PROYECTO FIN DE CARRERA

Título:	Desarrollo de un Framework HTML5 de Visualización y Consulta Semántica de Repositorios RDF
Autor:	Roberto Bermejo del Valle
Tutor:	Carlos Ángel Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Gregorio Fernández Fernández
Vocal:	Mercedes Garijo Ayestarán
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Marifeli Sedano Ruiz

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

PROYECTO FIN DE CARRERA

DESARROLLO DE UN FRAMEWORK HTML5 DE VISUALIZACIÓN Y CONSULTA SEMÁNTICA DE REPOSITORIOS RDF

Alumno: D. Roberto Bermejo del Valle

Tutor: D. Carlos Ángel Iglesias Fernández

Ponente: D. Carlos Ángel Iglesias Fernández

2014

Agradecimientos
A mis padres,
hermanos y a Sara

Agradecimientos

A mis padres. Gracias a ellos, a su apoyo y a las posibilidades brindadas, he llegado a ser quien soy hoy día. Tanto académicamente como personalmente.

A mis hermanos. Javi, compañero de profesión, gracias por la labor de guía en Teleco que has ejercido y por los empujones morales para acabar el PFC. Carlos, gracias por haber estado ahí también durante estos años.

A mi novia Sara. Gracias por el apoyo dado durante la carrera, por contrarrestar los días malos, y también por los ánimos en que acabase el PFC.

A mi tutor, Carlos Ángel Iglesias. Gracias por haberme ofrecido la realización de este PFC. Desde el primer momento me apoyaste y apostaste por mi. Sinceramente, muchas gracias por todo el conocimiento que he adquirido, por confiar en mi, y por la paciencia demostrada que has tenido al ver que se ha prolongado el mismo.

A los compañeros del Grupo de Sistemas Inteligentes (GSI). Han sido unos meses estupendos, en los que he aprendido con vosotros y siempre habéis estado ahí para echarme una mano en lo que hiciera falta.

Y en general, a todos los compañeros y amigos de universidad. ¡Gracias por estar ahí!

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

"Buen comienzo, agüero de buen término."

— Anónimo

En este primer capítulo se introduce el Proyecto Fin de Carrera realizado, presentando la motivación y necesidad de llevarlo a cabo, los objetivos marcados, conseguidos y la estructura de la memoria.

1.1. Motivación

Durante la última década la cantidad de datos disponibles en Internet ha crecido de manera exponencial. Ya en sus orígenes, con Tim Berners-Lee a la cabeza, se intentó incluir información semántica en la *World Wide Web* pero por diferentes causas tecnológicas esto no fue posible.

Pero antes de entrar en detalles, me gustaría plantear al lector la siguiente cuestión: ¿qué es la Web Semántica?

La Web Semántica (del inglés *semantic web*) es un conjunto de actividades desarrolladas en el seno de *World Wide Web Consortium* tendente a la creación de tecnologías para publicar datos legibles por aplicaciones informáticas.

Así pues, el concepto de Web Semántica implica una ampliación de la Web, en donde se añaden metadatos[1] semánticos a la ya conocida por todos *World Wide Web*. Este agregado describe el contenido, el significado y la relación que hay entre los datos disponibles. Los metadatos son proporcionados de manera formal, según estándares, para que los propios ordenadores puedan entender y procesar estos nuevos datos de manera automática.

El objetivo final de la Web Semántica es ampliar la interoperabilidad entre los agentes que constituyen la red de redes que es Internet. Los beneficios para el usuario final son claros: búsquedas más precisas y exactas de la información. Un reciente ejemplo de esto puede encontrarse en la nueva funcionalidad presentada por Google en su buscador denominada *Knowledge Graph* [1] (El Gráfico de Conocimiento) e integrada en *Google Now* en Android 4.1 *Jelly Bean*. El concepto tradicional de buscador que devuelve resultados ordenados por relevancia sintáctica se ve sustituido. Ahora las búsquedas que realices se entienden como cosas y conceptos, no solo palabras y por tanto los resultados ofrecidos son más completos ya que permite al usuario explorar las relaciones que ese concepto tiene con otros.

Por ejemplo, si buscamos "Madrid" podremos visualizar además de los resultados ya habituales, una segunda columna en la que aparece información básica sobre Madrid (con fuente la Wikipedia), un mapa, una serie de lugares de interés en la ciudad de Madrid y sugerencias de otras búsquedas relacionadas ("Comunidad de Madrid" en este caso). Esto posibilita el conocimiento exploratorio gracias a que, a día de hoy, el Gráfico de Conocimiento almacena más de 570 millones de elementos (lugares, cosas, personas...) además de 18 mil millones entre atributos y conexiones y sigue aumentando [2].

.

[1] del griego meta, son datos sobre datos

Figura 1.1: Búsqueda del término Madrid en Google

En el caso de *Google Now* en Android la funcionalidad es aún más sorprendente pues la lista de resultados se ve reemplazada por un único resultado: la respuesta exacta a tu consulta. Y todo esto gracias, entre otras tecnologías, a la Web Semántica.

Uno de los resultados más relevantes de la Web Semántica es la iniciativa *Linked Open Data* [3] que consiste en publicar de forma estructurada e interoperable datos de instituciones públicas y privadas, con el fin de que dicha información pueda ser consultada de forma distribuida. Así pues, *Linked Data* trata de conectar datos relacionados entre sí que previamente no lo estaban.

El éxito de esta iniciativa se hace patente con el siguiente dato: se han publicado más de 32 mil millones de tripletas RDF[2] en 2011. Las tripletas, explicadas en el Estado del Arte, son una de las maneras de representar una relación o sentencia RDF, mediante un sujeto, un predicado y un objeto. Pero indudablemente, para poder explotar las relaciones entre distintas tripletas, es conveniente la aplicación de técnicas de visualización que faciliten su comprensión. Este es uno de los objetivos primordiales del proyecto.

1.2. Objetivos

El objetivo principal del proyecto es ofrecer un entorno de visualización sencillo y fácilmente personalizable que ofrezca diferentes visualizaciones para la misma ontología sin necesidad de realizar una programación repetitiva por parte del usuario final.

A continuación se citan los diferentes objetivos marcados para este proyecto:

- Analizar el estado del arte y conocer en profundidad los estándares W3C (RDF, SPARQL y OWL) así como las herramientas de visualización ya existentes.
- Diseñar e implementar un Framework HTML5 de visualización de repositorios RDF.
- Integrar herramientas de automatización de proyectos y pruebas que garanticen la calidad del proyecto.
- Validar el proyecto en varios entornos. En concreto, será aplicado en el proyecto EPISTEME para la visualización de relaciones semánticas entre las empresas de la plataforma tecnológica INES; en el proyecto Financial Twitter Tracker para la visualización de la relación cronológica entre tweets de valores financieros y el sentimiento sobre dichos valores y en el proyecto Omelette para la visualización de las relaciones entre APIs.

[2] Resource Description Framework

1.3. Estructura de la memoria

Con el fin de realizar una introducción a los lectores a la presente memoria, se presenta a continuación la organización de sus capítulos.

La memoria está constituida por seis capítulos y cada uno de ellos está, a su vez, constituido por distintas secciones y apartados:

- Primer capítulo (1): se presenta este Proyecto Fin de Carrera exponiendo los motivos que han motivado su desarrollo y los objetivos marcados.
- Segundo capítulo (2): se proporciona una visión general relativa al mundo de la visualización de datos semánticos o *Linked Data*. Además, se introducen los estados del arte relativos a la Web semántica: RDF, OWL, SPARQL.
- Tercer capítulo (3): se desarrolla el estudio de los diferentes casos de uso y se detallan los requisitos y herramientas empleadas del proyecto a desarrollar.
- Cuarto capítulo (4): se detalla el diseño de la arquitectura del sistema realizado.
- Quinto capítulo (5): se muestran las distintas pruebas realizadas en entornos reales y los resultados de éstas.
- Sexto capítulo (6): se concluye el trabajo realizado, se citan las posibles mejoras del sistema y los trabajos futuros recomendados.

Capítulo 2

Estado del arte

"Primero resuelve el problema. Entonces, escribe el código."

— John Johnson

En este capítulo se presenta la base teórica sobre la que se sustenta este Proyecto Fin de Carrera. Se introducen los modelos estandarizados que constituyen la Web Semántica: RDF, OWL y SPARQL. Además, se analizan otras herramientas de visualización ya existentes para determinar qué funcionalidades ofrecen y el estado de desarrollo en el que se encuentran. Por último, se estudian los patrones de desarrollo Web actuales y se evalúan algunos *Frameworks* Web de desarrollo.

2.1. Introducción

La aparición de nuevas aplicaciones y plataformas en Internet tales como las redes sociales, hechos como la universalización de la banda ancha y el uso cada vez más generalizado de Internet en nuestro día a día ha propiciado que los datos, de muy distinto tipo disponibles en Internet, crezcan de manera exponencial.

Ante esta situación, se echa en falta una infraestructura sólida y estable que permita que la gestión, mantenimiento y recuperación de información no suponga un problema para los gestores de la información y para el usuario final consumidor de esta información.

La recuperación de información relevante resulta una tarea cada vez más compleja pues el conocimiento humano ya no está sólo determinado por unidades físicas de información. Éste se ha convertido en un conjunto de publicaciones electrónicas constituidas por imágenes, textos, vídeos, todos ellos con formatos heterogéneos, que conforman nuevas representaciones del conocimiento.

Desde el *World Wide Web Consortium* (W3C) [4] se impulsaron diferentes soluciones para mejorar la recuperación de información en Internet y ya son cada vez más los gobiernos, las entidades públicas y privadas que adaptan sus contenidos ya publicados en la Web incluyendo metadatos.

Los metadatos son estructuras que permiten describir distintos objetos de información distribuidos en la web, de tal forma que la búsqueda basada en estos metadatos mitiga el problema de la recuperación de información.

2.2. RDF: Resource Description Framework

Uno de estos lenguajes de marcado semántico es el *Resource Description Framework*[1] (Marco de Descripción de Recursos), un modelo de metadatos cuyo objetivo es mejorar la recuperación de información.

RDF surgió en 1997 en el seno del W3C. Se apoyó en trabajos previos de varios colectivos como otras iniciativas del W3C y, por supuesto, de los trabajos de la comunidad bibliotecaria en torno al Dublin Core (DC); uno de los modelos de metainformación que adoptó rápidamente la sintaxis RDF.

Desde Febrero de 1999, la especificación del modelo y la sintaxis de RDF eran una re-

[1] De ahora en adelante RDF

comendación del W3C [W3C-RDF-R], y su esquema era una propuesta de recomendación [W3C-RDFS-PR]. En 2004, RDF se convirtió en estándar y su esquema en una recomendación.

2.2.1. Especificación oficial

La especificación de RDF está recogida en los siguientes documentos catalogados como Recomendación por el W3C:

- *RDF Primer* [5]: Es una introducción de los conceptos básicos sobre los que se sustenta RDF. Proporciona el conocimiento básico para comprender RDF.
- *RDF Concepts and Abstract Syntax* [6]: En este documento se define la sintaxis abstracta en la que se basa RDF.
- *RDF/XML Syntax Specification (Revised)* [7]: Esta Recomendación define la sintaxis XML para RDF conocida como RDF/XML.
- *RDF Semantics* [8]: Explicación de la estructura y uso del vocabulario semántico que puede emplearse en RDF, como *RDF Schema*.
- *Vocabulary Description Language: RDF Schema* [9]: Esta Recomendación describe la extensión semántica de RDF, denominada *RDF Schema*. Es un lenguaje de propósito general y básico para la descripción de vocabularios.
- *RDF Test Cases* [10]: Se recogen casos de uso en los que se emplea RDF.

RDF resulta de gran utilidad para modelar la información que debe ser procesada por máquinas pues apenas resulta legible para humanos, pudiendo emplearse en distintos escenarios, como por ejemplo, en la recuperación de recursos en buscadores o agentes inteligentes.

2.2.2. Conceptos básicos

RDF está basado en la idea de identificar conceptos mediante los *Uniform Resource Identifiers*[2] (Identificador Uniforme de Recurso), y en describir los recursos en términos de propiedades simples y valores.

Esto permite que en RDF se puedan representar sentencias simples en forma de grafo, constituido por nodos y arcos que representan los recursos, sus propiedades y valores.

[2]De ahora en adelante URIs

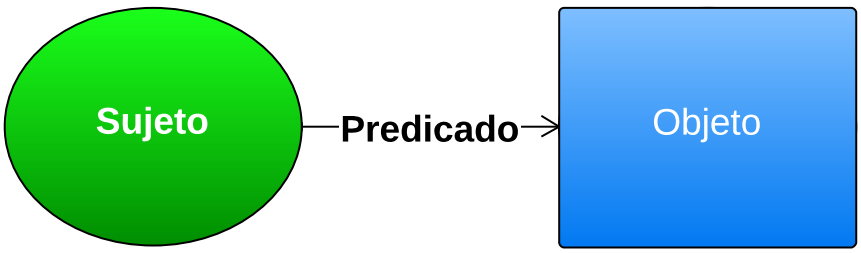

Figura 2.1: Esquema básico de una tripleta

Los sujetos y objetos son nodos, mientras que los predicados son arcos tal y como puede observarse en la figura 2.1, que representa una tripleta.

Con objeto de facilitar la comprensión del concepto al lector, veamos un ejemplo extraído de la especificación *RDF Primer* [5] en donde se muestra una serie de sentencias: "hay una persona identificada por el URI *http://www.w3.org/People/EM/contact#me*, cuyo nombre es Eric Miller, cuya dirección de correo electrónico es em@w3.org, y cuyo título es Dr.". Este ejemplo se representa en la figura 2.2:

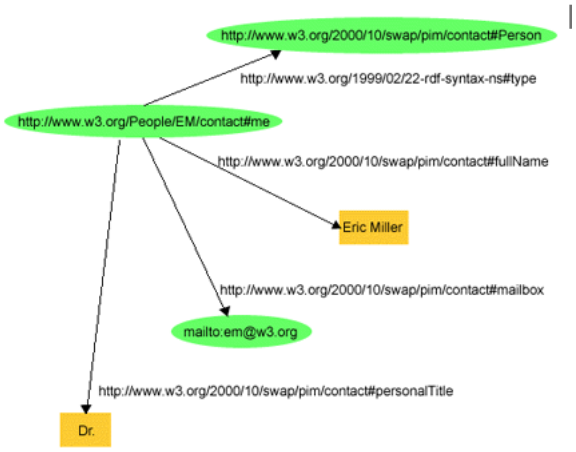

Figura 2.2: Grafo RDF que describe a Eric Miller

La figura 2.2 ilustra cómo RDF emplea los URIs para identificar:

- individuos, por ejemplo, Eric Miller, identificado por *http://www.w3.org/People/EM/contact#me*.
- clases de cosas, por ejemplo, Person, identificado por *http://www.w3.org/2000/10/swap/pim/contact#Person*.
- propiedades de estas cosas, por ejemplo, *mailbox*, identificado por *http://www.w3.org/2000/10/swap/pim/contact#mailbox*.
- valores de estas propiedades, por ejemplo, *mailto:em@w3.org* como el valor de la propiedad mailbox.

RDF también proporciona una sintaxis basada en XML, conocida como RDF/XML para representar estos grafos. A continuación una muestra de RDF en RDF/XML que corresponde al mismo grafo de la figura 2.2.

Código 2.1: RDF/XML que describe a Eric Miller

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
```

```
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Al igual que HTML, RDF/XML es también procesable por las máquinas. Mediante el uso de URIs se pueden establecer los ya habituales enlaces a través de la Web. Pero con una diferencia notable ya que los URIs empleadas en RDF pueden hacer referencia a cualquier cosa identificable, incluyendo cosas que pueden no ser recuperables de manera directa en la Web, como por ejemplo la persona Eric Miller.

El resultado es que además de describir cosas como páginas web, RDF también puede describir coches, negocios, personas... e incluso las propias propiedades de RDF.

2.2.3. El modelo RDF

Imaginemos ahora que queremos afirmar que alguien llamado John Smith ha creado una página web. En nuestro lenguaje esto lo podríamos expresar así:

http://www.example.org/index.html tiene un creador cuyo valor es John Smith

La declaración RDF en forma de tripleta sería la siguiente:

- el sujeto es la URL: *http://www.example.org/index.html*
- el predicado es la palabra creador
- y el objeto es la frase "John Smith"

Y haciendo uso de los URIs, la sentencia RDF podría formarse de esta manera:

- el sujeto: *http://www.example.org/index.html*
- el predicado: *http://purl.org/dc/elements/1.1/creator*
- y el objeto: *http://www.example.org/staffid/85740*

Y como se ha visto previamente, esta afirmación también puede ser expresada mediante un grafo con nodos y arcos tal y como se observa en la figura 2.3, en donde se puede distinguir:

- un nodo para el sujeto.
- un nodo para el objeto.
- un arco para el predicado desde el sujeto al objeto.

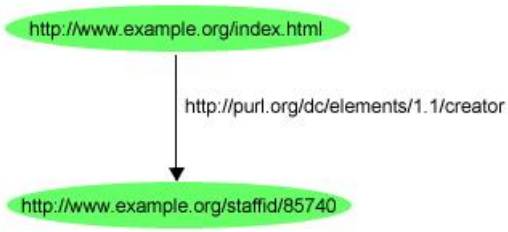

Figura 2.3: Grafo RDF simple

Dado el mismo sujeto, podemos añadir nuevas relaciones tal y como se aprecia en la figura 2.4. Por ejemplo:

http://www.example.org/index.html tiene una *creation-date* cuyo valor es August 16, 1999
http://www.example.org/index.html tiene un *language* cuyo valor es English

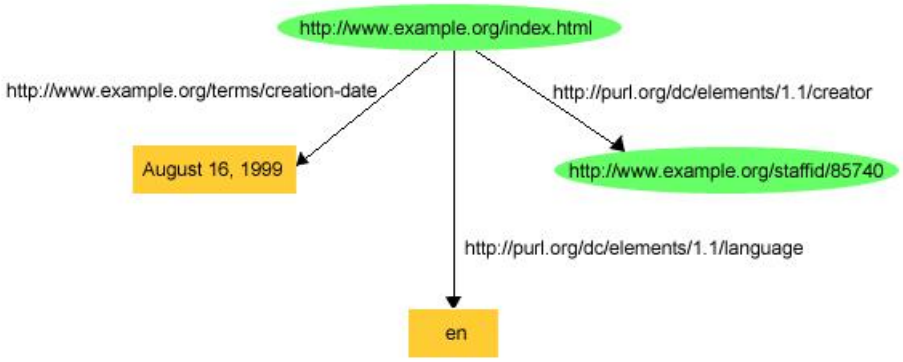

Figura 2.4: Grafo RDF con varias relaciones dado un sujeto

La figura 2.4 pone de manifiesto que los objetos declarados en sentencias o declaraciones RDF pueden ser URIs o valores constantes denominados literales.

Por lo tanto, los literales no pueden usarse como sujetos o predicados en sentencias RDF. A la hora de representar un grafo RDF, hay que tener en cuenta que los URIs se representan con elipses y los nodos que son literales se representan con cuadriláteros.

Según el contexto, a veces puede ser interesante evitar la representación mediante grafos. Una forma alternativa de representación es mediante el uso de las tripletas.

En la notación de tripletas, cada relación contenida en un grafo se escribe como una simple tripleta que incluye al sujeto, predicado y objeto correspondientes.

Por ejemplo, las 3 declaraciones mostradas en la figura 2.4 se representan de la siguiente forma:

Código 2.2: Tripletas

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740> .

<http://www.example.org/index.html> <http://www.example.org/terms/creation-date>
"August 16, 1999" .

<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/language>
"en" .
```

Como puede observarse, son dos maneras de representar la misma información; una manera gráfica y otra textual.

En el grafo el sujeto se representa una única vez y, en este caso, el sujeto debe incluirse en cada tripleta para poder especificar cuál es el sujeto de la misma.

Cabe destacar también la notación empleada en las tripletas. Los URIs se escriben generalmente entre ángulos. Pero esta manera de representar URIs puede generar tripletas muy largas y menos legibles para los humanos. La solución consiste en el empleo de prefijos, una manera de acortar estos URIs.

Las tres tripletas anteriores podrían representarse como se refleja a continuación:

Código 2.3: Tripletas con prefijos

```
prefix ex:, namespace URI: http://www.example.org/ .
prefix dc:, namespace URI: http://purl.org/dc/elements/1.1/ .
prefix exterms:, namespace URI: http://www.example.org/terms/ .
prefix exstaff:, namespace URI: http://www.example.org/staffid/ .

ex:index.html dc:creator exstaff:85740 .
ex:index.html exterms:creation-date "August 16, 1999" .
ex:index.html dc:language "en" .
```

De esta manera, *dc:creator* equivale a *<http://purl.org/dc/elements/1.1/creator>*; *dc:language* a *<http://purl.org/dc/elements/1.1/language>* y así sucesivamente.

Tal y como se puede apreciar la tripleta cuyo predicado es *dc:creator* tiene como objeto un URI y no un literal como cabría esperar (el nombre de John Smith).

La ventaja de este hecho es evidente pues al hacer referencia a un URI, éste podrá contener más información que un simple literal, pues se ha convertido en un recurso más identificable.

En este caso, el URI *exstaff:85740* hace referencia al número de empleado. Y se puede añadir toda la información que se desee sobre John Smith con sentencias RDF que tengan por sujeto ese URI, tal y como se representa en la figura 2.5.

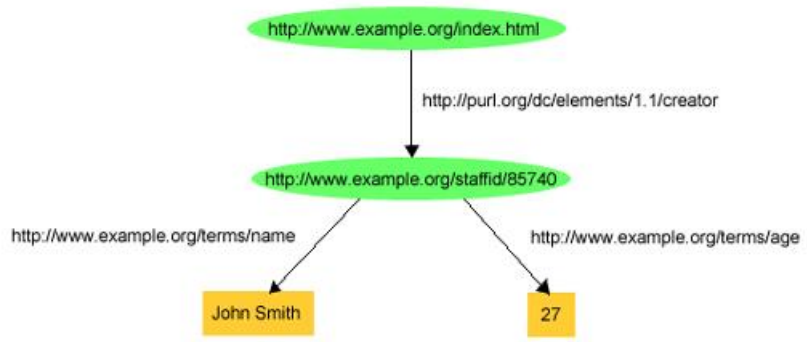

Figura 2.5: Información sobre John Smith

El uso del predicado *dc:creator* hace referencia de manera unívoca al atributo *creator* del conjunto de metadatos definidos en el *Dublin Core*, un conjunto de atributos utilizado de manera global para describir información de todo tipo.

Cualquier persona que conozca estos metadatos, podrá conocer el significado que otorga ese predicado a la relación entre el sujeto y el objeto. También se pueden desarrollar aplicaciones o programas que sean capaces de procesar tripletas y puedan distinguir las relaciones que se establecen gracias al uso de estos predicados predefinidos.

2.2.3.1. Nodos blancos

En ocasiones puede resultar de interés que algunos nodos intermedios no tengan un URI definido, convirtiéndose así en nodos blancos ya que nunca van a ser referenciados de manera directa y por tanto no necesitan un identificador üniversal".

Continuando con el ejemplo anterior, si se desea describir la dirección en la que vive John Smith, ésta se puede definir a través de estas tripletas:

```
exstaff:85740 exterms:address exaddressid:85740 .
exaddressid:85740 exterms:street "1501 Grant Avenue" .
exaddressid:85740 exterms:city "Bedford" .
exaddressid:85740 exterms:state "Massachusetts" .
exaddressid:85740 exterms:postalCode "01730" .
```

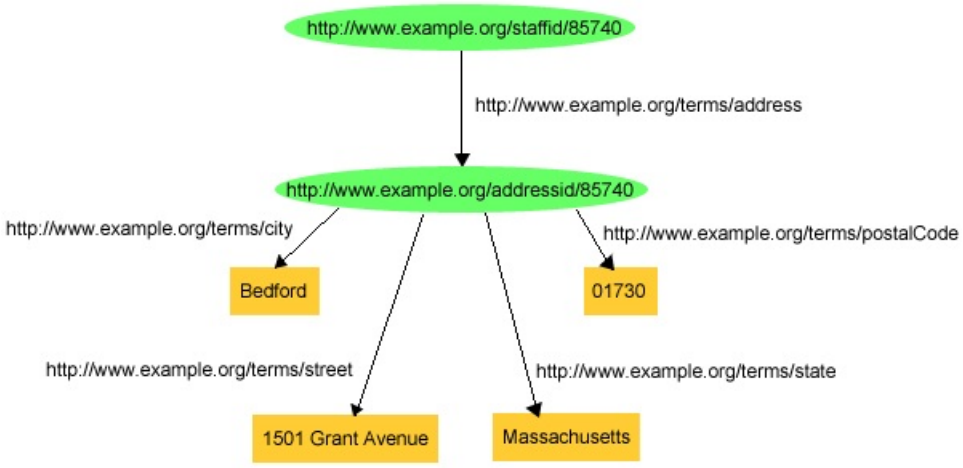

Figura 2.6: Desglose de la dirección de John Smith

Definido de esta manera, hemos generado un nodo intermedio con un identificador que probablemente nunca será referenciado de manera directa y por tanto no es necesario hacerlo así. Podemos emplear en su lugar un nodo blanco:

Código 2.4: Tripletas con un nodo blanco

```
exstaff:85740 exterms:address _:johnaddress .
_:johnaddress exterms:street "1501 Grant Avenue" .
_:johnaddress exterms:city "Bedford" .
_:johnaddress exterms:state "Massachusetts" .
_:johnaddress exterms:postalCode "01730" .
```

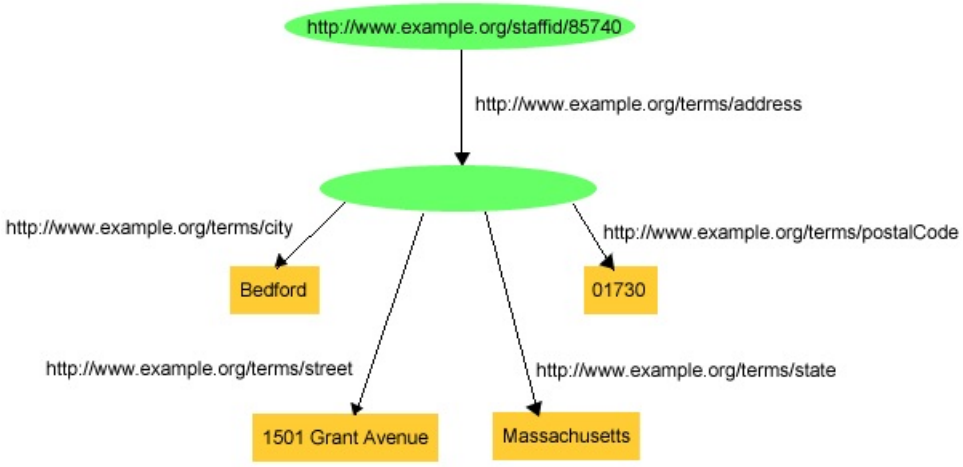

Figura 2.7: Dirección de John con un nodo blanco

A la vista de lo anterior, cabe mencionar que los nodos blancos se definen con esta sintaxis _:nombre. Es importante recalcar que los nodos blancos sólo pueden ser usados como sujetos u objetos en las tripletas.

2.2.3.2. Tipos de literales

Hasta ahora los literales empleados eran texto plano. En el ejemplo tratado hasta el momento convendría, por ejemplo, especificar que el objeto literal "27.[es] un número entero pues representa la edad de John Smith y que la fecha de creación de la página web es precisamente un objeto de tipo fecha:

```
prefix xsd:, namespace URI: http://www.w3.org/2001/XMLSchema# .

exstaff:85740 exterms:age "27"^^xsd:integer .
ex:index.html exterms:creation-date "1999-08-16"^^xsd:date .
```

Otros tipos de literales son: *xsd:string* o *xsd:boolean*.

2.2.4. Contenedores RDF

En determinados contextos puede ser necesario describir grupos de cosas, como por ejemplo: la lista de alumnos de un curso, los autores de un libro, etcétera.

Con este fin, RDF proporciona un vocabulario de contenedores que consiste en tres tipos de contenedores predefinidos.

El contenedor es el recurso que contiene a los elementos. Los miembros son el contenido, que pueden ser a su vez otros recursos (incluyendo nodos en blanco) o literales.

Los tres tipos de contenedores RDF se detallan a continuación:

- *rdf:Bag*: Se utiliza para indicar que un predicado tiene múltiples miembros en donde no es significativo el orden en el que se incluyen dichos miembros, pudiendo aparecer duplicados.
- *rdf:Seq*: Representa a un grupo de recursos o literales. Los miembros se pueden duplicar y el orden en el que se declaran si es significativo. Por ejemplo, una secuencia puede usarse para contener a un grupo de nombres ordenados alfabéticamente.

- *rdf:Alt*: Representa a un grupo de recursos o literales que son alternativas a un valor único de una propiedad. Por ejemplo, *rdf:Alt* podría usarse para describir el título de un libro en distintos lenguajes o para describir una lista de sitios alternativos en Internet en los que se puede encontrar un recurso determinado. Por tanto, la elección de cualquier valor alternativo es una elección correcta, aunque el valor por defecto es el que se declara en primer lugar.

Veamos a continuación un ejemplo con *rdf:Bag* en notación RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
         <rdf:Bag>
            <rdf:li rdf:resource="http://example.org/students/Amy"/>
            <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
            <rdf:li rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li rdf:resource="http://example.org/students/Phuong"/>
         </rdf:Bag>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

El código RDF/XML anterior representa la siguiente afirmación: *El curso 6.001 tiene los estudiantes Amy, Mohamed, Johann, Maria, y Phuong*. El curso se describe con el predicado *s:students*, cuyo objeto es un contenedor de tipo *rdf:Bag* que representa al grupo de estudiantes.

La visualización en forma de grafo se muestra en la figura **??**:

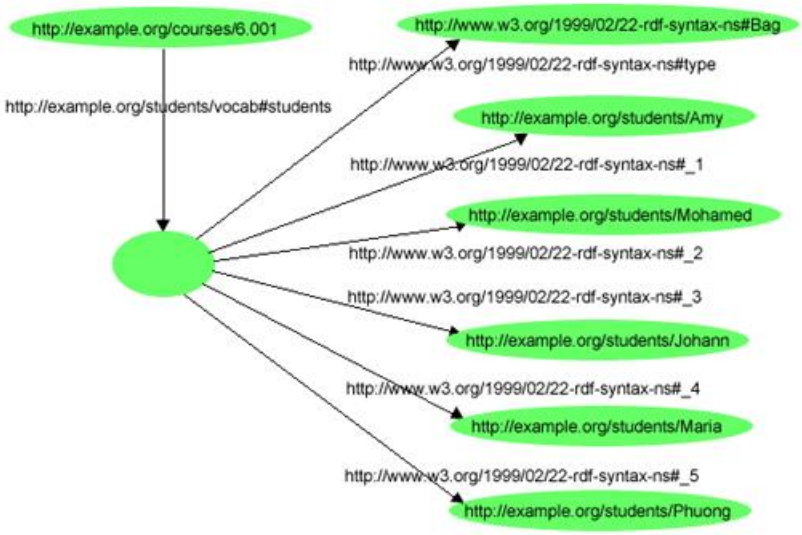

Figura 2.8: Ejemplo de contenedor *rdf:Bag*

Y a continuación, un ejemplo con *rdf:Alt* en notación RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

   <rdf:Description rdf:about="http://example.org/packages/X11">
      <s:DistributionSite>
         <rdf:Alt>
            <rdf:li rdf:resource="ftp://ftp.example.org"/>
            <rdf:li rdf:resource="ftp://ftp1.example.org"/>
            <rdf:li rdf:resource="ftp://ftp2.example.org"/>
         </rdf:Alt>
      </s:DistributionSite>
   </rdf:Description>
</rdf:RDF>
```

Este ejemplo representa las posibles ubicaciones de descarga en Internet de un paquete. La opción prioritaria (identificada por el predicado *rdf:_ 1*) es la que aparece en primer lugar dentro del contenedor *rdf:Alt*.

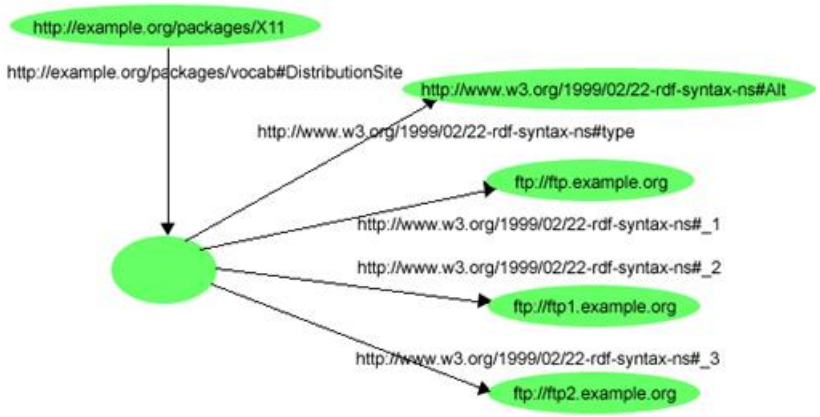

Figura 2.9: Ejemplo de contenedor *rdf:Alt*

Cabe destacar que los predicados para cada miembro son de la forma *rdf:_ n*, donde n es un número entero decimal más grande que cero: *rdf:_ 1*, *rdf:_ 2*, *rdf:_ 3*, y así sucesivamente, según el orden en el que se encuentren.

2.2.5. Colecciones RDF

La principal limitación de los contenedores RDF es que no existe la manera de acotarlos, es decir, concretar que "todos estos son los miembros del contenedor". Por tanto, un contenedor especifica a algunos recursos como miembros pero no asegura que otros recursos fuera de ese grafo puedan existir como miembros del mismo contenedor.

Esta carencia es la que suple las colecciones RDF: son agrupaciones de recursos representadas en forma de lista estructurada en un grafo RDF.

Esta lista se construye usando un vocabulario específico que consta del tipo *rdf:List*, las propiedades *rdf:first* y *rdf:rest*, y el recurso *rdf:nil*.

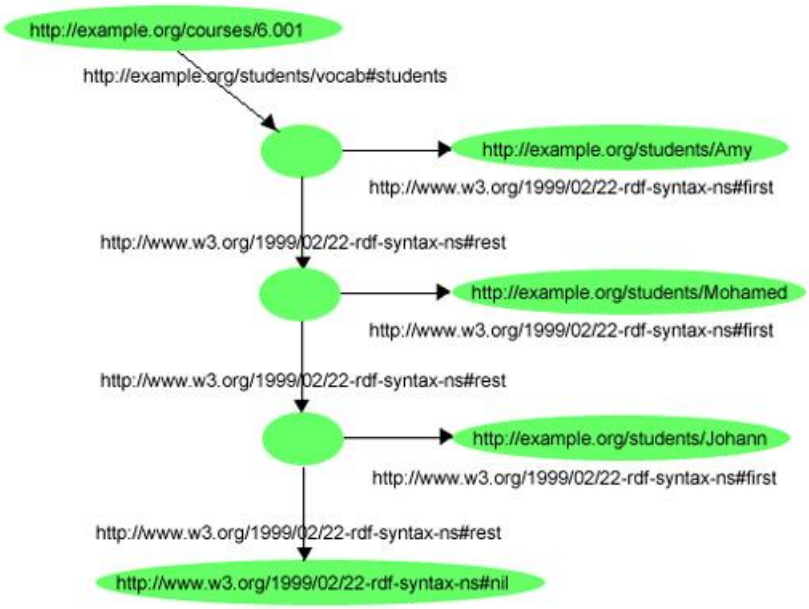

Figura 2.10: Colección RDF

En la figura 2.10 se representa la sentencia "Los estudiantes del curso 6.001 son Amy, Mohamed y Johann".

En este grafo cada miembro de la colección, como *s:Amy*, es el objeto del predicado *rdf:first* cuyo sujeto es un recurso (un nodo blanco en este caso) que representa una lista. Los enlaces entre sujetos que constituyen la lista se especifican gracias a la propiedad *rdf:rest*. El final de la lista se indica con un objeto de tipo *rdf:nil*.

En RDF/XML, una colección puede describirse por un elemento que tiene el atributo *rdf:parseType=Çollection"* (*rdf:parseType* indica que el contenido de este elemento se interpretará de forma especial) y que contiene un grupo de elementos anidados que constituye los miembros de la colección.

Este mismo ejemplo en sintaxis RDF/XML se declara así:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="\url{http://www.w3.org/1999/02/22-rdf-syntax-ns#}"
       xmlns:s="\url{http://example.org/students/vocab#}">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
        <s:students rdf:parseType="Collection">
               <rdf:Description rdf:about="http://example.org/students/Amy"/>
               <rdf:Description rdf:about="http://example.org/students/Mohamed"/>
               <rdf:Description rdf:about="http://example.org/students/Johann"/>
        </s:students>
   </rdf:Description>
</rdf:RDF>
```

Como puede apreciarse, al utilizar la propiedad *rdf:parseType=Çollection"* no se debe especificar ninguna propiedad específica como *rdf:rest* porque internamente ya se genera esa estructura.

Sin hacer uso de *rdf:parseType=Çollection"*, el código RDF/XML quedaría así:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:s="http://example.org/students/vocab#">

<rdf:Description rdf:about="http://example.org/courses/6.001">
   <s:students rdf:nodeID="sch1"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch1">
   <rdf:first rdf:resource="http://example.org/students/Amy"/>
   <rdf:rest rdf:nodeID="sch2"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch2">
   <rdf:first rdf:resource="http://example.org/students/Mohamed"/>
   <rdf:rest rdf:nodeID="sch3"/>
</rdf:Description>
```

```
<rdf:Description rdf:nodeID="sch3">
   <rdf:first rdf:resource="http://example.org/students/Johann"/>
   <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
</rdf:Description>
</rdf:RDF>
```

2.2.6. Herramientas de validación y visualizadores RDF

Existen varias herramientas de validación RDF vía web para poder descubrir así errores de manera eficaz y rápida en nuestros ficheros RDF/XML. El de uso más extendido, dado su origen, es el servicio en línea desarrollado por el W3C: el *RDF Validation Service*[3].

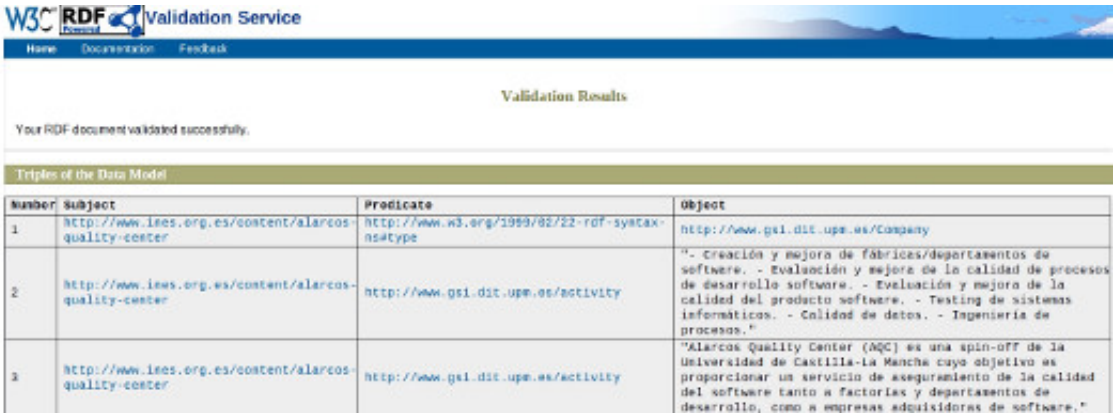

Figura 2.11: Validación RDF/XML de una empresa del proyecto INES. Se muestran las tripletas encontradas.

Este servicio, además de validar que el RDF en formato RDF/XML introducido es correcto, puede generar un grafo del RDF validado tal y como puede apreciarse en la figura 2.12.

Existen también entornos para visualizar relaciones y crear modelos RDF representados como grafos. Un buen ejemplo de ello es IsaViz[4], que resulta bastante completo.

[3] http://www.w3.org/RDF/Validator/
[4] http://www.w3.org/2001/11/IsaViz/

Figura 2.12: Gráfico generado a partir del RDF/XML analizado

2.3. RDF Schema

RDF Schema[5] (Esquema RDF) es una extensión semántica de RDF. Se puede pensar en un esquema como en una especie de diccionario que define los términos que se utilizarán en sentencias RDF para otorgarles significados específicos. Técnicamente, es un lenguaje primitivo de ontologías que proporciona los elementos básicos para la descripción de vocabularios.

El *Web Ontology Language* (Lenguaje de Ontología Web), tratado en la sección 2.4, es un lenguaje de ontologías más evolucionado que el aquí tratado.

Actualmente, RDFS es una recomendación publicada en Febrero de 2004 por la W3C [9].

2.3.1. Clases

Los recursos pueden clasificarse en grupos denominados clases. Los miembros de dichas clases se conocen como instancias de la clase.

Las clases son en sí mismas recursos, identificadas habitualmente por URIs y descritas usando propiedades RDF. La propiedad *rdf:type* se emplea para afirmar que un recurso es una instancia de una clase.

Los recursos siguientes son las clases principales que se definen como parte del vocabulario del esquema RDF:

- *rdfs:Resource*: Todas las cosas que se describan con expresiones RDF se denominan recursos, y son instancias de la clase *rdfs:Resource*. *rdfs:Resource* es una instancia de *rdfs:Class*.
- *rdfs:Class*: Corresponde con el concepto genérico de una categoría o tipo, semejante a la noción de Clase en los lenguajes de programación orientados a objetos. Las clases RDF pueden definirse para representar cualquier cosa: desde páginas web, personas, tipos de documentos, bases de datos a conceptos abstractos. La definición es recursiva pues rdf:Class es una instancia de *rdfs:Class*.
- *rdfs:Literal*: Es la clase de todos los valores literales como cadenas de texto y números enteros. *rdfs:Literal* es una instancia de *rdfs:Class* y es una subclase de *rdfs:Resource*.
- *rdfs:Datatype*: Es la clase que abarca los tipos de datos definidos en el modelo RDF. *rdfs:Datatype* es una instancia y una subclase de *rdfs:Class*. Cada instancia de *rdfs:Datatype* es una subclase de *rdfs:Literal*.

[5] De ahora en adelante RDFS

- *rdf:Property*: Es la clase de las propiedades RDF. *rdf:Property* es una instancia de *rdfs:Class*.

2.3.2. Propiedades

Las siguientes propiedades son instancias de *rdf:Property*.

- *rdfs:range*: indica que los objetos de un predicado son instancias de una o más clases. Por ejemplo, la tripleta P *rdfs:range* C indica que:
  - P es una instancia de la clase *rdf:Property*.
  - C es una instancia de la clase *rdfs:Class*.
  - Todos los objetos con predicado P son instancias de la clase C.
- *rdfs:domain* especifica el dominio de una propiedad P. Es decir, la clase de los recursos que aparecen como sujetos en las tripletas donde P es predicado. Por ejemplo, la tripleta P *rdfs:domain* C indica que:
  - P es una instancia de la clase *rdf:Property*.
  - C es una instancia de la clase *rdfs:Class*.
  - Todos los sujetos con predicado P son instancias de la clase C.
- *rdf:type* se usa para sentenciar que un recurso es una instancia de una clase. Una tripleta de la forma R *rdf:type* C indica que C es una instancia de *rdfs:Class* y R es una instancia de C.
- *rdfs:subClassOf* se emplea para especificar que todas las instancias de una clase son instancias de otra. Una tripleta de la forma C1 *rdfs:subClassOf* C2 indica que C1 y C2 son instancias de *rdfs:Class* y que C1 es una subclase de C2.

  La jerarquía de clases otorga la herencia de los dominios y rangos de las propiedades de una clase a sus subclases.
- *rdfs:subPropertyOf* permite definir jerarquías de propiedades.
- *rdfs:label* proporciona un nombre legible para los humanos dado el nombre de un recurso. En la tripleta R *rdfs:label* L, L sería la versión legible del nombre del recurso R.
- *rdfs:comment* proporciona una descripción legible para los humanos dado un recurso. En la tripleta R *rdfs:comment* L, L sería la descripción del recurso R.

2.3.3. Ejemplo práctico RDFS

A continuación se muestra un ejemplo en el que se emplean algunas de las propiedades y clases anteriormente descritas para reflejar su uso, dentro del siguiente escenario:

- Dog1 es un animal.
- Cat1 es un gato.
- Los gatos son animales.
- El zoo contiene a animales.
- Zoo1 contiene al Cat2.

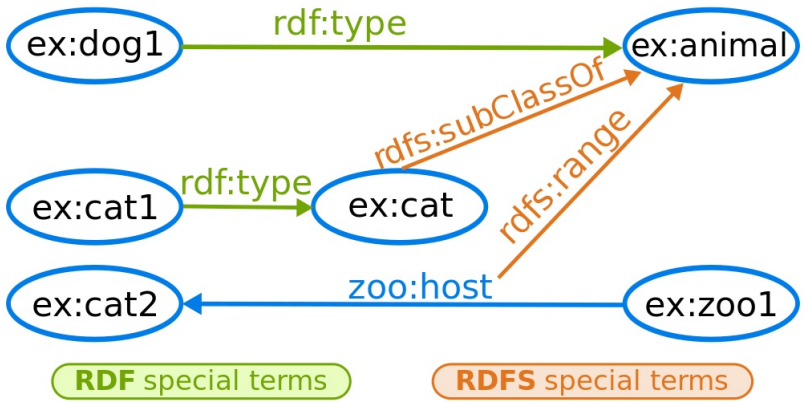

Figura 2.13: Ejemplo RDFS

La figura 2.13 es la representación de las siguientes tripletas:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:  <http://example.org/> .
@prefix zoo:  <http://example.org/zoo/> .

ex:dog1 rdf:type ex:animal .
ex:cat1 rdf:type ex:cat .
ex:cat rdfs:subClassOf ex:animal .
zoo:host rdfs:range  ex:animal .
ex:zoo1 zoo:host  ex:cat2 .
```

2.3.4. Resumen

A modo de resumen, se facilita en las siguientes tablas las clases y propiedades RDF analizadas.

2.3.4.1. Clases RDF

Nombre de la Clase	Comentario
rdfs:Resource	La clase de los recursos.
rdfs:Literal	La clase de los literales, por ejemplo, cadenas de texto y números enteros.
rdf:XMLLiteral	La clase de los valores literales XML.
rdfs:Class	La clase de las clases.
rdf:Property	La clase de las propiedades RDF.
rdfs:Datatype	La clase de los tipos de datos RDF.
rdf:Statement	La clase de las declaraciones RDF.
rdf:Bag	La clase de los contenedores desordenados.
rdf:Seq	La clase de los contenedores ordenados.
rdf:Alt	La clase de los contenedores de alternativas.
rdfs:Container	La clase de los contenedores RDF.
rdfs:ContainerMembership	La clase de las propiedades de los miembros de contenedores, *rdf:_1*, *rdf:_2*...
rdf:List	La clase de las listas RDF.

Tabla 2.1: Clases RDF

2.3.4.2. Propiedades RDF

Nombre de la Propiedad	Comentario	Dominio	Rango
rdf:type	El sujeto es una instancia de una clase.	*rdfs:Resource*	*rdfs:Class*
rdfs:subClassOf	El sujeto es una subclase de una clase.	*rdfs:Class*	*rdfs:Class*
rdfs:subPropertyOf	El sujeto es una subpropiedad de una propiedad.	*rdf:Property*	*rdf:Property*
rdfs:domain	Un dominio de la propiedad del sujeto.	*rdf:Property*	*rdfs:Class*
rdfs:range	Un rango de la propiedad del sujeto.	*rdf:Property*	*rdfs:Class*
rdfs:label	Un nombre del sujeto legible por los humanos.	*rdfs:Resource*	*rdfs:Literal*
rdfs:comment	Una descripción del sujeto.	*rdfs:Resource*	*rdfs:Literal*
rdfs:member	Un miembro del sujeto.	*rdfs:Resource*	*rdfs:Resource*
rdf:first	El primer elemento en la lista RDF.	*rdf:List*	*rdfs:Resource*
rdf:rest	El resto de la lista RDF después del primer elemento.	*rdf:List*	*rdf:List*
rdfs:seeAlso	Más información sobre el sujeto.	*rdfs:Resource*	*rdfs:Resource*
rdfs:isDefinedBy	La definición del sujeto.	*rdfs:Resource*	*rdfs:Resource*

rdf:value	Propiedad idiomática usada para valores estructurados.	*rdfs:Resource*	*rdfs:Resource*

Tabla 2.2: Propiedades RDF

2.4. OWL: Web Ontology Language

El *Web Ontology Language*[6] (Lenguaje de Ontología Web) está especialmente diseñado para aplicaciones en donde se requiere procesar información, en oposición a las situaciones en donde la información únicamente se presenta a los humanos.

OWL puede usarse para representar de manera explícita el significado de términos en vocabularios y las relaciones entre esos términos. La representación de estos términos y las relaciones entre ellos se denomina ontología.

OWL proporciona más funcionalidades que XML, RDF y RDFS. Es una revisión del lenguaje de ontologías web DAML+OIL[7] ya que subsana todos los errores aprendidos consecuencia del uso y explotación de DAML+OIL.

El lenguaje OWL está definido en un conjunto de documentos, cada uno de los cuales tiene un propósito diferente y está orientado a usuarios diferentes. Según el grado de complejidad técnica, ordenados de menor a mayor complejidad, encontramos los siguientes documentos:

- *OWL Overview* [11]: Ofrece una introducción simple a OWL para proporcionar una lista de características del lenguaje con una breve descripción.
- *OWL Guide* [12]: Demuestra el uso del lenguaje OWL mediante un ejemplo. Ofrece también un glosario de la terminología usada en estos documentos.
- *OWL Reference* [13]: Ofrece una descripción sistemática y resumida de todas las primitivas de modelado OWL.
- *OWL Semantics and Abstract Syntax* [14]: Es la normativa final.
- *OWL Web Ontology Language Test Cases* [**?**]: Contiene un conjunto de casos de prueba.
- *OWL Use Cases and Requirements* [15]: Contiene un conjunto de casos de uso y recopila un conjunto de requisitos.

El primer recurso que se necesita aplicar a RDF para hacer de la Web Semántica una realidad es un lenguaje de ontologías que pueda describir formalmente el significado de la terminología usada en los documentos web.

[6] De ahora en adelante OWL

[7] Más información: `http://www.w3.org/TR/daml+oil-walkthru/`

OWL forma parte de las sucesivas Recomendaciones del W3C relacionadas con la Web Semántica, resumidas a continuación:

- XML proporciona una sintaxis superficial para documentos estructurados, pero no impone restricciones semánticas en el significado de dichos documentos.
- *XML Schema* es un lenguaje para restringir la estructura de los documentos XML y también mejora XML incluyendo tipos de datos.
- RDF es un modelo de datos para objetos (recursos") y para las relaciones entre ellos. Proporciona una semántica simple y puede ser representado con sintaxis XML.
- *RDF Schema* es un vocabulario para describir propiedades y clases de recursos RDF.
- OWL es una evolución ya que añade más vocabulario para describir propiedades y clases. Por ejemplo relaciones entre clases, cardinalidad, igualdad, etcétera.

2.4.1. Las tres variantes OWL

Como se comentó anteriormente, OWL tiene en consideración las diferentes necesidades de los usuarios, ofreciendo por tanto tres variantes diseñadas según sus necesidades:

- *OWL Lite* da soporte a aquellos usuarios que necesitan una jerarquía básica de clasificación y restricciones simples. Por ejemplo, aunque soporta las limitaciones cardinales, solo permite los valores 0 o 1. Tiene una complejidad formal menor que OWL DL.
- *OWL DL* da soporte a aquellos usuarios que quieren las máximas posibilidades teniendo en cuenta las limitaciones computacionales de tiempo y recursos.
- OWL Full es la opción elegida por aquellos que quieren la máxima libertad sintáctica sin garantías computacionales. Es improbable que algún software de razonamiento o aplicación de soporte a todas las características ofrecidas por OWL Full.

Cada uno de estas variantes es una extensión de su predecesor. El conjunto de relaciones siguiente es correcto pero no su inversa:

- Cada ontología OWL Lite es una ontología OWL DL.
- Cada ontología OWL DL es una ontología OWL Full.
- Cada conclusión válida en OWL Lite es una conclusión válida en OWL DL.

- Cada conclusión válida en OWL DL es una conclusión válida en OWL Full.

Con el propósito de introducir al lector en el concepto de OWL esta sección se centra en el uso de OWL Lite que, aunque tenga más limitaciones que OWL DL u OWL Full, con OWL Lite ya se pueden definir relaciones jerárquicas entre conceptos aplicando una menor complejidad formal y facilitando así su comprensión.

2.4.2. Características e identificadores

2.4.2.1. Características que tienen que ver con igualdad o desigualdad en OWL Lite

- *equivalentClass*: Dos clases pueden ser equivalentes y por tanto, tienen las mismas instancias. La equivalencia puede usarse para crear clases de sinónimos. Por ejemplo, Coche puede ser una *equivalentClass* de Automóvil. Por tanto, una aplicación puede determinar que una instancia de Coche es también una instancia de Automóvil y viceversa.

- *equivalentProperty*: Dos propiedades pueden ser equivalentes. La equivalencia en este caso puede ser usada para crear propiedades que sean sinónimas. Por ejemplo, si tieneLider es *equivalentProperty* de tieneCabezilla, una aplicación determinará que si X está relacionado con Y por la tieneLider, también lo estará con tieneCabezilla y viceversa. Además, tieneLider es una subpropiedad de tieneCabezilla y tieneCabezilla también lo es de tieneLider.

- *sameAs*: Dos identidades pueden ser declarados como el mismo recurso. Por ejemplo, el individuo Fernando puede ser declarado como FernandoAlonso.

- *differentFrom*: Lo contrario a *sameAs*. Establecer de manera explícita que los individuos son diferentes puede ser importante cuando usamos lenguajes tales como OWL (y RDF) ya que éstos no asumen que los individuos tienen un único nombre.

- *AllDiferent*: En conjuntos de N individuos, no hace falta declarar varias sentencias *differentFrom*. Basta definir a los individuos como en el siguiente ejemplo:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <vin:WineColor rdf:about="#Red" />
    <vin:WineColor rdf:about="#White" />
    <vin:WineColor rdf:about="#Rose" />
  </owl:distinctMembers>
```

```
</owl:AllDifferent>
```

2.4.2.2. Identificadores especiales en OWL Lite que se usan para proporcionar información relativa a las propiedades y sus valores

- *inverseOf*: Una propiedad puede ser definida como la inversa de otra. Si la propiedad P1 es la inversa de P2, entonces si X está relacionado con Y por la propiedad P2, Y está relacionado con X por la propiedad P1. Por ejemplo, si tieneHijo es el inverso de tienePadre y Luis tienePadre Antonio, entonces una aplicación puede deducir que Antonio tieneHijo Luis.

- *TransitiveProperty*: Si una propiedad P es transitiva, entonces si el par (x,y) es una instancia de la propiedad transitiva P, y el par (y,z) es una instancia de P, entonces el par (x,z) es también una instancia de P. Por ejemplo, si Ancestro es una propiedad transitiva, y si Mayte es un Ancestro de Paloma - es decir, (Mayte, Paloma) es una instancia de la propiedad Ancestro - y Paloma es un Ancestro de Sara - es decir, (Paloma,Sara) es una instancia de la propiedad Ancestro - una aplicación puede deducir que Sara es un Ancestro de Mayte.

- *SymmetricProperty*: Si una propiedad P es simétrica, si el par (x,y) es una instancia de la propiedad P, entonces el par (y,x) es también una instancia de P. Por ejemplo, la propiedad Amigo podría ser simétrica. De este modo, si Francisco es Amigo de Carlos, la aplicación podría deducir que Carlos es Amigo de Francisco.

- *FunctionalProperty*: Una propiedad podría tener, por sus características y el contexto, únicamente un solo valor o ninguno. Por ejemplo, la propiedad Marido puede definirse como una *FunctionalProperty*. Como consecuencia, una instancia de Mujer puede tener un único Marido o no tener ninguno.

- *InverseFunctionalProperty*: Si una propiedad P se declara como *InverseFunctionalProperty*, el sujeto S ligado a un objeto O a través de esta propiedad P es único. Por ejemplo, el número de afiliación a la Seguridad Social es una propiedad con estas características ya que éste es único para cada persona. Una aplicación podrá deducir que dos instancias de Persona no pueden tener el mismo número de Seguridad Social y también que si dos instancias de Persona tienen el mismo número, entonces esas dos instancias se refieren al mismo individuo.

2.5. SPARQL

SPARQL es un acrónimo recursivo del inglés *SPARQL Protocol and RDF Query Language*. Se trata de un lenguaje estandarizado para la consulta de grafos RDF y que por tanto nos permitirá extraer información y conocimiento de la Web Semántica. Se constituyó como Recomendación oficial del W3C el 15 de Enero de 2008.

En un principio SPARQL únicamente incorpora funciones para la recuperación de grafos RDF. Sin embargo, algunas propuestas como SPARUL[8] (SPARQL/Update) de Hewlett-Packard también incluyen operaciones para otras funciones como la creación, modificación y borrado de datos en grafos RDF. Y precisamente, a día de hoy se ha integrado en lo que el W3C ha denominado *SPARQL 1.1 Update*, una propuesta de recomendación con estos fines.

El lenguaje SPARQL permite que el usuario se centre en la recuperación de información, abstrayéndose así del *backend* empleado (parámetros como la versión y el tipo de la Base de Datos, etcétera).

La especificación global de SPARQL está definida en tres especificaciones complementarias:

- *SPARQL Query Language for RDF*
- *SPARQL Query Language for RDF* [16]: Documento que explica la sintaxis de uso de SPARQL.
- *SPARQL Protocol for RDF* [17]: Documento técnico para desarrolladores que quieran conocer el funcionamiento del protocolo.
- *SPARQL Query Results XML Format* [18]: Descripción del formato utilizado en las respuestas XML que generan las consultas SELECT y ASK.

En el caso del proyecto aquí tratado, interesa conocer la sintaxis para poder formular las peticiones deseadas.

2.5.1. Componentes

Las consultas SPARQL pueden contener tres tipos de datos:

[8] http://en.wikipedia.org/wiki/SPARUL

- URIs: para especificar los recursos.
- Variables: precedidas por ¿", sirven para asignar valores de manera temporal, aplicar filtros o recuperar resultados.
- Literales: delimitados entre son cadenas de texto.

2.5.2. Sintaxis y ejemplos

SPARQL tiene una gran semejanza a otros lenguajes como SQL con la típica sintaxis basada en cláusulas como SELECT, FROM, WHERE y ORDER BY.

La estructura de una consulta SPARQL es la siguiente:

```
PREFIX pref1, pref2, ..., prefX
OPERACION var1, var2, ..., varX
WHERE {condiciones}
```

En PREFIX se definen los prefijos (ya explicados anteriormente en el fragmento de Código 2.3) que acortan los URIs usados en la consulta.

En el bloque OPERACION se especifica la operación a realizar sobre las variables que queremos obtener como resultado a la consulta.

Finalmente, dentro del bloque WHERE se detallan las condiciones que deben cumplir las tripletas (con estructura sujeto, predicado y objeto) para que éstas puedan ser devueltas al hacer la consulta.

Cuatro son los tipos de operaciones que pueden realizarse mediante las consultas SPARQL. A continuación se introduce cada uno de ellos acompañado de un pequeño ejemplo para facilitar la comprensión:

- SELECT: Es la operación más común. Devuelve las variables que se le indican para las tripletas que coinciden con la condición dentro del bloque WHERE. Al igual que en SQL, es posible especificar un carácter * para representar todos los elementos. Ejemplo:

  Datos

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
```

```
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

Consulta

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

Respuesta

```
name     mbox
"Johnny Lee Outlaw"  <mailto:jlow@example.com>
"Peter Goodguy"   <mailto:peter@example.org>
```

Como puede observarse, el nodo blanco identificado por _:c no aparece en la respuesta ya que carece de propiedad *foaf:name* y por tanto no cumple con la condición ubicada dentro del bloque WHERE.

Podría filtrarse también por un literal en concreto. Por ejemplo, tomando como base los mismos datos anteriores, vamos a buscar la dirección de correo electrónico de "Peter Goodguy":

Consulta

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?mbox
WHERE
  { ?x foaf:name "Peter Goodguy" .
    ?x foaf:mbox ?mbox }
```

Respuesta

```
mbox
<mailto:peter@example.org>
```

- CONSTRUCT: esta operación devuelve un grafo RDF construido según el patrón especificado. Ejemplo:

  Datos

```
@prefix org: <http://example.com/ns#> .

_:a org:employeeName "Alice" .
_:a org:employeeId 12345 .

_:b org:employeeName "Bob" .
_:b org:employeeId 67890 .
```

Consulta

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX org: <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE { ?x org:employeeName ?name }
```

Respuesta

```
@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

Respuesta en notación RDF/XML

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    >
  <rdf:Description>
    <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
    <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

- ASK: devuelve un valor verdadero o falso según si se ha encontrado o no coincidencia en la búsqueda según el patrón definido. Ejemplo:

  Datos

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:homepage <http://work.example.org/alice/> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
```

Consulta

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

ASK { ?x foaf:name "Alice" }
```

Respuesta

```
yes
```

En cambio, si hacemos la siguiente consulta:

Consulta

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

ASK { ?x foaf:name "Alice" .
      ?x foaf:mbox <mailto:alice@work.example> }
```

Respuesta

```
no
```

No se encuentra coincidencia ya que en los datos no se especifica la dirección de correo electrónico de Alice.

- DESCRIBE: el resultado es un grafo RDF describiendo los recursos relacionados dado un URI. El ejemplo más sencillo es el siguiente:

```
DESCRIBE <http://example.org/resource>
```

Volviendo a las consultas SPARQL y a su analogía con SQL, la consulta SPARQL más sencilla sería la siguiente:

```
SELECT ?sujeto ?predicado ?objeto
WHERE {
   ?sujeto ?predicado ?objeto .
}
```

En SQL, significaría un SELECT * FROM [TABLA].

Para concluir esta introducción a SPARQL, realicemos una consulta a la dbpedia[9], una versión estructurada de Wikipedia que pone a disposición de todos un punto de acceso SPARQL al que realizar consultas[10]:

```
SELECT *

WHERE {
  ?company a <http://dbpedia.org/ontology/Organisation> .
  ?company <http://dbpedia.org/ontology/foundationPlace>
  <http://dbpedia.org/resource/Madrid> .
}
```

Esta consulta devolvería todas las empresas fundadas en Madrid.

2.6. Visualización de Linked Data

2.6.1. Introducción

Permítame el lector comenzar esta introducción con un refrán popular: una imagen vale más que mil palabras. Es indudable que la visualización de una imagen puede proporcionar la misma información que una explicación de muchas palabras. En el ámbito particular del *Linked Data* en el que la unidad más básica es una tripleta, este hecho es más patente aún pues aplicar la visualización gráfica del *Linked Data* nos permitirá entender de una manera más intuitiva, sencilla y eficiente las relaciones existentes entre los datos.

Los requisitos para una experiencia de visualización completa requieren de múltiples opciones y perspectivas para representar los datos incrementando así la usabilidad de la aplicación, pudiendo representar los datos y sus relaciones según convenga en cada caso. Por contra, esto requiere de manera intrínseca que el usuario conozca los datos que está

[9] http://dbpedia.org/
[10] http://dbpedia.org/sparql

manejando y su estructura, tarea no siempre posible.

Las mejores técnicas de visualización no tienen por qué ser complejas. Algunas de estas técnicas son gráficos de barras, circulares o histogramas... que ofrecen análisis estadísticos simples o tendencias.

Las representaciones de árboles en 2D y 3D se emplean para mostrar datos estructurados con jerarquías (como ontologías y redes).

Podemos encontrar otros tipos de visualización tales como: matrices, líneas temporales, mapas, nubes de etiquetas (con gran popularidad aunque potencial reducido), *gauges*, etcétera. Cabe destacar que cada tipo de visualización se ajusta mejor a unos contenidos u otros. Por ejemplo, los gráficos de barras o los gráficos circulares son útiles cuando hay pocos datos que representar. Si el número es elevado, será mejor usar otras técnicas como los árboles.

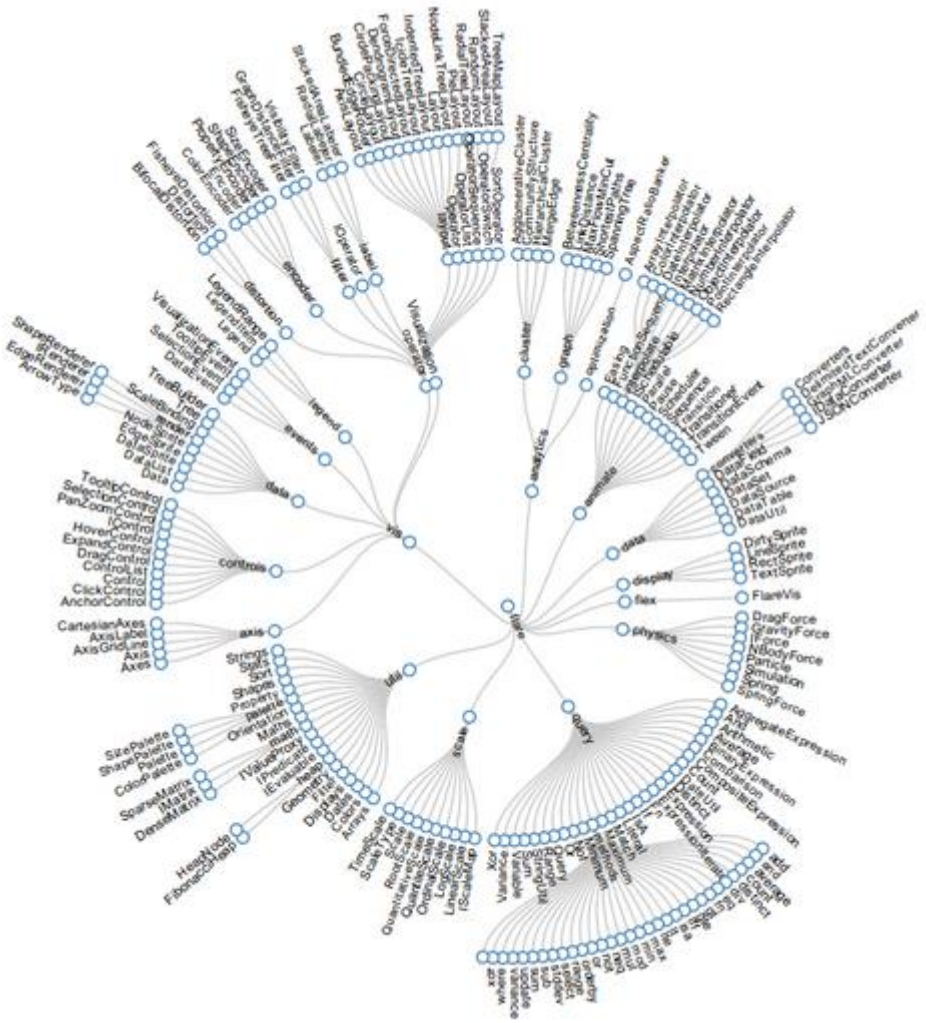

Figura 2.14: Árbol en 2D

2.6.2. Requisitos de diseño

Según un estudio de la Universidad Estatal de Pensilvania [19], a la hora de diseñar una herramienta de visualización de *Linked Data* debemos tener en cuenta algunos criterios o requisitos básicos a seguir para que la herramienta diseñada resulte de utilidad, de manera que el usuario entienda y capte la máxima información presente en los datos visualizados.

Dependiendo del grado de sofisticación de la interfaz de usuario podemos encontrar requisitos más duros o laxos. En nuestro caso, interesa que sea una aplicación sencilla para usuarios no expertos en el ámbito, sin sacrificar la calidad de los datos semánticos generados, y asegurando la correcta interconexión de los datos que estén relacionados.

A modo de resumen, estos serían los requisitos que una herramienta de visualización básica con los objetivos anteriormente mencionados debe proporcionar:

- Navegación intuitiva a través de las grandes cantidades de datos presentes mediante búsqueda semántica u otros métodos.
- Descubrimiento de conocimiento exploratorio, esto es, que las relaciones entre los distintos conceptos sean visualizadas y que podamos empezar visualizando un término y acabemos en otro distinto pero éste mantenía relación con él (directa o indirectamente).
- Soporte avanzado de consultas: desde técnicas de filtrado a consultas básicas de términos indexados.
- Análisis detallados de las regiones de interés para el usuario.
- Poder exportar los datos a un formato estándar que permita que otros usuarios o aplicaciones puedan reutilizarlos.
- Presentar los resultados del análisis a diferentes audiencias.

2.6.3. Resumen de herramientas

Las herramientas de visualización ya existentes pueden clasificarse en herramientas basadas únicamente en texto y en herramientas que ofrecen distintas opciones de visualización.

Dentro de las primeras podemos encontrar herramientas que proporcionan buenos resultados para los usuarios expertos; no para el usuario medio aunque estas herramientas puedan incluir listas, tablas o imágenes.

Las herramientas que ofrecen opciones de visualización son mucho más potentes de cara al usuario no experto por dos motivos principalmente:

- La visualización proporciona un mayor rango de posibilidades para mostrar los atributos tan heterogéneos de los datos *Linked Data* de manera que la representación sea más efectiva.

- Permite aprovechar la capacidad avanzada de percepción que tenemos los seres humanos reduciendo así la carga cognitiva. Como se comentó previamente "una imagen vale más que mil palabras", especialmente cuando hay que realizar un análisis de datos a gran escala.

Ejemplos: IsaViz[11] o RelFinder[12].

2.7. Patrones y Frameworks de desarrollo Web

Con la llegada de nuevas tecnologías como HTML5 y modelos de programas asíncronos (Node.js), las aplicaciones web modernas tienen la posibilidad de comportarse como aplicaciones de escritorio, dejando atrás las restricciones tradicionales del protocolo HTTP.

Como consecuencia, varios patrones de diseño, tanto añejos como recientes han cobrado importancia para facilitar el desarrollo y generar aplicaciones de fácil mantenimiento y extensibles.

Dados los requisitos técnicos impuestos por la herramienta de visualización objeto de desarrollo: interfaz dinámica, manejo intenso de datos, creación de gráficas, etcétera; es conveniente emplear algún *Framework* web que haga posible usar un patrón que separe el Modelo de los datos de la Vista (es decir, la interfaz de usuario).

Ante esta necesidad surgen las siguientes alternativas de patrones:

2.7.1. Patrón MVC

Este patrón posee tres componentes:

- La Vista: Es la interfaz de usuario.
- El Modelo: Los datos que la vista está mostrando.
- El Controlador: Contiene la lógica que modifica al modelo dependiendo de la acción que se ejecute desde la interfaz de usuario y viceversa.

[11] http://www.w3.org/2001/11/IsaViz/
[12] http://www.visualdataweb.org/relfinder.php

En este caso, el Controlador no conoce nada de la Vista, y la idea es que la Vista pueda estar relacionada con distintos Controladores (por ejemplo, dependiendo del usuario que ha iniciado sesión en el sistema). Gracias a esto, el mismo Controlador puede ser usado por múltiples Vistas.

La vista se suscribe a los cambios que se producen en el Modelo y por lo tanto, los datos (visualizados y contenidos en el Modelo) están sincronizados. Una de las desventajas del patrón MVC es que es complicado ejecutar tareas de pruebas.

Ejemplo de uso: El usuario presiona un botón en la interfaz y se genera un evento que llega al Controlador. El controlador modifica el valor en el Modelo. Y este cambio en el Modelo se traslada automáticamente a la Vista (cambiando por ejemplo el color de fondo de la página web).

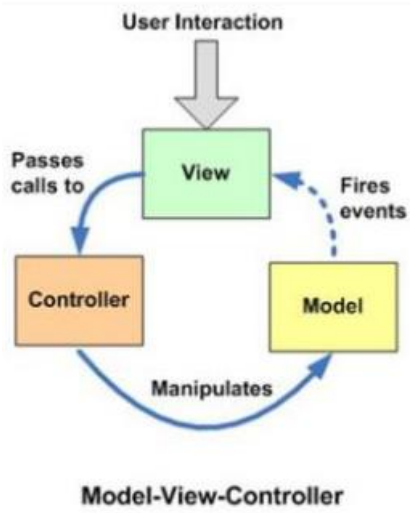

Figura 2.15: Patrón MVC

2.7.2. Patrón MVP

De nuevo se distinguen tres componentes. Pero las dependencias entre ellos cambian tal y como puede observarse en el diagrama 2.16:

La figura del Controlador se sustituye por el Presentador, encargado de actualizar los cambios hechos en el Modelo a la Vista. Por tanto, la principal diferencia entre los dos

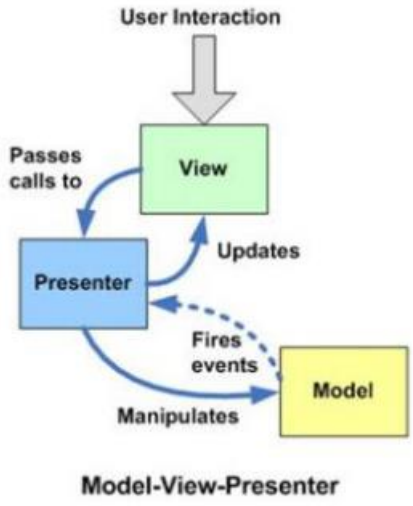

Figura 2.16: Patrón MVP

patrones analizados radica en que ahora el Presentador actualiza directamente la Vista mientras que el Controlador no.

Esto simplifica enormemente a la Vista y la ejecución de pruebas. El Presentador ahora tiene mayor responsabilidad y complejidad, pues tiene que manipular el Modelo y actualizar la Vista.

Este modelo es flexible y está en manos de los desarrolladores la tesitura de si simplificar la Vista, aumentando la complejidad del Presentador o viceversa.

Ejemplo de uso: El usuario presiona un botón en la interfaz y se genera un evento que llega al Presentador. El controlador modifica el valor en el Modelo, quien genera un evento que posibilita que el Presentador actualice la Vista.

2.7.3. Patrón MVVM

En el patrón Modelo-Vista-Vista Modelo se sustituye el Presentador por la Vista Modelo.

Este patrón añade a la Vista algo más de complejidad con nuevas propiedades como por ejemplo la propiedad *isChecked*. Estas nuevas propiedades están sincronizadas y son entendidas por la Vista Modelo quien se encarga de notificar al Modelo los cambios producidos.

El patrón MVVM es muy interesante para plataformas que necesiten soportar un *binding* bidireccional con menor esfuerzo. Este factor será determinante a la hora de elegir qué patrón seguir en el diseño y desarrollo de la herramienta de visualización aquí tratada.

Ejemplo de uso: el valor de un campo de entrada de texto puede ser modificado por el usuario manualmente (*binding* desde la Vista a la Vista Modelo) desde la interfaz web o bien presionar un botón que modifique este campo de entrada de texto (*binding* desde la Vista Modelo a la Vista).

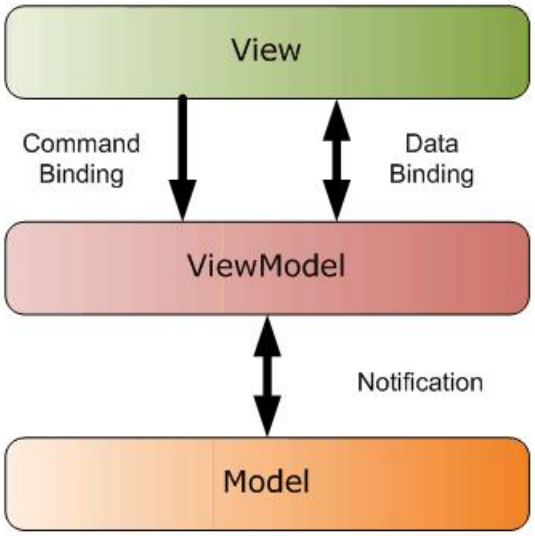

Figura 2.17: Patrón MVVM

2.7.4. Frameworks Web de desarrollo

Tras haber analizado los patrones existentes y habiendo concluido que lo óptimo es un patrón MVVM se procede a analizar dos alternativas válidas que hacen uso de dicho patrón: Knockout[13] y Backbone.js[14].

Como las dos alternativas resultan en principio válidas, los criterios de selección son la facilidad de aprendizaje y la legibilidad del código fuente resultante. Para ello, se realizan ejemplos a pequeña escala de funciones que la herramienta a desarrollar tendría que soportar.

Por ejemplo, para crear un campo de entrada de texto en la Vista y actualizar el Modelo cuando éste cambie, en Knockout se procede así:

```
<input id="test" type="text" data-bind="value: firstName" />

<script type="text/javascript">
```

[13] http://knockoutjs.com/
[14] http://backbonejs.org/

```
var model = { firstName : ko.observable("Default_Value") };
ko.applyBindings(model);
</script>
```

Mientras que en Backbone.js, la misma solución toma la siguiente forma:

```
<input id="test" name="firstName" type="text">

<script type="text/javascript">
var Model = Backbone.Model.extend();
var model = new Model();

$('input').bind( "change", function() {
var obj = {};
obj[$(this).attr('name')] = $(this).val();
model.set(obj); });
</script>
```

Tal y como puede apreciarse, Knockout además de ser una solución más legible es la solución más rápida y sencilla. El aprendizaje de Knockout es relativamente rápido frente al de Backbone.js y en este caso, dado que la herramienta a desarrollar no es excesivamente compleja, no se justifica el uso de Backbone.js (que parece ser más flexible para soluciones complejas) frente a Knockout.

2.7.5. Conclusiones

A primera vista la forma más efectiva para visualizar *Linked Data* es mediante el uso de gráficos y otras técnicas que aprovechan la capacidad de percepción del ser humano en vez de presentar grandes tablas y listas con datos que dificultan un análisis inmediato de los datos allí presentados.

Pero esto no siempre es así pues, cuando nos encontramos ante grandes cantidades de datos, las técnicas gráficas de visualización también tienen sus inconvenientes como hemos visto anteriormente.

Así pues, se puede concluir que la presentación debe estar basada en los requisitos del usuario final y sus tareas. Se necesita proporcionar una interfaz flexible y totalmente personalizable, compatible con la visualización de los datos de distinta naturaleza (cadenas de texto, números, coordenadas geográficas e imágenes).

Capítulo 3

Análisis de Requisitos

"Si hubiésemos preguntado a los usuarios qué era lo que necesitaban, su respuesta habría sido: "caballos más rápidos"."

— Henry Ford

En este capítulo se lleva a cabo una de las fases más importantes dentro del mundo del desarrollo del Software: la captura y análisis de los requisitos que sirven como base y rigen el Software a desarrollar.

Para hacer posible esta labor, en primer lugar se realiza un análisis detallado de los Casos de Uso que puedan tener lugar en cada escenario, apoyándose en el Lenguaje Unificado de Modelado (*UML*).

Así pues, los Casos de Uso permiten especificar y documentar un sistema mediante lenguaje gráfico centrándose en las necesidades del usuario. El fin de este capítulo es la captura completa de requisitos, los cuales determinarán la fase de diseño e implementación del Software a realizar.

3.1. Casos de uso

Semantic Faceted Rapid Application Development (SEFARAD) es la herramienta a desarrollar objetivo de este Proyecto Fin de Carrera.

Tal y como el nombre de la herramienta indica, se trata de una aplicación de desarrollo rápida que permite el filtrado simultáneo de múltiples facetas, o propiedades, de cualquier repositorio semántico.

Con el fin de obtener una lista de requisitos que determinen el funcionamiento y comportamiento de SEFARAD, siempre desde un punto de vista del usuario, a continuación se presentan varios Casos de Uso que ilustran los requisitos del sistema al mostrar cómo reacciona éste a eventos que se producen en su ámbito o en él mismo.

Tras la introducción y presentación de los actores, llevada a cabo en la siguiente sección, se procederá a detallar cada Caso de Uso siguiendo un esquema predefinido en el que se detallara el Caso de Uso. De manera opcional, se incluye un diagrama representativo para facilitar la comprensión del mismo.

3.1.1. Diccionario de los actores

En la tabla 3.1 se recogen los actores primarios y secundarios que intervienen en los Casos de Uso que se detallan posteriormente.

Identificador del actor	Papel	Descripción
ACT-1	Usuario	Usuario del sistema
ACT-2	Administrador	Administrador del sistema
ACT-3	Código Fuente	El código fuente de SEFARAD.
ACT-4	Punto de Acceso *SPARQL*	Repositorio RDF que permite consultas *SPARQL*.

Tabla 3.1: Diccionario de los actores

3.1.2. Modo pasivo

El modo pasivo se caracteriza por ser pasivo de cara al Usuario final. Es el Administrador del sistema quien realiza la subyacente gestión técnica.

- CU-1: Creación de una demostración (tabla 3.2)

  Para que el Administrador pueda crear una demostración concreta, únicamente deberá editar el código fuente, añadiendo una ruta en SEFARAD y los *Widgets* necesarios para que otros usuarios puedan acceder a la misma.

  Este Caso de Uso se presenta en la tabla 3.2.

Identificador	CU-1
Nombre	Creación de una demostración.
Descripción	El administrador puede crear tantas demostraciones como desee. Éstas serán accesibles públicamente a través de una URL determinada.
Actores	Administrador y Código Fuente.
Precondiciones	Un punto de acceso *SPARQL* que funcione correctamente.
Flujo normal	1. El administrador crea una nueva ruta. 2. El administrador configura el punto de acceso *SPARQL* y la consulta. 3. El administrador configura *Widgets* y otros parámetros que considere oportunos.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.2: Caso de uso número 1

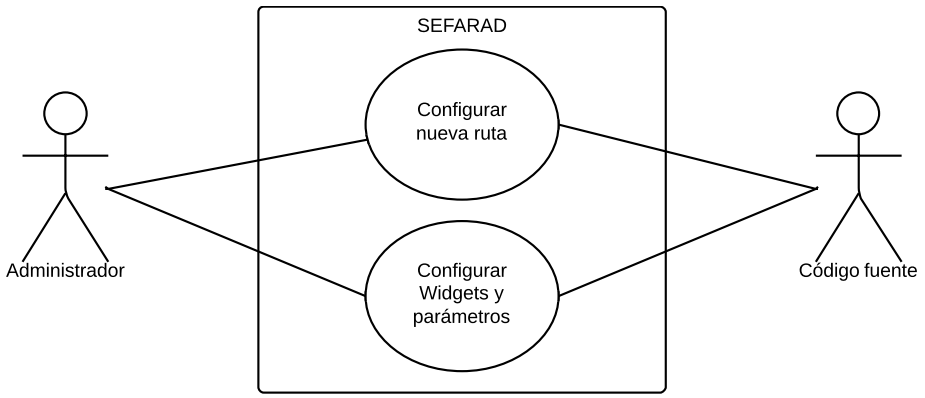

Figura 3.1: Caso de uso número 1

- CU-2: Visualización de una demostración (tabla 3.3)

Identificador	CU-2
Nombre	Visualización de una demostración.
Descripción	El usuario accede a la ruta y visualiza lo configurado por el administrador. Podrá añadir sus propios *Widgets* en cada sesión.
Actores	Usuario y Base de Datos
Precondiciones	Un punto de acceso *SPARQL* que funcione correctamente.
Flujo normal	1. El usuario accede a la URL de la demostración. 2. El usuario podrá añadir sus *Widgets* propios pero sólo serán válidos en esa sesión.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.3: Caso de uso número 2

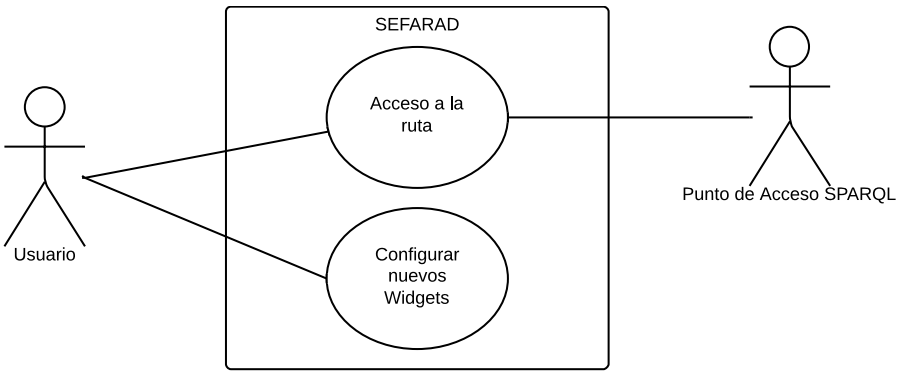

Figura 3.2: Caso de uso número 2

- CU-3: Búsqueda y filtrado por facetas (tabla 3.4)

Identificador	CU-3
Nombre	Búsqueda y filtrado por facetas.
Descripción	Se describe la experiencia de usuario.
Actores	Usuario.
Precondiciones	Un entorno de visualización configurado correctamente.
Flujo normal	1. El usuario puede filtrar por facetas de manera múltiple e ilimitada. 2. El usuario puede filtrar por palabras clave.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.4: Caso de uso número 3

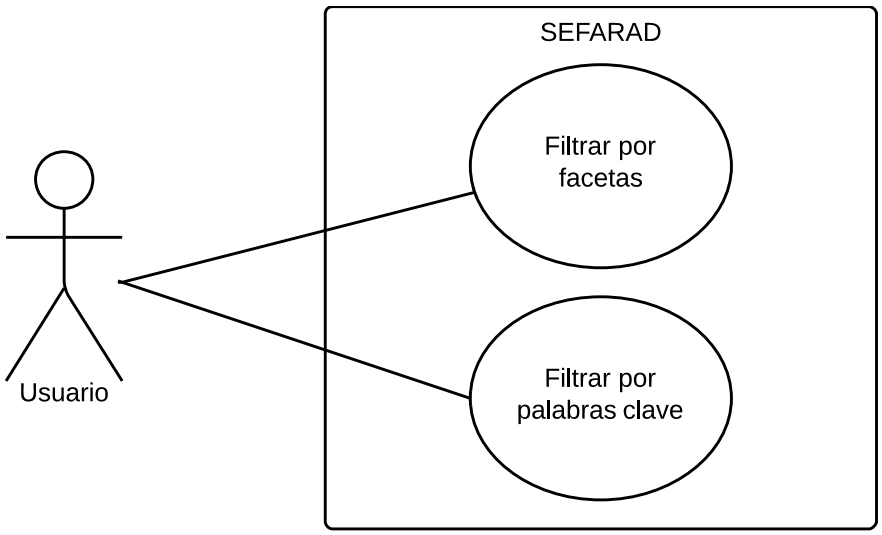

Figura 3.3: Caso de uso número 3

- CU-4: Visualización gráfica (tabla 3.5)

  Dado el contexto de grandes cantidades de datos que se manejan en este tipo de repositorios, y sus relaciones, debe ofrecerse la posibilidad de generar gráficos de manera automática.

  El resumen de este caso de uso se presenta en la tabla 3.5.

Identificador	CU-4
Nombre	Visualización gráfica.
Descripción	El usuario podrá visualizar de manera automática, por ejemplo, la proporción de elementos de las facetas deseadas en gráficos de tipo tarta.
Actores	Usuario.
Precondiciones	Un entorno de visualización configurado correctamente.
Flujo normal	1. El usuario escoge qué gráficos visualizar. 2. El usuario aplica filtros a través de los *Widgets*. 3. El usuario visualizará en tiempo real cómo varían las proporciones de las facetas escogidas.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.5: Caso de uso número 4

3.1.3. Modo activo

El modo activo requiere de conocimientos técnicos que tienen que aplicarse.

- CU-5: Consultas *SPARQL* a un repositorio externo (tabla 3.6)

  Este Caso de Uso es válido tanto para Usuarios finales como para el Administrador.

  El resumen de este caso de uso se presenta en la tabla 3.6.

Identificador	CU-5
Nombre	Consultas *SPARQL* a un repositorio externo.
Descripción	Permite ejecutar consultas *SPARQL* a un punto de acceso configurado previamente.
Actores	Administrador, Usuario y Base de Datos.
Precondiciones	Un punto de acceso *SPARQL* que funcione correctamente.
Flujo normal	1. El usuario o administrador configura el punto de acceso SPARQL al que enviar las consultas. 2. El usuario o administrador ejecuta la consulta. 3. El usuario o administrador configura la interfaz de visualización a su gusto. 4. El usuario o administrador ejecuta una nueva consulta. El contenido visualizado cambiará de forma dinámica.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.6: Caso de uso número 5

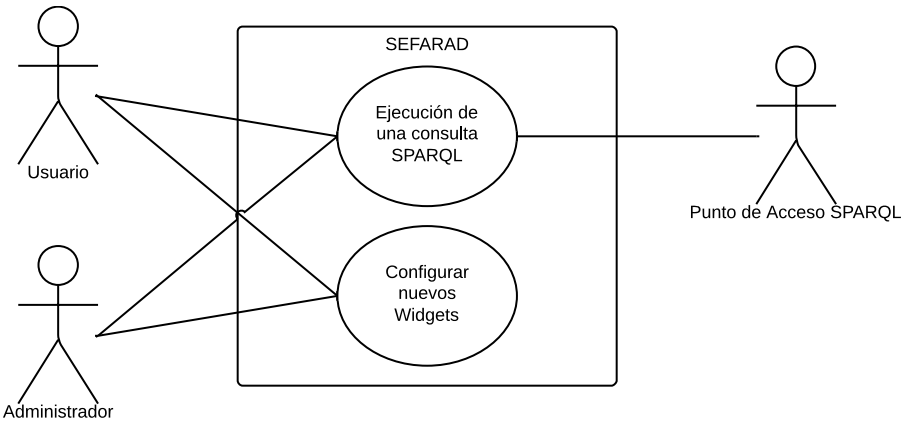

Figura 3.4: Caso de uso número 5

- CU-6: Creación de gráficos mediante consultas *SPARQL*.

Identificador	CU-6
Nombre	Creación de gráficos mediante consultas *SPARQL*.
Descripción	Se crean gráficos a petición a través de la interfaz de usuario.
Actores	Administrador, Usuario y Base de Datos.
Precondiciones	Un punto de acceso *SPARQL* que funcione correctamente.
Flujo normal	1. El Usuario o Administrador configura el punto de acceso SPARQL al que enviar las consultas. 2. El Usuario o Administrador ejecuta la consulta SPARQL cuyos resultados quiere visualizar de manera gráfica.
Flujo alternativo	-
Poscondiciones	-

Tabla 3.7: Caso de uso número 6

3.2. Análisis y captura de requisitos

Gracias a los Casos de Uso anteriores se está en disposición de recopilar las especificaciones generales que debe cumplir el sistema, teniendo siempre presentes los aspectos técnicos recogidos en el Estado del Arte en lo referente a la Visualización de Linked Data.

A modo de recordatorio, estos aspectos eran:

- Navegación intuitiva a través de las grandes cantidades de datos presentes mediante búsqueda semántica u otros métodos.
- Descubrimiento del conocimiento mediante técnicas exploratorias, esto es, que las relaciones entre los distintos conceptos sean visualizadas y que podamos empezar visualizando un término y acabemos en otro distinto que mantenía relación con él (directa o indirectamente).
- Soporte avanzado de consultas: desde técnicas de filtrado a consultas básicas de términos indexados.
- Análisis detallados de las regiones de interés para el usuario.
- Poder exportar los datos a un formato estándar que permita que otros usuarios o aplicaciones puedan reutilizarlos.
- Presentar los resultados del análisis a diferentes audiencias.

Los Casos de Uso anteriormente desarrollados proporcionan una visión más cercana al Usuario final. En base a estos, podemos concluir con los requisitos recogidos en la siguiente tabla:

Id	Descripción	Tipo de Requisito	Priori-dad	Trazabili-dad
1	Facilitar al Administrador del sistema la posibilidad de crear una visualización completa de una determinada fuente de datos semánticos que pueda ser compartida fácilmente a otros usuarios	Funcional	Deseable	CU1
2	Acceso libre a la visualización de una demostración ya creada a través de la ruta apropiada	Funcional	Deseable	CU-1 y CU-2
3	Permitir el acceso a una interfaz donde se pueda configurar el punto de acceso al que poder ejecutar las consultas deseadas y poder visualizar las respuestas de manera dinámica	Funcional	Obligato-ria	CU-5
4	Se debe tener en cuenta la posibilidad de repositorios con grandes cantidades de datos ofreciendo alguna alternativa de interacción de manera paginada con el servidor	Rendimiento	Opcional	CU-5

Tabla 3.8: Requisitos SEFARAD

Capítulo 4

Arquitectura y descripción

"La arquitectura tiene como objetivo la eternidad"

— Alexis Carrel

En este capítulo se describe la arquitectura y se dan detalles del trasfondo técnico del proyecto.

La arquitectura muestra, desde un punto de vista más general, la integración necesaria entre recursos, aplicaciones, bases de datos y herramientas para poder desarrollar un sistema que cumpla con los requisitos previamente acordados.

4.1. Diseño arquitectónico

4.1.1. Introducción

Para poder llevar a cabo el desarrollo de SEFARAD, tras el análisis de otras herramientas de visualización realizado en el Estado del Arte (capítulo 2.6) y la captura de requisitos (tabla 3.8), se concluye que es necesario disponer de los siguientes recursos o herramientas:

- Un *Framework* que permita el almacenamiento y el procesamiento de datos semánticos. Debe contar con, al menos, una interfaz que soporte consultas SPARQL.
- Un *Framework* Web de desarrollo que haga posible la experiencia de visualización dinámica que debe ofrecer SEFARAD. El patrón MVVM es el más idóneo tal y como se analizó en el Estado del Arte (capítulo 2.7.3).
- Un conjunto de librerías como jQuery para manejar Javascript y Ajax de manera más sencilla y eficiente; jQueryUI[1] para crear una interfaz simple, elegante y fácilmente adaptable a nuevos temas de colores con herramientas como *jQueryUI ThemeRoller*.

4.1.2. Herramientas empleadas

4.1.2.1. Knockout

Knockout es la herramienta elegida como *Framework* Web tal y como se justificó previamente en el capítulo 2.7.4, dada la sencillez y legibilidad que aporta.

A modo de una pequeña introducción, Knockout es una implementación JavaScript de código abierto del patrón Modelo-Vista-VistaModelo (MVVM)2.7.3 ideal para crear de manera sencilla y eficaz plataformas en las que la interfaz de usuario cambia de forma dinámica (por ejemplo cuando el usuario realiza alguna acción o cuando la fuente de datos cambia).

Sus principales características son:

- Proporciona un seguimiento de dependencias eficaz: Cuando el modelo de datos cambia, automáticamente se actualizan las partes de la interfaz que muestran estos datos.
- Uso de *observables* (enlaces declarativos): La manera más obvia y simple de conectar la interfaz al modelo de datos. Se pueden construir complejas interfaces de usuario dinámicas.

[1] Más información: `http://jqueryui.com/`

- Es fácilmente extensible: Con solo unas pocas líneas de código se pueden crear enlaces personalizados en el modelo de datos.
- Es una librería JavaScript pura: Compatible por tanto con cualquier servidor o tecnología en el lado cliente.
- Compacta: sólo ocupa 13kb.
- Funciona en cualquier navegador: Internet Explorer 6 en adelante, Firefox, Chrome, Safari...

4.1.2.2. LMF: Linked Media Framework

En cuanto al *Framework* o servidor RDF, éste debe ser capaz de almacenar y procesar datos semánticos además de ofrecer una interfaz en la que poder realizar consultas SPARQL desde SEFARAD (lado cliente) en donde se visualizarán los resultados correspondientes a dichas consultas.

Tras el estudio de diversas alternativas, la opción elegida ha sido el *Linked Media Framework*[2]. Los motivos son los siguientes:

- Es un proyecto de código abierto.
- El proyecto está activo. Prueba de ello es el lanzamiento de hasta cuatro versiones en cuatro meses.
- Soporte técnico de las propios creadores.
- Proporciona numerosos módulos relacionados con el procesamiento semántico. Por cada módulo se proporciona una interfaz de tipo *REST* a la que se puede consultar desde SEFARAD.
- Dispone de un módulo de búsqueda semántica basado en Apache SOLR. Este módulo es el idóneo para cumplir con el requisito de poder trabajar con grandes cantidades de datos.

4.1.3. Descripción global del sistema

Con el propósito de dar una visión global del sistema, a continuación se describe de manera breve la arquitectura global que constituye a SEFARAD.

[2]De ahora en adelante LMF

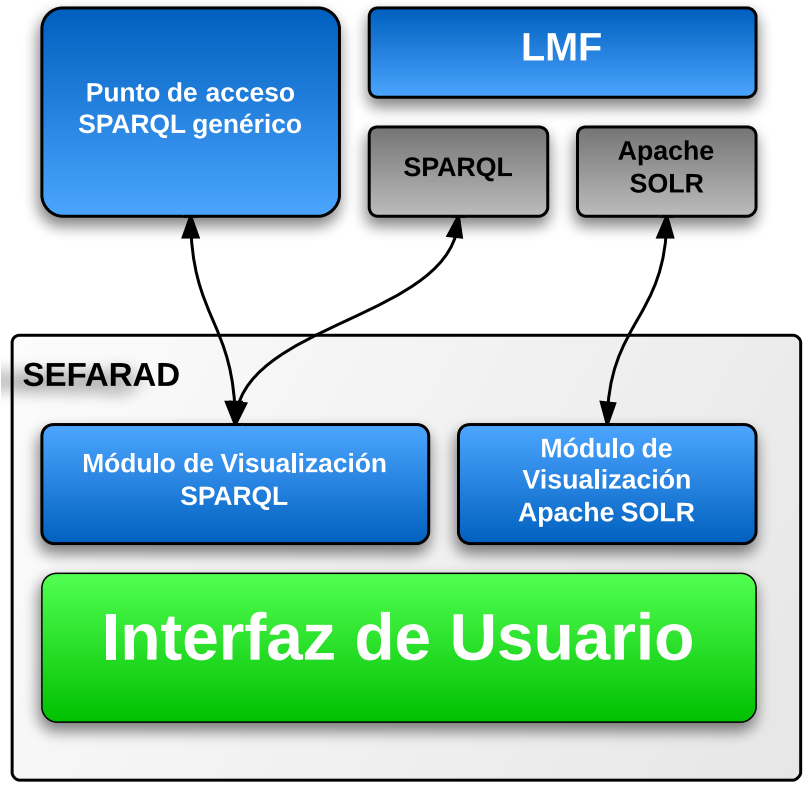

Figura 4.1: Arquitectura general

Según su cometido se distinguen cinco módulos básicos, algunos de los cuales se describen con detalle en la siguiente sección:

- Punto de acceso SPARQL genérico: Cualquier punto de acceso SPARQL puede constituir una fuente de datos que puede ser visualizada desde SEFARAD.

- LMF: Este *Framework* proporciona su propio punto de acceso SPARQL ligado a la base de datos integrada en la herramienta. Además, ofrece también un punto de acceso al "servidor de búsquedas" (a través Apache SOLR) que realiza una tarea de indexación de datos semánticos totalmente configurable por el administrador.

- Módulo de Visualización SPARQL: Encargado de procesar cualquier respuesta SPARQL y de aplicar cualquier tipo de filtrado por parte del usuario.

- Módulo de Visualización Apache SOLR: Su principal tarea es la de procesar las respuestas del servidor y generar nuevas consultas al servidor (aplicar filtros, realizar búsquedas...) para su posterior visualización.

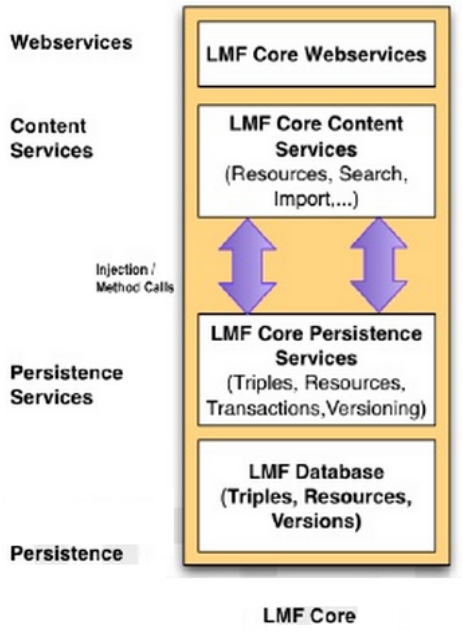

Figura 4.2: Núcleo LMF

- Interfaz de Usuario: La parte visual de SEFARAD. Gracias al patrón MVVM proporcionado por Knockout, la interfaz es completamente dinámica y flexible. El acceso a una determinada URL u otra determinara qué módulo de visualización está activo.

4.2. Diseño detallado

4.2.1. LMF: Linked Media Framework

Linked Media Framework (LMF) permite configurar un servidor avanzado que da soporte a las diferentes tecnologías de Web Semántica necesarias para ofrecer todos los servicios a continuación descritos.

Su arquitectura se divide en dos partes principales: el núcleo LMF constituye la base de la aplicación y los diferentes módulos avanzados contribuyen a añadir más funcionalidades.

4.2.1.1. Núcleo LMF

El núcleo del *Linked Media Framework* no es más que un servidor *Linked Data* que permite procesar datos de acuerdo a los siguientes principios establecidos:

- Emplear URIs como nombres de las cosas.
- Usar URIs HTTP para poder visualizar esos nombres.
- Dar soporte a RDF y SPARQL.
- Devolver enlaces a otras URIs relacionadas para poder descubrir más información dentro de ese contexto.

4.2.1.2. Módulos LMF

Los módulos del *Linked Media Framework* son los encargados de extender las funcionalidades básicas del servidor *Linked Data* y son los que realmente hacen que LMF sea una herramienta útil. Además, al igual que el núcleo, todos los módulos proporcionan una interfaz *REST* que permite la interacción de manera sencilla y eficaz.

4.2.1.2.1. Importer

De manera nativa se puede importar fichero a fichero. Para extender esta funcionalidad, se ha desarrollado un pequeño *script* que importa cómodamente todos los ficheros que se encuentran en un directorio, haciendo uso así de la API REST ofrecida por LMF.

4.2.1.2.2. Google Refine

Utilidad que permite importar cualquier tipo de fichero (TSV, CSV, Excel, XML, RDF sobre XML, JSON...) al formato propio que maneja Google Refine.

Una vez importados, se crea un nuevo proyecto con dichos datos y se visualizan en la pantalla principal en una tabla en la que por cada "sujeto" (fila) se muestran en distintas columnas las propiedades del objeto y su valor.

Una posibilidad interesante que ofrece la herramienta es la de filtrar por propiedad del objeto. En nuestro caso concreto de datos INES por ejemplo podemos filtrar por gsi:type y ver las que son pequeñas y medianas empresas o universidades.

También se puede aplicar cualquier tipo de transformación y edición a las celdas, aplicando expresiones regulares por ejemplo.

Tras haber completado las modificaciones requeridas, no hace falta que exportemos los cambios a un nuevo RDF e importarlos en LMF, se pueden salvar directamente en LMF.

4.2.1.2.3. Classification

LMF te permite crear clasificadores entrenados con frases que están asociados a un concepto. De esta manera, cuando el entrenamiento se ha realizado con éxito, tras alimentarlo con 10 frases de ejemplo, podemos obtener las probabilidades de pertenencia a un concepto para un texto nuevo dado.

Es importante hacer un *retrain* tras haber terminado el entrenamiento para que los cambios sean aplicados.

4.2.1.2.4. SPARQL

Gracias a este módulo se pueden realizar consultas SPARQL sobre los datos que tengamos importados en la base de datos que también provee LMF.

Este módulo incorpora varios editores web a través de los que podemos hacer consultas y obtener las respuestas en el formato deseado (html, json o xml), así como una API REST usada por SEFARAD para uno de sus modos de funcionamiento.

4.2.1.2.5. Semantic Search

Este módulo proporciona una funcionalidad extra basada en Apache SOLR que se utiliza en otro de los modos de funcionamiento de SEFARAD.

El concepto subyacente es sencillo: este módulo indexa lo configurado en cada uno de los "cores", asociando una palabra clave a la etiqueta RDF deseada de manera periódica con la Base de Datos que contiene dichos ficheros RDF.

Gracias a este módulo, pueden configurarse distintos "cores" para distintos escenarios y, mediante consultas estandarizadas (Apache SOLR) a una interfaz REST, podemos obtener los datos deseados de manera cómoda sin necesidad de realizar complejas consultas SPARQL.

En el siguiente ejemplo se configura un core para obtener información de un RDF que contiene información de una empresa según una ontología propia:

Código 4.1: RDF importado en LMF

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:gsi="http://www.gsi.dit.upm.es/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <gsi:Company rdf:about="http://www.ines.org.es/content/alarcos-quality-center">
    <gsi:name>Alarcos Quality Center</gsi:name>
    <gsi:address>Paseo de la Universidad 4</gsi:address>
    <gsi:province>Ciudad Real</gsi:province>
    <gsi:type>Pequena y mediana empresa</gsi:type>
    <gsi:web>www.alarcosqualitycenter.com</gsi:web>
  </gsi:Company>
</rdf:RDF>
```

Código 4.2: Core configurado en el módulo Semantic Search de LMF

```
@prefix gsi : <http://www.gsi.dit.upm.es/> ;
@filter rdf:type is gsi:Company ;
name = gsi:name :: xsd:string ;
address = gsi:address :: xsd:string ;
province = gsi:province :: xsd:string ;
type = gsi:type :: xsd:string ;
web = gsi:web :: xsd:string ;
```

4.2.1.2.6. Permisos/seguridad

LMF implementa su propio mecanismo de autorización y autenticación. Hay 3 perfiles predefinidos: - Simple (usado por defecto): acceso de lectura para todos y de escritura solo en localhost. - Standard: acceso de lectura para todos y de escritura para usuarios autenticados con rol de manager. - Restricted: permite solo el acceso para miembros autenticados.

Para cambiar el perfil se puede hacer desde la interfaz gráfica o bien, si LMF está corriendo en un servidor remoto, hay que hacer acceder al mismo y ejecutar el siguiente comando:

```
curl -X POST -H "Content-Type: application/json" -d '["standard"]'
http://<MACHINE_NAME>:<PORT>/LMF/config/data/security.profile
```

Tras este paso, podremos iniciar sesión con el usuario de administración configurado por defecto, cuyos credenciales son:

- Usuario: admin
- Contraseña: pass123

4.2.1.3. Apache SOLR y Consultas SPARQL

Tal y como se ha comentado con anterioridad, la plataforma LMF permite disponer de interfaces o puntos de acceso a los que ejecutar diferentes consultas.

En SEFARAD se emplean las interfaces que ponen a disposición estos dos módulos: Semantic Search y SPARQL para poder acceder a los datos necesarios en cada caso.

4.2.1.3.1. SPARQL

Como ya se ha visto anteriormente, SPARQL se puede utilizar para expresar consultas que permiten obtener información de diversas fuentes de datos, si los datos se almacenan de forma nativa como RDF o son definidos mediante vistas RDF a través de algún sistema *middleware*.

El módulo de SPARQL de LMF proporciona las siguientes interfaces:

- /sparql/select

  En este caso, se hace uso de la interfaz de la siguiente manera: GET /sparql/select?query=...&output=...

  El campo *query* hace referencia a una query que cumple con la especificación SPARQL 1.1[3] y el campo output especifica el formato de la respuesta que generará el servidor ("html", "json" o "xml").

  Existen otras alternativas como: POST /sparql/select?output=...

  En este caso, la *query* SPARQL va en el "body" de la petición POST.

- /sparql/snorql

[3] http://www.w3.org/TR/sparql11-query/

Su uso se realiza de una única manera a través de una petición POST: POST /sparql/snorql

Los campos query y output se encuentran en el "body" de la petición.

Aunque parezca redundante con la interfaz anterior, esta interfaz es la que emplea el submódulo "snorql" que permite realizar consultas SPARQL a través del panel de administración de LMF y visualizar el resultado.

- /sparql/update Este interfaz no proporciona ninguna respuesta salvo la evidente que reporta si ha tenido éxito o no la acción deseada.

  El uso de la interfaz se realiza de la siguiente manera: GET /sparql/update?query=...

  La query debe cumplir con la sintaxis SPARQL 1.1 Update (`http://www.w3.org/TR/sparql11-update/`). Este conjunto de "querys update" permiten tareas importantes como eliminar tripletas ya incluidas en algún grafo RDF.

4.2.1.3.2. Apache SOLR

Apache SOLR es una plataforma de búsquedas basada en Apache Lucene, que funciona como un "servidor de búsquedas". Sus principales características incluyen búsquedas de texto completo, resaltado de resultados, clustering dinámico y manejo de documentos como Word y PDF. SOLR es escalable, permitiendo realizar búsquedas distribuidas y actualmente se está usando en muchos de los sitios más grandes de Internet:

- www.whitehouse.gov Utiliza SOLR a través de Drupal para la búsqueda del sitio
- AT & T.
- AOL.
- Netflix.

SOLR está escrito en Java y se ejecuta como un servidor de búsqueda de texto completo independiente dentro de un contenedor de servlets como Tomcat.

La principal característica de SOLR (o al menos la más útil) es su API estilo REST, ya que en vez de usar drivers o APIs programáticas para comunicarnos con SOLR podemos hacer peticiones HTTP y obtener resultados en XML o JSON.

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

- Las respuestas vienen en formato XML o JSON, que hoy en día pueden ser interpretados por casi cualquier cosa. La métrica generalmente es JavaScript: si podemos interpretarlo con JavaScript dentro de todas las limitaciones que impone un navegador, entonces lo podemos interpretar en cualquier otro lugar. Por supuesto, JavaScript tiene soporte nativo para JSON y XML.

- Aunque esté escrito en Java, podemos usarlo con nuestro lenguaje de programación preferido, simplemente usando las peticiones GET para realizar búsquedas en el índice, y POST para agregar documentos.

Por supuesto suele haber una librería para cada lenguaje de programación que encapsula la interfaz REST, pero es muy simple construir una librería con, por ejemplo, JavaScript, e igualmente no es obligatorio usarla.

4.2.2. Módulo de Visualización SPARQL

Al acceder a la ruta en la que el módulo está activo y tras configurar el punto de acceso SPARQL y realizar la consulta deseada, cualquier filtrado se realizará en local gracias a las funciones definidas que actualizan la vista automáticamente al modificar las variables del Modelo.

La ruta de acceso es la siguiente: GET /sparql

En este módulo las posibilidades de personalización son menores, tanto el título como el logo están ocultos. En la parte superior izquierda de la interfaz, hay un desplegable que permite ejecutar una petición SPARQL al endpoint configurado desde la Configuración accesible en la interfaz.

Cada vez que escribimos una petición, se ejecuta una petición GET al punto de acceso con la *query* introducida y pidiendo una respuesta JSON. La respuesta JSON se trata y se *mapea* en el Modelo de Datos. Por tanto, todos los Widgets se actualizarán automáticamente.

En cuanto al filtrado en local, éste se realiza mediante expresiones regulares. Por tanto, en este modo no debemos visualizar grandes cantidades de datos porque el rendimiento que percibamos dependerá de la capacidad de nuestro equipo.

4.2.3. Módulo de Visualización Apache SOLR

El acceso a este módulo implica, como ya se ha visto, la sucesión de conexiones al servidor Apache SOLR (integrado en LMF) para requerir la información deseada. Asi pues,

todo filtrado se realiza mediante peticiones GET al servidor siguiendo la sintaxis definida por SOLR.

Este módulo consiste en la extensión e integración del proyecto Ajax-SOLR[4] con Knockoutjs.

Al acceder por primera vez, se trata de cargar la configuración guardada previamente. Esta configuración almacena la siguiente información:

- Variables de configuración como: título y logo de la página, idioma y autocompletado.
- Todos los Widgets con su configuración asociada.

Si no existe ninguna guardada con anterioridad, porque es la primera vez que accedemos al core por ejemplo, se cargará una configuración por defecto que no es más que un Widget de Resultados vertical.

El punto de acceso viene se configura a través de la variable serverURL del fichero mvvm.js

Al contrario que en el módulo anterior, el filtrado no se realiza en local, se realiza haciendo una nueva petición al servidor Apache SOLR por lo que podemos trabajar con grandes cantidades de datos; la potencia asociada al procesamiento la asume el lado servidor. Esta petición devuelve un JSON asociado que es tratado y *mapeado* en el Modelo de Datos. Por tanto, como en el caso anterior, todos los Widgets se actualizarán automáticamente.

4.2.3.1. Almacenamiento de la configuración

La configuración de los Widgets y demás variables de un entorno configurado funcionando en Apache SOLR se almacena mediante una petición POST en la siguiente ruta: config/data/search.config.<nombrecore>.

4.2.4. Interfaz de Usuario y Modelo

La interfaz de usuario está constituida por *observables* (elementos declarativos) de Knockout. Como se vio previamente, estos elementos facilitan la actualización automática de la interfaz ante cualquier cambio en el Modelo y viceversa. Según su funcionalidad asociada pueden clasificarse en:

[4] https://github.com/evolvingweb/ajax-solr

- Variables de control.
- Variables globales.

4.2.4.1. Variables de control

Realizan una tarea elemental: distinguir el estado en el que se encuentran diversos elementos funcionales o de interfaz. Por ejemplo:

- Controlar si el panel de configuración está visible o no.
- Distinguir qué módulo de visualización está activo: SPARQL o Apache SOLR.

4.2.4.2. Variables globales

Este tipo de variables recogen elementos tan dispares como el título de la página que se visualiza, el punto de acceso SPARQL configurado o el idioma seleccionado.

4.2.4.3. Arrays

Los denominados *observableArrays* contienen la información más importante en SEFARAD: como por ejemplo la visualizada y la de los filtros activos.

A modo de ejemplo, en el módulo de visualización SPARQL, se emplean hasta tres *observableArrays* distintos para poder visualizar los datos en el Widget de Resultados: viewData, filteredCategory y filteredData.

El array viewData contiene todos los datos que se proporcionan en la respuesta desde el servidor. En el caso del módulo Apache SOLR como hemos visto, el filtrado se realiza en el lado servidor y por tanto el resto de *observableArrays* no son necesarios. En el caso del módulo SPARQL, como hemos visto esta información esta sin filtrar. El posterior filtrado se realiza en el lado cliente, entrando en juego los siguientes:

- filteredCategory: es el resultado de filtrar viewData según los diferentes Widgets de tipo nube de etiquetas.
- filteredData: resultado visualizado en el Widget, tras filtrar filteredCategory con la búsqueda textual.

Para determinar qué etiquetas de los Widgets de tipo nube de etiquetas están efectuando un filtrado, se ha consultado al *observableArray* activeWidgets que almacena qué filtros están activos en todo momento.

Capítulo 5

Experimentación y validación

"Si la depuración es el proceso de eliminar errores, entonces la programación debe ser el proceso de introducirlos."

— Edsger W. Dijkstra

En este capítulo se detallan los distintos modos de uso que proporciona SEFARAD. También se valida el proyecto en varios entornos reales.

5.1. Introducción

Gracias a la flexibilidad con la que se ha diseñado y desarrollado SEFARAD, tal y como se ha especificado en el capítulo anterior, es posible emplear SEFARAD para visualizar cualquier repositorio de datos semánticos siempre y cuando exista un punto de acceso SPARQL o se haya configurado correctamente un Core en el módulo de búsqueda semántica que proporciona LMF, independientemente de la estructura de los datos a manejar.

Dada esta flexibilidad y atendiendo al modo en el que se realiza la ejecución de SEFARAD, se pueden distinguir hasta tres escenarios de funcionamiento distintos introducidos a continuación:

- Modo demostración: Se requiere que el Administrador configure una ruta para acceder a la demostración creada, esto es, se generará una URL única e introducir unas pequeñas líneas de código para personalizar el punto de acceso SPARQL, la consulta, los *Widgets* y campos visibles. Esta URL se puede compartir y los usuarios que accedan podrán navegar a través de los datos gracias a los *Widgets* configurados.
- Modo local: Esta modalidad no requiere modificar ni añadir ninguna línea de código. Desde la interfaz visual de SEFARAD, se configura el punto de acceso SPARQL y se ejecutan las consultas deseadas. El usuario determinará en tiempo real qué *Widgets* son los adecuados para que la visualización de dichos datos sea la más oportuna.
- LMF + SEFARAD: Esta opción es la más compleja desde el punto de vista técnico para el usuario pero también es la que más posibilidades ofrece. Los pasos a seguir para llevar a cabo la instalación y configuración de esta modalidad vienen descritos con detalle en el Anexo A. En el caso de tener que acceder a grandes colecciones de datos, esta debe ser la opción elegida pues se hace uso del módulo de búsqueda semántica que ofrece LMF y que está basado en Apache SOLR.

5.1.1. Funcionalidades

En conjunto, los distintos modos de uso hacen posible las siguientes funcionalidades, clasificadas según su complejidad:

5.1.1.1. Funcionalidades básicas

- Navegación intuitiva a través de las grandes cantidades de datos presentes mediante búsqueda semántica, filtrados múltiples o simples y funcionalidades básicas como

ordenación por etiqueta y autocompletado del campo de búsqueda.

- Soporte avanzado de consultas: se permite realizar consultas SPARQL y consultas semánticas totalmente personalizables por etiquetas.
- El análisis detallado de las regiones de interés para el usuario es posible gracias a la gran capacidad de personalización que ofrece SEFARAD pues es éste, el usuario final, quien decide qué es lo que quiere visualizar y dónde.
- Las diferentes audiencias están tenidas en cuenta gracias al modo de visualización simple/complejo y los permisos o roles de usuario.
- Internacionalización en inglés y español.
- Soporte en móviles y tabletas.

5.1.1.2. Funcionalidades avanzadas

- Posibilidad de añadir gráficos estáticos mediante consultas *SPARQL*.
- Configuración y personalización simples desde el punto de vista técnico.

5.2. Experimentación

Tras conocer los modos de uso y las funcionalidades que SEFARAD es capaz de ofrecer, en esta sección se pretende experimentar con la plataforma.

La experimentación es el proceso que permite asegurar que la plataforma es eficaz y carece de errores.

Con este fin presento las siguientes pruebas personales llevadas a cabo.

5.2.1. Consultas a la DBpedia

El proyecto DBpedia[1] trata de extraer el contenido generado en la Wikipedia de manera estructurada. Este proyecto ha generado durante mucho tiempo información semántica a partir de la versión inglesa de Wikipedia. Desde Junio de 2011, el proceso de extracción de información se hace también en la versión española, dando lugar a DBpedia en español[2],

[1] http://dbpedia.org/About
[2] http://es.dbpedia.org/

además de otros tantos idiomas. El comité de internacionalización de DBpedia ha asignado un sitio web y un punto de acceso SPARQL para cada uno de estos idiomas.

De este modo podemos emplear SEFARAD en modo local para realizar consultas de diversa temática al punto de acceso *http://es.dbpedia.org/sparql* o *http://dbpedia.org/sparql*.

5.2.1.1. Equipos de fútbol

En este ejemplo, vamos a obtener una lista de los jugadores que pertenecen a la selección española de fútbol. Para ello, extraeremos todos los objetos que cumplen esta condición y, para cada uno de ellos, extraeremos información útil como su nombre, el lugar de nacimiento, una pequeña biografía, en qué club juegan y una fotografía.

Todo lo anterior se obtiene ejecutando la siguiente consulta:

Código 5.1: Consulta SPARQL para filtrar a los jugadores de la selección española de fútbol

```
PREFIX prop: <http://dbpedia.org/property/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT * WHERE {
      <http://dbpedia.org/resource/Spain_national_football_team> prop:name
      ?playerobj ;
      rdfs:label ?name .
      ?playerobj dbpedia-owl:birthPlace ?cityobj ;
      rdfs:label ?city .
      ?playerobj dbpedia-owl:birthDate ?date ;
      prop:currentclub ?clubsource ;
      rdfs:label ?club .
      ?playerobj rdfs:comment ?comment ;
      dbpedia-owl:thumbnail ?photo
FILTER ( lang(?name) = 'es' && lang(?city) = 'es' && lang(?club) = 'es' &&
lang(?comment) = 'es' )
 }
```

Punto de acceso: http://dbpedia.org/sparql

Tras configurar los Widgets pertinentes para sacar el máximo provecho a estos datos, el resultado es el que se puede observar en la figura 5.1.

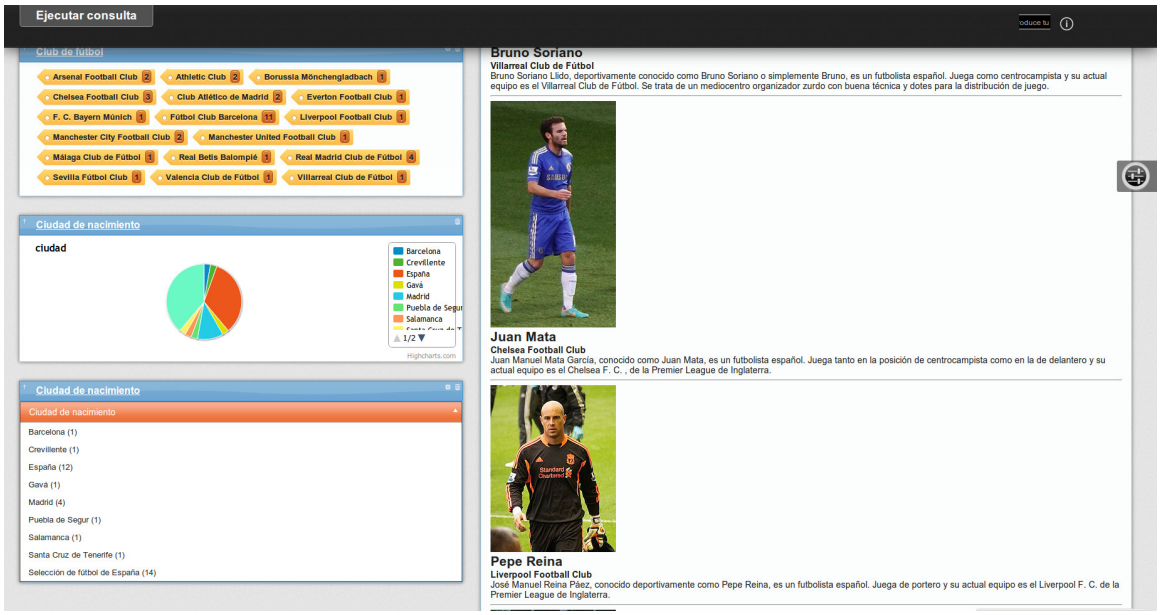

Figura 5.1: Jugadores de la Selección Española de Fútbol

.

De esta manera, podremos conocer qué provincia es la que ha dado más internacionales o en qué equipos y ligas están jugando.

5.2.1.2. Películas

Esta consulta resultará de utilidad a todos los cinéfilos:

Código 5.2: Consulta SPARQL para obtener un listado de películas

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpontology: <http://dbpedia.org/ontology/>
PREFIX dbpproperty: <http://es.dbpedia.org/property/>

SELECT ?titulo ?director ?pais ?estreno ?genero
WHERE {
        ?s rdf:type dbpontology:Film .
        OPTIONAL {
                ?s dbpproperty:título ?titulo .
        }
        OPTIONAL {
                ?s dbpproperty:dirección ?director .
```

```
        }
        OPTIONAL {
                ?s dbpproperty:país ?pais .
        }
        OPTIONAL {
                ?s dbpproperty:estreno ?estreno .
        }
        OPTIONAL {
                ?s dbpproperty:género ?genero .
        }
FILTER ( ?estreno >= xsd:integer(1920) )
}
LIMIT 200
```

Punto de acceso: http://es.dbpedia.org/sparql

Tal y como puede observarse, para cada sujeto clasificado como película, se extrae su título, la dirección, el país, el año de estreno y el género al que pertenece.

Tras configurar los Widgets pertinentes para sacar el máximo provecho a estos datos, el resultado es el que se puede observar en la figura 5.2.

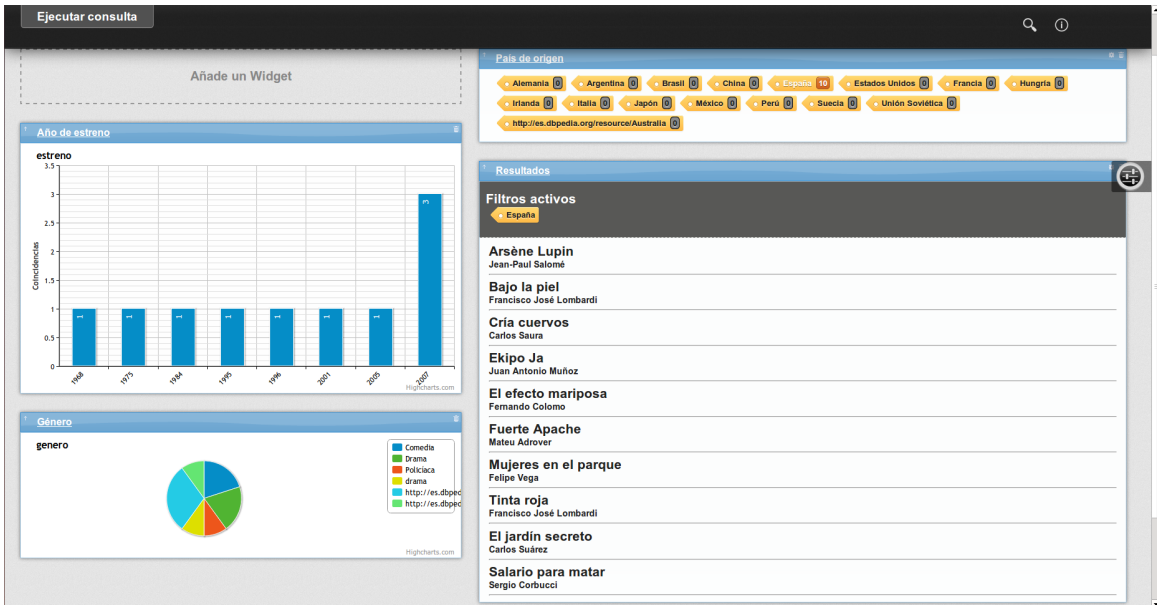

Figura 5.2: Películas

Nótese que en la consulta se ha limitado a 200 el número de coincidencias devueltas para no saturar el cliente.

Las posibilidades ofrecidas son enormes: descubrir nuevas películas filtrando por categoría, año de estreno, país de origen... incluso descubrir al director de nuestras películas favoritas.

5.2.2. KukiNet

KukiNet una red social basada en la distribución de frases en donde cada usuario puede recibir y añadir sus propias Kukis para que sean repartidas de manera global. Cuando hablamos de Kukis nos referimos a frases cortas y originales que puede hacer reír, provocar reflexiones o evocar sentimientos (según su categoría).

Como KukiNet es un proyecto personal y dispongo del acceso necesario a la Base de Datos en donde se almacenan conjuntos de datos propensos a la visualización, tan sólo se necesita migrar este contenido almacenado en una base de datos relacional a un conjunto de datos en formato RDF que posteriormente se importarán en LMF.

El formato RDF predefinido que describe una Kuki es el siguiente:

Código 5.3: RDF de una Kuki

```
<?xml version='1.0' encoding='utf-8'?>

<rdf:RDF
        xmlns:kuki='http://www.kukinet.com/'
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>

        <kuki:Item rdf:about='http://www.kukinet.com/kuki1'>
                <kuki:text>Todo gran proyecto tiene un principio. Este es nuestro
                comienzo del cual tú puedes escribir la historia. </kuki:text>
                <kuki:score>53</kuki:score>
                <kuki:category>2</kuki:category>
                <kuki:delivered>192</kuki:delivered>
                <kuki:latitude>0</kuki:latitude>
                <kuki:longitude>0</kuki:longitude>
                <kuki:username>KukiNet</kuki:username>
                <kuki:userdescription>Cuenta oficial de
                KukiNet</kuki:userdescription>
                <kuki:usercountry>ESP</kuki:usercountry>
                <kuki:uservoted>0</kuki:uservoted>
                <kuki:userreceived>0</kuki:userreceived>
                <kuki:userscore>10</kuki:userscore>
        </kuki:Item>
</rdf:RDF>
```

Como puede observarse, por cada Kuki extraída obtenemos la siguiente información:

- Identificador único que constituye el sujeto de la tripleta.
- Texto de la Kuki.
- Puntuación que tiene la Kuki dentro de la plataforma, determinada por las votaciones de otros usuarios.
- Categoría a la que pertenece la Kuki, determinada por las votaciones de los usuarios.
- Número de veces que se ha recibido por otros usuarios.
- Lugar en donde se escribió (latitud y longitud)
- El nombre del usuario creador de la Kuki en cuestión.
- Descripción del usuario.
- Nacionalidad del usuario.

- Número de votos que ha realizado ese usuario.
- Número de Kukis que ha recibido ese usuario.
- Puntuación que tiene el usuario en la plataforma.

La utilidad práctica que proporciona la visualización de estos datos está más que justificada: se pueden filtrar las Kukis por categorías, analizar los mejores usuarios y las mejores frases, realizar búsquedas por palabras clave o nombre de usuario e incluso estudiar la categoría de frases que predomina en cada país; entre muchos otros usos.

Dado el número elevado de frases extraídas (aproximadamente unas dos mil) se procede a configurar un Core para especificar qué parámetros se quieren indexar.

Código 5.4: Core de KukiNet

```
@prefix kuki : <http://www.kukinet.com/> ;
@filter rdf:type is kuki:Item ;

        text = kuki:text :: xsd:string ;
        score = kuki:score :: xsd:string ;
        category = kuki:category :: xsd:string ;
        delivered = kuki:delivered :: xsd:string ;
        latitude = kuki:latitude :: xsd:string ;
        longitude = kuki:longitude :: xsd:string ;
        username = kuki:username :: xsd:string ;
        userdescription = kuki:userdescription :: xsd:string ;
        usercountry = kuki:usercountry :: xsd:string ;
        uservoted = kuki:uservoted :: xsd:string ;
        userreceived = kuki:userreceived :: xsd:string ;
        userscore = kuki:userscore :: xsd:string ;
```

Tras esto, basta con acceder a la ruta correspondiente al Core y configurar los Widgets pertinentes para sacar el máximo provecho a estos datos. Una configuración posible puede observarse en la figura 5.3.

Figura 5.3: Kukinet

5.3. Validación en proyectos reales

El proceso de validación, en el contexto de la ingeniería de software, es el proceso de revisión gracias al cual se comprueba que el sistema de software producido cumple con las especificaciones y, por tanto, con su cometido.

Por ello, la mejor manera de validar SEFARAD es mediante su aplicación directa en distintos proyectos que se están desarrollando en el Departamento y que tratan con datos semánticos de diversa índole.

5.3.1. Episteme

5.3.1.1. Introducción

Episteme es un creador de oportunidades que posibilita la creación de consorcios entre empresas. Está diseñado para sugerir, dado un consorcio configurado, las empresas que mejor encajan en dicho consorcio gracias a la información semántica que se tiene de cada empresa.

5.3.1.2. Aplicando SEFARAD

SEFARAD se aplica en este caso para posibilitar la visualización de relaciones semánticas entre las empresas que forman parte en la plataforma tecnológica INES.

El RDF que describe a una empresa cualquiera es el siguiente:

Código 5.5: RDF de una empresa (Proyecto Episteme)

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ecos="http://kmm.lboro.ac.uk/ecos/1.0#"
xmlns:v="http://www.w3.org/2006/vcard/ns#">
  <ecos:Enterprise rdf:about="http://www.ines.org.es/content/atos">
    <ecos:General>
      <rdf:Description>
        <ecos:Address>
          <rdf:Description>
            <v:VCard>
              <rdf:Description>
                <v:adr>
                  <rdf:Description>
                    <v:locality>Madrid</v:locality>
                    <v:postal-code>28037</v:postal-code>
                    <v:street-address>Albarracín, 25</v:street-address>
                  </rdf:Description>
                </v:adr>
              </rdf:Description>
            </v:VCard>
          </rdf:Description>
        </ecos:Address>
        <ecos:CompanyName>
          <rdf:Description>
            <ecos:name>ATOS</ecos:name>
            <ecos:uri>http://www.atos.net</ecos:uri>
          </rdf:Description>
        </ecos:CompanyName>
      </rdf:Description>
    </ecos:General>
    <ecos:Specific>
      <rdf:Description>
        <ecos:Plan>
```

```
      <rdf:Description>

      </rdf:Description>
    </ecos:Plan>
    <ecos:Skill>
      <rdf:Description>
        <rdf:Bag>
          <rdf:Description>
        <ecos:detail>El principal reto para una empresa de consultoría de
        tecnologías de la información no es tanto el presente como el futuro.
        El conocimiento del estado del arte, de las necesidades de sus
        clientes, y de los entornos económico y legal de los escenarios de
        aplicación no son suficientes. Es preciso el desarrollo de unas
        acciones a futuro que incluyan la apertura de nuevos caminos
        tecnológicos mediante las actividades de investigación, el análisis
        de la evolución de las necesidades de sus clientes a largo plazo, la
        perspectiva económica que rodeará el desarrollo de los negocios a
        cinco y diez años vista, y el conocimiento de las políticas
        comunitarias que son de aplicación en los próximos
        ejercicios.</ecos:detail>
          <rdf:li>
              <rdf:Description>
                <ecos:name>Estándares para portar aplicaciones</ecos:name>
                <ecos:level>Intermediate</ecos:level>
              </rdf:Description>
            </rdf:li>
                          [...]
            <rdf:li>
              <rdf:Description>
                <ecos:name>Sistemas de inferencia automáticos para validar y
                completar la información</ecos:name>
                <ecos:level>Intermediate</ecos:level>
              </rdf:Description>
            </rdf:li>
          </rdf:Description>
        </rdf:Bag>
      </rdf:Description>
    </ecos:Skill>
  </rdf:Description>
</ecos:Specific>
<v:VCard>
```

```
      <rdf:Description>
        <v:adr>
          <rdf:Description>
            <v:locality>Madrid</v:locality>
            <v:postal-code>28037</v:postal-code>
            <v:street-address>Albarracín, 25</v:street-address>
          </rdf:Description>
        </v:adr>
        <v:fn>ATOS</v:fn>
        <v:logo>http://www.ines.org.es/sites/default/files/Logo_ATOS_PANTONE3015C.jpg?1315309194</v:l
        <v:org>
          <rdf:Description>
            <v:organisation-name>ATOS</v:organisation-name>
            <v:organisation-unit>Gran Empresa</v:organisation-unit>
          </rdf:Description>
        </v:org>
      </rdf:Description>
    </v:VCard>
  </ecos:Enterprise>
</rdf:RDF>
```

Como puede observarse, se hace uso de la ontología vcard y ecos. Los datos que vamos a manejar son: nombre de la empresa, provincia, tipo de empresa, logotipo y actividad.

Con el fin de poderlo visualizar en SEFARAD podríamos utilizar una consulta SPARQL que recogiera los campos anteriores para posteriormente aplicar los filtrados que se deseen.

La consulta SPARQL tendría la siguiente estructura:

Código 5.6: Consulta SPARQL Episteme

```
SELECT ?nombre ?provincia ?tipo ?logotipo ?actividad
WHERE {
        ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://kmm.lboro.ac.uk/ecos/1.0#Enterprise> .
        ?s <http://www.w3.org/2006/vcard/ns#VCard> ?vcard .
    OPTIONAL{
            ?vcard <http://www.w3.org/2006/vcard/ns#logo> ?logotipo .
    }
    ?vcard <http://www.w3.org/2006/vcard/ns#fn> ?nombre .
    ?vcard <http://www.w3.org/2006/vcard/ns#adr> ?direccionnodo .
        ?direccionnodo <http://www.w3.org/2006/vcard/ns#locality> ?provincia .
```

```
    ?vcard <http://www.w3.org/2006/vcard/ns#org> ?org .
        ?org <http://www.w3.org/2006/vcard/ns#organisation-unit> ?tipo .
        ?s <http://kmm.lboro.ac.uk/ecos/1.0#Specific> ?specific .
        ?specific <http://kmm.lboro.ac.uk/ecos/1.0#Plan> ?plan .
               ?plan <http://kmm.lboro.ac.uk/ecos/1.0#detail> ?actividad
}
```

Y de manera inmediata podemos comenzar a explorar y filtrar las empresas visualizadas mediante la configuración de *Widgets* que se considere oportuna. Un ejemplo de configuración es el mostrado en la figura 5.4.

Otra alternativa es la del uso en LMF de un Core en el módulo de búsqueda semántica para especificar qué parámetros se quieren indexar.

El Core configurado tendría la siguiente estructura:

Código 5.7: Core Episteme

```
@filter rdf:type is <http://kmm.lboro.ac.uk/ecos/1.0#Enterprise> ;

        nombre = <http://www.w3.org/2006/vcard/ns#VCard> /
        <http://www.w3.org/2006/vcard/ns#fn> :: xsd:string ;
        provincia = <http://www.w3.org/2006/vcard/ns#VCard> /
        <http://www.w3.org/2006/vcard/ns#adr> /
        <http://www.w3.org/2006/vcard/ns#locality> :: xsd:string ;
        logotipo = <http://www.w3.org/2006/vcard/ns#VCard> /
        <http://www.w3.org/2006/vcard/ns#logo> :: xsd:string ;
        tipo = <http://www.w3.org/2006/vcard/ns#VCard> /
        <http://www.w3.org/2006/vcard/ns#org> /
        <http://www.w3.org/2006/vcard/ns#organisation-unit> :: xsd:string ;
        actividad = <http://kmm.lboro.ac.uk/ecos/1.0#Specific> /
        <http://kmm.lboro.ac.uk/ecos/1.0#Plan> /
        <http://kmm.lboro.ac.uk/ecos/1.0#detail> :: xsd:string ;
```

Tal y como se observa en la figura 5.5, los diferentes filtros aplicados permitirán descubrir empresas según su tipo y provincia del país. Se pueden efectuar también búsquedas textuales según la actividad de la empresa.

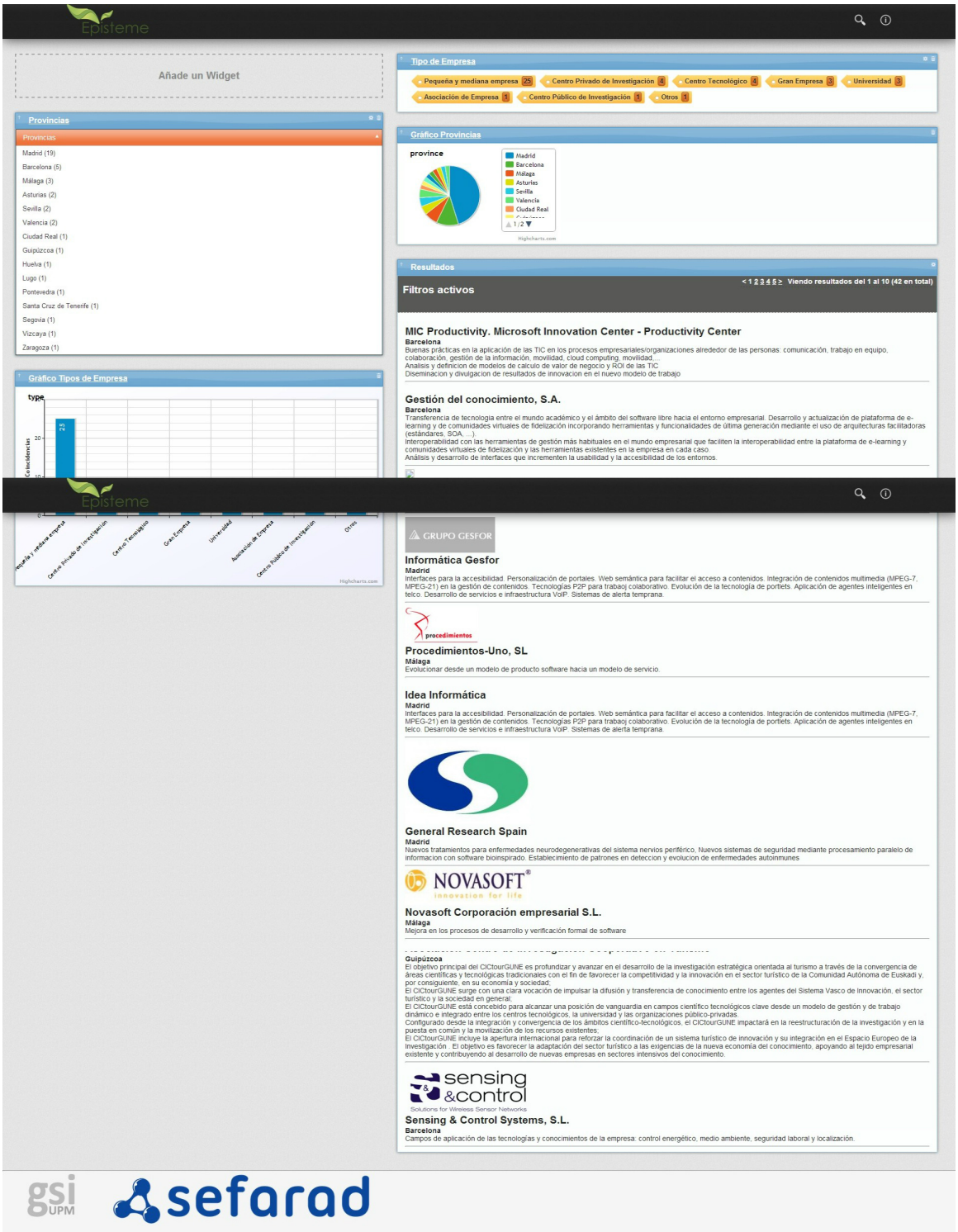

Figura 5.4: Listado de todas las empresas

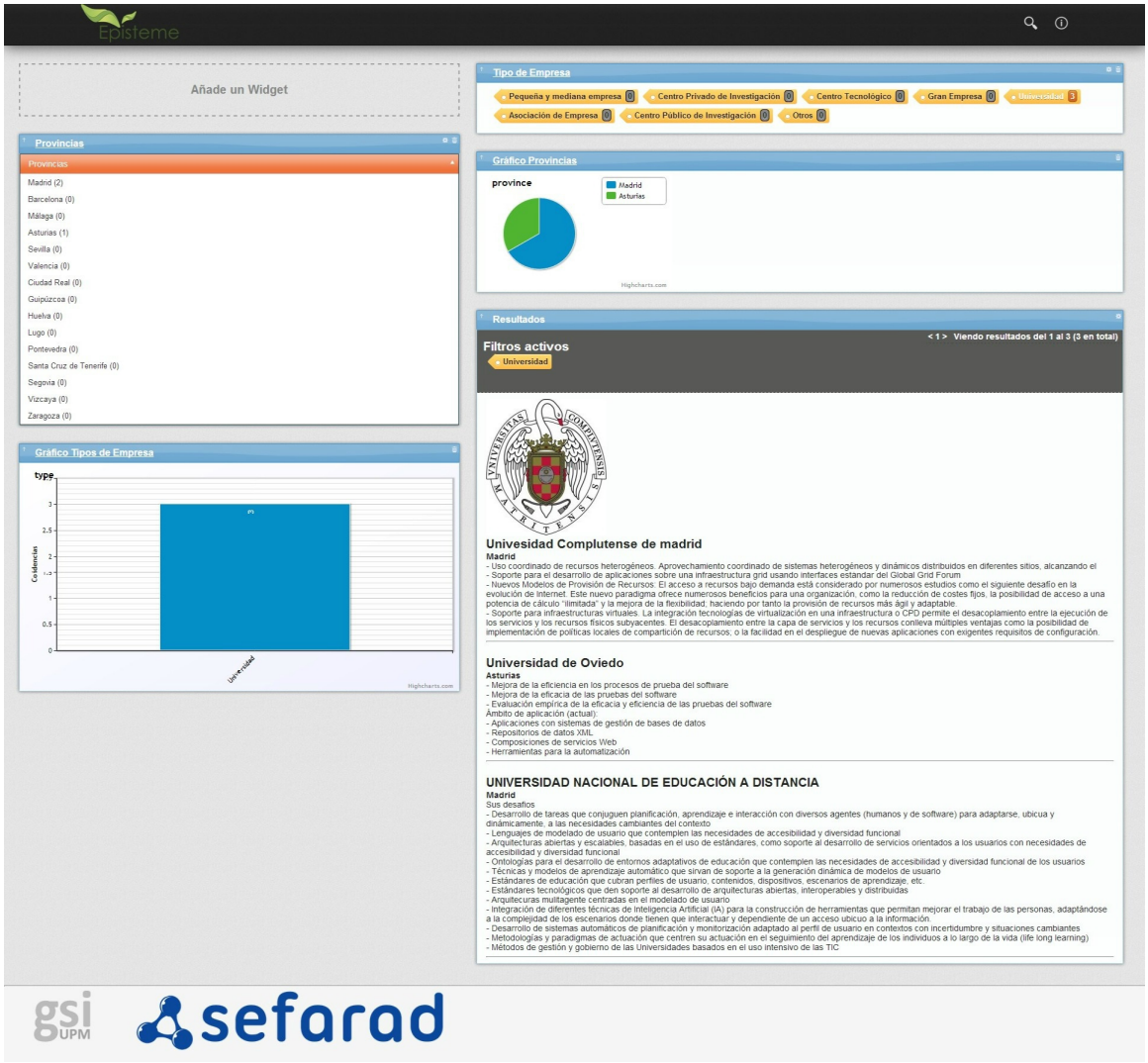

Figura 5.5: Lista de universidades

5.3.2. Financial Twitter Tracker

5.3.2.1. Introducción

El objetivo principal del proyecto *Financial Twitter Tracker* es el enriquecimiento de contenidos financieros con información extraída de medios sociales de Twitter, así como la detección de demanda de nuevos contenidos financieros en determinados tópicos. Con este fin, se realizará un sistema de análisis y evaluación de información financiera en Twitter que extraiga, monitorice, visualice y evalúe el peso, variación, propagación temporal y las relaciones de los datos.

5.3.2.2. Aplicando SEFARAD

La aplicación de SEFARAD en este caso se limita a la visualización de los diferentes Tweets extraídos y a su filtrado manual.

Cada Tweet extraído posee las siguientes propiedades:

Código 5.8: RDF de un Tweet (Proyecto Financial Twitter Tracker)

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
        xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:sioc="http://rdfs.org/sioc/ns#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:marl="http://purl.org/marl/"
        >

        <rdf:Description
        rdf:about="https://api.twitter.com/1/statuses/show/268792368378028032.json">
                <rdf:type
                rdf:resource="http://rdfs.org/sioc/types#MicroblogPost"/>
                <dc:title>Bankia</dc:title>
                <dcterms:created
                rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2009-11-30T21:03:25.194-
                <sioc:has_creator
                rdf:resource="https://twitter.com/andrespuentes"/>
                <marl:opinionText>Bankia sube</marl:opinionText>
                <marl:polarityValue>-0.2</marl:polarityValue>
```

```
            <marl:minPolarityValue>-1</marl:minPolarityValue>
            <marl:maxPolarityValue>1</marl:maxPolarityValue>
            <marl:hasPolarity
            rdf:resource="http://purl.org/marl/ns#Negative"/>
        </rdf:Description>
</rdf:RDF>
```

Podremos por tanto filtrar cómodamente aquellos Tweets con un sentimiento o polaridad positivos o negativos y también por entidades bancarias, como se visualiza en la figura PPP.

Sin embargo, el desarrollo de un *Widget* específico para *Financial Twitter Tracker* realizado en el Departamento tomando como base una versión beta de SEFARAD, permite una visualización más gráfica de estos datos:

Figura 5.6: Financial Twitter Tracker extendido

El esfuerzo que requiere la implementación de este nuevo *Widget* es realmente bajo, basta con nutrir al gráfico con los datos ya filtrados visualizados.

Capítulo 6

Conclusiones y trabajos futuros

"La vida es el arte de sacar conclusiones."

— Samuel Butler

En este último capítulo se resumen las conclusiones obtenidas tras la elaboración del proyecto y se evalúan los objetivos conseguidos. A continuación, se plantean una serie de líneas de trabajo futuras con el fin de mejorar la herramienta desarrollada.

6.1. Conclusiones

El concepto de Web Semántica no debe asociarse a un concepto novedoso que haya surgido en los últimos años. Como se ha visto, la Web Semántica se ha convertido, con el paso de los años, en una evolución de una serie de propuestas desde el seno del W3C hasta convertirse, en el año 2008, en estándares y recomendaciones constituyendo los cimientos de la Web Semántica.

Este hecho ha posibilitado que el uso de datos semánticos se haya ido extendiendo con el paso del tiempo. Cada vez son más las iniciativas que crean o transforman datos planos en datos con contenido semántico con el fin de hacerlos accesibles a otros, como la espectacular DBpedia.

En este contexto, cabe destacar el movimiento *Open Data*, al que se han adherido un gran número de administraciones públicas de todo el mundo. Se está dedicando esfuerzos para centralizar sus bancos de datos y para ofrecerlos públicamente, sin restricciones, en formatos que permitan su procesamiento.

Los datos recopilados desde el sector público se presuponen fiables, veraces y exhaustivos; por lo que pueden gran aportar valor a las aplicaciones y servicios ofrecidos por las administraciones públicas o por terceros, además de jugar un papel clave en la transparencia respecto a la gestión de recursos públicos.

En nuestro país algunos buenos ejemplos del movimiento *Open Data* vienen dados por el Ayuntamiento de Zaragoza[1] o el Principado de Asturias[2].

Así mismo, el número de herramientas que hacen uso de estos datos con contenido semántico ha ido creciendo en paralelo al aumentar el número de datos disponibles, ofreciendo funcionalidades cada vez más interesantes, como la expuesta en la introducción dada por *Google Now*.

En cuanto a SEFARAD, la herramienta desarrollada consecuencia de este proyecto, se han visto implementados de manera satisfactoria todos los requisitos capturados en el capítulo 3. De este modo, SEFARAD se ha convertido en una herramienta polivalente que resulta de gran utilidad para poder crear visualizaciones de repositorios de datos semánticos con extrema facilidad, gracias a la flexibilidad con la que ha sido desarrollada: podrá visualizarse cualquier conjunto de datos semánticos independientemente del número y estructura de los datos que lo compongan.

[1] http://www.zaragoza.es/ciudad/risp/
[2] http://risp.asturias.es/catalogo/index.html

6.2. Trabajos futuros

En todo proyecto de investigación, hay margen para la mejora y para continuar con el trabajo realizado. En el caso de SEFARAD, las mejoras se pueden resumir en las siguientes líneas:

- Aumentar el número de *Widgets* disponibles. Aunque hay 32 widgets disponibles en total, de los cuales 11 son dinámicos y 21 estáticos; podrían incluirse más variantes de gráficos que ayuden a la visualización de los datos que se quieran analizar.
- Mejoras en *Widgets* ya existentes. Por ejemplo, el *Widget* del Mapa podría resolver direcciones, en vez de aceptar únicamente coordenadas (latitud y longitud).
- Permitir una mayor personalización en relación a la interfaz de usuario: ofrecer la posibilidad de seleccionar en la propia interfaz de SEFARAD entre varios temas de colores y fuentes disponibles.

Bibliografía

[1] Google. Knowledge Graph. http://www.google.com/insidesearch/features/search/knowledge.html

[2] Google España. Presentamos el Gráfico de conocimiento en español: cosas, en vez de solo palabras. http://googleespana.blogspot.com.es/2012/12/presentamos-el-grafico-de-conocimiento.html

[3] Linked Open Data http://linkeddata.org/home

[4] W3C. World Wide Web Consortium - Semantic Web. http://www.w3.org/standards/semanticweb/

[5] W3C. RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[6] W3C. RDF Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/

[7] W3C. RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/

[8] W3C. RDF Semantics. http://www.w3.org/TR/rdf-mt/

[9] W3C. RDF Schema. http://www.w3.org/TR/rdf-schema/

[10] W3C. RDF Semantics. http://www.w3.org/TR/rdf-testcases/

[11] W3C. OWL Overview. http://www.w3.org/TR/owl-features/

[12] W3C. OWL Guide. http://www.w3.org/TR/owl-guide/

[13] W3C. OWL Reference. http://www.w3.org/TR/owl-ref/

[14] W3C. OWL Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/

[15] W3C. OWL Use Cases and Requirements. http://www.w3.org/TR/webont-req/

[16] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.

[17] W3C. SPARQL Protocol for RDF. http://www.w3.org/TR/rdf-sparql-protocol/.

[18] W3C. SPARQL Query Results XML Format. http://www.w3.org/TR/rdf-sparql-XMLres/.

[19] Krzysztof Janowicz, Pennsylvania State University, USA. Approaches to visualising Linked Data: A survey. http://iospress.metapress.com/content/2822p340453463g1/fulltext.pdf.

Apéndice A

Installation guide

If you want to learn how to install SEFARAD and, to extend possibilities, how to install *Linked Media Framework* (LMF) this is the guide you are looking for.

A.1. Installing SEFARAD

This is going to be a short section. Unzip SEFARAD.zip file and... that's all! You don't even need to include SEFARAD in any running Apache Server.

You can access to index.html and play with some demos. Check our faq.html to understand all the given posibilities and needed HTTP routes that make SEFARAD work. But, if you already now how the routes work, go ahead!

A.2. Installing LMF

A.2.1. Introduction

Installing LMF is not necessary at all in order to get SEFARAD working. If you already have an SPARQL endpoint and you know how to make SPARQL queries, then you can skip this section.

But if you don't, or if you have a long collection of semantic data that you want to visualize, you should use the SEFARAD + LMF scheme.

The Linked Media Framework is an easy-to-setup server application that bundles central Semantic Web technologies to offer advanced services.

LMF currently supports the following tasks out of the box:

- Publishing Legacy Data as Linked Data: Whenever you have legacy datasets in CSV, Excel, XML or similar and want to publish it as Linked Data.
- Building Semantic Search over your Data: You have data in some format and you want to enrich it with content from the Linked Data Cloud to provide advanced Semantic Search.
- Using a SKOS Thesaurus for Information Extraction: You have a custom thesaurus and you want to analyse and automatically interlink content based on its concepts.

And more features that you can discover on its official website[1].

In this guide, you are going to learn how to do the basic setup on LMF so you can make queries to its SPARQL endpoint. You will also learn how to setup Cores that work with its

[1] http://code.google.com/p/lmf/

Semantic Search module (Apache SOLR) so you can handle big amounts of data server-side keeping the client lightweight.

A.2.2. Requirements

As stated on the LMF website, the requirements are the following:

Hardware related

- Standard workstation (Dual-Core or similar)
- 1GB main memory
- About 100MB hard disk

Software related

- Java JDK 6 or higher
- Java Application Server (Tomcat 6.x or Jetty 6.x)
- Database (PostgreSQL, MySQL - if not explicitly configured, an embedded H2 database will be used)

A.2.3. How to: Installation steps

First of all, you must go to the official's website downloads section[2].

As you can see, there are three different ways of installation for each version. In this guide, we are going to explain two of them. It will be easy for you to choose, depending on if you already have an Apache server running.

If you do, you should download the .war file and deploy it as usual.

If you don't, you should download the .jar file, that's a standalone installer that has Apache Tomcat included.

A.2.3.1. First option: deploying war file

Please, follow the next steps:

[2] http://code.google.com/p/lmf/downloads/list

1. Deploy the LMF.war file into the desired Tomcat.

2. Add the following settings to catalina.sh:
   export LMF_HOME=YOUR_ROUTE
   export JRE_HOME=YOUR_ROUTE
   export JAVA_OPTS=Djava.net.preferIPv4Stack=true -Xmx1024m -XX:PermSize=128m
   -XX:MaxPermSize=256m -XX:+UseConcMarkSweepGC -XX:+CMSPermGenSweepingEnabled
   -XX:+CMSClassUnloadingEnabled"

3. Start the server and shutdown it so the initial files and directories are created.

4. Edit the $LMF_HOME/system-config.properties file and configure the following properties:
   kiwi.context = tomcat_url/LMF/
   kiwi.host = tomcat_url/LMF/
   prefix.knowledgespace = tomcat_url/LMF/knowledgespace
   prefix.context = tomcat_url/LMF/context
   prefix.local = tomcat_url/LMF/local

5. Relaunch your Tomcat and go to the kiwi.host url. A login form will popup. Default credentials are ”admin” and ”pass123”. At this point, you should know that default security profile is ”simple”. What does this mean? Let’s explain the three different security profiles available:

   - Simple: Simple security profile allowing read access by anyone and write access only from localhost.
   - Standard: Security profile allowing read access by anyone and write access only from authenticated users of the manager role.
   - Restricted: Security profile allowing read access only from authenticated users and write access only from authenticated users of the manager role.

   So if you are having some issues because you computer is not at localhost where the server is, you should change this ”simple” profile to any other. Here is an small example that shows how to change the security profile to ”standard”:
   Remote connect through ssh to the hosting machine and run the following:

```
curl -X POST -H "Content-Type: application/json" -d '["standard"]'
http://<MACHINE_NAME>:<PORT>/LMF/config/data/security.profile
```

A.2.3.2. Second option: running .jar file

This option will open a setup installer. Follow the steps and you are done because installation process also installs an optional Apache Tomcat. Best option for installing LMF at localhost on your own machine.

A.2.4. Control panel

Once you start Apache Tomcat, you can access to kiwi.host route and LMF control panel will show up (figure A.1).

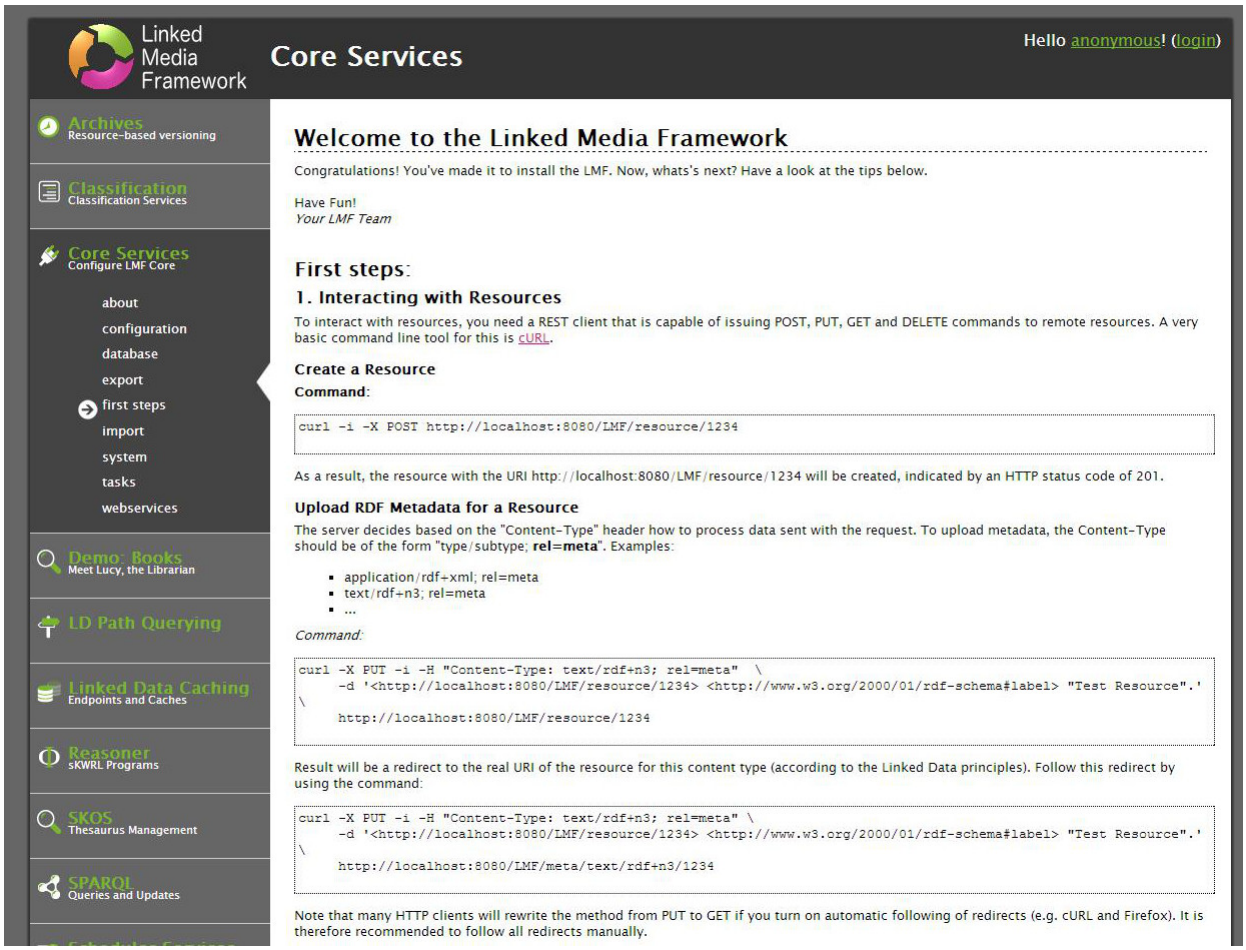

Figura A.1: LMF Control Panel

You can browse through all modules from this screen. With SEFARAD, you are probably going to use this three modules mainly:

- Core Services: You can import data into database from here.
- SPARQL: Here you can make your own SPARQL queries and observe plain results using snorql or you can even make graphs of your data with sgvizler. Make those queries from SEFARAD and explore your results.
- Semantic Search: You must create a new Core using LDPath query language[3] so you can specify which fields you want Apache SOLR to index. Then you can visualize this data from SEFARAD.

Note: Mind that, by default, in Semantic Search configuration solr.local_only is set to true. You might need to switch it to false, so all resources will be considered by the indexer.

[3] http://code.google.com/p/ldpath/

Apéndice B

User manual

This small guide is addressed to all users who want to know all the possibilities that SEFARAD can provide.

As it is a small guide, there is no index. Start reading from beginning to discover all features and enjoy it!

B.1. First steps

First off, you may know that you can use SEFARAD in three different ways. Please, open index.html file to discover them:

- Demo mode: This is a basic example of what you can do in SEFARAD thanks to an automatic query to dbpedia's SPARQL endpoint. You, as an user, can instantly browse through all data. And you, as an administrator, can create new routes that link to your own demos.
- Explore your own queries: If you know an interesting SPARQL endpoint and you have SPARQL querying knowledge, then this is an interesting choice for you. After this small setup, create your own Widgets and browse through all data.
- LMF + SEFARAD: Firstly, you should follow the previous appendix A in order to install LMF. Then, you can import your semantic data and explore it thanks to SEFARAD. You have two possibilities, you can do SPARQL queries to LMF's endpoint or you can configure a new Semantic Search Core and visualize it from SEFARAD.

Let's study this three possibilities in detail.

B.2. Demo mode

This is what you can see when opening this demo:

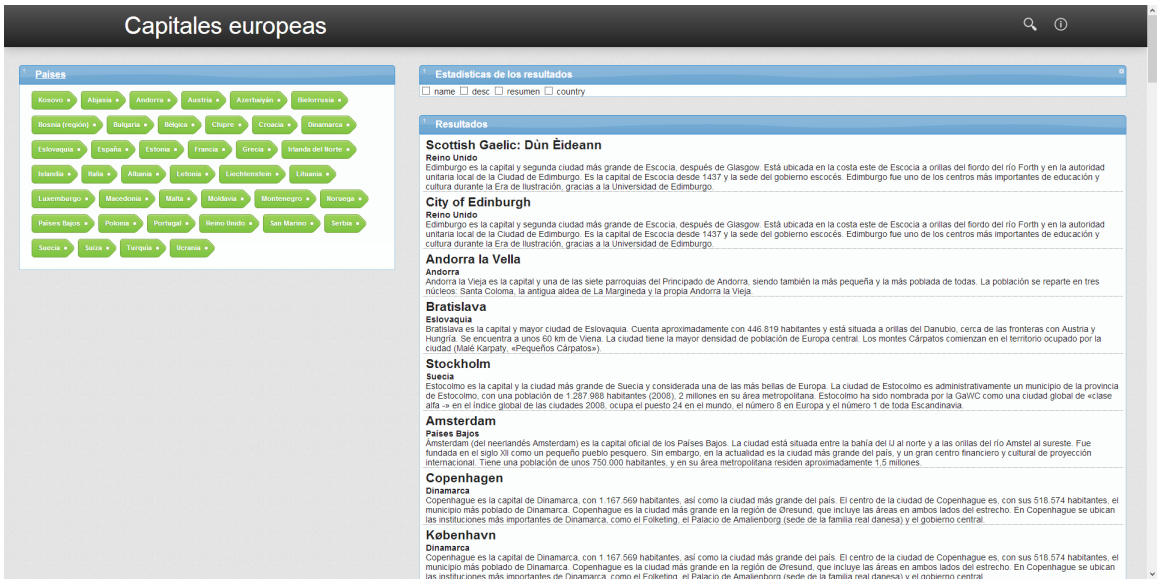

Figura B.1: Demo screen

As you can perceive, there are two main columns. In the left one, there are movable Widgets that let you filter the data showed in the Results Widget, placed at the right column. Clicking on the different tags will let you filter/unfilter data with that tag content.

There is a search box at the top of the page and a small help guide icon also.

Finally, in the result's stats Widget you can generate pie charts so you can see different results' data proportion in a easy way.

B.3. Explore your own SPARQL queries

First of all, you must configure the SPARQL endpoint where you are going to make your queries. To do so and make your first query, please open the configuration panel clicking at the icon on the right side of the screen as shown in the next screenshot:

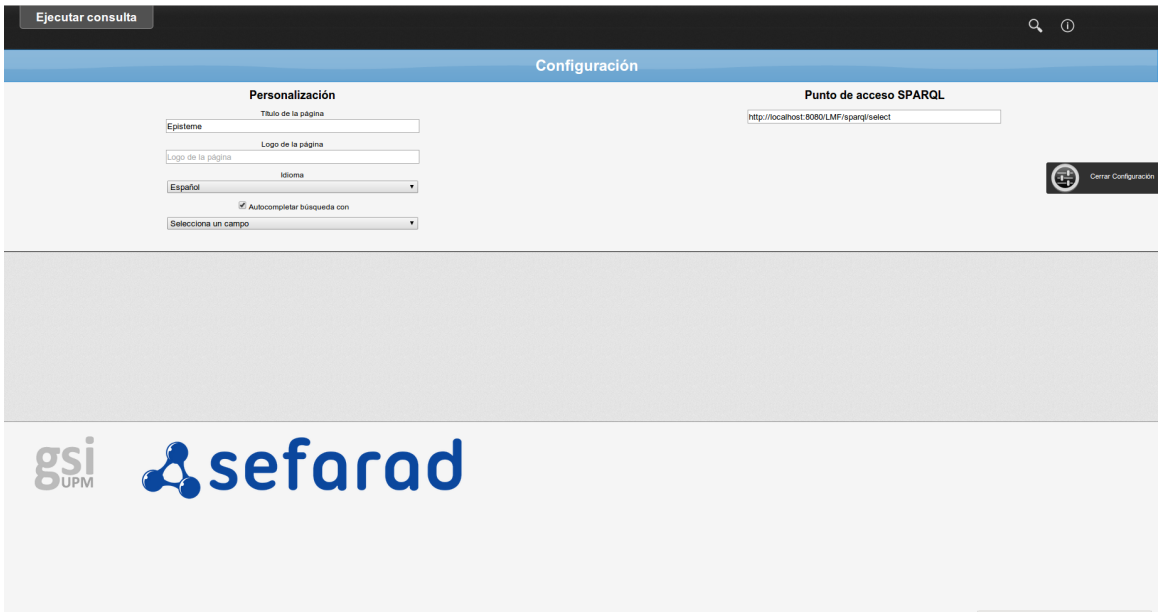

Figura B.2: Configuration panel

Add the SPARQL endpoint and make your own SPARQL query. You will have to configure the Results Widget so you can specify what fields you want to visualize.

Then you can add as many Widgets you want. Clicking on "Add widget" will open a wizard as following:

B.3.1. Types of Widgets

As seen on the above screenshot, there are many types of Widgets available from Search tab:

- Results Widget: Vertical and grid styles, this is where you visualize the current results after applying all desired filters.
- Tagcloud Widget: The name defines itself. You select which field you want to generate a tagcloud from. This Widget is also available with a vertical list style.
- Numeric filter Widget: If you have some fields with numeric values then you might be interested on filtering by range.
- Gauge: Shows the current number of results.
- Map Widget: If you have latitude and longitude fields, what's better than a map in order to visualize them.

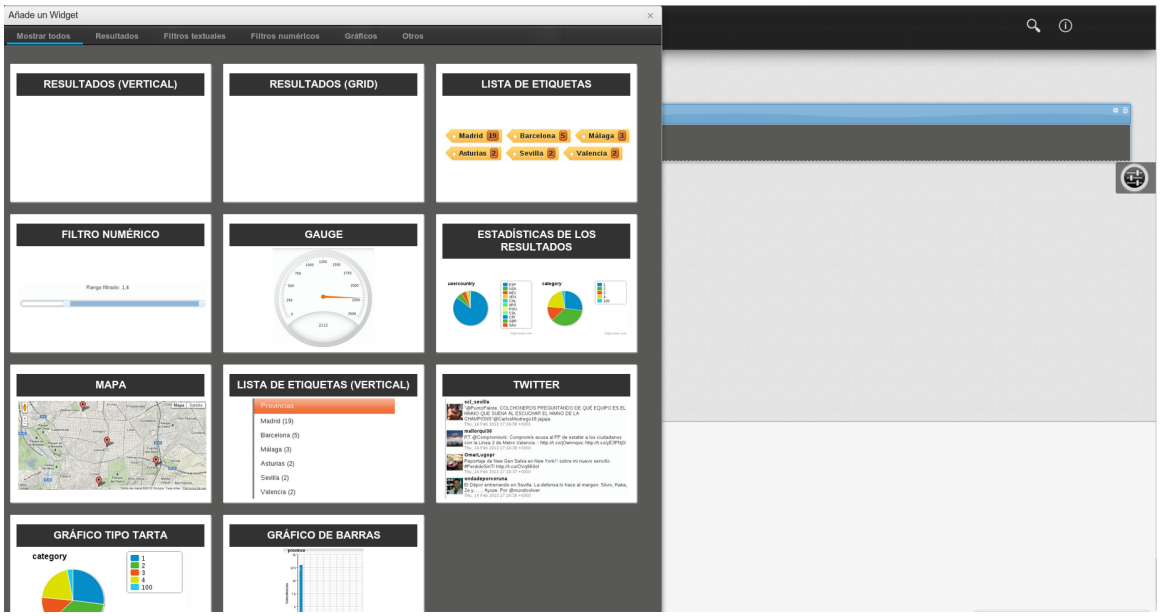

Figura B.3: Widget's wizard

- Twitter Widget: See whats going on about the field content you choose in this popular social network. For example, if you have a field "name" that contains names of enterprises, you might be interested on seeing what are Twitter users saying about them.
- Graphs: With pie and vertical bars style.

At the control panel tab, you can create many static widgets (SPARQL knowledge required). For example, you can generate your own static pie chart using a SPARQL query.

B.4. SEFARAD + LMF

This mode requires some previous steps before working, listed below:

- Install LMF. See previous appendixA.
- Import your semantic data through LMF control panel.
- Configure a new Semantic Search Core so you can index the fields you want.
- Access to SEFARAD and choose the appropriate Core.
- Enjoy!