizadas por el usuario.

3.2 Búsqueda de la información

El tratamiento de la información es la parte fundamental de la herramienta. El usuario podrá realizar una serie de búsquedas entre distintas fuentes de información predefinidas por el mismo. Estas consultas tienen como objetivo explotar las principales características del estándar RDF aunque se podrán realizar todas aquellas que hayan sido configuradas por el usuario siempre que la información lo permita.

3.3 Recolección de la información

Las fuentes de información deberán ser proporcionadas por el usuario del software. Se deberán facilitar tanto la configuración como el uso de herramientas auxiliares para recopilar la información. Es importante destacar dos conceptos distintos, la fuente y la información. Es posible que de una fuente de información solo sea interesante recopilar una determinada información. Por lo tanto el software deberá ofrecer la posibilidad de seleccionar la información a coleccionar dentro de una fuente.

Esta información puede estar codificada tanto en formato RDF como en cualquier otro. Por lo tanto el software deberá proveer las características necesarias para que independientemente de la fuente y el tipo de información, esta pueda ser procesada.

La fuente de información se podrá encontrar tanto online como offline. Cuando la fuente sea online se realizará la conexión con una determinada web para extraer la información. Las posibilidades que brinda la web pueden ser amplias. La información puede ser recuperada por medio de una API, proporcionada por la fuente de datos, o interpretada directamente de un fichero disponible online (ej., HTML, CSV, etc.) El formato de la información puede estar contenido en otros estándares como pueden ser XML o JSON.

En el caso de que la información se encuentre offline, es decir, estará en un archivo local, el programa deberá abrir el archivo, leerlo y hacer la conveniente conversión al formato RDF.

También hay que tener en cuenta la posibilidad de que la información, independientemente del medio en el que se encuentre, pueda ya estar en formato RDF. Es importante a la hora del diseño que una misma fuente puede estar tanto online como offline, por ejemplo si se descarga un archivo XML independientemente del medio donde estuviera, su traducción a RDF sería la misma.

3.4 Transformación de la información

El programa deberá crear y gestionar la información para su conversión a RDF. Una vez que se han seleccionado las fuentes de datos estás deben ser adaptadas tanto a la herramienta como al estándar RDF. Esto quiere decir que se deberá establecer el espacio de nombres, vocabularios y las distintas propiedades de acompañan a cada fuente de información.

Adicionalmente, se le podrá proporcionar una ontología para cada fuente de datos. El programa deberá ajustar los datos de acuerdo a dicha ontología. De esta manera se podrán establecer las restricciones del dominio convenientes.

3.5 Requisitos de desarrollo

En este apartado se hace hincapié en las necesidades ya mencionadas anteriormente. El software deberá proporcional una configuración orientada a módulos ampliables. La herramienta deberá proporcionar una manera sencilla de cambiar las diferentes configuraciones por parte del usuario.

3.6 Otros requisitos

El programa será desarrollado en el lenguaje de programación Java. Al ser un lenguaje multiplataforma podrá ejecutarse bajo los distintos sistemas operativos que lo soportan.

En cuanto a la interfaz de usuario el único requisito de la herramienta es que se maneje por línea de comandos. Tampoco existe ningún requisito en relación con el rendimiento. Estos apartados quedan a determinar en futuras versiones.

Finalmente, el código de la herramienta se publicara bajo dominio público.

3.7 Caso de uso

El software parte del requisito de poder recorrer diversas fuentes de información y extraer resultados de una serie de búsquedas. El propósito de este apartado es establecer un caso de uso desde donde se construirá la herramienta y sus configuraciones. Se establecerán una serie de fuentes de información y consultas para poder ser mostradas como ejemplo.

Un usuario desea obtener una serie de información de las fotos que son subidas a la web www.flickr.com. Esta información comprende estadísticas como por ejemplo cuál es la cámara más usada o cual es el software de edición más utilizado en las poblaciones con un rango determinado de habitantes. Para determinar las localidades se consultará DBpedia.org donde aparece la población. Como requisito imprescindible las fotos deben estar geolocalizadas para poder saber que fueron tomados en el rango de poblaciones que se analizan.

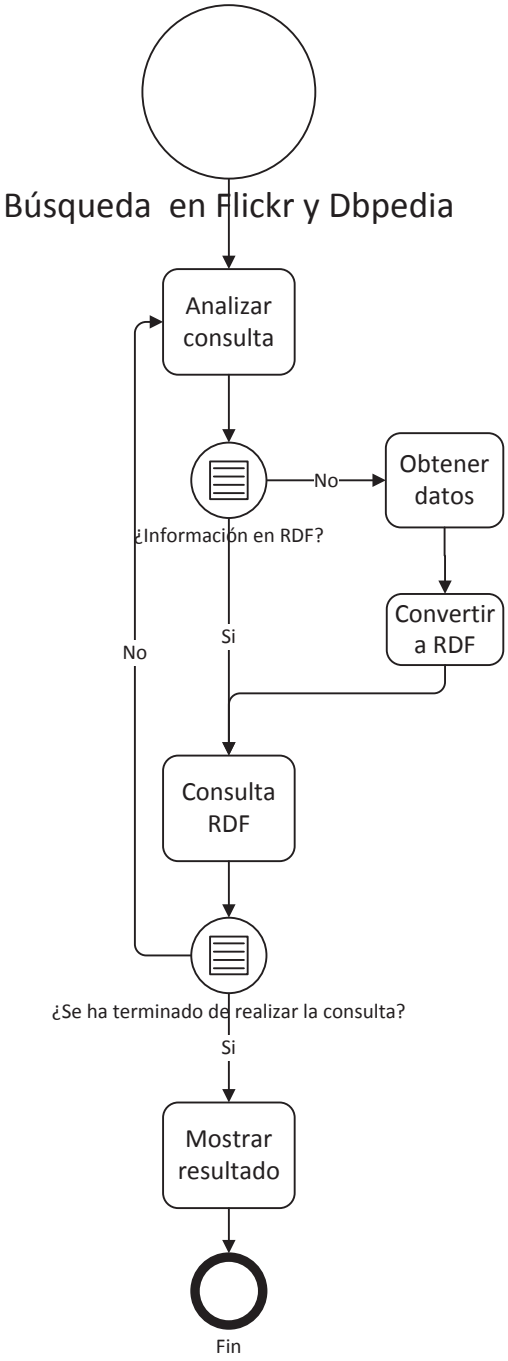

Ilustración 5 Diagrama de caso de uso

4 DISEÑO

4.1 Introducción

En el presente capítulo, partiendo de los requisitos establecidos en el apartado anterior, se procederá a describir la arquitectura diseñada para la herramienta. Se empezará desde su arquitectura y lo más general al diseño de su clases y lo más particular para poder desarrollar cada uno de los diversos subsistemas que lo forman.

4.2 Arquitectura

La herramienta cuenta con diversas capas, estas están pensadas para que su ampliación y posible reconfiguración sea lo más sencilla posible. Es por ello que algunos de los subsistemas pueden ser replicados para dotar de otras funcionalidades lo que haría crecer de manera horizontal pero manteniendo intacta su lógica vertical.

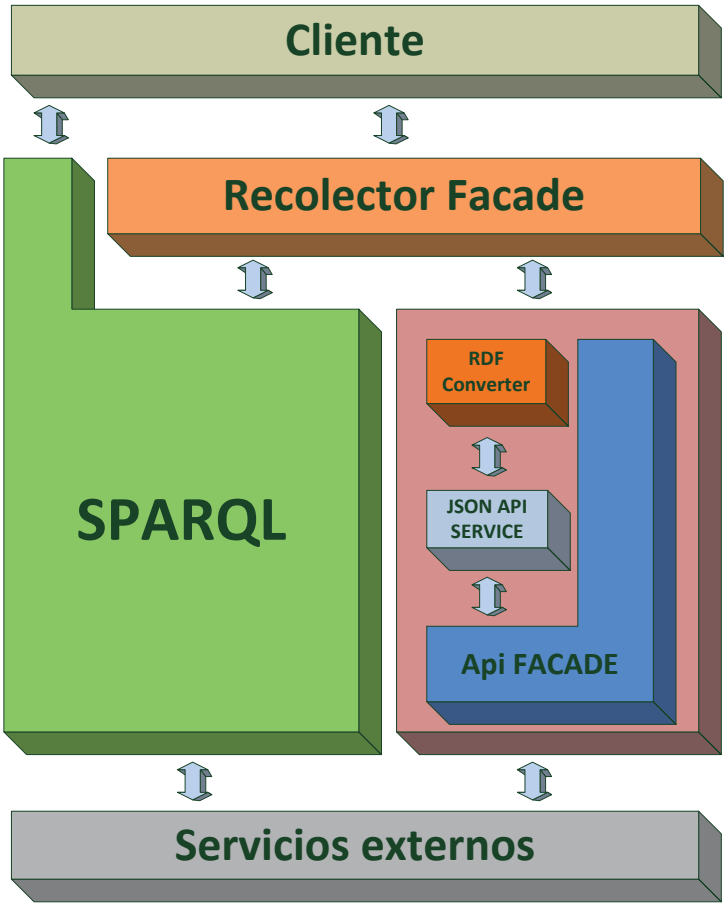

Ilustración 6 Diagrama de la arquitectura

La Ilustración 6 muestra el esquema de arquitectura para el caso más simple. Algunos de los conjuntos que forman la arquitectura se pueden ver replicados para acoger más funcionalidades sobre otros servicios. Estas capas, en general, irían encajadas entre los servicios externos y la capa Recolector *facade*. Es importante indicar que no se replicaría el módulo SPARQL, éste simplemente vería ampliada su funcionalidad. A continuación se detallaran las diversas capas que forman la arquitectura.

4.2.1 Cliente

Esta capa es la que permite gestionar la parte gráfica de la herramienta. A su vez también es responsable de recoger las entradas por parte del usuario, las consultas que realiza el usuario entre las diversas fuentes de datos disponibles. Estas consultas deberán ser entregadas a la capa Recolector *facade*, encargada de procesar la consulta. Dichas consultas deberán estar en el formato estándar SPARQL; es por ello que el módulo Cliente puede apoyarse en el módulo SPARQL para generar las consultas ajustándose al estándar.

4.2.2 Recolector facade

Este módulo permite al cliente abstraerse del funcionamiento de los diversos subsistemas que forman en su conjunto la herramienta. Es por definición la parte encargada de manejar las comunicaciones entre los diversos módulos y le dota del nivel lógico más superior. Como se puede ver en la Ilustración 6, este módulo tiene accesibilidad a todos los subsistemas a excepción de los servicios externos.

4.2.3 SPARQL

Como de su nombre se puede derivar, este subsistema se encarga de toda la gestión de consultas del sistema. Desde generar correctamente las propias cadenas de consulta ciñéndose al estándar homónimo a realizar las consultas a los diversos modelos RDF.

4.2.4 Manejador API

El conjunto que forman API f*acade*, JSON API *service* y RDF *converter* permite gestionar desde las comunicaciones con el servicio API que se ha configurado, hasta su conversión final a RDF.

De esta manera API *facade* es la capa que permite acceder directamente a los servicios de la API, en el caso de que sea JSON se apoya en el módulo JSON API *service*. Este módulo es donde se produce la transformación del JSON a objetos y permite abstraerse del modelo de datos de la información.

Por último se encuentra el convertidor de RDF, este es el más independiente y alejado del servicio de la API y se encarga de usar JSON API *service* para alimentarse de los datos necesarios y formar el documento RDF.

4.3 Paquetes y clases

El diseño de clases se deriva a partir de los diversos subsistemas descritos en la sección arquitectura del diseño. De esta manera se profundiza de lo más general a las clases que forman parte del diseño. Cada apartado describe un paquete que puede corresponder a una capa de la arquitectura o ser simplemente un apoyo a otro paquete o al conjunto de la herramienta.

El diseño de clases y paquetes ha sido ajustado para principalmente ajustarse al caso de uso descrito en la memoria. Es por ello que los nombres que aparecen hacen referencia al caso particular. Sin embargo, estas pueden ser generalizadas sencillamente como será explicado en capítulos posteriores.

4.3.1 Console

Este paquete contiene las clases que corresponden con la capa cliente de la arquitectura. En este caso el diseño del cliente está enfocado para una interfaz basada en consola. Para ello se cuenta con 3 clases.

La clase principal la forma *ConsoleController*, la cual se comunica con el resto de módulos del sistema y gestiona la lógica de la interfaz a través de su función *main()*. Esta clase se apoya en *ConsolePainter* para representar la información al usuario y leer las entradas.

ConsolePainter proporciona una serie de métodos que permiten imprimir los mensajes y leer las consultas que realiza el usuario. Estos métodos se apoyan en la clase *ConsoleMessages* para escribir los mensajes. De esta manera la clase que contiene los mensajes puede ser muy fácilmente configurada para cambiar el idioma de la herramienta.

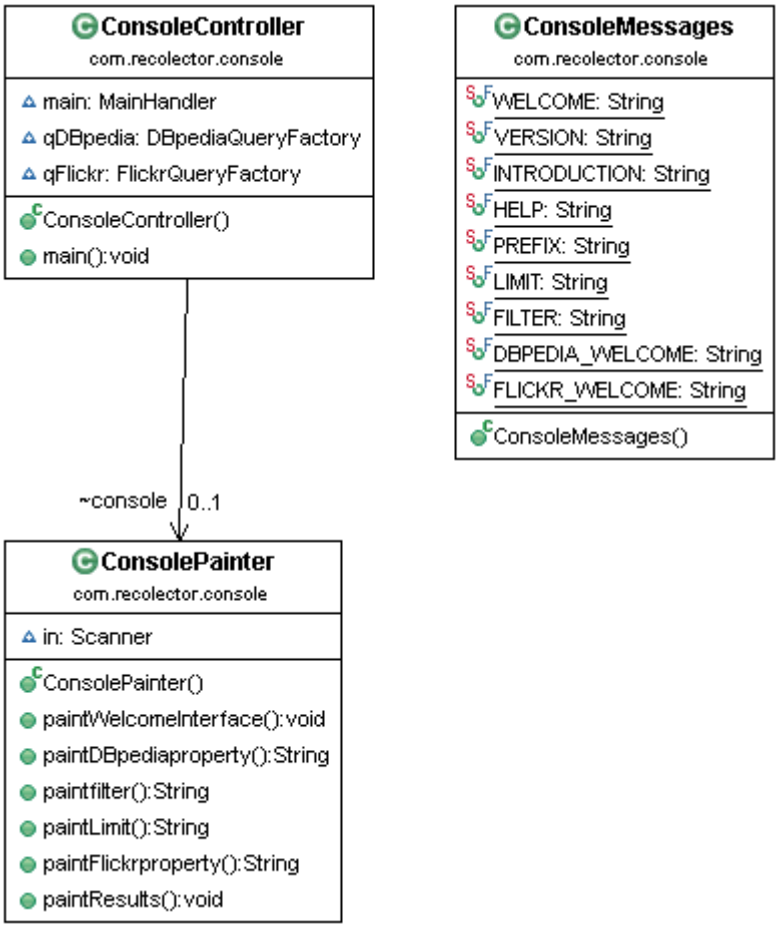

Ilustración 7 Diagrama de clases UML del paquete com.recolector.console

4.3.2 Recolector facade

Como se deriva de la arquitectura el paquete se corresponde con su capa homónima. El caso de uso provoca que se implementen 3 clases. Aunque en el Diagrama de clases UML del paquete com.recolector.facade se puede observar que existe una clase principal, *MainHandler*.

La clase principal gestiona las diversas fuentes de datos que son controladas por los diversos manejadores. De esta manera en la clase *MainHandler* se diseña toda la lógica de conexionado de las fuentes, es decir los diversos manejadores que se diseñan. Esta clase debe conocer los métodos lanzadores de información de esas fuentes de datos.

Ambas clases que son gestionadas por la principal implementan el método *getResults*. Este proporciona el análisis de los datos obtenidos en el módulo SPARQL según el modulo consultado.

Adicionalmente la propiedad principal de la clase *FlickrHandler* es contener el modelo RDF. Esta clase gestiona dicho modelo para dotar los métodos que necesita *MainHandler* para recuperar la información y rellenar con datos el citado modelo a través de los módulos de conversión y las capas que gestionan la API.

La clase *DBpediaHandler*. A diferencia de la anterior no gestiona ningún modelo puesto que este se encuentra remotamente y no puede ser manipulado por el software.

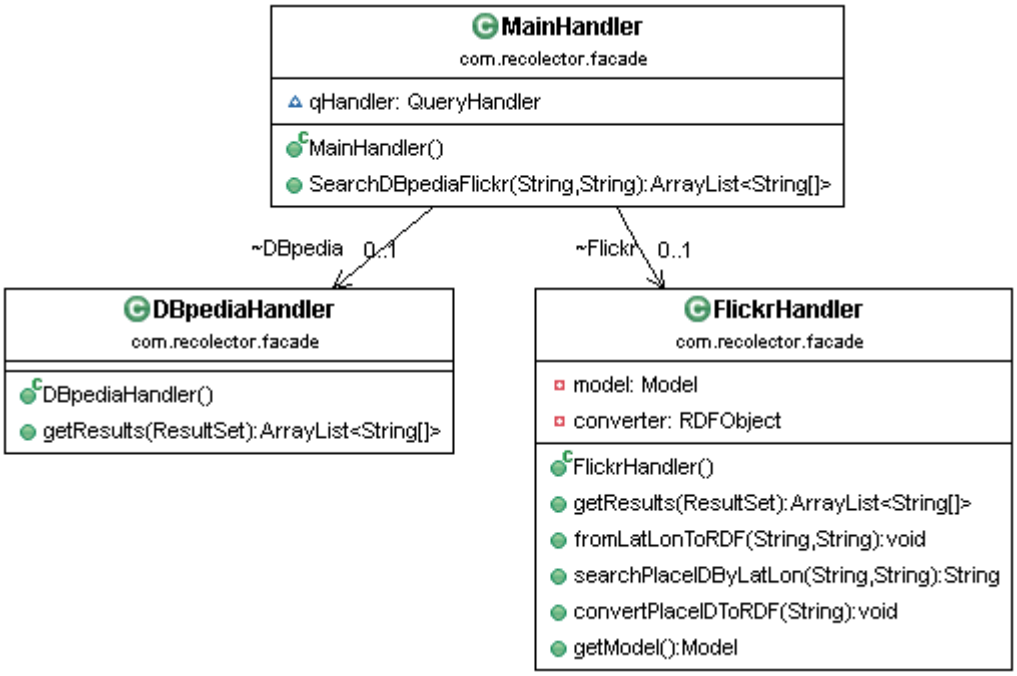

Ilustración 8 Diagrama de clases UML del paquete com.recolector.facade

4.3.3 SPARQL

Su correspondencia en la arquitectura es la capa de igual nombre que se presenta en la Ilustración 6. Se puede dividir en 2 funcionalidades principales. Por una parte la gestión de la cadena de consultas y por otra la consulta sobre un documento RDF.

Para gestionar las cadenas de consultas se parte de la clase abstracta *StringQueryFactory*. Esta implementa una gestión de las cadenas independientemente del

modelo. Las clases que derivan la clase anterior permiten una gestión de la cadena de consulta mucho más específica al modelo RDF que la implementa. Estas clases se apoyan en las constantes definidas en la clase *DefaultQuery* que cuenta con las diversas palabras reservadas del estándar y las consultas predefinidas por el software.

Por otra parte clase *QueryHandler* gestiona los métodos que permiten realizar los dos tipos de consultas, bien a un modelo interno de la herramienta o a un servicio por medio de un *endpoint*.

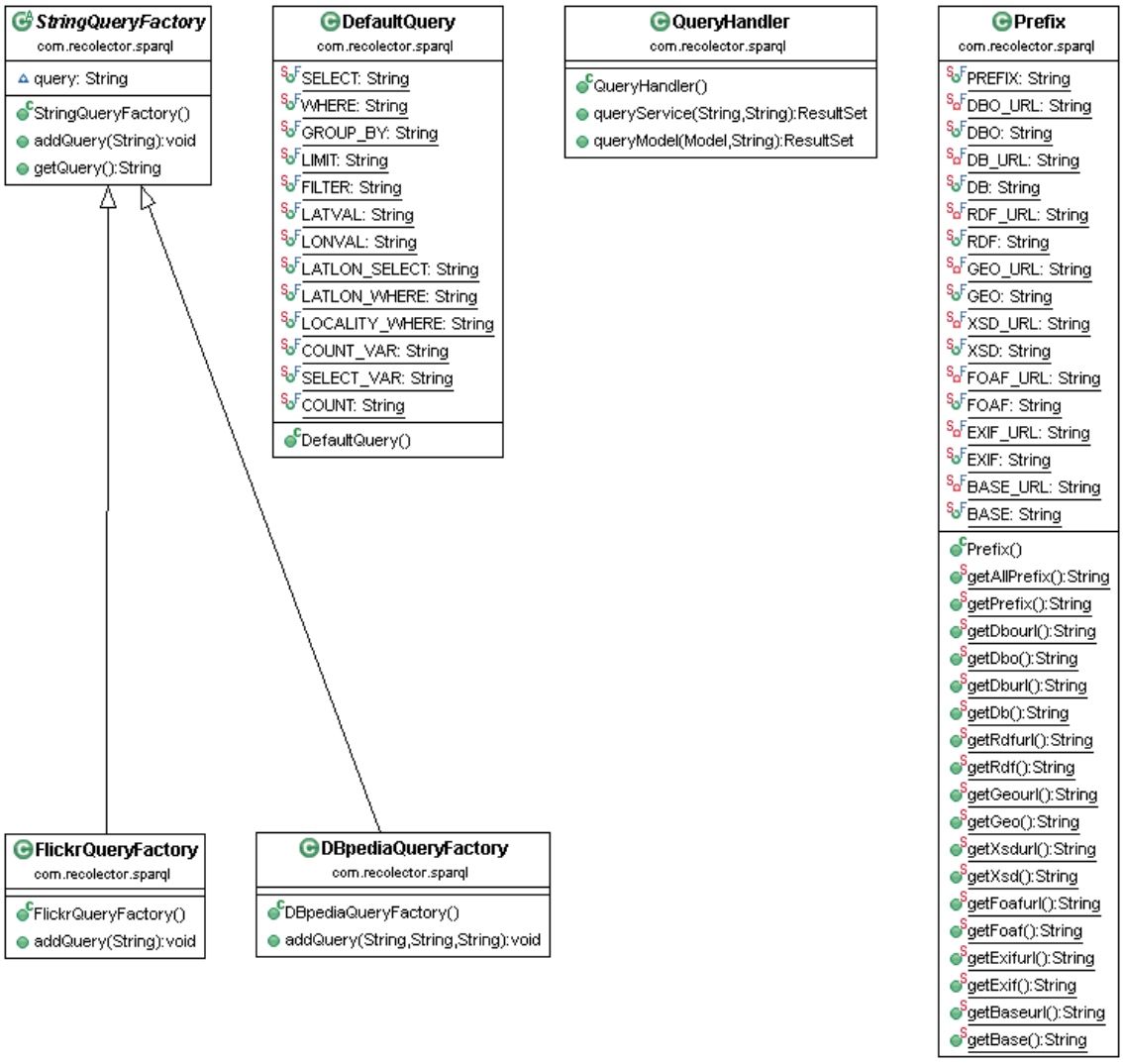

Ilustración 9 Diagrama de clases UML del paquete com.recolector.sparql

4.3.4 Flickr facade

De igual nombre que en la arquitectura, este paquete proporciona en su clase principal, *ApiFlickr*, el gestor de la API con el servicio online directamente. Proporciona los métodos para llamar a todas las funciones de la API que van a ser implementadas. De igual manera gestiona la capa de comunicación con el servicio API.

Las otras dos clases, *FlickrSearch* y *FlickrPlaces*, proporcionan una implementación de un *iterador* para manejar la lista de resultados obtenidos de los métodos de los servicios una vez que ya han sido procesados, en este caso transformados a objetos.

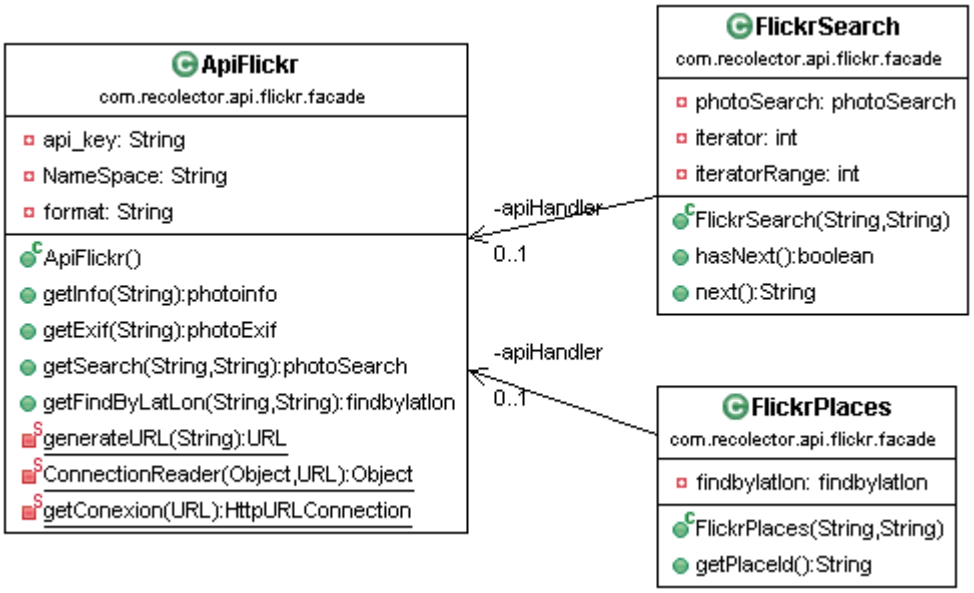

Ilustración 10 Diagrama de clases UML del paquete com.recolector.api.flickr.facade

4.3.5 Flickr service

Este conjunto de clases, que en la arquitectura corresponde a JSON API *service*, proporciona el proveedor de recursos, esto es la devolución de la información por medio de métodos *getters*. Este proveedor de recursos además se apoya en un conjunto de clases que presentan toda la arquitectura donde se apoya la librería GSON para generar la arquitectura de objetos que se obtiene de los JSON de la API, en este caso particular de flickr.com.

La clase *FlickrResources*, el proveedor de recursos, se encarga de hacer la consulta a los métodos de la API que proporcionan información sobre una foto se puede invocar devolver el valor de la información del método deseado.

GSON necesita un conjunto de clases que su nombre corresponde con el nombre en el servicio que proporciona la API. Por lo tanto el diseño es una simple transformación de la estructura JSON a Objetos. El JSON obtenido tiene más valores pero como no son necesarios no se implementan las clases extras.

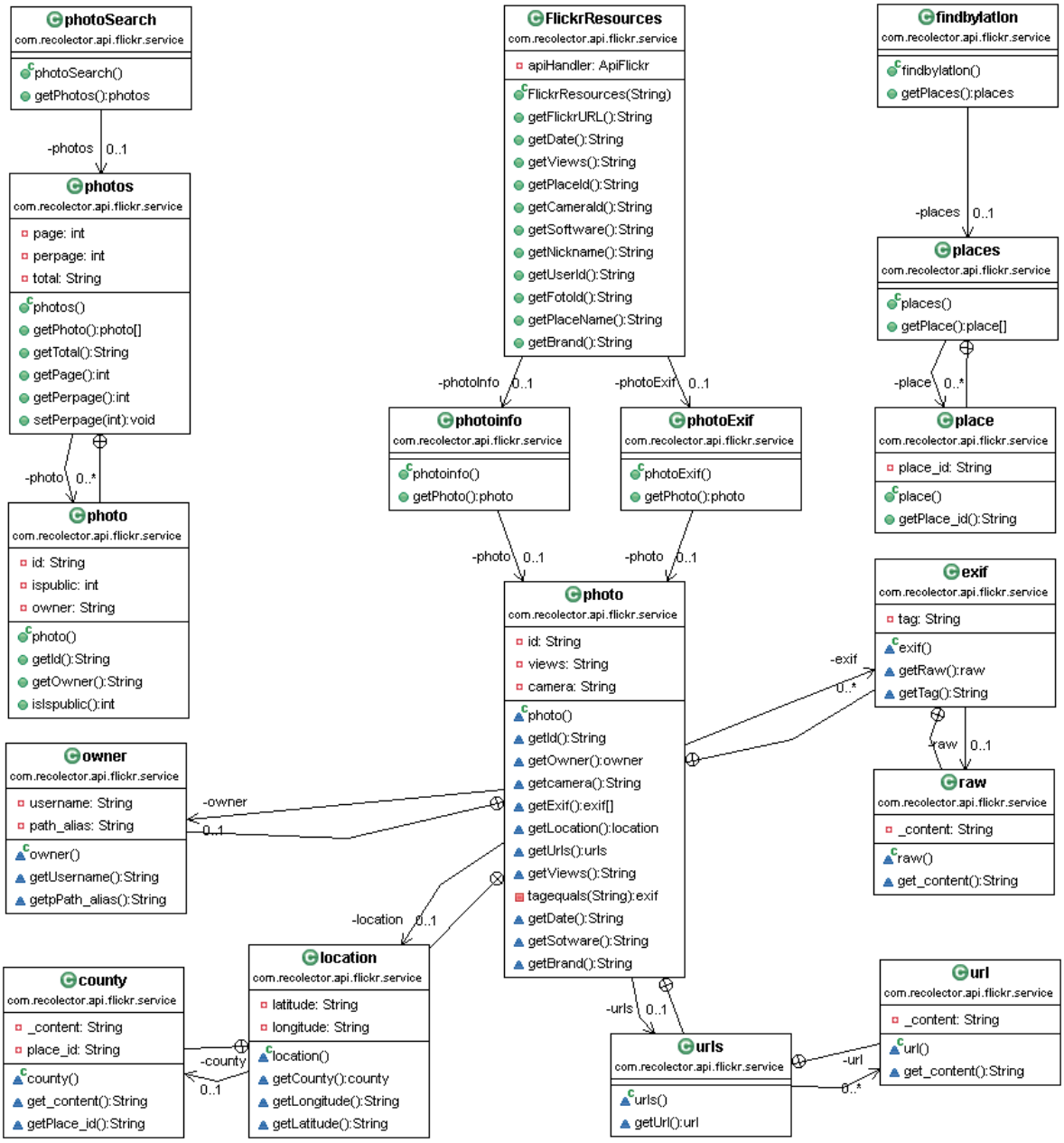

Ilustración 11 Diagrama de clases UML del paquete com.recolector.api.flickr.service

4.3.6 RDF

El paquete RDF es el encargado del denominado en la arquitectura RDF *converter*. Este necesita un modelo RDF a rellenar y un proveedor de recursos. De esta manera por medio de una configuración predeterminada por el usuario en función de la fuente de datos y adicionalmente de una ontología va obteniendo datos del proveedor y rellenando el modelo proporcionado.

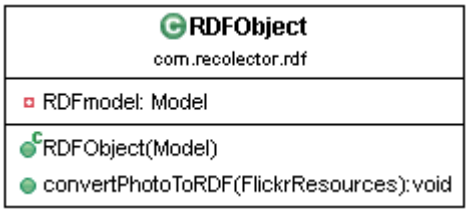

Ilustración 12 Diagrama de clases UML del paquete com.recolector.rdf

En el caso concreto del caso de uso para el servicio Flickr se ha diseñado un modelo RDF. Se puede observar en la Ilustración 13 los diferentes recursos, literales y propiedades que forman el documento RDF que se debe implementar en este paquete.

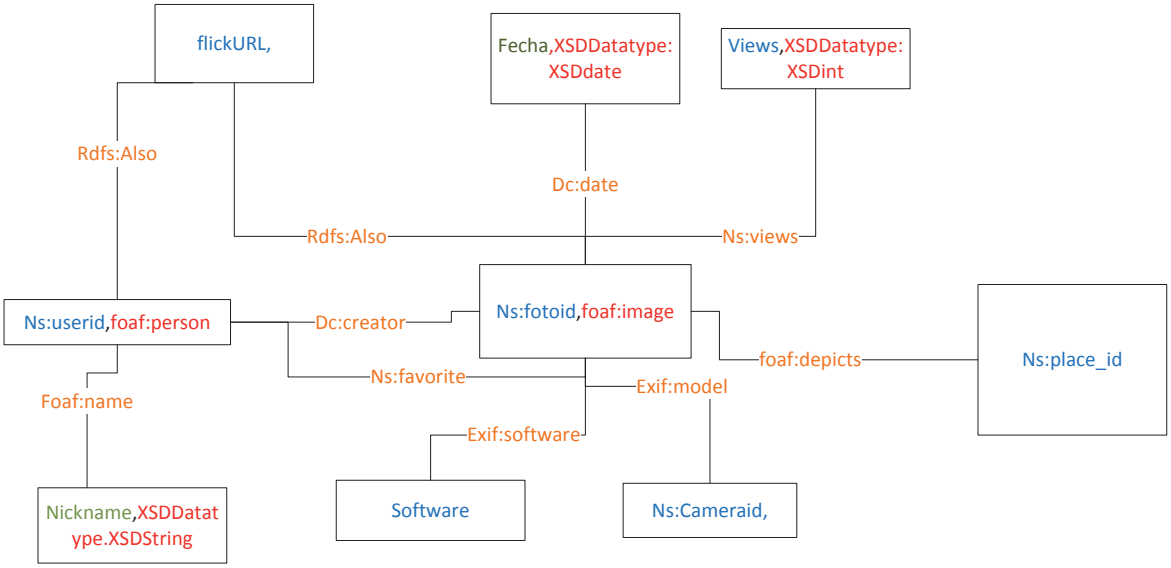

Ilustración 13 Esquema RDF flickr

4.3.7 Properties

Este paquete es simplemente un módulo auxiliar del conversor a RDF que permite definir las diversas propiedades. En este caso se definen las propiedades que no están disponibles en la librería Jena y que son necesarias en el caso de uso. En el caso de *Exif* es una implementación de un vocabulario disponible [EXIF].

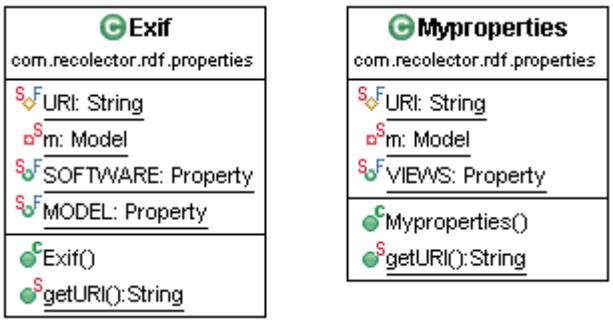

Ilustración 14 Diagrama de clases UML del paquete com.recolector.rdf.properties

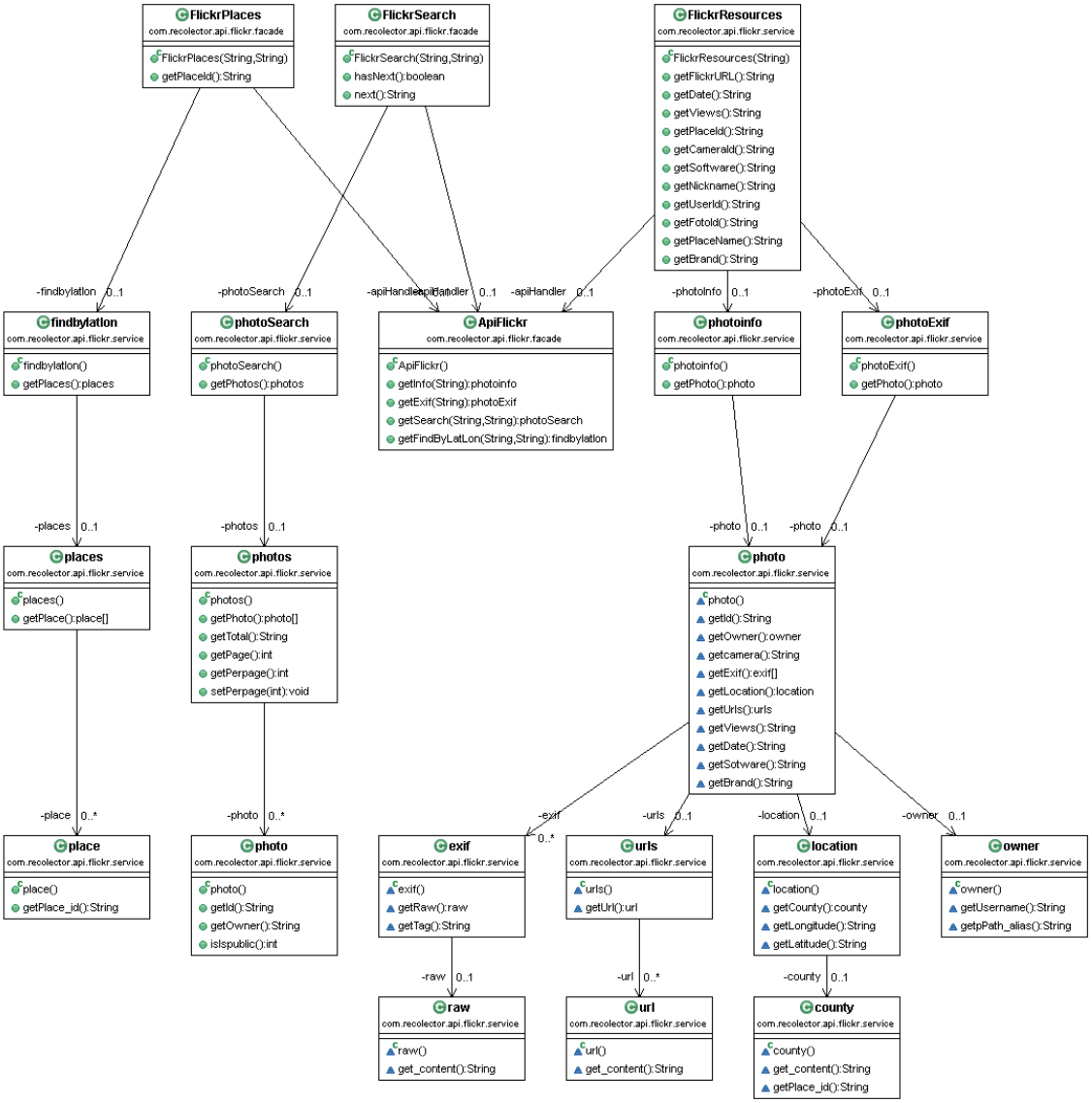

Ilustración 15 Diagrama de clases UML del conjunto de la API

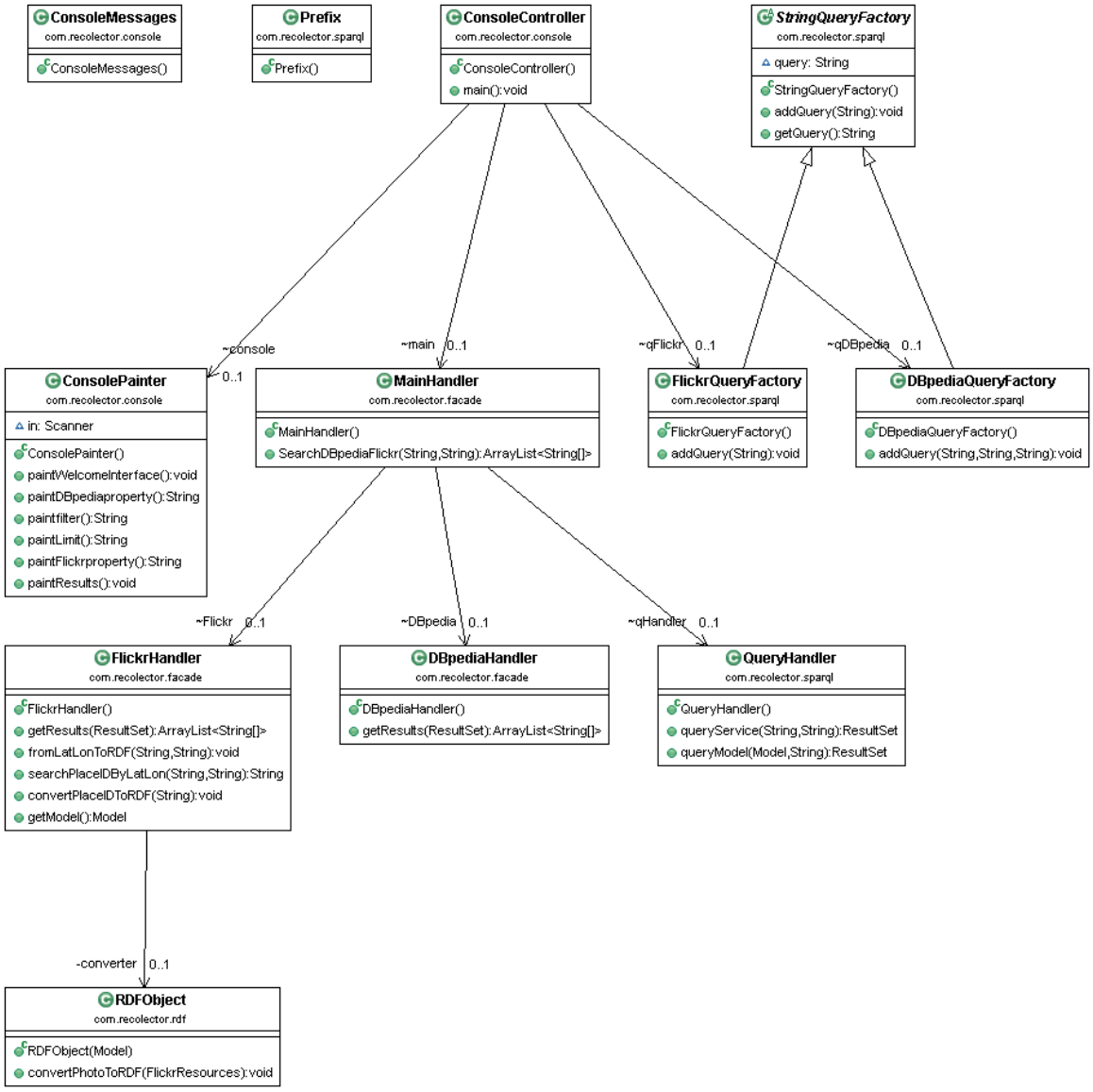

Ilustración 16 Diagrama general de clases UML

4.4 Diagrama de secuencia

Para mostrar un mejor entendimiento entre como los diversos módulos, paquetes y clases se comunican, a continuación, se muestran una serie de diagramas de secuencia en los que se pueden ver el diseño de conexionado entre clases.

El Diagrama de secuencia general muestra el comportamiento que sigue la herramienta desde que un usuario introduce una consulta hasta que le es devuelta la solución. Como implica muchas clases y paquetes se presenta con una visión más general en paquetes y los métodos más representativos.

El Diagrama de secuencia Flickr muestra el proceso de comunicación que se da en el manejador API para convertir un conjunto de datos a RDF. Este diagrama muestra el detalle que queda por definir en el diagrama general durante la ejecución del bucle. El lanzador del proceso es la clase FlickrHandler que se encuentra en el paquete recolector *facade*.

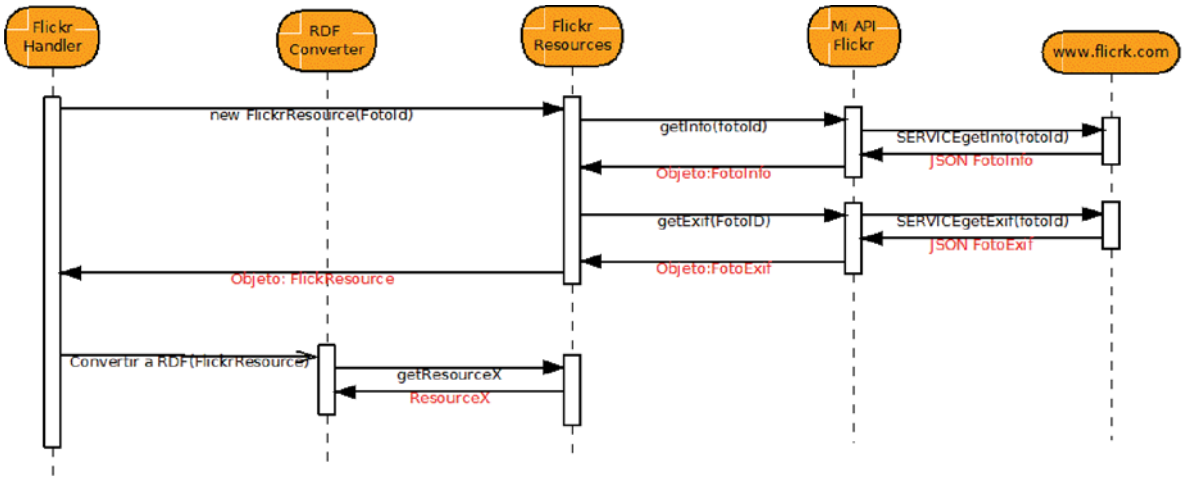

Ilustración 17 Diagrama de secuencia Flickr

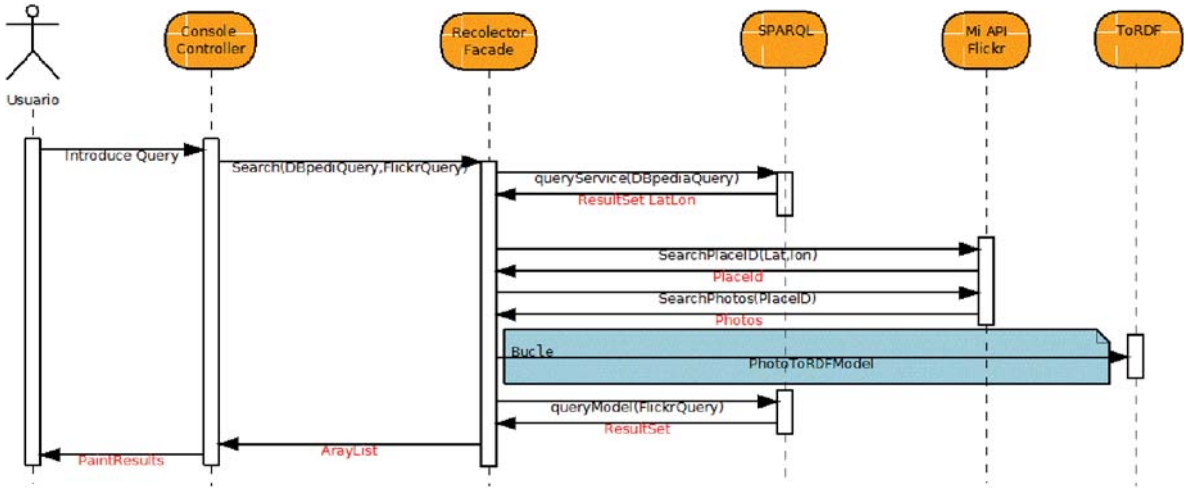

Ilustración 18 Diagrama de secuencia general

5 IMPLEMENTACIÓN

5.1 Introducción

El software se ha implementado siguiendo los requisitos establecidos y un diseño inicial. Durante la implementación ha habido aspectos del diseño original que se han visto variados y el diseño final que se sigue es el presentado en el apartado correspondiente.

En este apartado además se describirán una serie de limitaciones que tiene la implementación. También se explicará cómo tienen que ser implementadas las ampliaciones del software.

La herramienta se encuentra desarrollada en el lenguaje de programación Java, más concretamente en su versión 8. Además hace uso de dos librerías externas; Jena, para el manejo de los documentos RDF y las consultas del estándar SPARQL y GSON para la conversión de los datos JSON a objetos Java.

Todo el código está disponible para su consulta pública en el servicio *github*. El cual se encuentra bajo el nombre de *RDFrecolector* en el repositorio de *AlvaroMoreno*. El enlace para su acceso es el siguiente: https://github.com/AlvaroMoreno/RDFrecolector.

5.2 Limitaciones

El diseño proporciona, en general, el cumplimento de todos los requisitos establecidos, sin embargo, existen una serie de limitaciones técnicas a la hora de la implementación del caso de uso. Estas limitaciones influyen sobre todo en la rapidez del software y en la calidad de los datos obtenidos.

El método que proporciona la API de flickr.com para acceder a sus fotos devuelve las ocurrencias que han sido subidas más recientes. Se puede ir iterando para aumentar la cantidad de fotos obtenidas.

La razón de que no haya un requisito específico en cuanto el número de fotos recuperadas es principalmente por la idea de recoger la mayor cantidad que técnicamente se pueda. Es por ello que se debe establecer un compromiso en el cual ni se sature el servicio y se alargue excesivamente la espera de la herramienta y además la propia limitación de memoria que se pueda presentar.

De igual manera si son proporcionadas de la DBpedia muchas localizaciones para recuperar sus fotos también puede verse comprometido el servicio por lo que se aconseja en la implementación mantener un compromiso para la estabilidad de la herramienta.

5.3 Configuración de ampliaciones

Uno de los compromisos de la herramienta es la posibilidad de que fácilmente se pueda ampliar y cambiar el origen de las fuentes de datos. Los tipos de fuentes de datos más desarrollados para recuperar información son aquellos que recuperan la información de un *endpoint*, el cual puede ser consultado por medio del estándar SPARQL, o las API *REST* que devuelvan un documento JSON.

Si se quiere añadir un servicio que dispone de *endpoint* simplemente se necesitaría modificar la lógica del *mainHandler* y añadir su propio *Handler* con las funciones de tratamiento de resultados que fueran necesarias. Para hacer la consulta simplemente habría que invocar el método *QueryHandler* indicando la dirección del *endpoint*.

La adicción de una nueva fuente de datos que proporciona sus datos en JSON es algo más compleja pues se tiene que diseñar toda la lógica del modelo RDF y la ontología que se quiere implementar.

Para ello se puede observar el ejemplo del diseño de Flickr. En la clase *ApiFlickr* habría que implementar los métodos que proporcionan la fuente de datos que se quiere conectar, sin embargo no haría falta implementar la lógica de la conexión ni del tratamiento de datos. Si es necesario se implementarían las clases con *iterador* si estas fueran necesarias.

Es imprescindible analizar el documento JSON proporcionado por la API para generar el conjunto de clases y atributos que se quieren recoger. De este análisis también se deberá diseñar la conversión a RDF, implementando el proveedor de recursos, definiendo las propiedades que se necesiten y añadiendo la información al modelo en el módulo RDF.

A continuación se muestran dos tablas de modo de resumen con los pasos que habría que seguir para realizar una de las ampliaciones descritas anteriormente.

Ampliación: servicio con endpoint SPARQL		
Paquete	**Clase**	**Acción**
com.recolector.sparql	ServicioQueryFactory	Extender si se necesita la clase abstracta StringQueryFactory
com.recolector.facade	ServicioHandler	Crear clase. Implementar método getResults(ResultSet)
com.recolector.facade	MainHandler	Añadir la lógica para llamar a ServicioHandler y método queryServiceQueryHandler con la dirección del servicio endpoint

Tabla 10 Ampliación servicio con endpoint

Ampliación: servicio con API con respuesta JSON		
Paquete	**Clase**	**Acción**
com.recolector.Serviciofacade	ApiHandler	Crear la clase. Ayuda consultar Flickr.
com.recolector.Serviciofacade	ServicioMétodo	Crear iterador para las estructuras que lo requieran
com.recolector.Servicio.Service	*	Crear estructura de clases a partir de JSON
com.recolector.Servicio.Service	ResourceProvider	Crear la clase que implemente los métodos get de los recursos necesarios
com.recolector.rdf	RDFObject	Añadir el método que genere el modelo RDF a partir de ResourceProvider
com.recolector.sparql	ServicioQueryFactory	Extender si se necesita la clase abstracta StringQueryFactory.
com.recolector.facade	ServicioHandler	Crear clase. Implementar método getResults(ResultSet) y los necesarios para manejar las consultas a ApiHandler
com.recolector.facade	MainHandler	Añadir la lógica para llamar a ServicioHandler y método queryModel de QueryHandler

Tabla 11 Ampliación servicio API JSON

6 CASOS DE PRUEBA

Para poder asegurarse de la calidad del software se han llevado a cabo una serie de pruebas en los diversos subsistemas que lo componen. Cada una tiene como objetivo probar el correcto funcionamiento del módulo analizado.

Caso de prueba 1	
Nombre de la prueba	Conexión API flickr.com
Módulo	flickr.facade
Dependencias	API flickr.com
Descripción	Se comprueba que se descarga un archivo JSON desde la API de flickr.com
Entrada	Servicio requerido más el id de una foto de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	El JSON es igual que el obtenido en la API.
Resultado	Correcto

Tabla 12 Prueba: Conexión API flickr.com

Caso de prueba 2	
Nombre de la prueba	De JSON getInfo a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio getInfo a los objetos Java correspondientes por medio de la librería GSON
Entrada	Servicio requerido más el id de una foto de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que coinciden con la muestra.
Resultado	Correcto

Tabla 13 Prueba: Nombre de la prueba De JSON getInfo a objeto Java

Caso de prueba 3	
Nombre de la prueba	De JSON getInfo incompleto a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio getInfo a los objetos Java correspondientes por medio de la librería GSON. Previamente se comprueba que el id no tiene algún valor.
Entrada	Servicio requerido más el id de una foto de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que se gestiona correctamente los valores nulos de la muestra.
Resultado	Correcto

Tabla 14 Prueba: De JSON getInfo incompleto a objeto Java

Caso de prueba 4	
Nombre de la prueba	De JSON getExif a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio getExif a los objetos Java correspondientes por medio de la librería GSON
Entrada	Servicio requerido más el id de una foto de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que coinciden con la muestra.
Resultado	Correcto

Tabla 15 Prueba: De JSON getExif a objeto Java

Caso de prueba 5	
Nombre de la prueba	De JSON getExif incompleto a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio getExif a los objetos Java correspondientes por medio de la librería GSON. Previamente se comprueba que el id no tiene algún valor.
Entrada	Servicio requerido más el id de una foto de ejemplo, con algún valor modulo ya comprobado en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que los datos nulos se manejan correctamente.
Resultado	Correcto

Tabla 16 Prueba: De JSON getExif incompleto a objeto Java

Caso de prueba 6	
Nombre de la prueba	De JSON search a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio search a los objetos Java correspondientes por medio de la librería GSON
Entrada	Servicio requerido más el id de una lugar de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que coinciden con la muestra.
Resultado	Correcto

Tabla 17 Prueba: De JSON search a objeto Java

Caso de prueba 7	
Nombre de la prueba	De JSON findByLatLon a objeto Java
Módulo	flickr.facade
Dependencia	Api flickr.com, GSON, flickr.service
Descripción	Se realiza la transformación del JSON obtenido de la API de flickr.com del servicio findByLatLon a los objetos Java correspondientes por medio de la librería GSON
Entrada	Servicio requerido más dos coordenadas, latitud y longitud, de ejemplo, ya comprobado su correcto funcionamiento en la API.
Condición de éxito	Se realizan impresiones de los datos almacenados en los objetos y se comprueba que coinciden con la muestra.
Resultado	Correcto

Tabla 18 Prueba: De JSON findByLatLon a objeto Java

Caso de prueba 8	
Nombre de la prueba	RDF a través de flickr.service y Jena
Módulo	Rdf
Dependencia	Api flickr.com, GSON, Jena, flickr.*, properties
Descripción	Se transforman los datos obtenidos de la API de flickr.com y los servicios getInfo y getExif a un modelo RDF por medio de la librería Jena. Se imprime un grafo RDF en formato Turtle.
Entrada	Se necesita un id de foto correcto y enlazar los módulos de flickr con los de rdf.
Condición de éxito	Los datos mostrados en formato RDF corresponde tanto en valor como en disposición al grafo esperado
Resultado	Correcto

Tabla 19 Prueba: RDF a través de flickr.service y Jena

Caso de prueba 9	
Nombre de la prueba	RDF nodos en blanco
Módulo	Rdf
Dependencia	Api flickr.com, GSON, Jena, flickr.*, properties
Descripción	Se transforman los datos obtenidos de la API de flickr.com y los servicios getInfo y getExif con una foto con datos incompletos a un modelo RDF por medio de la librería Jena. Se imprime un grafo RDF en formato Turtle.
Entrada	Se necesita un id de foto correcto pero con datos incompletos y enlazar los módulos de flickr con los de rdf.
Condición de éxito	Los datos mostrados en formato RDF corresponden tanto en valor como en disposición al grafo esperado con especial atención a los valores nulos.
Resultado	Correcto

Tabla 20 Prueba: RDF nodos en blanco

Caso de prueba 10	
Nombre de la prueba	Generación de consulta a la DBpedia
Módulo	Sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta al endpoint Virtuoso de la DBpedia para obtener unas coordenadas a partir de la población de las localidades
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas. Se parametriza poniendo una rango de población
Condición de éxito	Se obtiene una lista de coordenadas que corresponde con recursos con la población determinada.
Resultado	Correcto

Tabla 21 Prueba: Generación de consulta a la DBpedia

Caso de prueba 11	
Nombre de la prueba	Generación de consulta errónea la DBpedia
Módulo	Sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta al endpoint Virtuoso de la DBpedia para obtener unas coordenadas a partir de la población de las localidades pero con unos valores incorrectos.
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas. Se parametriza poniendo una rango de población incorrecto.
Condición de éxito	No se obtiene nada
Resultado	Correcto

Tabla 22 Prueba: Generación de consulta errónea la DBpedia

Caso de prueba 12	
Nombre de la prueba	Consulta básica a la DBpedia
Módulo	sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta básica al endpoint Virtuoso de la DBpedia para obtener unas coordenadas.
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas.
Condición de éxito	Se obtiene una lista de coordenadas.
Resultado	Correcto

Tabla 23 Prueba: Consulta básica a la DBpedia

Caso de prueba 13	
Nombre de la prueba	Consulta limitada a la DBpedia
Módulo	sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta al endpoint Virtuoso de la DBpedia para obtener unas coordenadas.
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas. Además se añade un límite al número de resultados obtenidos.
Condición de éxito	Se obtiene una lista de coordenadas con el número límite de resultados.
Resultado	Correcto

Tabla 24 Prueba: Consulta limitada a la DBpedia

Caso de prueba 14	
Nombre de la prueba	Consulta parametrizada a la DBpedia
Módulo	Sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta al endpoint Virtuoso de la DBpedia para obtener unas coordenadas a partir de la población de las localidades
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas. Se parametriza poniendo una rango de población
Condición de éxito	Se obtiene una lista de coordenadas que corresponde con recursos con la población determinada.
Resultado	Correcto

Tabla 25 Prueba: Consulta parametrizada a la DBpedia

Caso de prueba 15	
Nombre de la prueba	Consulta avanzada a la DBpedia
Módulo	Sparql
Dependencia	Jena, Virtuoso
Descripción	Se realiza una consulta al endpoint Virtuoso de la DBpedia para obtener unas coordenadas a partir de una serie de propiedades.
Entrada	La consulta se realiza con los parámetros para obtener recursos de tipo seatlement y seleccionar las coordenadas. Se parametriza poniendo de condición una propiedad de tipo meteorológico
Condición de éxito	Se obtiene una lista de coordenadas que corresponde con recursos con la población determinada.
Resultado	Correcto

Tabla 26 Prueba: Consulta avanzada a la DBpedia

Caso de prueba 16	
Nombre de la prueba	Lista de fotos a RDF flickr
Módulo	Facade
Dependencia	Api flickr.com, GSON, Jena,flickr.*, properties
Descripción	Se gestiona la construcción de un modelo RDF con los fotoId de flickr.com obtenidos a partir de un placeId. Se imprime el modelo RDF.
Entrada	Un placeId para llamar a la API de flickr
Condición de éxito	Se obtiene el modelo RDF correspondiente al placeId consultado.
Resultado	Correcto

Tabla 27 Prueba: Lista de fotos a RDF flickr

Caso de prueba 17	
Nombre de la prueba	De coordenadas a RDF flickr
Módulo	facade
Dependencia	Api flickr.com, GSON, Jena,flickr.*, properties
Descripción	Se recupera un placeId a partir de unas coordenadas para obtener una lista de fotos que se transforma a RDF.
Entrada	Unas coordenadas para llamar a la API de flickr.
Condición de éxito	Se obtiene el modelo RDF correspondiente a las coordenadas consultadas.
Resultado	Correcto

Tabla 28 Prueba: De coordenadas a RDF flickr

Caso de prueba 18	
Nombre de la prueba	De lista de coordenadas a RDF flickr
Módulo	Facade
Dependencia	Api flickr.com, GSON, Jena,flickr.*, properties
Descripción	Se recupera los correspondientes placeId a partir de una lista de coordenadas. Para que las fotoid se transformen a RDF.
Entrada	Una lista de coordenadas para llamar a la API de flickr.
Condición de éxito	Se obtiene el modelo RDF correspondiente a las coordenadas consultadas.
Resultado	Correcto

Tabla 29 Prueba: De lista de coordenadas a RDF flickr

Caso de prueba 19	
Nombre de la prueba	Consulta básica RDF flickr
Módulo	Sparql
Dependencia	Api flickr.com, GSON, Jena,flickr.*, properties,facade
Descripción	A partir de un modelo RDF con datos de flickr se realiza una consulta para recuperar la propiedad de modelo de cámara.
Entrada	El modelo RDF a consultar y la consulta con el parámetro de un recurso o literal del modelo para extraer.
Condición de éxito	Se obtienen todas los recursos que cumplen la condición
Resultado	Correcto

Tabla 30 Prueba: Consulta básica RDF flickr

Caso de prueba 20	
Nombre de la prueba	Consulta repeticiones en RDF flickr
Módulo	Sparql
Dependencia	Api flickr.com, GSON, Jena,flickr.*, properties,facade
Descripción	A partir de un modelo RDF con datos de flickr se realiza una consulta para recuperar el número de ocurrencias que presenta una propiedad de modelo de cámara.
Entrada	El modelo RDF a consultar y la consulta con el parámetro de un recurso o literal del modelo para extraer y la propiedad de contar las repeticiones.
Condición de éxito	Se obtienen el número de entradas del recurso seleccionado.
Resultado	Correcto

Tabla 31 Prueba: Consulta repeticiones en RDF flickr

Caso de prueba 21	
Nombre de la prueba	Enlace básico DBpedia y flickr
Módulo	console
Dependencia	Api flickr.com, GSON, Jena, flickr.*, properties, sparql, facade, VIRTUOSO
Descripción	Se realiza una consulta. Primero la parte en la DBpedia, donde se limita a uno el resultado, y a partir de esta se lanza la creación del modelo RDF y se realiza la parte de la consulta al modelo RDF obtenido a partir de flickr.cm
Entrada	La consulta, para la DBpedia parámetros de seatlement, latitud, longitud, población con rango y limite a 1. Para el modelo de flickr ocurrencias de cámaras.
Condición de éxito	Se obtienen el número de entradas del recurso seleccionado de los datos que corresponde a la consulta de DBpedia
Resultado	Correcto

Tabla 32 Prueba: Enlace básico DBpedia y flickr

Caso de prueba 22	
Nombre de la prueba	Enlace múltiple DBpedia y flickr
Módulo	console
Dependencia	Api flickr.com, GSON, Jena, flickr.*, properties, sparql, facade, VIRTUOSO
Descripción	Se realiza una consulta. Primero la parte en la DBpedia, y a partir de esta se lanza la creación del modelo RDF y se realiza la parte de la consulta al modelo RDF obtenido a partir de flickr.cm
Entrada	La consulta, para la DBpedia parámetros de seatlement, latitud, longitud, población con rango. Para el modelo de flickr ocurrencias de cámaras.
Condición de éxito	Se obtienen el número de entradas del recurso seleccionado de los datos que corresponde a la consulta de DBpedia
Resultado	Correcto

Tabla 33 Prueba: Enlace múltiple DBpedia y flickr

Caso de prueba 23	
Nombre de la prueba	Enlace múltiple DBpedia y flickr
Módulo	console
Dependencia	Api flickr.com, GSON, Jena, flickr.*, properties, sparql, facade, VIRTUOSO
Descripción	Se realiza una consulta. Primero la parte en la DBpedia, y a partir de esta se lanza la creación del modelo RDF y se realiza la parte de la consulta al modelo RDF obtenido a partir de flickr.cm
Entrada	La consulta, para la DBpedia parámetros de seatlement, latitud, longitud, población con rango. Para el modelo de flickr ocurrencias de software.
Condición de éxito	Se obtienen el número de entradas del recurso seleccionado de los datos que corresponde a la consulta de DBpedia
Resultado	Correcto

Tabla 34 Prueba: Enlace múltiple DBpedia y flickr

7 CONCLUSIONES

En este TFG se ha desarrollado una herramienta capaz de explotar uno de los principales beneficios de la web semántica. Esto es la posibilidad de combinar la información entre diversas fuentes de datos. En particular, el caso de uso desarrollado ha permitido combinar datos de fuentes tan distintas como son el servicio de alojamiento de fotos flickr.com y la enciclopedia DBpedia.

Concretamente la herramienta permite realizar consultas para conocer una de las características más usadas en las fotos tomadas en las poblaciones con unas determinadas propiedades. Las características de la foto pueden ser por ejemplo el modelo de cámara más usado o el software de edición utilizado. Por parte de la población se puede seleccionar según un rango de población. De esta manera podríamos conocer cuáles son las cámaras usadas en las poblaciones más utilizadas en las poblaciones de un millón de habitantes. Incluso se podría ser más específico con alguna otra propiedad presente en DBpedia.

A su vez se ha hecho un esfuerzo para que la herramienta siga unos patrones de diseño que consigan una estandarización que permita añadir nuevas y distintas fuentes de datos. Estas se han centrado sobre todo en aquellas que tienen accesibles un *endpoint* SPARQL y las API que devuelven resultados en JSON.

De esta manera por ejemplo si se añade una fuente de datos que contenga el tiempo en las poblaciones podemos además filtrar por esa característica. Por ejemplo se podría consultar cuál es la cámara más utilizada en las poblaciones de más de un millón de habitantes los días más lluviosos.

No solo se tiene que tener en cuenta la posibilidad de ampliar a diversas fuentes. Una misma fuente puede ser a su vez ampliada para contener más o nuevos datos. Toda esta posibilidad de actualización y no control por parte del desarrollador dificulta la creación de una herramienta perfecta.

La heterogeneidad que presentan las diversas fuentes de datos provoca grandes dificultades a la hora de proponer un estándar definitivo desde una primera versión del producto. Futuras iteraciones en el desarrollo del software permitirían una mayor homogenización.

La web semántica cada vez está más presente y desarrollada, sin embargo, aún los usuarios no pueden verse beneficiados de las bondades que presenta esta tecnología. No en todas las ocasiones es debido a que la fuente no dispone la información de manera fácilmente accesible, sino porque las licencias de distribución no permiten su consulta.

La mayoría de la web semántica está desarrollada en el mundo académico. Uno de los problemas se encuentra en el poder que puede ofrecer una buena combinación de datos. Es difícil parar el avance de la tecnología y seguramente cada vez más empresas hagan esfuerzos para su beneficio.

8 LÍNEAS FUTURAS

8.1 Introducción

En este apartado se presentará una serie de funcionalidades que han sido o bien descubiertas en las distintas iteraciones del desarrollo del programa y no han sido posibles de implementar o funcionalidades adicionales a las requeridas pero que por motivos de tiempo no han sido diseñadas.

8.2 Comprobador de cadena de consulta

El estándar SPARQL proporciona la sintaxis para realizar las consultas. En general debido a las peculiaridades de las fuentes de datos, en la herramienta, las consultas son más limitadas que todas las posibilidades de combinaciones que permite el estándar.

Esta mejora se encontraría dentro del módulo SPARQL y más concretamente en las clases que implementan las factorías de consultas. La herramienta dispone una aproximación a un comprobador de semántica, éste funciona, pero está demasiado acoplado al caso de uso particular. Esto requiere que por cada nueva fuente de datos adicional se requiera un diseño demasiado complejo y en general poco funcional.

Sería interesante elaborar un diseño para que a través de una sencilla configuración por parte del usuario se pudiera comprobar la correcta sintaxis y semántica para la fuente de datos para la que se genera la consulta.

De esta manera se conseguiría la mayor flexibilidad deseada para las ampliaciones deseadas y una mayor seguridad frente a los errores que puedan ser provocados por parte del usuario al introducir las consultas.

8.3 Hilos para recuperar la información

La implementación de la herramienta se ha llevado a cabo con un esquema de implementación secuencial para simplificar su desarrollo. Esto provoca que el rendimiento no sea el más adecuado.

En la herramienta encontramos una gran oportunidad para explotar la funcionalidad de la implementación de hilos. El software debe hacer consultas a un servicio externo, esto provoca que se quede esperando mientras que el servicio responde. El gran número de peticiones provoca que el tiempo perdido sea considerable.

Una gestión en paralelo para realizar tanto las peticiones como el rellenado de datos podría suponer una mejora sustancial en desempeño de la herramienta. Principalmente se debería estudiar si se puede dar alguna inconsistencia al construir el modelo.

8.4 Mejora del módulo RDF

Actualmente la clase que transforma los datos en documentos RDF cumple con los requisitos, puede ser ampliable y no es excesivamente difícil de ampliar. Sin embargo, la manera con la que se amplía puede ser abstraída para el usuario final. Ahora mismo se encuentra demasiado asociada a la clase.

Una posible manera que ha sido explorada durante el desarrollo de la herramienta es el uso de la reflexión que proporciona Java. Esto permitirá que el usuario no tuviera que modificar la clase sino un archivo de configuración.

El citado archivo de configuración podría estar codificado por ejemplo en JSON. De esta manera el documento contendría una raíz que sería el recurso principal y posteriormente cada nodo debería indicar el nombre del método proveedor del valor.

8.5 Almacenamiento de documentos RDF

Como ya fue explicado en el apartado limitaciones, el hecho que las fotos en el caso de uso, o en general cualquier otro conjunto de información sea muy extenso hace que se tengan que realizar muchas consultas sobre servicios online.

Para ofrecer un compromiso entre tiempo de ejecución y riqueza de los datos una posible solución sería almacenar los datos ya consultados anteriormente para que puedan ser recuperados en posteriores ejecuciones.

El diseño e implementación no es muy complejo a priori pues la librería Jena permite tanto el volcado como recuperación de documentos RDF. El principal esfuerzo seria el diseño del sistema de archivos y la coherencia de los documentos con el paso de tiempo.

8.6 Interfaz

El desarrollo de una interfaz atractiva daría un uso más comercial y cercano al usuario final. Ahora mismo la interfaz por medio de consola cumple su prometido y permite tanto realizar las consultas como mostrar los resultados.

El poder tener una interfaz, principalmente que pudiera ser online, por medio de una web permitiría una mayor accesibilidad de la herramienta y acercarse a lo que sería un buscador semántico configurable.

Este TFG puede ser tomado como una primera toma de contacto para un desarrollador, esto es debido a que otra mejora relacionada con una interfaz mucho más atractiva seria la posibilidad de realizar las ampliaciones por medio de esa misma interfaz.

8.7 Módulos

Las distintas representaciones de datos no siempre se van a encontrar en un documento JSON o accesibles por medio de un *endpoint*. Es por ello que se hace necesario implementar los diversos módulos para la recogida y conversión de datos.

Uno de los principales formatos en la que podemos encontrar la información es sin ningún formato, directamente en HTML. Esto requerirá de un gran esfuerzo que en este TFG no se ha tenido cabida. Se requiere que la implementación sea muy flexible puesto que existe un ecosistema de representación muy complejo.

Algunos otras fuentes de información si la información está codificada similar a JSON no requerirá tanto esfuerzo en el diseño, simplemente se deberán tener en cuenta las peculiaridades sintácticas y si existen librerías que soporten su tratamiento.

9 ANEXO

9.1 Índice de tablas

9.2 Índice de ilustraciones

10 BIBLIOGRAFÍA

[RDF] - Richard Cyganiak; David Wood; Markus Lanthaler(2014 Febrero 25) RDF 1.1 Concepts and Abstract Syntax. [Online] URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[RDF-INT] – JoshData (2014 Mayo 21) What is RDF and what is it good for? [Online] URL: https://github.com/JoshData/rdfabout/blob/gh-pages/intro-to-rdf.md#

[TURTLE] - Eric Prud'hommeaux; Gavin Carothers. (2014 Febrero 25) RDF 1.1 Turtle: Terse RDF Triple Language. [Online] URL:http://www.w3.org/TR/2014/REC-turtle-20140225/

[TRIG] - Gavin Carothers; Andy Seaborne (2014 Febrero 25) RDF 1.1 TriG RDF Dataset Language [Online] URL: http://www.w3.org/TR/2014/REC-trig-20140225/

[JSON-LD] - Manu Sporny; Gregg Kellogg; Markus Lanthaler(2014 Enero 16) JSON-LD 1.0: A JSON-based Serialization for Linked Data[Online] URL:http://www.w3.org/TR/2014/REC-json-ld-20140116/

[RDFS] - Dan Brickley; R.V. Guha (2014 Febrero 25) RDF Schema 1.1 [Online] URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[OWL] - Pascal Hitzler; Markus Krötzsch; Bijan Parsia;Peter F. Patel-Schneider; Sebastian Rudolph (2012 Diciembre 11) OWL 2 Web Ontology Language Primer (Second Edition) [Online] URL: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/

[DCAT] - Fadi Maali; John Erickson (16 Enero 2014) Data Catalog Vocabulary (DCAT) [Online] URL: http://www.w3.org/TR/2014/REC-vocab-dcat-20140116/

[SPARQL] - Steve Harris; Andy Seaborne (2013 Marzo 21) SPARQL 1.1 Query Language [Online] URL: http://www.w3.org/TR/sparql11-query/

[SPARQL-EX] Lee Feigenbaum; Eric Prud'hommeaux (2013 Mayo 30) SPARQL By Example [Online] URL: http://www.cambridgesemantics.com/semantic-university/sparql-by-example

[JENA] - The Apache Software Foundation. An Introduction to RDF and the Jena RDF API [Online] URL: https://jena.apache.org/tutorials/rdf_api.html

[GSON] - Inderjeet Singh, Joel Leitch, Jesse Wilson. Gson User Guide [Online] URL: https://sites.google.com/site/gson/gson-user-guide

[EXIF] – Kanzaki (2004 Febrero 16) Exif vocabulary workspace [Online] URL: http://www.w3.org/2003/12/exif/

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jun 25 18:18:07 CEST 2015
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)