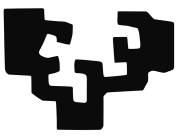
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Prototipo para la integración de datos públicos

Autor

Unai Garciarena Hualde

2015

Resumen

La herramienta pretende integrar datos de la UPV/EHU públicamente accesibles y establecer puentes entre distintos repositorios de información del ámbito académico. Para ello, aprovecharemos el *Linked Data*, "la forma que tiene la Web Semántica de vincular los distintos datos que están distribuidos en la Web"[1], y los estándares que éste define.

Los repositorios elegidos para este trabajo han sido el ADDI, Bilatu, las páginas de todos los centros de la UPV/EHU en el Campus de Gipuzkoa y la DBLP.

La mayoría de las funcionalidades de esta aplicación son genéricas, por lo que podrían fácilmente aplicarse a repositorios de otras instituciones. El sistema es un prototipo que demuestra la factibilidad del objetivo de integración y que está abierto a la incorporación de más conjuntos de datos, siguiendo la misma metodología empleada en el desarrollo de este proyecto.

Índice general

Resumen I

Índice general III

Índice de figuras VII

Indice de tablas IX

1. Introducción 1

2. Alcance del proyecto 7

2.1. Objetivo principal 7

2.2. Alcance mínimo 8

2.3. Posibles mejoras 8

2.3.1. Relación entre recursos 8

2.3.2. Obtención de información 8

2.3.3. Consultas 9

2.4. Estructura de descomposición de trabajo 9

2.5. Tareas 9

2.6. Diagrama de Gantt 11

2.7. Riesgos 12

3. Contexto tecnológico **13**

3.1. Web Semántica 13

3.2. RDF 15

3.3. Linked Data 17

3.4. SPARQL 18

3.5. Stardog 20

3.6. Sails.js 21

3.7. DSpace 21

3.8. Protégé 23

3.9. Antecedentes 23

4. Desarrollo del proyecto **25**

4.1. Obtención de la información 25

4.1.1. DSpace 25

4.1.2. Primera aproximación 27

4.1.3. Segundo acercamiento 30

4.1.4. Método definitivo 32

4.1.5. Bilatu 33

4.1.6. Páginas de las facultades 33

4.1.7. DBLP 33

4.2. Arquitectura 34

4.2.1. Capa de presentación 34

4.2.2. Capa de negocio 34

4.2.3. Capa de acceso a otras fuentes 35

4.3. Desarrollo de ontologías 36

4.3.1. Ontología interna 36

4.3.2. Ontología externa . . . 38
4.4. Parte cliente . . . 41
4.4.1. Sails.js . . . 41
4.4.2. Consultas cocinadas . . . 42
4.4.3. Consultas crudas . . . 42
4.5. Gestión . . . 44
4.5.1. Seguimiento y control . . . 44

5. Conclusiones y vista al futuro **47**
5.1. Conclusiones . . . 47
5.1.1. Utilidad de la tecnología de la Web Semántica . . . 47
5.1.2. Genericidad de la metodología . . . 47
5.1.3. Utilidad de la aplicación . . . 48
5.2. Lineas de desarrollo . . . 48
5.2.1. Otros repositorios externos . . . 48
5.2.2. Otros repositorios internos . . . 49
5.2.3. Otros repositorios DSpace . . . 49
5.2.4. Interfaz . . . 49
5.2.5. Aplicación . . . 49

Anexos

A. Funciones **53**

B. Ontologías **63**

Bibliografía **69**

Índice de figuras

1.1. Datasets publicados en 2007 4

1.2. Datasets publicados en 2011 4

1.3. Datasets publicados en 2014 5

2.1. Estructura de Descomposición de Trabajo 10

2.2. Diagrama de Gantt 11

3.1. Calidad de la información según Berners-Lee 17

3.2. Diagrama de funcionamiento del DSpace 22

4.1. Pantalla inicial DSpace 26

4.2. Metadatos en RDF 27

4.3. Próxima página 27

4.4. Guardar Página 29

4.5. Acceder a la próxima página 29

4.6. Ejemplo de archivos después de la ejecución de DSpace.java 35

4.7. Arquitectura de la aplicación 36

4.8. Términos más útiles para nosotros en la ontología HERO 39

4.9. Términos interesantes para nuestra información en la ontología VIVO . . 39

4.10. Diagrama de la ontología 40

4.11. Ejemplo de la interfaz 43

4.12. Uso de la aplicación 43

4.13. Linked Data 44

4.14. Diagrama de Gantt 45

Indice de tablas

2.1. Tabla de estimación temporal 9

2.2. Tabla riesgos 12

3.1. Triples 16

4.1. Horas aproximadas por tareas 46

CAPÍTULO 1

Introducción

Encontramos los orígenes del Linked Data en un artículo escrito por *el padre de la web* Tim Berners-Lee en 2006 . En él sentó las bases para lo que considera "*la Web Semántica bien hecha*", el Linked Data.

La popularidad del paradigma ha ido aumentando exponencialmente desde entonces, como podemos comprobar en estos diagramas creados por Max Schmachtenberg, Christian Bizer, Anja Jentzsch y Richard Cyganiak en `http://lod-cloud.net/`, que representan datasets publicados respetando la directrices descritas por Berners-Lee, en 2007 Figura 1.1, 2011 Figura 1.2 y 2014 Figura 1.3.

Encontramos en el DSpace un repositorio que recientemente se ha unido al movimiento, y que desde la versión 1.5 (publicada en marzo de 2008 [2]) ofrece una interfaz pública en la que podemos obtener metadatos del contenido del que dispone en distintos formatos, entre los que está el RDF, el cual aprovecharemos para realizar este trabajo.

El modelo RDF ha sido escogido por varias razones.

- DSpace ya nos ofrece los metadatos en formato RDF.
- Nos permite realizar consultas complejas.
- Los metadatos siguen ya una ontología estandarizada, la del Dublin Core
- Es un estándar extendido que podrá ser utilizado por otros trabajadores de la Web Semántica.
- Facilidades en futuras migraciones.

Pongamos que un alumno de la Facultad de Psicología elige comenzar con su TFG sobre Psicología Infantil, y desea hacerlo en euskera. Podría, por ejemplo, navegar por la página de su facultad, hasta llegar al departamento que trata ese tema y descubrir qué profesores podrían ser potenciales directores. Luego, para empezar a documentarse, buscaría en *ADDI* trabajo ya existente sobre el tema, para ver hasta dónde ha llegado la investigación en este ámbito, y poder continuar desde ese punto, sin repetir trabajo ya realizado. Después, debería consultar *Bilatu*, para encontrar el correo y el número de teléfono de su despacho. Además, tendrá que saber también si es bilingüe. Para finalizar, quizá le resulte interesante las publicaciones que tenga dicho profesor en distintos repositorios.

En total, el alumno tendrá que visitar cuatro páginas, suponiendo que estén todas activas en el momento que desee usarlas. Esta aplicación simplificará este proceso, integrando en un único *contenedor* los datos de todas estas páginas. Dada su estructura de grafo, podremos obtener toda esta información desde cualquiera de *puntos de acceso*. Por ejemplo, podremos buscar el nombre del departamento y obtener los nombres de todos los profesores que están adjudicados a él. Y desde el profesor, podremos ver si domina el euskera, las publicaciones que tiene, los modos de contacto que tenemos. Pero también podremos buscar entre los recursos "Psicología Infantil" y recuperar del repositorio todos los trabajos que tienen algo que ver con el tema. Y de ahí podríamos tirar del hilo, hasta la página de un profesor.

Nuestro punto de partida ha sido la instancia del DSpace para la UPV/EHU, el ADDI (*Archivo Digital para la Docencia y la Investigación de la Universidad del País Vasco*). El ADDI es un depósito de documentos digitales, cuyo objetivo es organizar, archivar, preservar y difundir en modo de acceso abierto la producción intelectual resultante de la actividad docente e investigadora ejercida en la Universidad del País Vasco[3].
Del ADDI hemos extraído los metadatos de todos los recursos registrados desde que se abrió en 2010. Después, obtenemos la información que necesitemos del *Staff Directory* y la anotamos, relacionando los resultados con una URI. Si el profesor pertenece a algún centro del Campus de Guipuzcoa, buscamos su nombre en la página web de dicho centro y así podemos conocer el tipo de contrato que lo une a la universidad, además de si posee un doctorado y si habla euskera. Como extra, encontramos una última funcionalidad, exclusiva para los profesores de nuestra facultad: buscar la página del profesor en la DBLP, y si existe, la anota junto con el resto de información del individuo.

Una vez que tenemos los datos anotados, entra en juego la "parte cliente"de la aplicación. Ésta se alimenta de los datos recogidos anteriormente para poder realizar consultas más ricas, que hasta ahora solo podíamos resolver mediante una navegación y búsqueda manual por las páginas correspondientes.

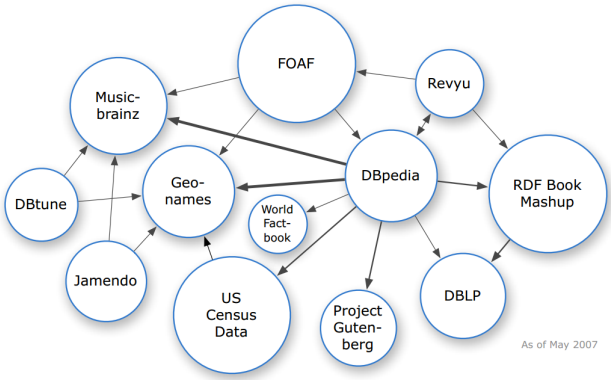


Figura 1.1: Datasets publicados en 2007

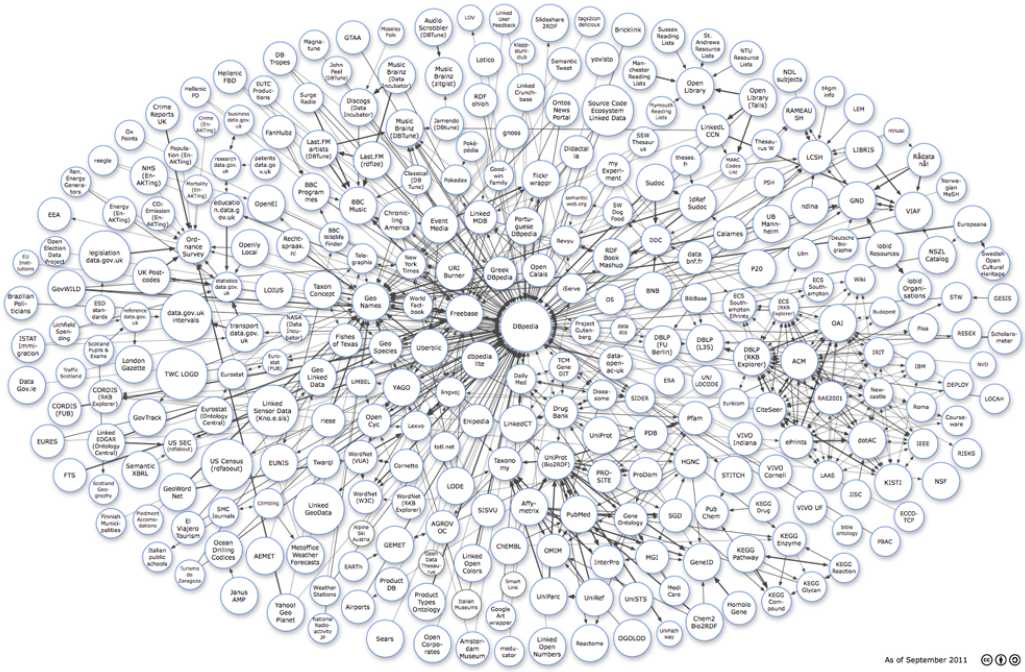


Figura 1.2: Datasets publicados en 2011

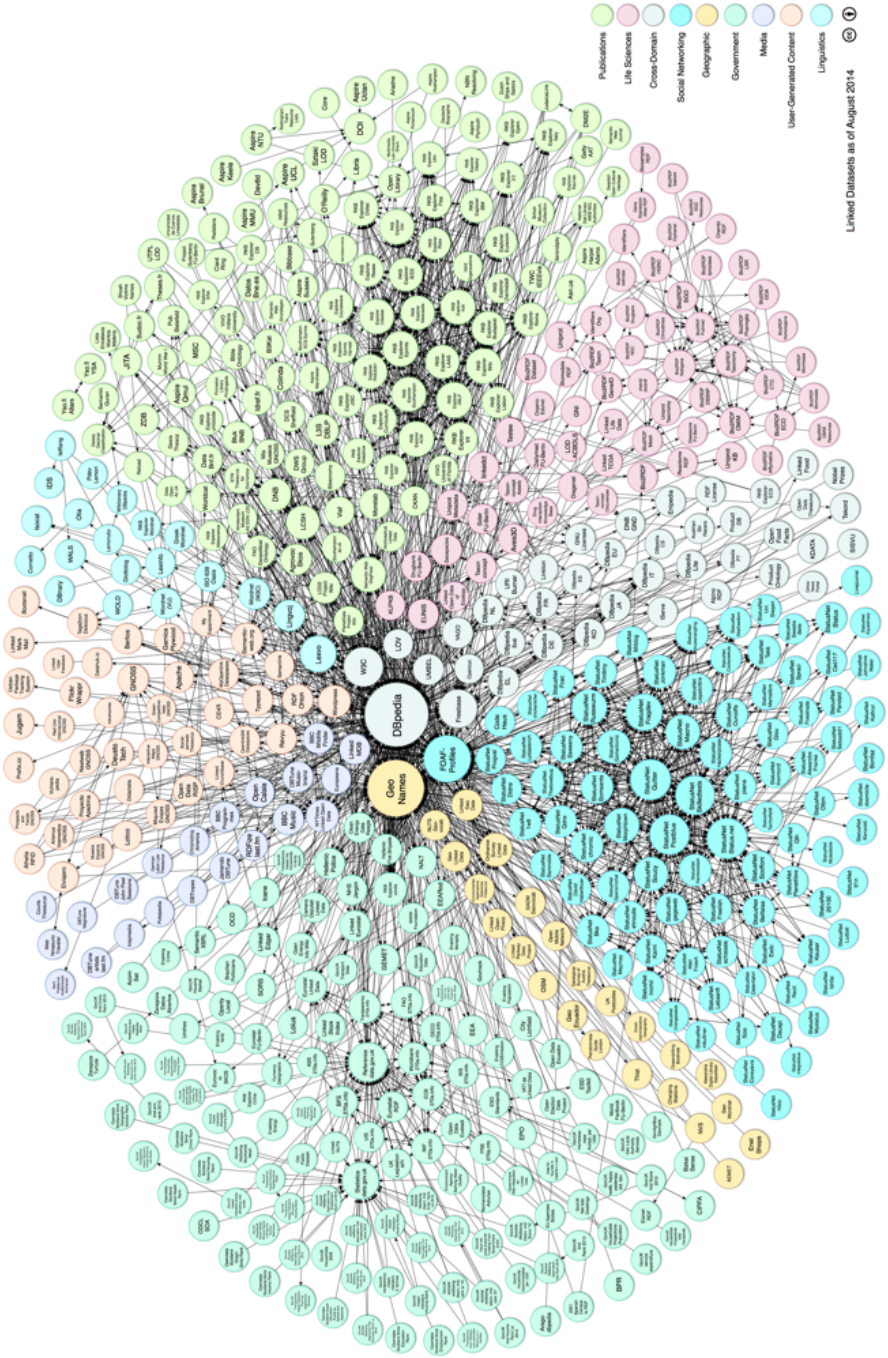


Figura 1.3: Datasets publicados en 2014

CAPÍTULO 2

Alcance del proyecto

Este capítulo comienza fijando el objetivo principal y el alcance mínimo propuesto para el proyecto al principio del mismo. Después encontramos objetivos intermedios, es decir, el punto en el que estaremos al final de cada fase de trabajo. También están contenidos en el capítulo cuestiones relacionadas con la gestión del tiempo en lo que se refiere al proyecto, como la Estructura de Descomposición del Trabajo, Diagrama de Gantt o una previsión de riesgos y una serie de soluciones para disminuir su efecto en la medida de lo posible.

2.1. Objetivo principal

El objetivo principal del proyecto es el de desarrollar una aplicación que sea capaz de integrar datos de distintos repositorios del ámbito académico. Para ello, comenzaremos por obtener metadatos de los recursos almacenados en el ADDI (potencialmente de cualquier servidor que utilice DSpace) y otros de distintas bases de datos para relacionarlos de forma que nos aporte facilidades a la hora de consultar la información. Finalizaremos presentando los datos en una "parte cliente"que realizará dichas consultas.

2.2. Alcance mínimo

El alcance mínimo se ha fijado en relacionar los datos de al menos dos depósitos. El primero el ADDI, el principal. El segundo será el Bilatu, que es un repositorio en el que encontramos mucha información concentrada, y de una forma casi trivial. El puente entre éstos dos serán los individuos que sean *Autores* y *Colaboradores* de todos los recursos contenidos en el primero, y tengan una entrada en el segundo.

Creando estas relaciones conseguimos hacer búsquedas que hasta el momento era imposible hacer en una única navegación, utilizando datos de los que la universidad ya disponía. La aplicación dispone de una interfaz gráfica, pensada para usuarios sin conocimientos altos de informática, y una opción de introducir una consulta en *SPARQL* para gente que sí los tenga.

2.3. Posibles mejoras

Si al cumplir los mínimos de una fase disponemos de más tiempo, iremos mejorando, añadiendo más funcionalidades, o fuentes de datos, de la siguiente manera:

2.3.1. Relación entre recursos

En este aspecto la mejora consistiría en incluir información de otros repositorios que estén implementados con DSpace, para poder así enriquecer la base de datos de la aplicación, y relacionar recursos de distintas instituciones.

2.3.2. Obtención de información

En el caso de que sobre tiempo en la fase en la que obtendremos la información del ADDI y Bilatu, se buscarán más fuentes de información que explotar. Bases de datos similares al DSpace, que contengan recursos e información sobre sus creadores, o solo información académica y profesional sobre una persona, como LinkedIn.

Otra vía de mejora en este aspecto podría ser la de obtener información sobre integrantes de otras universidades que utilicen el DSpace como herramienta para disponer de sus recursos *on-line*. Es decir, el equivalente a Bilatu de otras universidades. De esta forma, sería igualmente válida tanto para la UPV/EHU como para otras instituciones.

2.3.3. Consultas

Primeramente se implementarán las opciones de búsqueda que ofrece el ADDI. El implementar más criterios de búsqueda se plantea como un mínimo del proyecto, pero también es una vía de mejora, ya que si sobrase tiempo al final del proyecto, se dedicaría a implementar más consultas distintas.

2.4. Estructura de descomposición de trabajo

La siguiente figura 2.1 contiene la EDT del proyecto:

2.5. Tareas

A continuación, encontraremos la tabla 2.1 que relaciona las tareas especificadas en la EDT anterior con una primera estimación de dedicación de tiempo.

Tarea	**Estimación (en horas)**
Formación sobre el Linked Data	30
Formación DSpace	10
Formación Stardog	10
Formación clases Web de Java	10
Formación Jena	30
Obtener información DSpace	40
Obtener información Bilatu	20
Implementación de la aplicación	50
Diseño de la interfaz	20
Reuniones + Seminarios	10
Redacción de la Memoria	100
Total	330

Tabla 2.1: Tabla de estimación temporal

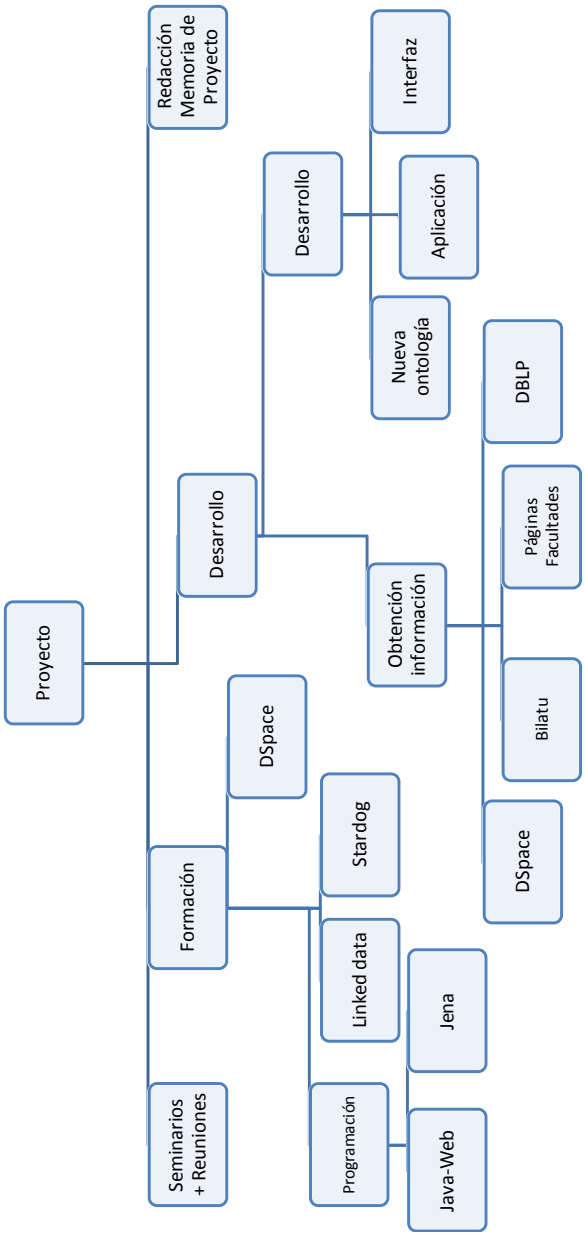


Figura 2.1: Estructura de Descomposición de Trabajo

2.6. Diagrama de Gantt

Ahora plasmaremos esas tareas en la siguiente figura 2.2, un diagrama de Gantt, para ver esa información gráficamente.

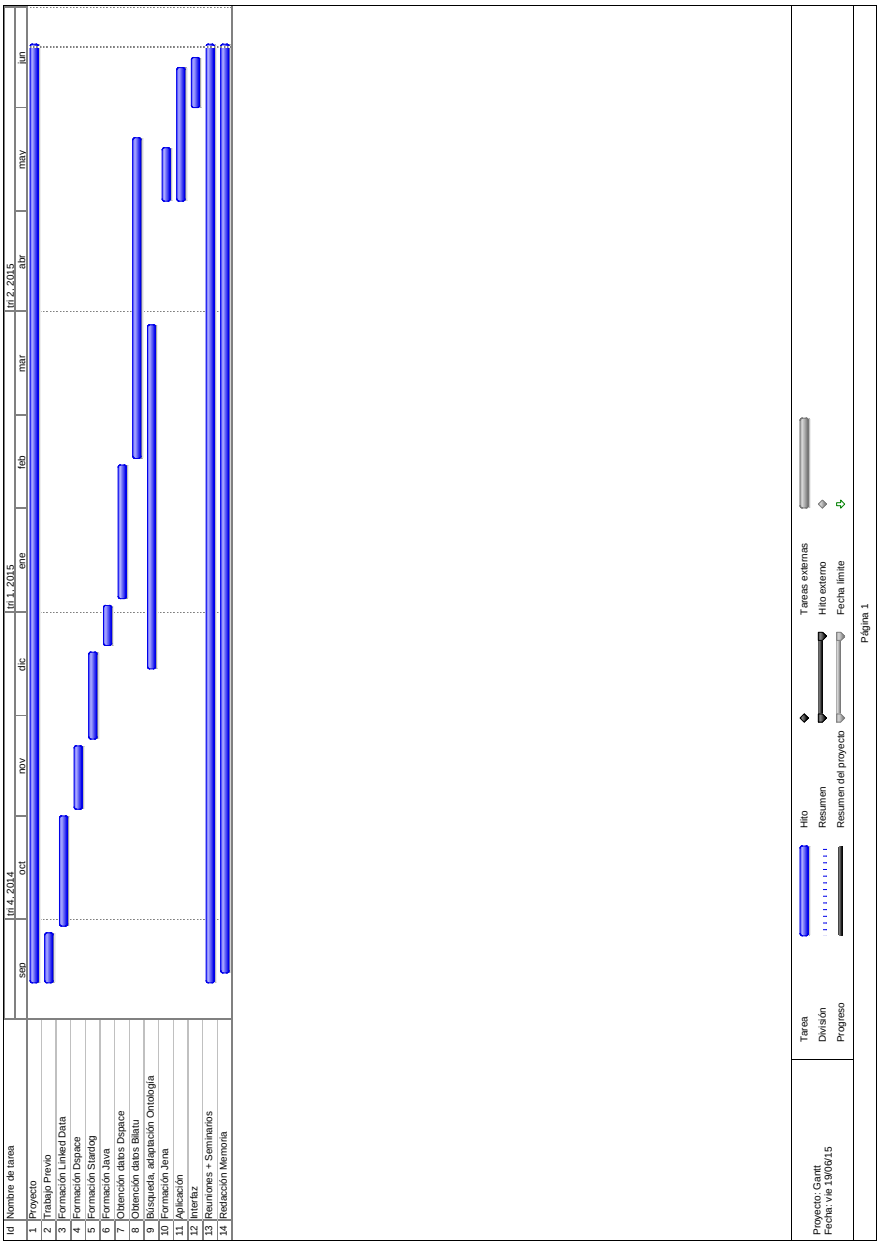


Figura 2.2: Diagrama de Gantt

2.7. Riesgos

Como todo proyecto, es necesario hacer una previsión de riesgos que cubra posibles eventos que puedan afectar negativamente al desarrollo del proyecto. En este apartado intentaremos cumplir esto último, mencionando posibles problemas que podamos encontrarnos, el efecto que pueda tener en la previsión de tiempo que hemos hecho antes, y posibles medidas que podamos tomar para paliarlo.

Situación	**Efecto**	**Medida**
Mala planificación	Alto	No se puede hacer nada.
Malas decisiones de diseño	Alto	Dependerá del momento en el que nos demos cuenta del error. Cuanto más tarde, más afectará.
Pérdida del código, diseño, o memoria	Bajo	Copias de seguridad semanales
Ausencia de conexión a internet	Alta al principio, baja al final	Terminar la parte de obtención de información mientras estoy en el campus

Tabla 2.2: Tabla riesgos

CAPÍTULO 3

Contexto tecnológico

Para entender la motivación del proyecto, el por qué se ha hecho y se ha realizado de esta manera, comenzaremos por introducir el marco tecnológico que se propuso desde un principio. La realización del trabajo en este marco era, a la vez, una oportunidad de aprendizaje de algo totalmente desconocido para el autor en aquel momento, y un reto de conseguir una aplicación con ventajas funcionales frente a lo actualmente existente para resolver el escenario propuesto. Para ello, comenzaremos con el marco general de las tecnologías relativamente recientes que vamos a usar y el estado actual del Linked Data.

3.1. Web Semántica

La Web Semántica (Semantic Web) es extensión de la World Wide Web que permite superar los límites de las aplicaciones y sitios web actuales [4]. Según se describió en el artículo seminal del área, publicado en la revista *Scientific American Magazine* [5]

> La Web semántica no es una web distinta a la actual, sino una extensión de ella, en la cual la información dada tiene un significado bien definido, haciendo así más fácil el trabajo colaborativo de personas y máquinas.

Podríamos hacer una comparación con la web actual, la web del hipertexto. Esta está basada en documentos enlazados mediante *links* que no tienen interpretación y que, por tanto, los programas no pueden procesar significativamente, simplemente son capaces de seguir los enlaces y navegar de una página a otra. En cambio, la Web Semántica propone aumentar esto con una web de datos en la que los recursos enlazados son los propios datos y, además, esos enlaces pueden interpretarse significativamente en base a vocabularios y ontologías establecidas públicamente y de uso compartido. [1]

La Web Semántica nos proporciona un estándar para poder relacionar todos esos datos entre ellos para poder saber, por ejemplo, que un gasto de la tarjeta se corresponde con una foto que se hizo, o dónde se estaba un día concreto [6].

Quizá sea más fácil entender este tema mediante un ejemplo. Imaginémonos que queremos comprar un televisor. En nuestra pared como mucho entraría una de 70 pulgadas y queremos que al menos tenga 55 para que no se nos quede pequeña. Además, queremos que tenga un *input* HDMI o más, y que sea una *Smart TV*. Sin duda, las marcas de televisión tienen bases de datos en las que, con distintos diseños y términos, podríamos realizar consultas para poder estrechar el cerco alrededor de nuestra *tele* ideal. Pero en la situación actual, debemos ir página a página, marca por marca, *Samsung*, *Philips*, *LG*... hasta que nos hartemos de buscar, ya que hoy en día las empresas que producen dispositivos electrónicos son infinitas. Este método de búsqueda es absurdo dados los avances tecnológicos que hemos experimentado los últimos tiempos.

¿No sería mejor poder hacer una única consulta, y que en el resultado encontrásemos todos los televisores que se ajustasen a las características que buscamos? Pues ése es el objetivo que persigue la Web Semántica: mediante ontologías estandarizadas, poder hacer una pregunta basándonos en ellas, y que se nos sea devuelta información que las marcas hayan publicado ciñéndose a estas reglas.

3.2. RDF

RDF, cuyas siglas corresponden a ***R**esource **D**escription **F**ramework* (Marco de Descripción de Recursos), es el modelo de datos en el que la información está estructurada a modo de grafo dirigido y etiquetado que sirve para representar información en la red.

> "La especificación RDF 1.1 consiste en una serie de recomendaciones definida por la World Wide Web Consortium"[7]

Está basado en el concepto de triple, que tiene la siguiente estructura:

- **Sujeto**: Aquello que se va a describir.
- **Predicado**: Propiedad que relaciona el sujeto y el objeto.
- **Objeto**: Valor o recurso relacionado mediante el predicado con ese sujeto anterior

Veamos un ejemplo con el que trabajará la aplicación:

Listing 3.1: Ejemplo de RDF

```
<rdf:RDF xmlns:rdf="http://www.openarchives.org/OAI/2.0/rdf/"
xmlns:doc="http://www.lyncode.com/xoai"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ds="http://dspace.org/ds/elements/1.1/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:ow="http://www.ontoweb.org/ontology/1#"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/rdf.xsd">
<ow:Publication rdf:about="oai:addi.ehu.es:10810/2089">
<dc:title>Proyecto titulado "Penetración en Euzkadi. Apuntes Generales". Se trata de una
corrección a los anteriores Proyectos del 4 y 19 de marzo de 1938</dc:title>
<dc:creator>Beldarrain, Pablo</dc:creator>
<dc:subject>Guerrilla</dc:subject>
<dc:subject>País Vasco Historia 1936-1939 (Guerra civil)</dc:subject>
<dc:description>Fecha: 13-6-1938 / Unidad de instalación: Carpeta 11 - Expediente 1-3 /
N º de pág.: 11 (10 Mecanografiadas, 1 Manuscrita)</dc:description>
<dc:date>2011-03-22T17:45:34Z</dc:date>
<dc:date>2011-03-22T17:45:34Z</dc:date>
<dc:date>2011-03-22T17:45:34Z</dc:date>
<dc:type>info:eu-repo/semantics/other</dc:type>
<dc:identifier>http://hdl.handle.net/10810/2089</dc:identifier>
<dc:language>spa</dc:language>
<dc:relation>Fondo: BELDARRAIN, Pablo / Sección: 1 / Serie: Proyectos</dc:relation>
<dc:rights>info:eu-repo/semantics/restrictedAccess</dc:rights>
</ow:Publication>
</rdf:RDF>
```

En este caso el elemento a describir lo encontramos en la octava línea, en el atributo rdfabout, de la etiqueta <ow:Publication>; ”oai:addi.ehu.es:10810/2089”

Los predicados son las etiquetas que están anidadas en la que contiene el elemento, en este ejemplo, todas las que empiezan por ”dc”.

Y por último, el objeto es la información que encontramos dentro de las etiquetas de predicado.

La información que describen las líneas 1-7, dan significado a los prefijos. Entre los que se encuentran los que hemos mencionado hasta ahora, ”dc:” y ”ow:”, líneas 5 y 6 respectivamente.[1]

Que aplicado, los triples que obtendríamos serían los siguientes:

Sujeto	Predicado	Objeto
oai:addi.ehu.es:10810/2089	http://purl.org/dc/elements/1.1/title	Proyecto titulado...
oai:addi.ehu.es:10810/2089	http://purl.org/dc/elements/1.1/creator	Beldarrain, Pablo
oai:addi.ehu.es:10810/2089	http://purl.org/dc/elements/1.1/subject	Guerrilla
oai:addi.ehu.es:10810/2089	http://purl.org/dc/elements/1.1/subject	País Vasco...
.	.	.
...	...	...

Tabla 3.1: Triples

[1]”Entrando en las URIs que definen los prefijos encontraremos el significado de los predicados.”

3.3. Linked Data

Los Datos Enlazados (Linked Data en inglés) son un conjunto de buenas prácticas que se deberían seguir a la hora de publicar información en la web.

Las bases del Linked Data las sentó Tim Berners-Lee en un artículo [8] que fue publicado en 2006, en el que especificaba reglas que los datos deben cumplir para poder obtener la calificación de Linked Open Data, y poder subir de nivel dentro de ella. Eran las siguientes:

1. Usar URIs como nombres para los elementos
2. Las URIs deberán ser HTTP, para que la gente pueda encontrar descripciones de esos nombres
3. Cuando una persona busca una URI, proporcionar información útil, usando los estándares.
4. Incluir enlaces a otras URIs, para que podamos descubrir más contenido.

La primera regla es entendida como el uso de la tecnología de la Web Semántica. Si no se usan URIs, no podemos considerarlo Web Semántica.

En la tercera, los estándares mencionados son los establecidos por el consorcio W3C (RDF, RDFS, SPARQL, OWL...).

Además, Berners Lee ideó un sistema de calificación para los datos publicados (figura 3.1), en el que el contenido iba subiendo de categoría (estrellas), según iba cumpliendo requisitos,:

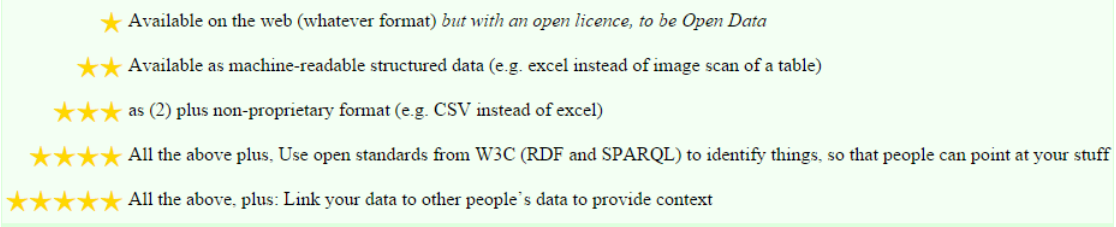


Figura 3.1: Calidad de la información según Berners-Lee

Traducido:

1. Hacer que los datos estén en la web con licencia abierta.
2. Dichos datos deben ser legibles por una máquina (Excel en vez de una imagen de una tabla, por ejemplo)
3. El punto anterior pero el formato debe ser abierto (CSV en vez de excel)
4. Las anteriores, más el uso de estándares del W3C (RDF y SPARQL) para identificar cosas, así el resto podrá hacer referencias a tus datos.
5. Hacer referencia a otros datos.

3.4. SPARQL

SPARQL, **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage (Protocolo SPARQL y Lenguaje de Consulta RDF), es un lenguaje de consultas diseñado para cumplir con los casos de uso y requerimientos identificados por el *RDF Data Access Working Group*.

El mejor modo de entender cómo funciona este lenguaje es mediante un ejemplo:

Listing 3.2: Ejemplo de consulta SPARQL simple

```
1 PREFIX  dc: <http://purl.org/dc/elements/1.1/>
2 PREFIX  ow: <http://www.ontoweb.org/ontology/1>
3 SELECT  ?x
4 WHERE   { <oai:addi.ehu.es:10810/2089> dc:creator ?x }
```

Como se puede observar, tiene una sintaxis similar a SQL, pero vamos a entender cómo funciona. Las dos primeras líneas son parecidas a las que hemos visto antes en el ejemplo de RDF (*Listing* 3.1). Significan que posteriormente, al escribir por ejemplo "*dc:creator*"en realidad será interpretado como "*http://purl.org/dc/elements/1.1/creator*".

Después encontramos la sentencia *SELECT*. En ella colocaremos las variables que queremos mostrar, de un modo similar a como lo hacemos en SQL. Sin embargo, aquí debemos poner un signo de interrogación antes de la variable, para que sea interpretada como tal. En este ejemplo *?x* sería la única variable.

Al final, encontramos la sentencia *WHERE*. Dentro de sus corchetes deberemos escribir nuestra consulta. El modo de hacerlo es mediante un *Triple pattern*. Esto es una lista de Sujeto-Predicado-Objeto, separada por espacios en blanco. En este ejemplo, vemos uno,

<oai:addi.ehu.es:10810/2089> → *dc:creator* → *?x*

Como ya hemos dicho, "*?x*"es una variable, que en este caso toma el papel de Objeto que hemos mencionado antes. Por lo tanto, ésta no añadirá restricción alguna a nuestra búsqueda. En el centro, encontramos "*dc:creator*", que como ha sido explicado antes, es interpretado como "*http://purl.org/dc/elements/1.1/creator*". Bien, pues este elemento cumple las veces de Predicado en nuestra lista.

Y al principio tenemos "*oai:addi.ehu.es:10810/2089"* en la posición de Sujeto.

Encontramos esta definición del funcionamiento de *SPARQL* en la página del *W3C*:

> La mayoría de las formas de consulta en SPARQL contienen un conjunto de patrones de tripleta (triple patterns) denominadas patrón de grafo básico. Los patrones de tripleta son similares a las tripletas RDF, excepto que cada sujeto, predicado y objeto puede ser una variable. Un patrón de grafo básico concuerda con un subgrafo de datos RDF cuando los términos RDF (RDF terms) de dicho subgrafo pueden ser sustituidos por las variables y el resultado es un grafo RDF equivalente al subgrafo en cuestión.[9]

La función es que el resultado de la consulta se restrinja los elementos no variables (que no empiecen por '?') y coloque en estos la parte de los triples que cumplan los requisitos que les corresponda.

El único triple que se ciñe a los requerimientos (Sujeto = `<oai:addi.ehu.es:10810/2089>` y Predicado = "http://purl.org/dc/elements/1.1/creator") es el segundo de la tabla 3.1 (el que relaciona los Sujeto y Predicado anteriores) Por lo tanto, ése es el triple que nos interesa. Una vez sabiendo las tuplas que nos son interesantes, "llenamos"la variable (?x en este caso) con los valores que le corresponden a su posición, en este caso la del Objeto (la tercera). Si ejecutásemos esta consulta contra nuestra base de datos, obtendríamos el *String* "Beldarrain, Pablo".

Hay muchas combinaciones más, como por ejemplo fijar un único elemento y poner dos variables, seleccionar más de una para que se muestre (igual que en SQL), utilizar *Union*, combinar consultas...

Listing 3.3: Consulta con dos variables

```
1 PREFIX  dc: <http://purl.org/dc/elements/1.1/>
2 PREFIX  ow: <http://www.ontoweb.org/ontology/1>
3 SELECT  ?x ?y
4 WHERE   { <oai:addi.ehu.es:10810/2089> ?y ?x }
```

En este ejemplo se mostrarían todos los triples que tuviesen oai:addi.ehu.es:10810/2089 como Sujeto. Y se nos mostrarán dos columnas. Una con el contenido de las variables "?y" (los predicados de los triples que tengan ese Sujeto) y "?x" (los Objetos).

Listing 3.4: Ejemplo de consulta SPARQL múltiple

```
1 PREFIX  dc: <http://purl.org/dc/elements/1.1/>
2 PREFIX  ow: <http://www.ontoweb.org/ontology/1>
3 SELECT  ?y
4 WHERE   { <oai:addi.ehu.es:10810/2089> dc:creator ?x
5           ?x dc:creator ?y}
```

Aquí vemos una consulta múltiple. En ella, primero se coloca en "*?x*" el contenido que le corresponde. Después, para cada identificador que contenga esa variable, se ejecutará la segunda, completando así "*?y*". Por lo tanto, ésta última, que será la que se muestra, contendrá todos los recursos que haya *creado* el autor del recurso <*oai:addi.ehu.es:10810/2089*>.

3.5. Stardog

Stardog es una base de datos de grafos semánticos implementada en Java. Este sistema soporta el modelo de grafo RDF, consultas SPARQL, protocolo HTTP para acceso remoto, y ontologías OWL, además de reglas definidas por el usuario.

En nuestro caso dos de sus funcionalidades son los que nos interesan. La primera, como servidor local, para hacer pruebas durante el desarrollo del proyecto. La otra, mediante la utilización de la librería **Stardog.js** para **Node.js**, haremos que sea nuestro servidor para la obtención de datos de forma remota.

3.6. Sails.js

Sails.js es un entorno de programación para *back-end web*, es decir, el "servidor". En otras palabras, sirve para ”crear” APIs, "lanzar"páginas HTML... Y está orientado al Modelo-Vista-Controlador. En él, encontraremos variedad de paquetes, entre los que se incluye Node.js, el necesario para trabajar con la librería de *Stardog* que hemos mencionado antes.

Este entorno será en el que nos apoyemos para desarrollar la parte cliente de la aplicación. Diseñaremos una web orientada a eventos, que se encargará de (crear y) trasladar la consulta a nuestro servidor *Stardog*.

3.7. DSpace

Como ya se mencionó en la introducción, el DSpace es un software pensado para organizaciones académicas y sin ánimo de lucro para construir repositorios digitales abiertos.

DSpace nos provee de un acceso fácil a todo tipo de contenidos digitales, como texto, imágenes, vídeos y datasets. Además, cuenta con una comunidad de desarrolladores en crecimiento, enfocados a expandir y mejorar el software.

En la figura 3.2, se muestra cómo funciona gráficamente la aplicación.

Unos ejemplos de aplicaciones que tiene el DSpace:

1. Repositorio institucional: El ADDI, sobre el que vamos a trabajar, sirve para tener acceso a material docente, Proyectos de Fin de Carrera/Trabajos de Fin de Grado y Máster, Fondos Digitalizados, Trabajos de Investigación, Revistas y Tesis Doctorales.

2. Repositorio de imágenes: La universidad de Rice (Houston) usa DSpace como un archivo digital. TIMEA contiene guías de viaje, catálogos de museos, fotos e imágenes dibujadas a mano de Egypto.

3. Repositorio de audio y vídeo: SMARTech de la biblioteca de Georgia, conecta material digital existentes en el campus para crear un repositorio útil, cohesionado y sostenible.

4. Museo: La Biblioteca Digital de Afghanistan, de la universidad de Nueva York contiene y restaura material publicado en Afghanistan en peligro de desaparecer. Su objetivo es el de recoger, catalogar, digitalizar y dar acceso a cuantas publicaciones sea posible para así reconstruir una parte esencial del partimonio cultural Afgano.

5. Reportes y Registros gubernamentales: Policy Archive

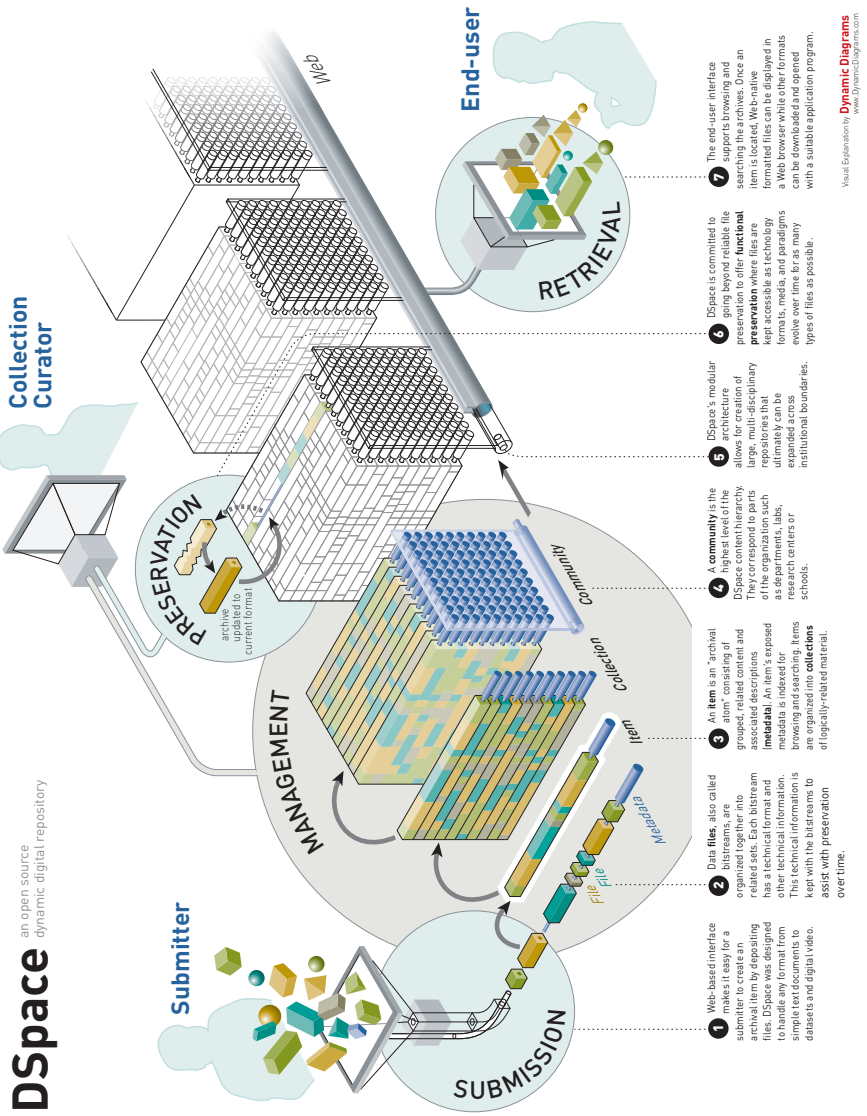


Figura 3.2: Diagrama de funcionamiento del DSpace

3.8. Protégé

Protégé[10] es un *Framework* desarrollado por la Universidad de Stanford con la colaboración de la Universidad de Manchester. Proporciona una interfaz gráfica para visualizar y editar ontologías. Es "la herramienta líder en lo que se refiere a ontologías en ingeniería"[11]

3.9. Antecedentes

La idea de este trabajo viene de un artículo [12] publicado por unos investigadores de la Universidad Técnica Nacional de Atenas . En él, mencionan la tendencia que bibliotecas como la Deutsche National Bibliothek, la Biblioteca Nacional De España o la British Library han tomado estos últimos tiempos, la de almacenar sus metadatos como Linked Data.

Esta propuesta menciona también las principales ventajas que que adquiriría un sistema DSpace al integrarse con otra aplicación externa que implementase Linked Data:

1. Facilitaría la integración de otros repositorios que tuviesen metadatos en formato RDF, sin necesidad de que fuera una instancia DSpace.
2. Aportaría herramientas para la gestión, proceso, visualización y análisis de la información.
3. Facilitaría el descubrimiento bibliográfico por las razones anteriores
4. Dar a conocer nuevo material sería más sencillo para los gestores de contenido, ya que, si se reutilizan las ontologías, el contenido recién introducido tendría la misma relación con un concepto que otro recurso que lleve más tiempo.
5. Dispondría de una base sólida desde la que empezar, ya que la ontología que aplica DSpace, la Dublin Core, es la primera que encontramos en la lista de las mejores ontologías del *World Wide Web Consortium*[13]

Aunque también apunta que esto requeriría un gran cambio, puesto que migrar un sistema que gestiona todos los contenidos de un repositorio es muy costoso. Por ello, la propuesta extiende otra posibilidad: una aproximación modular.

CAPÍTULO 4

Desarrollo del proyecto

En este capítulo encontraremos toda la documentación sobre el desarrollo del proyecto. Podremos ir conociendo todas las actividades que se han realizado, sin contar con las fases de formación y aprendizaje previas, ya que no son objeto de esta memoria. Empezaremos por la obtención de la información, seguiremos con su tratamiento, y terminaremos explicando cómo llevamos a cabo la aplicación que utilizará el usuario para consultar los datos de una forma más eficaz gracias a un buen uso de los metadatos.

4.1. Obtención de la información

4.1.1. DSpace

Como hemos mencionado DSpace ya nos ofrece una interfaz en la que podemos leer los metadatos que pone a nuestra disposición en varios formatos. Para acceder a ella debemos entrar en el dominio de la instancia DSpace (en el caso de nuestra universidad, `https://addi.ehu.es/`), y luego añadirle "*oai/request?verb=Identify*". Nos encontramos con la pantalla que podemos ver en la figura 4.1

En ella encontramos datos de interés general, de los que nosotros no obtendremos información.

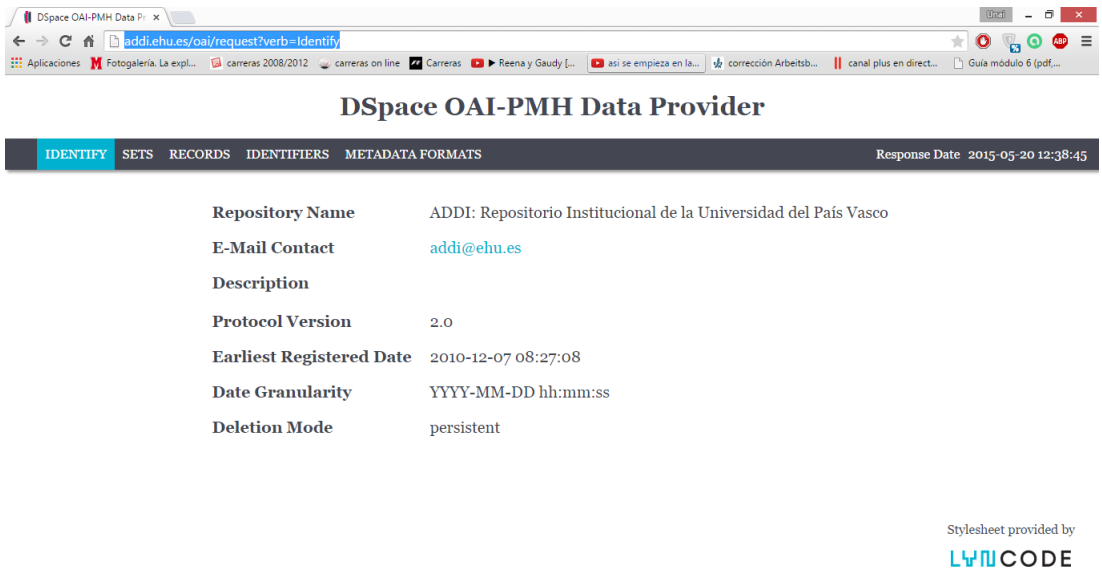


Figura 4.1: Pantalla inicial DSpace

Si hacemos click en **RECORDS**, encontraremos los metadatos que estamos buscando, aunque posiblemente no estén en el formato que nos interesa. Pero si seleccionamos la pestaña **METADATA FORMATS** veremos la cantidad de formatos en los que nos es ofrecida la información. Si miramos todas las opciones, veremos que una de las alternativas es **RDF**. Si la seleccionamos, somos redirigidos a la pestaña **RECORDS**, y esta vez sí, la información estará en RDF, y usando el vocabulario de Dublin Core. Podremos comprobarlo seleccionando **Metadata** en cualquiera de los recursos, como representa la figura 4.2.

Anteriormente ya hemos comprobado los metadatos que nos ofrece DSpace (en este trozo de código 3.1), aunque no es así como nos los proporciona . La manera de obtener esta información es guardando la página web que estamos viendo en cada momento. Esto nos genera un archivo con extensión *.xml* que posteriormente trataremos.

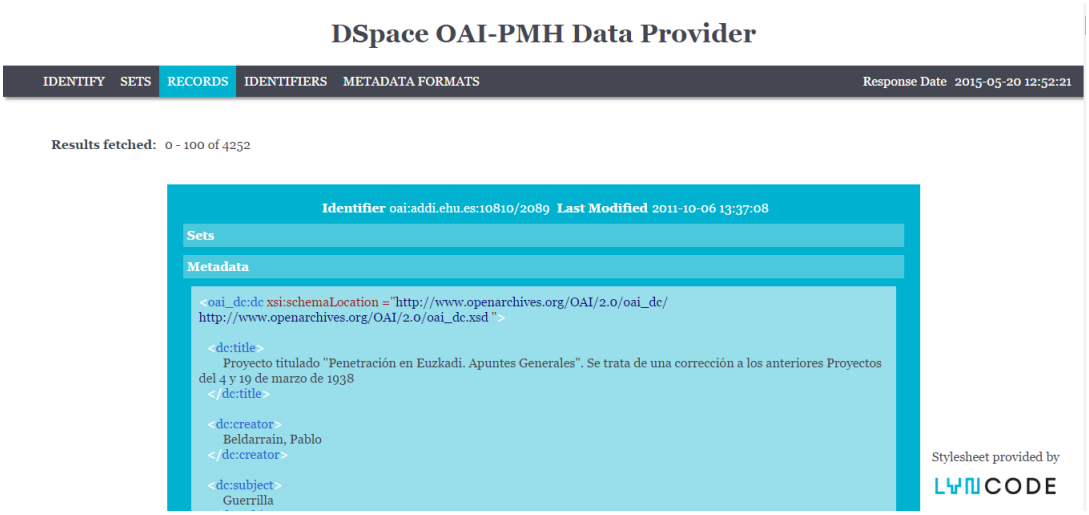


Figura 4.2: Metadatos en RDF

4.1.2. Primera aproximación

Cada página de la interfaz nos muestra metadatos de 100 recursos, más un enlace para acceder a la próxima página 4.3, en la que encontraríamos otras 100 entradas y otro *link* a la siguiente.

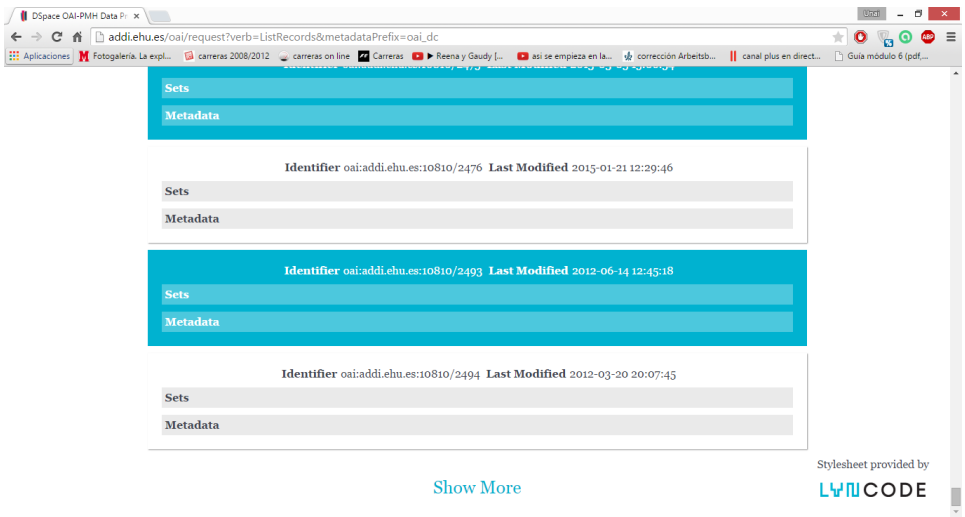


Figura 4.3: Próxima página

La primera aproximación para obtener la información fue el programa Auto KeyBoard. Esta aplicación permite programar las acciones que deben realizar el ratón y el teclado. En este primer intento, se programó el ratón para que, estando activa la pestaña en la que vemos la primera página de metadatos (4.2) seleccionase **Archivo** en el navegador, luego **Guardar página como...**, y luego **Guardar** otra vez, como vemos en la siguiente figura 4.4. Esto nos genera el archivo *.xml* que buscábamos, con el nombre *request.xml*.

El próximo paso era acceder a la siguiente página, que contendría la metainformación de los próximos 100-200 elementos del *ADDI*. Para ello, se añadieron varias instrucciones más a la secuencia que debían seguir el teclado y el ratón. Después de guardar la página, el teclado debía pulsar repetidamente la barra espaciadora, lo que provocaría que el *scroll* de la página fuera bajando. Una vez que estuviese en la parte inferior, veríamos el enlace a la próxima página. Solo debíamos pulsar en él, para acceder a más información. Una vez estando en la próxima, solo había que repetir el ciclo de acciones que han sido descritas hasta ahora.

Vemos el final del proceso descrito en esta figura 4.5.

La condición de parada se fijaba manualmente, viendo de cuántas páginas disponía el *ADDI*.

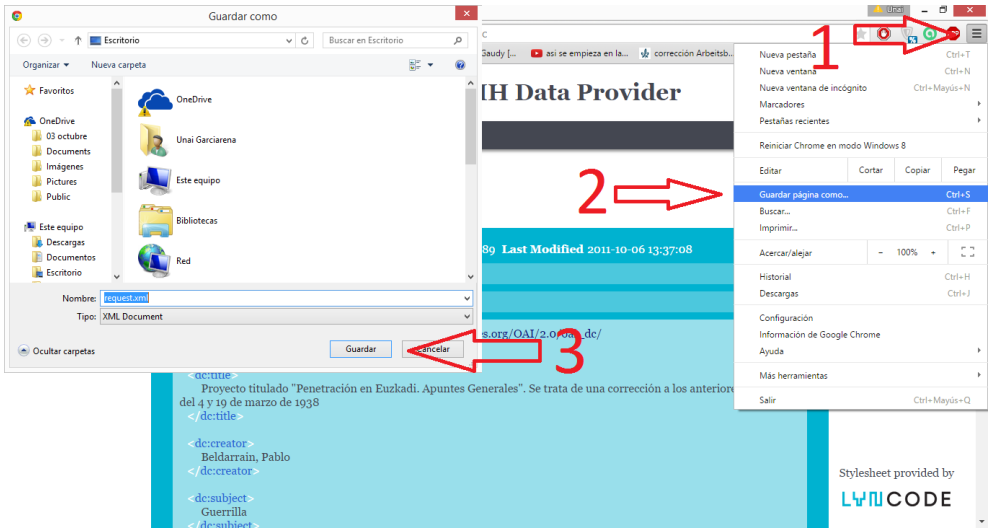


Figura 4.4: Guardar Página

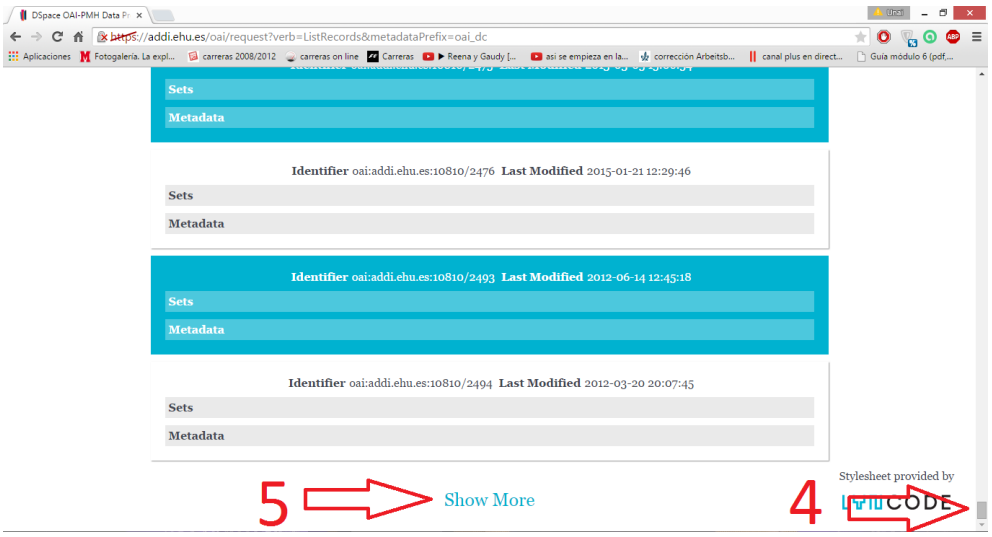


Figura 4.5: Acceder a la próxima página

De esta manera, se generaban tantos archivos como páginas tuviese el ADDI;

Número de páginas de metadatos = n = $\lceil \frac{X}{100} \rceil$

Siendo 'X' el número de recursos que tiene el DSpace. Los nombres de los archivos serán gestionados de distinta manera, dependiendo del navegador que se utilice. *Google Chrome*, por ejemplo, no cambia el nombre del primer archivo, que lo deja en *request.xml*, y a los siguientes les asigna un número, añadiéndoselo al final del nombre, antes de la extensión entre paréntesis. Por lo tanto, de completar todo el proceso sin errores, encontraríamos una carpeta con varios *.xml*s, cuyos nombres tendrían la siguiente progresión:

request.xml, *request (1).xml*, *request (2).xml... request (n-1).xml*, significando n lo mismo que en la ecuación anterior.

Con todo esto, ya teníamos todo lo necesario para empezar a trabajar con la información. Pero no es una solución digna de un TFG de Grado en Ingeniería Informática.

4.1.3. Segundo acercamiento

Sin embargo, este trabajo no fue tiempo perdido. En una modificación posterior del programa descrito en la sección anterior, conseguimos que la URI en la que obteníamos la información apareciese en el mismo archivo. Tiempo después, en la etapa en la que ya la información estaba siendo tratada, se observó un patrón en las URI. Éstas eran iguales, salvo una letra que va variando, en orden alfabético. Aquí un ejemplo del valor de las variables GET de las URIs (la dirección que utiliza estos valores para mostrar unos recursos u otros es "*http://addi.ehu.es/oai/request?*"):

2: *verb=ListRecords&resumptionToken=MToxMDB8Mjp8Mzp8NDp8NTpvYWlfZGM=*
3: *verb=ListRecords&resumptionToken=MToyMDB8Mjp8Mzp8NDp8NTpvYWlfZGM=*
4: *verb=ListRecords&resumptionToken=MTozMDB8Mjp8Mzp8NDp8NTpvYWlfZGM=*
5: *verb=ListRecords&resumptionToken=MTo0MDB8Mjp8Mzp8NDp8NTpvYWlfZGM=*
...
9: *verb=ListRecords&resumptionToken=>MTo5MDB8Mjp8Mzp8NDp8NTpyZGY=*

Como vemos, las direcciones son calcadas, salvo un par de carácteres en el valor de la segunda variable GET (*resumptionToken*). El tercer y cuarto valor vemos que va variando entre ”ox”, ”oy”, ”oz” y ”o0”. A partir de ahí, la progresión no es alfabética, pero sí sigue siendo estable (la dirección que hace uso sigue siendo "*http://addi.ehu.es/oai/request?*");

12: *verb=ListRecords&resumptionToken=>MToxMjAwfDI6fDM6fDQ6fDU6cmRm*
13: *verb=ListRecords&resumptionToken=>MToxMzAwfDI6fDM6fDQ6fDU6cmRm*
14: *verb=ListRecords&resumptionToken=>MToxNDAwfDI6fDM6fDQ6fDU6cmRm*
15: *verb=ListRecords&resumptionToken=>MToxNTAwfDI6fDM6fDQ6fDU6cmRm*
16: *verb=ListRecords&resumptionToken=>MToxNjAwfDI6fDM6fDQ6fDU6cmRm*
17: *verb=ListRecords&resumptionToken=>MToxNzAwfDI6fDM6fDQ6fDU6cmRm*
18: *verb=ListRecords&resumptionToken=>MToxODAwfDI6fDM6fDQ6fDU6cmRm*

Aquí observamos que son el quinto y sexto los que van variando. El sexto hace ciclos entre [’D’, ’T’, ’j’, ’z’], mientras que el quinto sigue una progresión alfabética una vez más, [’M’, ’N’, ’O’, ...].

Una vez habiendo descubierto que las direcciones siguen un patrón relativamente descifrable, se procede a idear un nuevo modo para automatizar la obtención de información.

Programa Java

El método elegido es programar una aplicación Java que accediese a cada URI calculable, obtener el código *.xml* y guardarlo en la máquina. Finalmente, el resultado sería similar al logrado con el método anterior, aunque el proceso sería válido para cualquier máquina con Java instalado, además de consistir en un proceso mucho más elegante y fácilmente mejorable.

4.1.4. Método definitivo

Finalmente, tras un estudio más a fondo de los recursos que obtenemos del DSpace, se encuentra nueva información que nos será de mucha utilidad. Al final de cada archivo *.xml* que guardamos (trozo de código 4.1), observamos que al final encontramos una etiqueta especial;

Listing 4.1: Final de un archivo request.xml

```
    <dc:type>info:eu-repo/semantics/bachelorThesis</dc:type>
    <dc:identifier>GKINTZOU, Christina. Geometric documentation and reconstruction of the church
     of the Monastery of San Prudencio (La Rioja, Spain). Proyecto fin de carrera de Ingeniería T
     écnica en Topografía. Escuela de Ingeniería de Vitoria-Gasteiz (Universidad del País Vasco-
     Euskal Herriko Unibertsitatea UPV/EHU). 2011.</dc:identifier>
    <dc:identifier>http://hdl.handle.net/10810/7096</dc:identifier>
    <dc:language>eng</dc:language>
    <dc:relation>LDGP_pfc_006</dc:relation>
    <dc:relation>http://dspace.lib.ntua.gr/handle/123456789/6168</dc:relation>
    <dc:relation>http://hdl.handle.net/10810/9906</dc:relation>
    <dc:relation>http://hdl.handle.net/10810/9168</dc:relation>
    <dc:rights>info:eu-repo/semantics/openAccess</dc:rights>
    <dc:publisher>Laboratorio de Documentación Geométrica del Patrimonio (LDGP)</dc:publisher>
</ow:Publication>
</rdf:RDF>
</metadata>
</record>
    <resumptionToken completeListSize="4197" cursor="8">MTo5MDB8Mjp8Mzp8NDp8NTpyZGY=</
     resumptionToken>
</ListRecords>
</OAI-PMH>
```

En este ejemplo de código, vemos el final de los metadatos de el último recurso de una página, más una etiqueta especial, *resumptionToken*. Tiene como atributos *completeListSize*, que indica el total de recursos que contiene la instancia del DSpace en el momento en el que se descarga la página, y *cursor*, que nos muestra la página en la que nos encontramos. En este caso, el ADDI contaba con 4197 en aquel momento, y el ejemplo está extraído de la octava página. Aunque lo que nos interesa de la etiqueta es el *String* que contiene. Este es el valor de la variable GET *resumptionToken*, de la que hemos hablado antes, que da acceso a la próxima página, lo que hace aún más fácil el acceso a el resto de datos empezando desde la página principal. Por lo tanto podemos decir ya que estamos en el punto óptimo de información para comenzar con la parte de programación definitivamente.

4.1.5. Bilatu

Para la extracción de la información de Bilatu, se ha tenido que desarrollar otro programa similar al anterior. La base es igual, de hecho varias funciones se utilizan en ambos sistemas. Aun así, este método es más complicado. El funcionamiento de esta parte comienza mientras se están uniendo los archivos *requestXXX.xml*. Cada vez que el programa detecta una etiqueta *dc:creator* o *dc:contributor*, hace una petición al buscador de Bilatu con el nombre que encuentra en la etiqueta como parámetro. Cuando el buscador procesa la petición y devuelve una respuesta, el sistema comprueba que en la página hay algún resultado. Si es así, vuelve a hacer otra petición con el enlace que contiene el primer resultado de la búsqueda. Al recibir esta segunda respuesta, es procesada de un modo similar al sistema descrito anteriormente, buscando patrones de atributos en las etiquetas, para extraer los valores necesarios.

4.1.6. Páginas de las facultades

Para los profesores del *Campus de Gipuzkoa* se añadió la funcionalidad que aportaba información sobre si es doctor, si domina el euskera y el contrato que tiene con la universidad. Para ello, se han tenido que encontrar las páginas en las que obtenemos esta información, en la guía docente de cada titulación. Ejemplo: Informática, Química

Estos ejemplos son los únicos encontrados que tienen una URI como identificador de esta página, el resto tienen una complicada y larga lista de parámetros tanto *GET* como *POST*.

Una vez obtenidas las páginas, se aplicaba el proceso habitual de extracción de información de un *html*.

4.1.7. DBLP

Finalmente, la última base de datos consultada. Ésta ha resultado la más sencilla, ya que el parámetro va insertado en la URI, y además el resultado viene ya en formato *RDF*, por lo que no es necesario ningún procesamiento de información.

Ejemplo de URI: http://dblp.uni-trier.de/pers/xr/b/Berm=uacute=dez:Jes=uacute=s.rdf

Los elementos a cambiar son el carácter 'b', que es la primera letra del apellido del autor, y el nombre, que se escribe en formato Apellido:Nombre, tratándose los caracteres con tilde como =Xacute= (X es el carácter que debería llevarla.)

4.2. Arquitectura

Una vez decidido el camino que vamos a seguir, deberemos diseñar la arquitectura que seguirá la aplicación, para separar bien las funcionalidades, y que así sea más fácil modularizar. Con la consiguiente simplificación del proceso de corregir errores. El modelo escogido es el de Programación por Capas, que es suficiente para esta tarea.

4.2.1. Capa de presentación

Como esta parte de la aplicación será solo para obtener la información y no para mostrarla, la capa de presentación no será necesaria, por lo que su función será ligeramente modificada.

En este caso, esta capa será la que interactúa con el ADDI. Se encargará de conseguir la información y de guardarla para que las otras capas trabajen con ella. Además, contendrá la función maestra, la que llamará al resto. En el programa está representada por el fichero *DSpace.java*.

Lo que hará el programa será ir guardando una a una las páginas que nos ofrezca la instancia, con los siguientes nombres: *request*λ*.xml*, siendo λ un número entre 1 y la cantidad de páginas que nos ofrezcan. Podemos ver un ejemplo de ejecución en la figura 4.6. Además, encontramos en el *Listing* A.1 las cabeceras de cada función con una pequeña descripción de lo que hacen.

4.2.2. Capa de negocio

Esta capa se encargará de leer la información de los contenidos del DSpace y tratarlos, ya que no todos los metadatos que nos ofrece la interfaz del DSpace nos son útiles. Además se encargará de gestionar las llamadas a las funciones de la capa de acceso a otras fuentes, y de gestionar sus resultados. Vemos las cabeceras de las funciones junto con una pequeña descripción de la tarea que llevan a cabo en los siguentes Listings, en los anexos; A.2, A.3 y A.4

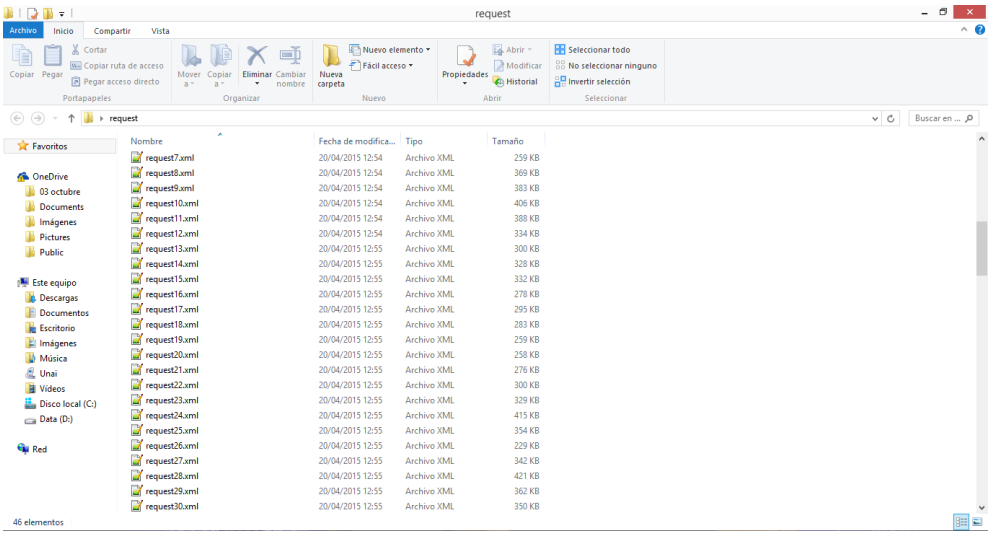


Figura 4.6: Ejemplo de archivos después de la ejecución de DSpace.java

4.2.3. Capa de acceso a otras fuentes

El fichero *form.java* se encargará de completar y enviar peticiones a las distintas fuentes de información de las que vayamos a registrar datos. Después de enviarlas, será responsable de recibir las respuestas, y de pasárselas a la Capa de negocio, para que ésta las procese. Vemos las cabeceras y descripciones del funcionamiento de los métodos de esta clase en los siguientes *Listings* A.5, A.6, A.7, A.8, A.9 y A.10

Para resumir en una imagen el funcionamiento global de la aplicación, podemos basarnos en este diagrama: 4.7

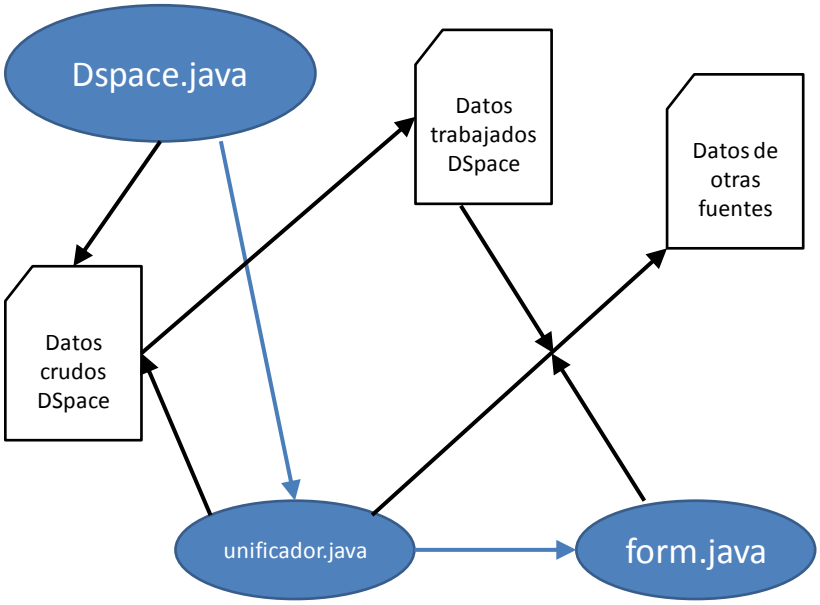


Figura 4.7: Arquitectura de la aplicación

4.3. Desarrollo de ontologías

4.3.1. Ontología interna

Como ya hemos mencionado previamente, los metadatos del DSpace están anotados con el vocabulario de la ontología *Dublin Core (DC)* [14]. Ésta es a menudo calificada como la mejor ontología por varias fuentes, como por ejemplo W3C [13]. Cita. Por esta razón, se ha elegido adoptarla, en vez de buscar otra. Sin embargo, se han introducido nuevos datos entre la información que nos provee el ADDI, como el tamaño y el formato, por lo que es necesario buscar términos que se adecúen a estos datos. La otra alternativa es modificar la ontología, pero se ha decidido no hacerlo, ya que se han encontrado términos similares, como <dc:format>para el formato y <dc:extent>para el tamaño del archivo. El término *format* es definido por DC de la siguiente manera: *El formato físico del archivo o dimensiones del recurso.* Por lo tanto, es completamente válido para el uso que le estemos dando.

El otro término introducido es <dc:extent>, que es definido como *El tamaño o duración del recurso*. El uso que le damos es el del tamaño, por lo tanto es también adecuado.

En resumen, los términos más utilizados en nuestra clasificación, son los siguientes:

- http://purl.org/dc/elements/1.1/title
- http://purl.org/dc/elements/1.1/contributor
- http://purl.org/dc/elements/1.1/creator
- http://purl.org/dc/elements/1.1/subject
- http://purl.org/dc/elements/1.1/description
- http://purl.org/dc/elements/1.1/date
- http://purl.org/dc/elements/1.1/type
- http://purl.org/dc/elements/1.1/identifier
- http://purl.org/dc/elements/1.1/extent
- http://purl.org/dc/elements/1.1/format
- http://purl.org/dc/elements/1.1/language
- http://purl.org/dc/elements/1.1/relation
- http://purl.org/dc/elements/1.1/rights
- http://purl.org/dc/elements/1.1/publisher

Todos ellos dentro de la clase

- http://www.ontoweb.org/ontology/1#Publication

4.3.2. Ontología externa

Además de haber extendido los términos que utilizamos de la ontología DC, se ha introducido una cantidad de datos considerable al sistema, extraídos de Bilatu, las páginas de los centros y la DBLP. Para gestionar todos esos nuevos recursos, se ha llevado a cabo un estudio de ontologías ya existentes con términos que pudiesen corresponderse con los nuevos datos que íbamos a integrar de estas nuevas fuentes. Después de haber rastreado la web en busca de ellas, se llegó a reducir el número de opciones a dos; la ontología VIVO [15] y la HERO [16]. Las dos necesitaban nuevos términos para que nos fueran totalmente válidas, por lo que eso no era un factor determinante a la hora de escoger una u otra.

El criterio principal es que cuantos menos términos tengamos que añadir mejor, para evitar pérdida de información a terceras personas que quieran utilizar nuestros datos. La nueva información que queremos añadir es la recabada de las páginas de las facultades, el Bilatu, y el link de la DBLP. 10 nuevos términos en total;

- Nombre
- Centro
- Departamento
- Campus
- Teléfono
- Email
- Dominio del Euskera
- Doctorado
- Enlace DBLP
- Clase de empleado

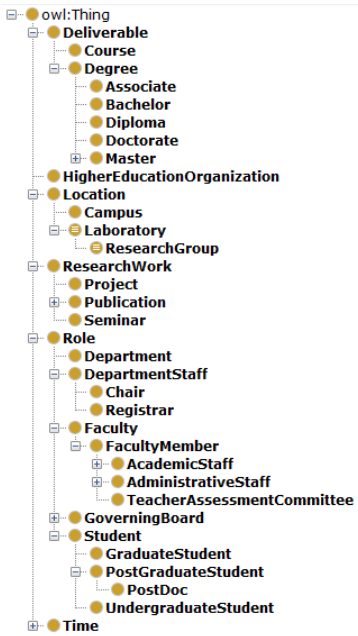


Figura 4.8: Términos más útiles para nosotros en la ontología HERO

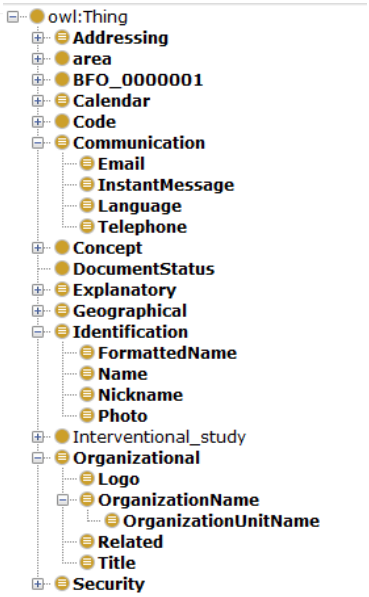


Figura 4.9: Términos interesantes para nuestra información en la ontología VIVO

Se decide que la clase de empleado sea el término principal, es decir, la clase de los Sujetos de los triples dependerá del tipo de relación que les una con la UPV/EHU. Explicado gráficamente en la figura 4.10

Y con esto ya son 9 los nuevos elementos a añadir. De los cuales, la ontología VIVO tiene integrados 5 (Nombre, Teléfono, Campus, Centro, Email), siendo flexible (Tomar *Institution* como Campus, por ejemplo). Mientras que HERO tiene 6 (Los mismos que la VIVO, más el Departamento), y más exactos. Además, las siglas HERO corresponden a *Higher Education Reference Ontology*.

Habiendo decidido que la HERO es más apropiada para este trabajo, pasamos a la fase de adecuación. Ésta consiste en añadir algunas propiedades y clases, para poder gestionar todos los tipos nuevos de datos. Todos los términos añadidos son subclase de una que ya existe, por lo que terceros que quieran realizar consultas sobre nuestra información podrán hacerlo sobre ese término que ya existe. Los términos son los distintos papeles que desempeñan los profesores en nuestra universidad. Éstos serán sin duda distintos a los de otras universidades, por lo que la información que nos es útil aquí, posiblemente no lo sea en otras instituciones. Por lo tanto, hemos hecho que así como nosotros podemos preguntar por ”catedráticos”, ”investigadores”, ”agregados”... trabajadores ajenos a este sistema podrán preguntar por "*profesores"* en general.

Ya que la ontología está en inglés sería adecuado que los nuevos términos también lo estuvieran. Esto conlleva muchas dificultades, ya que se pretendía clasificar a los profesores por el tipo de vínculo que les une a la universidad, y poco (o nada) tienen que ver las denominaciones que recibe cada profesor debido a su contrato en nuestra universidad, con los que pueda haber en Estados Unidos o Inglaterra, por ejemplo. La primera decisión tomada fue la de dar prioridad a los papeles de profesores equivalentes de EEUU sobre los del resto de países de habla inglesa. El motivo principal es que es el inglés más estandarizado.

A continuación encontramos un diagrama que describe la estructura definida a partir de la ontología HERO modificada 4.10.

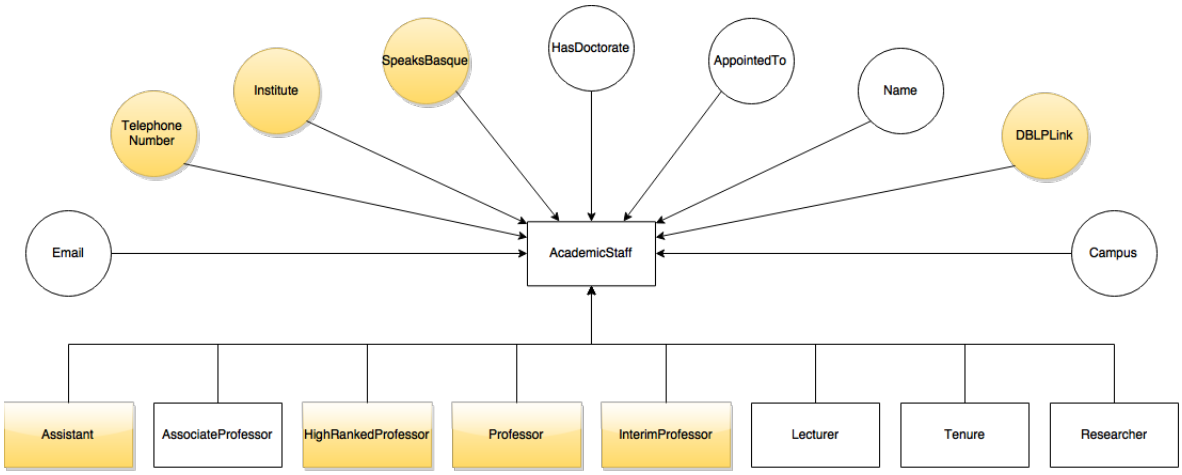


Figura 4.10: Diagrama de la ontología

Nosotros nos hemos centrado en una sola parte de la ontología HERO. Como se ve en la figura 4.8, hay una clase *Academic Staff* que es donde hemos introducido los nuevos términos, las clases que están en amarillo en el diagrama anterior, representadas por rectángulos (4.10).

De la misma manera, se han añadido varias propiedades (representadas por círculos), que podemos igualmente distinguir por el color. Para ver el significado de cada una de las clases y propiedades, además de la forma en la que se han implementado, tenemos los siguientes *Listings* B.1 y B.2. También podemos ver la ontología aplicada en metadatos (B.4).

Se ha utilizado también la ontología de la DBLP [17] para clasificar los datos que nos proporcionan. De momento, solo se han obtenido datos de autores, y no de los recursos, por lo que la utilización de ésta ha sido muy limitada. Para anotar esta información solo nos han sido necesarios los términos que describen la URI de la persona, su nombre y la última modificación que ha habido, y sus trabajos (tanto los que ha creado como en los que ha colaborado). Sin embargo, aunque no muy extensa, ésta es una ontología con bastante variedad de elementos, como por ejemplo la noción de *subpropiedad*. De forma análoga a como ocurre con las clases, si P1 es subpropiedad de P2, P2 tendrá, al menos el mismo rango y dominio que P1. Ejemplo de aplicación en B.3.

4.4. Parte cliente

Una vez que tenemos todos los datos que se buscaban integrados, toca darles uso. Para ello, primeramente se pensaba hacer una pequeña aplicación con una interfaz gráfica en Java. Con este objetivo se estudió el *framework Java* Apache Jena [18], que son unas librerías con objetos y métodos orientados a tratar información en RDF. Se llevó a cabo un pequeño ejecutable en el que pasándole una consulta SPARQL, la ejecutaba sobre nuestros metadatos y nos mostraba el resultado. Pero pronto se cambió de opinión, ya que posiblemente fuese más cómodo para un usuario final entrar en una página web que descargarse un programa de escritorio. Además, actualmente las posibilidades de la Web son enormes.

Por ello, decidimos aprovecharnos de la condición de servidor que tiene Stardog. La idea era que en la interfaz web, un usuario pudiese introducir una consulta completa, o un String junto con unas preferencias de búsqueda, y que se le fueran devueltos los resultados. El método que tiene Stardog para este tipo de consultas es recibirlas en la misma URI. Aun así, parece que esta funcionalidad no estaba muy desarrollada cuando se estaba realizando este trabajo, pues daba igual lo que se le preguntara, devolvía siempre todos los triples que tenía en la base de datos.

4.4.1. Sails.js

Indagando por internet, se descubrió la librería *Stardog.js* [19] [20]. Estaba construida para Node.js, lo que acarrearía más trabajo de estudio de tecnología, pero siendo la única manera que había en ese momento, no hubo más remedio.

Por recomendación de otra estudiante que ya había trabajado con *Node.js*, se decidió utilizar *Sails.js*, que, como ya se ha explicado en un capítulo anterior, es otro entorno que lleva *Node* incorporado.

En este caso, la aplicación no ha sido muy complicada de hacer, ya que simplemente necesitamos una página que sea capaz de enviar una consulta SPARQL (y, si es necesario, crearla) y recibir y mostrar los datos.

Ya que esta parte no es más que un elemento extra, que no tiene nada que ver con la Integración de Datos, no ahondaremos mucho en su desarrollo.

4.4.2. Consultas cocinadas

Este sistema de búsqueda está pensado para usuarios sin conocimientos SPARQL. En él, se dará la opción de el usuario introduzca un *String* y una opción. El primero será el elemento a buscar, y el segundo, dónde buscar. Estas opciones determinarán dónde se busca. Puede ser un departamento (<hero:AppointedTo>), un nombre de profesor (<hero:Name>), una temática de un recurso del *ADDI* (<dc:subject>)...

4.4.3. Consultas crudas

Pero también se añadirá un campo para que un usuario con conocimientos *SPARQL* pueda introducir consultas más complejas, como consultas con múltiples restricciones, con *UNION*...

Esta parte queda abierta al desarrollo, ya que en el alcance del proyecto no se encontraba una prueba con usuarios reales, con el fin de mejorar la interfaz.

En la figura (4.11) vemos como podemos introducir un *String* y un criterio de búsqueda, y la consulta se genera automáticamente.

Si después hacemos click en una URI que se nos muestre en los resultados, la consulta se vuelve a cambiar (4.13),

Y así podemos enlazar datos.

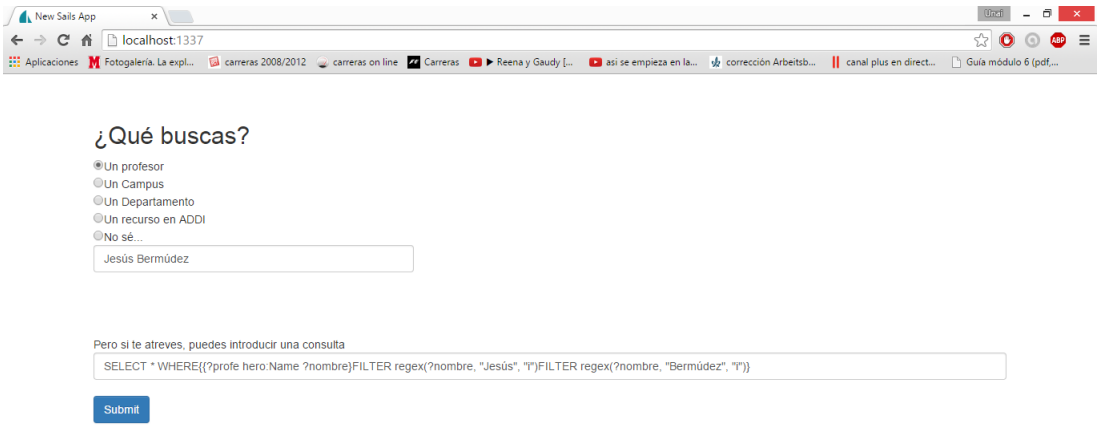


Figura 4.11: Ejemplo de la interfaz

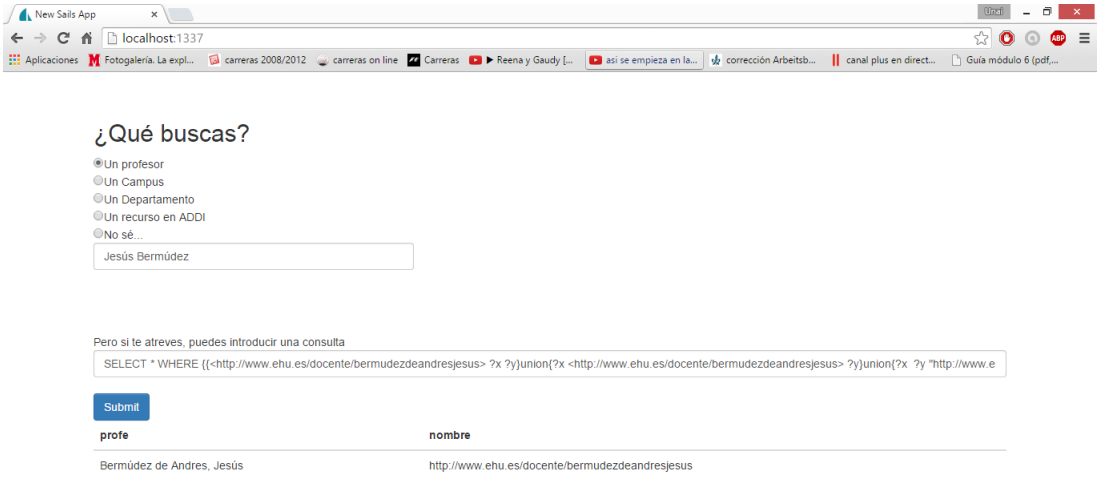


Figura 4.12: Uso de la aplicación

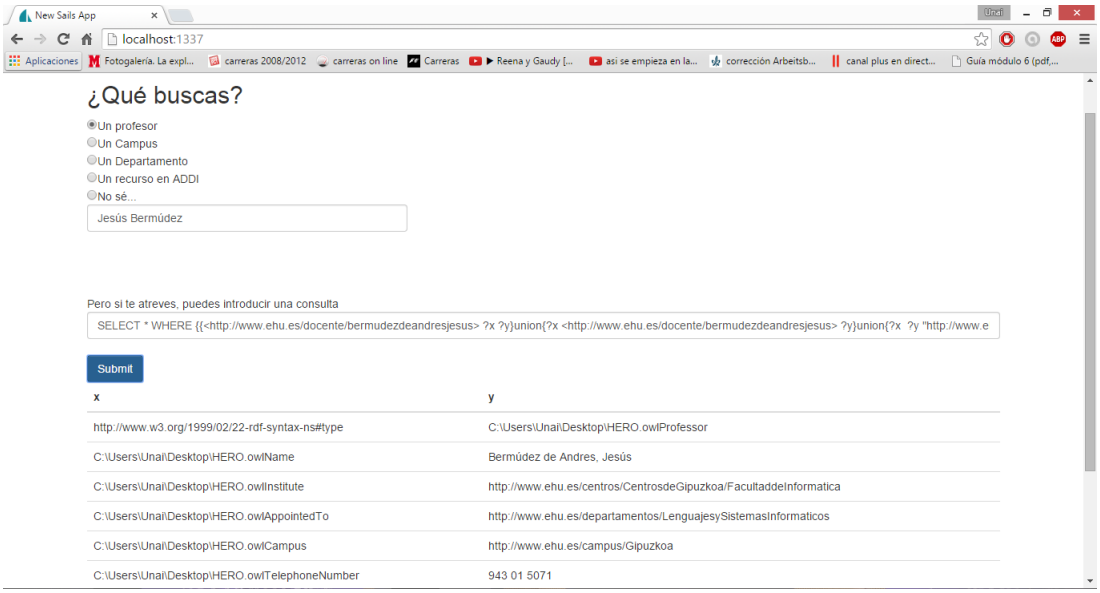


Figura 4.13: Linked Data

4.5. Gestión

4.5.1. Seguimiento y control

Finalmente, mostraremos las dedicaciones al proyecto en general y a las tareas la EDT 2.1 desgranadas, mediante otro diagrama de Gantt 4.14:

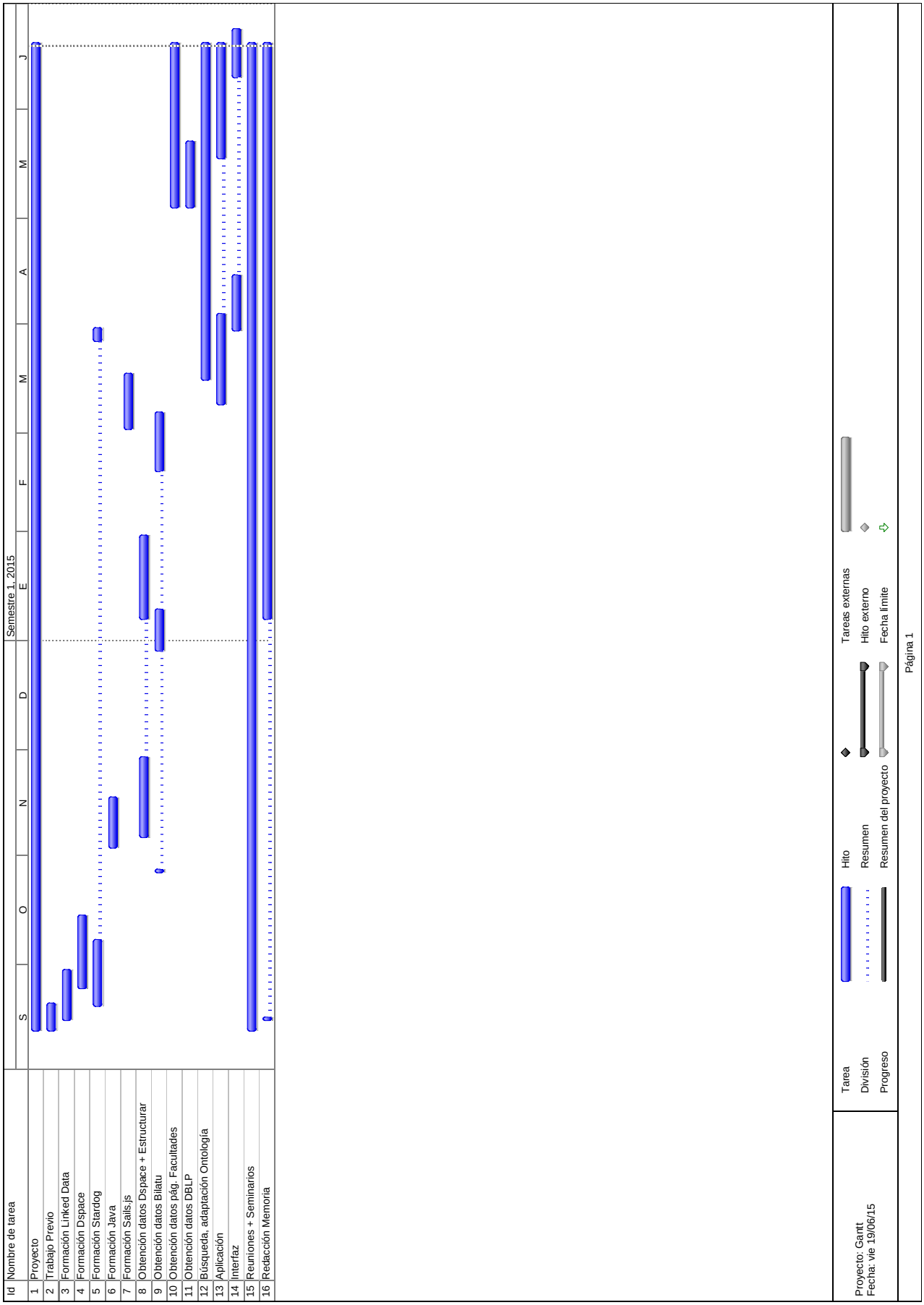


Figura 4.14: Diagrama de Gantt

Y reflejamos las horas aproximadas invertidas en cada una de las tareas en la siguiente tabla 4.1

Tarea	Dedicación (horas)
Trabajo previo	5
Formación Linked Data	10
Formación DSpace	5
Formación Stardog	20
Formación Java	5
Formación Sails.js	10
Obtención de datos DSpace	40
Obtención Datos Bilatu	30
Obtención Datos pág. Facultades	40
Obtención Datos DBLP	5
Búsqueda + Adaptación Ontología	10
Aplicación	20
Interfaz	10
Reuniones + Seminarios	10
Redacción Memoria	100
Total Proyecto	320

Tabla 4.1: Horas aproximadas por tareas

CAPÍTULO 5

Conclusiones y vista al futuro

Una vez terminado el proyecto, echamos la vista atrás para poder obtener conclusiones, y adelante para ver hacia dónde puede crecer la aplicación.

5.1. Conclusiones

5.1.1. Utilidad de la tecnología de la Web Semántica

Aunque también sería posible hacerlo mediante otras técnicas, esta extensión de la web tradicional es una herramienta excepcional para llevar a cabo tareas de integración de datos. Además, como se puede ver en los gráficos incrustados en la introducción, está en pleno auge. Aunque queda camino por recorrer. Buena prueba de ello es que esta aplicación no se haya desarrollado hasta el momento, o al menos no está integrada en esta universidad.

5.1.2. Genericidad de la metodología

Sin duda, este programa se puede aplicar fácilmente a otra institución que disponga de los datos de los que dispone nuestra Universidad. Por partes, la interfaz de los metadatos de todas las instancias de DSpace es la misma.

Si otra institución contase con otra página web equivalente a nuestro *Bilatu*, podríamos *acoplarla* sin problemas a nuestra aplicación con algunos cambios simples en ella, que serían la URI que identifica la página, los parámetros que hay que enviarle, y los patrones para rastrear dentro de las respuestas *.html*.

5.1.3. Utilidad de la aplicación

Estamos delante de una aplicación potencialmente aplicable al mundo real. el ejemplo que planteábamos en la introducción de esta memoria es perfectamente realista. Podríamos, por ejemplo, dar una solución sencilla a la pregunta que planteábamos al inicio de esta memoria, en la introducción (1).

Al haber integrado datos de la DBLP, *Bilatu*, las páginas de las facultades y el ADDI, el alumno podría escribir el *String* ”Psicología de la educación” en el cuadro de búsqueda, y seleccionar la opción que coloca el ADDI como rango en el que buscar (o la opción que busca en toda la base de datos). De esta manera, le saldrían varios triples con el identificador de un recurso como sujeto. Después podría hacer *click* en él, y buscar otra vez. De esta manera encontraría todos los datos sobre el recurso, entre los que estaría el creador y el contribuyente (el que posiblemente quisiera consultar). Ya que hemos integrado datos tanto de las páginas de las facultades, Bilatu y ADDI.

5.2. Lineas de desarrollo

La *Genericidad* comentada antes hace que este programa pueda ser desarrollado casi en direcciones infinitas, pero vamos a enunciar las más evidentes, o cercanas a este ámbito.

5.2.1. Otros repositorios externos

Mediante la integración de los datos de la DBLP se ha demostrado la posibilidad de utilizar este tipo de recursos. En este caso se ha elegido este repositorio por la cercanía al mundo de la informática, pero se podría fácilmente aplicar a otros campos. Por ejemplo, encontramos Tendencias Pedagógicas, una revista electrónica del mundo de la educación, que si bien no sería tan sencillo como fue la incorporación de datos de la DBLP (por aquello de que ésta ofrece los datos anotados ya en RDF), sería fácil utilizar los programas desarrollados para extraer datos de Bilatu para hacerlo de este tipo de repositorios.

La incorporación de datos de otras páginas relacionadas con la Web Semántica también es considerada, aunque no aporten metadatos. Es decir, un nivel más de abstracción. Por ejemplo, podríamos integrar otra base de datos, como SameAs que nos aportaría riqueza terminológica.

5.2.2. Otros repositorios internos

De la misma manera, se podrían integrar también datos de la UPV/EHU. Por ejemplo, se podrían integrar horarios de asignaturas, si se publicasen de otro modo que no fuese un *.pdf* o de forma distribuida como están ahora.

5.2.3. Otros repositorios DSpace

El método de extracción de metadatos de DSpace es totalmente válido para todas las instancias DSpace, por lo que se podrían extraer los metadatos de otras instituciones que lo utilicen (repositorios de UPV (Universidad politécnica de Valencia) y MIT (Massachusetts Institute of technology) han sido consultados y el método sería similar) y establecer nuevos enlaces.

5.2.4. Interfaz

Aunque *usable*, la interfaz creada no es amigable para un usuario sin conocimientos de informática. Por ello, la creación de una que si lo fuera queda como posible desarrollo futuro.

5.2.5. Aplicación

Relacionado con el punto anterior, se podría desarrollar una versión móvil de la aplicación. Si bien es cierto que la interfaz se adapta bien a una pantalla de un dispositivo de este tipo, una aplicación siempre es más atractiva.

Anexos

ANEXO A

Funciones

Listing A.1: Cabeceras y descripciones de las funciones de *DSpace.java*

```
1
2  public class Dspace {
3
4  //Función que calcula el tiempo de ejecución y llama al resto de funciones
5
6  public static void main(String[] args) throws Exception {
7
8
9  //Función que descarga todos los metadatos del DSpace y los guarda en tantos
10 //archivos como páginas tenga.
11
12 /*Tiene como parámetro de entrada una url, que será la primera página del DSpace.
13 *Así podremos cambiar de instancia
14 *cambiando simplemente una URI en el programa "main" anterior, sin tener que
15 *meternos al código.
16 *Además, podremos, mediante el parámetro i, determinar a partir de qué página
17 *empezará la ejecución del
18 *programa, útil si queremos comprobar algún evento específico, para no tener que
19 *empezar desde el principio.
20 *La ejecución del programa oscila entre 30 y 60 minutos, dependiendo de la
21 *conexión, la máquina, la cantidad de tareas
22 *que esté llevando a cabo...
23 */
24 public static String importProjects (String url, int i) throws IOException{
25
26 }
```

Listing A.2: Cabeceras y descripciones de las funciones de *unificador.java* I

```
1
2  //Método para hacer pruebas, símplemente llama a unificar.
3
4  public static void main(String[] args) throws Exception
5
6  /*Input: Páginas request.xml
7  *Output: Un úncio fichero con todos los datos que nos interesen de
8  *todos los archivos request.xml, con las URIs para profesores,
9  *centros... generadas, más otro fichero con la información extra
10 *de los profesores (Bilatu, páginas de las facultades, DBLP)
11 *
12 * Función principal de la aplicación. la funcionalidad principal
13 *la de unificar y limpiar de paja todos los archivos que ha generado
14 *DSpace.java en uno solo. Sin embargo, durante ese proceso, cada
15 *vez que detecta una etiqueta <dc:creator> o <dc:contributor> busca en
16 *las otras fuentes de datos información sobre ellos.
17 *
18 * También es responsable de conectar toda la información generando URIs
19 *
20 * Podemos elegir la página de la que partirá el programa con el
21 * parámetro i.
22 */
23 public static void unificar(int i) throws Exception {
```

Listing A.3: Cabeceras y descripciones de las funciones de *unificador.java* II

```
1
2  /*Input: La línea en la que nos encontramos y un booleano que describe
3  * el interés que teníamos en la anterior (y presumiblemente en la actual).
4  * Output: Otro booleano que indica el interés real que tenemos en la
5  * etiqueta de la línea.
6  *
7  * Subfunción de unificar, que hace que toda la información que no nos
8  * interesa y está en los request.xml no aparezca en el fichero
9  * definitivo en el que trabajaremos.
10 *
11 * El parámetro linea contiene la línea en la que nos encontramos,
12 * mientras que el booleano indica si estamos entre información
13 * interesante o no.
14 *
15 * Devuelve otro booleano indicando lo mismo que el anterior,
16 * el interés que tenemos en la etiqueta de la línea actual.
17 */
18
19 public static boolean dentroFuera (String linea, boolean b)
```

Listing A.4: Cabeceras y descripciones de las funciones de *unificador.java* III

```
1
2  /*Input: Líneas de código XML crudo, como nos lo presenta DSpace
3  * Output: Líneas de código tratado, si caracteres extraños que
4  * un gestor de bases de datos no relacionales no pueda interpretar
5  */
6
7  public static String transformar (String s)
8
9
10 /*Input: Un array con atos del profesor:
11 * Datos[0] = Centro
12 * Datos[1] = Departamento
13 * Datos[2] = Campus
14 * Datos[3] = Teléfono
15 * Datos[4] = Correo
16 * Datos[5] = Nombre
17 * Un puntero al fichero en el que se escribirá, y el nombre del Profesor
18 *
19 * Output: Todos esos datos del profesor escritos en el fichero que se le
20 * indica, más otros datos que buscará en la página del centro al que pertenezca:
21 *
22 * Doctorado/a
23 * Vascoparlante
24 * Tipo de contrato
25 * Página de la DBLP (solo en caso de pertenecer a la FISS)
26 */
27
28 public static void escribirProfe(String[] datos, PrintWriter out1, String nomProfAux, String
       nomProfesor) throws MalformedURLException, IOException{
```

Listing A.5: Cabeceras y descripciones de las funciones de *form.java* I

```
1
2  //Programa main para pruebas
3
4  public static void main(String[] args) throws Exception {
5
6  /*
7  * Input: Nombre del profesor
8  * Output: Un array de Strings con datos referentes al profesor extraídos de Bilatu
9  *           datos[0] = Centro
10 *           datos[1] = Departamento
11 *           datos[2] = Campus
12 *           datos[3] = Teléfono
13 *           datos[4] = Correo Electrónico
14 *           datos[5] = Nombre completo
15 *
16 *   Esta función realiza dos peticiones POST a Bilatu. La primera es una petición al
17 *  buscador. En ella, envía como parámetro el nombre y apellidos del profesor,
18 *  simulando que un usuario los ha introducido en la herramienta para buscar.
19 *
20 *  Después, rastrea los resultados de Bilatu, y si hay alguna coincidencia, hace
21 *  la segunda petición, esta vez con el enlace encontrado, con la información del
22 *  profesor. Finalmente, obtiene de esa seguna página los datos mencionados arriba.
23 */
24 public static String[] buscarDatosProfesor(String nombre) throws Exception {
25
26
27
28
29
30 /*
31 * Input: URI de un profesor en Bilatu
32 * Output: Datos relativos al profesor:
33 *
34 *           datos[0] = Centro
35 *           datos[1] = Departamento
36 *           datos[2] = Campus
37 *           datos[3] = Teléfono
38 *           datos[4] = Correo Electrónico
39 *           datos[5] = Nombre completo
40 *
41 * Función auxiliar a la anterior, que hace la segunda llamada que mencionábamos.
42 */
43 public static String[] datosProfesor(String url) throws IOException, InterruptedException
```

Listing A.6: Cabeceras y descripciones de las funciones de *form.java* II

```
1
2  /* Input: URI de un recurso DSpace
3  * Output: Tamaño del recurso
4  *
5  * Accede a la interfaz habitual para el usuario de un recurso DSpace (único lugar
6  * en el que encontramos el tamaño del recurso), y extrae de ella el tamaño del recurso
7  */
8
9  public static String obtenerTamano(String url) throws InterruptedException, IOException{
10
11 /* Input: URI de un recurso DSpace
12 * Output: Formato del recurso
13 *
14 * Función similar a la anterior, accede a la interfaz habitual para el usuario de un recurso
15 * DSpace, y extrae de ella el tamaño del recurso
16 */
17
18 public static String obtenerFormato(String url) throws InterruptedException, IOException{
19
20 /*
21 * Input: Dos strings, uno corto, y uno largo
22 * Output: Busca el dato aguja en el String pajar y lo devuelve
23 *
24 * Función orientada a HTML, rastrea una etiqueta en un fichero HTML, y
25 * devuelve su valor. Se utiliza para buscar los datos Bilatu de un profesor
26 */
27 public static String buscarDato(String aguja, String pajar)
28
29
30 /*
31 * Input: Dos strings, uno corto, y uno largo, y un punto de partida.
32 * Output: Busca el dato aguja en el String pajar y lo devuelve.
33 *
34 * Función parecida a la anterior, solo que esta vez se le da un punto de partida.
35 * Se utiliza para rastrear las páginas de las facultades. Se utiliza para encontrar
36 * datos de un profesor concreto. Se le pasa a la fución qué buscar, dónde y desde qué
37 * punto (en este caso, el índice del string en el que encontramos el nombre del profesor)
38 */
39
40 public static String buscarDatoDesde(String aguja, String pajar, int desde)
```

Listing A.7: Cabeceras y descripciones de las funciones de *form.java* III

```
1
2  /*
3  * Input: Un punto de partida y un String
4  * Output: El dato a partir del punto de partida.
5  *
6  * Función auxiliar a las dos anteriores. Se le pasa el índice desde
7  * el que empieza el dato, y devuelve el String que contiene hasta
8  * el cierre de la etiqueta.
9  */
10
11 public static String dato (int indice, String respuesta)
12
13 /*
14 * Sin input
15 * Output: Guarda todas las páginas de las facultades, haciendo una petición por cada una.
16 */
17
18 public static void inicializarPaginasEHU() throws InterruptedException, IOException{
19
20 /*
21 * Input: una URI y varios parámetros.
22 * Output: Código HTML de una página
23 *
24 * Función auxiliar a la anterior. Se le pasa la URI de una página de un centro, y
25 * varios parámetros POST necesarios para que la página devuelva  lo deseado,
26 * y distintos para cada facultad. Realiza la petición y devuelve el código HTML.
27 *
28 */
29
30 public static String paginaEHU(String url, String redirect, String centro, String plan, String
       origen) throws InterruptedException, IOException
31
32
33
34 /*
35 * Input: Lista de parámetros
36 * Output: String único con parámetros
37 *
38 * Función que utiliza la anterior, convierte todos los parámetros POST en uno para
39 * poder hacer la petición.
40 */
41
42 private static String getQuery(List<NameValuePair> params) throws UnsupportedEncodingException
```

Listing A.8: Cabeceras y descripciones de las funciones de *form.java* IV

```
1
2  /*
3  * Input: Conexión HTTP e intento
4  * Output: Buffer con el resultado de la petición
5  *
6  * Función utilizada por el resto para hacer peticiones. Devuelve null en caso de no poder
7  * completar el proceso.
8  */
9
10 public static BufferedReader pruebaIn(HttpURLConnection con, int i) throws InterruptedException
11
12 /*
13 * Input: Una conexión y el número de intentos
14 * Output: Datos preparados para ser enviados para una petición
15 *
16 * Función recursiva, que en caso de fallar se llama a sí misma hasta que tiene éxito.
17 */
18
19 public static DataOutputStream pruebaOut(HttpURLConnection con, int i) throws
       InterruptedException{
20
21 /*
22 * Input: Código HTML
23 * Output: Existencia de una URI
24 *
25 * Función que determina la existencia de una URI. Algunos portales tienen
26 * controlado el acceso a páginas que no existen o restringidas, por
27 * lo que la petición no termina con un código de error 404, u otro tipo,
28 * como algunos sitios sí hacen. Esta función comprueba que la página no tenga
29 * mensajes de error en el HTML, como, por ejemplo, "Invalid URL".
30 * Se utiliza para comprobar la existencia de páginas de la DBLP.
31 */
32
33 public static boolean existeURL(String urlString) throws MalformedURLException, IOException,
       InterruptedException {
34
35 /*
36 * Input: Nombre de un profesor, y centro al que pertenece
37 * Output: Datos "de segunda generación" del profesor
38 *
39 * El programa busca la página del centro indicado, y rastrea el nombre del profesor en ella.
       Devuelve los datos
40 * que podemos encontrar en ella.
41 */
42
43 public static String[] profesEHU(String nom, String centro) throws MalformedURLException,
       IOException, InterruptedException{
```

Listing A.9: Cabeceras y descripciones de las funciones de *form.java* V

```
1
2  /*
3  * Input: Nombre del profesor
4  * Output: Pruebas de dirección de DBLP
5  *
6  * Con el nombre del profesor, prueba varias opciones de enlaces en la DBLP, y si encuentra alguno
       válido, lo devuelve.
7  */
8  public static String obtenerDBLP(String nomProfesor) throws MalformedURLException, IOException,
       InterruptedException{
9
10
11 /*
12 * Input: Nombre completo de un profesor,
13 * con formato 1er Apellido 2do Apellido, Nombre
14 * Output: El nombre
15 */
16
17 public static String obtenerNombre(String nom){
18
19 /*
20 * Input: Nombre completo de un profesor, con formato 1er Apellido 2do Apellido, Nombre
21 * Output: El primer apellido
22 */
23
24 public static String obtener1Apellido(String nom){
25
26
27 /*Input: String
28 * Output: String adaptado a las URIs de la DBLP
29 *
30 * Cambia los caracteres con tilde al formato que usa la DBLP; por ejemplo,
31 *
32 * "á" => "=aacute="
33 */
34
35 public static String transformarDBLP (String s)
36
37
38 /*
39 * Input: String
40 * Output: String sin caracteres con tilde o ñ
41 */
42
43 public static String transformar (String s)
```

Listing A.10: Cabeceras y descripciones de las funciones de *form.java* VI

```
1
2  /*
3  * Input: Nombre del profesor
4  * Output: Pruebas de dirección de DBLP
5  *
6  * Con el nombre del profesor, prueba varias opciones de enlaces en la DBLP, y si encuentra alguno
       válido, lo devuelve.
7  */
8  public static String obtenerDBLP(String nomProfesor) throws MalformedURLException, IOException,
       InterruptedException{
9
10
11 /*
12 * Input: Nombre completo de un profesor,
13 * con formato 1er Apellido 2do Apellido, Nombre
14 * Output: El nombre
15 */
16
17 public static String obtenerNombre(String nom){
18
19 /*
20 * Input: Nombre completo de un profesor, con formato 1er Apellido 2do Apellido, Nombre
21 * Output: El primer apellido
22 */
23
24 public static String obtener1Apellido(String nom){
25
26
27 /*Input: String
28 * Output: String adaptado a las URIs de la DBLP
29 *
30 * Cambia los caracteres con tilde al formato que usa la DBLP; por ejemplo,
31 *
32 * "á" => "=aacute="
33 */
34
35 public static String transformarDBLP (String s)
36
37
38 /*
39 * Input: String
40 * Output: String sin caracteres con tilde o ñ
41 */
42
43 public static String transformar (String s)
```

Listing A.11: Cabeceras y descripciones de las funciones de *form.java* VII

```
1
2  /*
3  * Input: Nombre de un profesor y la página del centro al que pertenece
4  * Output: Un array de strings con los datos extraíbles de la página
5  *
6  * Función que sigue el algoritmo descrito en las funciones anteriores, rastrea el nombre, y
7  * proporciona los datos que le siguen a continuación;
8  *
9  *       datos[0] = Doctorado
10 *       datos[1] = Vascoparlante
11 *       datos[1] = Categoría
12 */
13
14 public static String[] datosEHU (String nombre, String pagina)
```

ANEXO B

Ontologías

Listing B.1: Clases utilizadas y añadidas a la ontología HERO

```
1
2  Utilizado como Profesor Adjunto:
3
4  <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Assistant">
5  <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
6  <rdfs:comment>Adjunto/Ayudante</rdfs:comment>
7  </owl:Class>
8
9  Utilizado como Colaborador:
10
11 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#AssociateProfessor">
12 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
13 </owl:Class>
14
15 Utilizado como Catedrático
16
17 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#HighRankedProfessor">
18 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
19 <rdfs:comment>Catedrático</rdfs:comment>
20 </owl:Class>
21
22 Utilizado como Profesor Interino:
23
24 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#InterimProfessor">
25 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
26 </owl:Class>
27
28 Utilizado como Profesor Titular:
```

```
29
30 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Professor">
31 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
32 <rdfs:comment>Titular</rdfs:comment>
33 </owl:Class>
34
35 Súperclase:
36
37 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Tenure">
38 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
39 <rdfs:comment>Agregado</rdfs:comment>
40 </owl:Class>
41
42 Utilizado como Investigador
43
44 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Researcher">
45 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
46 </owl:Class>
```

Listing B.2: Atributos utilizados y añadidos a la ontología HERO

```
1
2  Utilizado para describir el Nombre:
3
4  <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#Name"/>
5
6
7
8  Utilizado para describir el Centro:
9
10 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#Institute"/>
11
12 Utilizado para describir el Departamento:
13
14 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#AppointedTo">
15 <rdfs:range rdf:resource="http://www.UniversityReferenceOntology.org/HERO#Department"/>
16 <rdfs:domain rdf:resource="http://www.UniversityReferenceOntology.org/HERO#Teacher"/>
17 </owl:ObjectProperty>
18
19 Utilizado para describir el Campus:
20
21 <owl:Class rdf:about="http://www.UniversityReferenceOntology.org/HERO#Campus">
22 <rdfs:subClassOf rdf:resource="http://www.UniversityReferenceOntology.org/HERO#Location"/>
23 <rdfs:comment xml:lang="en">The U.S. collegiate experience is strongly shaped by a residential
       tradition.[...] </rdfs:comment>
24 <rdfs:isDefinedBy xml:lang="en">a field on which the buildings of a university are situated</
       rdfs:isDefinedBy>
25 </owl:Class>
26
27 Utilizado para describir el número de teléfono:
28
29 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#TelephoneNumber"/>
30
31 Utilizado para describir el Email:
32
33 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#Email"/>
34
35 Utilizado para describir el conocimiento de Euskera:
36
37 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#SpeaksBasque"/>
38
39 Utilizado para describir si es Doctor/a o no:
40
41 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#HasDoctorate">
42 <rdfs:domain rdf:resource="http://www.UniversityReferenceOntology.org/HERO#AcademicStaff"/>
43 <rdfs:range rdf:resource="http://www.UniversityReferenceOntology.org/HERO#Doctorate"/>
44 </owl:ObjectProperty>
45
46 Utilizado para describir el Link a la DBLP:
47
48 <owl:ObjectProperty rdf:about="http://www.UniversityReferenceOntology.org/HERO#DBLPLink"/>
```

Listing B.3: Ejemplo de uso de la ontología HERO modificada

```
<dblp:Person rdf:about="http://dblp.uni-trier.de/pers/c/Calvo:Borja">
    <dblp:personLastModifiedDate>Tue, 03 Feb 2015 17:33:33 +0100</dblp:personLastModifiedDate>
    <dblp:primaryFullPersonName>Borja Calvo</dblp:primaryFullPersonName>
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/conf/antsw/BlumBC14" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/conf/pkdd/SechidisCB14" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/cmpb/FloresILC13" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/conf/ae/BlumBC13" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/kais/CalvoILL12" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/tcbb/IrurozkiCL11" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/conf/ai/IrurozkiCL11" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/prl/CalvoLL09" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/titb/ArmananzasCILMUBFLZ09
    " />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/cmpb/CalvoLFLL07" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/prl/CalvoLL07" />
    <dblp:authorOf rdf:resource="http://dblp.uni-trier.de/rec/journals/bib/LarranagaCSBGILASMR06"
     />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/a/Arma=ntilde=anzas:Rub=
    eacute=n" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/b/Bernales:Irantzu" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/b/Bielza:Concha" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/b/Blesa:Maria_J=" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/b/Blum:Christian" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/b/Brown:Gavin" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/f/Flores:Jose_Luis" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/f/Fullaondo:Asier" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/f/Furney:Simon_J=" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/g/Galdiano:Josu" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/i/Inza:I=ntilde=aki" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/i/Irurozki:Ekhine" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/l/Larra=ntilde=aga:Pedro" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/l/L=oacute=pez=Bigas:N=uacute
    =ria" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/l/L=oacute=pez=Hoyos:Marcos"
    />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/l/Lozano:Jos=eacute=_Antonio"
     />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/m/Mart=iacute=nez:Aritz_P=
    eacute=rez" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/m/Mart=iacute=nez=Taboada:V=
    iacute=ctor" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/r/Robles:V=iacute=ctor" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/s/Santaf=eacute=:Guzm=aacute=
    n" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/s/Santana:Roberto" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/s/Sechidis:Konstantinos" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/u/Ucar:Eduardo" />
    <dblp:coCreatorWith rdf:resource="http://dblp.uni-trier.de/pers/z/Zubiaga:Ana_M=" />
    <dcterms:license rdf:resource="http://www.opendatacommons.org/licenses/by/" />
</dblp:Person>
```

Listing B.4: Ejemplo de uso de la ontología HERO modificada

```
<hero:Assistant rdf:about="http://www.ehu.es/docente/calvomolinosborja">
    <hero:Name>Calvo Molinos, Borja</hero:Name>
    <hero:Institute>http://www.ehu.es/centros/CentrosdeGipuzkoa/FacultaddeInformatica</hero:
     Institute>
    <hero:AppointedTo>http://www.ehu.es/departamentos/
     CienciadelaComputacioneInteligenciaArtificial</hero:AppointedTo>
    <hero:Campus>http://www.ehu.es/campus/Gipuzkoa</hero:Campus>
    <hero:TelephoneNumber>943 01 5013</hero:TelephoneNumber>
    <hero:Email>borja.calvo@ehu.eus</hero:Email>
    <hero:SpeaksBasque>Yes</hero:SpeaksBasque>
    <hero:HasDoctorate>Yes</hero:HasDoctorate>
    <hero:DBLPLink>http://dblp.uni-trier.de/pers/c/Calvo:Borja</hero:DBLPLink>
</hero:Assistant>
```

Bibliografía

[1] W3C. Guía Breve de Linked Data; 2010. Available from: http://www.w3c.es/Divulgacion/GuiasBreves/LinkedData.

[2] DuraSpace. DSpace1.5; 2008. Available from: "http://dspace.org/sites/dspace.org/files/archive/1_5_2Documentation/index.html".

[3] UPV/EHU. ADDI. https://addiehues/. 2010;.

[4] Institut für Angewandte Informatik und Formale Beschreibungsverfahren. Página de la iniciativa Semantic Web; 2012. Available from: http://semanticweb.org/wiki/Main_Page.

[5] Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific American Magazine. 2001 May;.

[6] W3C. Semantic Web. http://wwww3org/2001/sw/. 2013;.

[7] W3C. Página sobre RDF de W3; 2014. Available from: http://www.w3.org/RDF/.

[8] Berners-Lee T. Linked Data-Design Issues. http://wwww3org/DesignIssues/LinkedDatahtml. 2006;.

[9] W3C. Documentación de SPARQL; 2009. Available from: http://skos.um.es/TR/rdf-sparql-query/.

[10] Stamford University, Página de Protégé; 2013. Available from: http://protege.stanford.edu/.

[11] Gaševic D, Djuric D, Devedžic V. Model Driven Engineering and Ontology Development. 2009;.

[12] Spanos D, Konstantinou N, Houssos N, Mitrou N. Exposing Scholarly Information as Linked Open Data: RDFizing DSpace contents. The Electronic Library Journal. 2014 Oct;.

[13] W3C. Mejores ontologías según W3C; 2015. Available from: http://www.w3.org/wiki/Good_Ontologiese.

[14] ASIST. Portal del Dublin Core; 2003. Available from: http://dublincore.org/.

[15] DuraSpace. Documentación de la ontología VIVO;.

[16] DuraSpace. Página de la ontología HERO; 2013. Available from: http://sourceforge.net/projects/heronto/.

[17] de Trier U. Ontología DBLP;. Available from: http://dblp.uni-trier.de/rdf/schema-2015-01-26#.

[18] DuraSpace. Página de Apache Jena; 2009. Available from: https://jena.apache.org/.

[19] Complexible. Documentación de Stardog.js; 2015. Available from: http://complexible.github.io/stardog.js/docs/stardog.html.

[20] Complexible. PáginaStardog.js; 2015. Available from: https://github.com/Complexible/stardog.js/blob/develop/README.md.