litτera
sistema para la federación de depósitos de datos enlazados

Proyecto Fin de Carrera

10 de julio de 2013

Alberto Calvo García

Directora:
Arantza Illarramendi Echave

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Abstract

English

In this project, conceived under the semantic web philosophy, a system that allows one to federate datasets of linked data has been developed. With this system, called litτera, it is possible to query datasets of the domain of the literature seamlessly through a web interface, and build collaboratively a new dataset of open linked data.

The analysis performed and the techniques used to develop the system are not exclusive to the domain chosen to do the tests, so they can be easily extrapolated to other use cases.

Castellano

En este proyecto, enmarcado dentro del campo de la web semántica, se ha desarrollado un sistema llamado litτera que permite la federación de depósitos de datos enlazados. Dicho sistema permite consultar depósitos del dominio de la literatura de forma unificada a través de una interfaz web, así como construir colaborativamente un nuevo depósito de datos abiertos y enlazados.

Los análisis realizados y las técnicas utilizadas para el desarrollo del sistema que se recogen en esta memoria no son particulares del dominio elegido para hacer las pruebas, por lo que pueden ser fácilmente extrapolables a otros casos de uso.

Agradecimientos

Gracias a mi directora, Arantza, por su ayuda, sus consejos y su paciencia.

Gracias a mis padres, Begoña e Isidro, por su apoyo incondicional, y a mi hermana Laura también por haber contribuido en este proyecto dándole el nombre que tiene.

Gracias a todos los que han participado, han hecho posible, o simplemente han creído en esa ilusionante realidad llamada Magna SIS: Gorka, Ander, Jon, Fernando, Jere, Manex, José Miguel, el Decanato de la Facultad de Informática, el Consejo de Estudiantes de Gipuzkoa, la Federación Vasca de Junior Empresas, Dinitek, Arteverse y un largo etcétera; porque gracias a ellos he tenido un último curso muy diferente y mucho más enriquecedor de lo que podría haber llegado a imaginar.

Gracias a aquellos que han compartido estos años de carrera conmigo, sean profesores, compañeros, o amigos y conocidos de fuera de la universidad, porque todos ellos han dejado, en mayor o menor medida, su impronta en mí.

Gracias a todos aquellos que se interesen por este proyecto y lean más allá de esta primera página, porque dan sentido al trabajo realizado.

Zuei guztiei, mila esker.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde la aparición de los primeros depósitos de datos abiertos y enlazados hace poco más de cinco años, esta forma de publicación de datos se ha popularizado enormemente. Gobiernos, instituciones, empresas e iniciativas privadas han puesto en marcha proyectos para la publicación de millones de datos sobre los más diversos temas: información geográfica, gubernamental, editorial, científica, etc.

El diagrama de la figura 1.1, elaborado por Richard Cyganiak y Anja Jentzsch en septiembre de 2011, muestra los depósitos de datos enlazados más importantes en aquel momento y las relaciones existentes entre ellos. Teniendo en cuenta el número de depósitos reflejados en dicho diagrama (295), el tiempo transcurrido desde entonces (casi dos años) y los depósitos menores que por su tamaño nunca llegarán a verse reflejados en un diagrama de estas características, podemos deducir que en la actualidad existen miles de depósitos que tienen datos publicados en la web.

Todos ellos se basan en una misma filosofía: datos con licencias libres (para poder *consumirlos* y reutilizarlos sin impedimentos legales) y enlazados con terceros depósitos (para poder descubrir información adicional sobre los mismos). Sin embargo, la forma de publicación, las plataformas y los mecanismos para acceder a dichos datos son muy dispares entre ellos. Ante tal situación, surge la necesidad de herramientas que permitan acceder a los datos de forma unificada y consumir simultáneamente de distintos depósitos. Es decir, surge la necesidad de *federar* los depósitos.

El trabajo realizado en este Proyecto Fin de Carrera ha tenido como objetivo analizar la problemática que supone acometer la tarea de la federación de depósitos heterogéneos de datos enlazados y desarrollar un sistema para

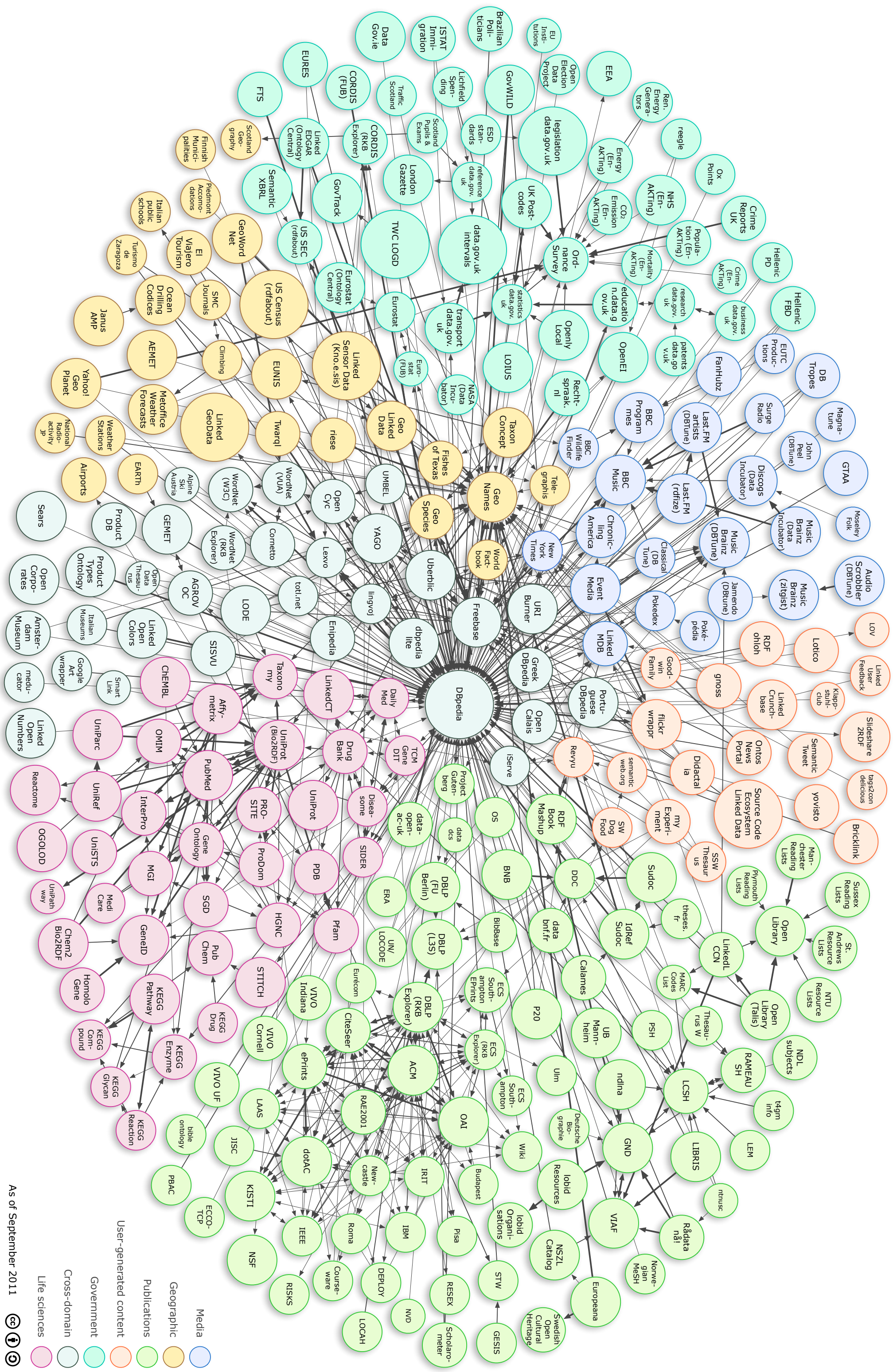


Figura 1.1: Diagrama con los depósitos de datos abiertos y enlazados

un dominio concreto (en este caso la literatura) que permita a los usuarios consultar diversos depósitos de datos mediante una interfaz unificada; teniendo siempre presente que los usuarios finales no tienen por qué conocer aspectos técnicos propios del campo de los datos enlazados, ni conocer las particularidades de los depósitos de datos federados.

Este documento, que es la memoria del trabajo realizado por Alberto Calvo García bajo la dirección de la doctora Arantza Illarramendi durante el segundo cuatrimestre del curso académico 2012–2013, está estructurado de la siguiente forma:

- Se comienza con el documento de objetivos del proyecto, en el cual se describe la planificación confeccionada al inicio del mismo.
- A este le sigue un capítulo que comienza introduciendo el contexto tecnológico en el que se enmarca el proyecto, seguido de una exposición del trabajo existente hasta el momento, y un análisis del dominio, la arquitectura y las tecnologías elegidas para desarrollar el sistema.
- Después se encuentra un apartado relativo al desarrollo del sistema en el que se detallan, entre otros aspectos, los casos de uso planteados, el modelo de clases confeccionado y algunos de los algoritmos utilizados.
- A continuación se incluye un capítulo con detalles de la implementación, tales como el modelo de desarrollo, el sistema de pruebas utilizado para cerciorar su correcto funcionamiento, los requisitos mínimos necesarios para su uso y los resultados empíricos obtenidos con la aplicación.
- Seguidamente se hace una breve valoración final del proyecto y se enumeran algunas ideas y líneas de mejora que podrían implementarse para seguir trabajando en el sistema.
- En el último capítulo se incluye la bibliografía con un apartado relativo a las referencias en Internet.
- Por último se ha incluido como apéndice el seguimiento y control del proyecto (que detalla y reflexiona sobre los cambios acontecidos con respecto al plan inicial del documento de objetivos del proyecto), y un manual para la instalación y administración del sistema desarrollado.

Capítulo 2

Documento de objetivos

En este capítulo se detalla la planificación confeccionada al inicio del proyecto. Se comienza enumerando los objetivos del mismo, tanto los directos como los indirectos, para continuar con la definición del alcance del proyecto y los entregables que generará el mismo. A continuación se incluye la estructura de descomposición del trabajo, la planificación temporal y los planes de calidad y comunicaciones. Para terminar se analizan los riesgos a los que se enfrenta el proyecto, categorizándolos y midiendo su posible impacto.

2.1. Objetivos

El objetivo del proyecto es desarrollar un sistema que permita realizar consultas cuyas respuestas se obtengan accediendo a distintos depósitos de datos enlazados (*linked data*). El sistema deberá mostrar los resultados de forma integrada y completa, con el ánimo de simplificar y automatizar dicha labor.

La infraestructura para la federación de los depósitos de datos deberá ser lo más flexible y escalable posible, con el fin de permitir tanto la mejora futura de las funcionalidades existentes y la adición de nuevas características, como la incorporación de nuevos depósitos de datos.

Asimismo, se marcan como objetivos indirectos la familiarización del desarrollador con el campo de los datos enlazados y el manejo del sistema Virtuoso, así como la adquisición de experiencia en el desarrollo web, en los procedimientos ingenieriles para el desarrollo de software, y en la edición de documentos con LaTeX.

2.2. Alcance

A continuación se marca el alcance mínimo del sistema que se plantea, las posibles líneas de ampliación que se prevén, y los aspectos que en todo caso quedan fuera del alcance del proyecto.

2.2.1. Mínimo

El sistema a desarrollar deberá contar, al menos, con las siguientes funcionalidades:

- Soporte para la federación de dos depósitos de datos.
- Interfaz web para para la realización de consultas, compatible con las últimas versiones de los navegadores web más utilizados.
- Procesamiento de preguntas que integre los resultados de cada depósito y posteriormente proporcione una visualización web.

2.2.2. Líneas de ampliación

Se proponen las siguientes ampliaciones de funcionalidades:

- Federación de un depósito almacenado en una base de datos NoSQL.

2.2.3. Exclusiones

Queda fuera del alcance del proyecto la modificación o ampliación de los depósitos ya existentes.

2.3. Entregables

Memoria documento formal y detallado que contenga toda la información relativa al proyecto, así como lo acaecido durante el transcurso del mismo.

Sistema instancia en funcionamiento del sistema, que sirva como demostración del trabajo realizado.

Código fuente código fuente del sistema desarrollado.

2.4. Estructura de descomposición del trabajo

En la figura 2.1 se muestra la estructura de descomposición del trabajo (EDT) de este proyecto, en la cual se pueden apreciar los principales componentes o grupos de tareas de los que está compuesto y tener así una perspectiva global del mismo.

2.5. Planificación temporal

En esta sección se detalla la planificación temporal, que consta de los hitos clave del proyecto, la estimación del tiempo que deberá de invertirse para completar cada tarea, y el marco temporal en el se planea realizar estas tareas.

2.5.1. Hitos

En el cuadro 2.1 se enumera la lista de hitos junto con sus respectivas fechas, todas ellas correspondientes al año 2013.

Hito	Fecha
Inicio del desarrollo	20 de febrero
Prototipo	21 de marzo
Análisis de ampliación	2 de mayo
Finalización implementación	3 de junio
Finalización de todos los entregables	30 de junio
Defensa del proyecto	19 de julio

Cuadro 2.1: Hitos

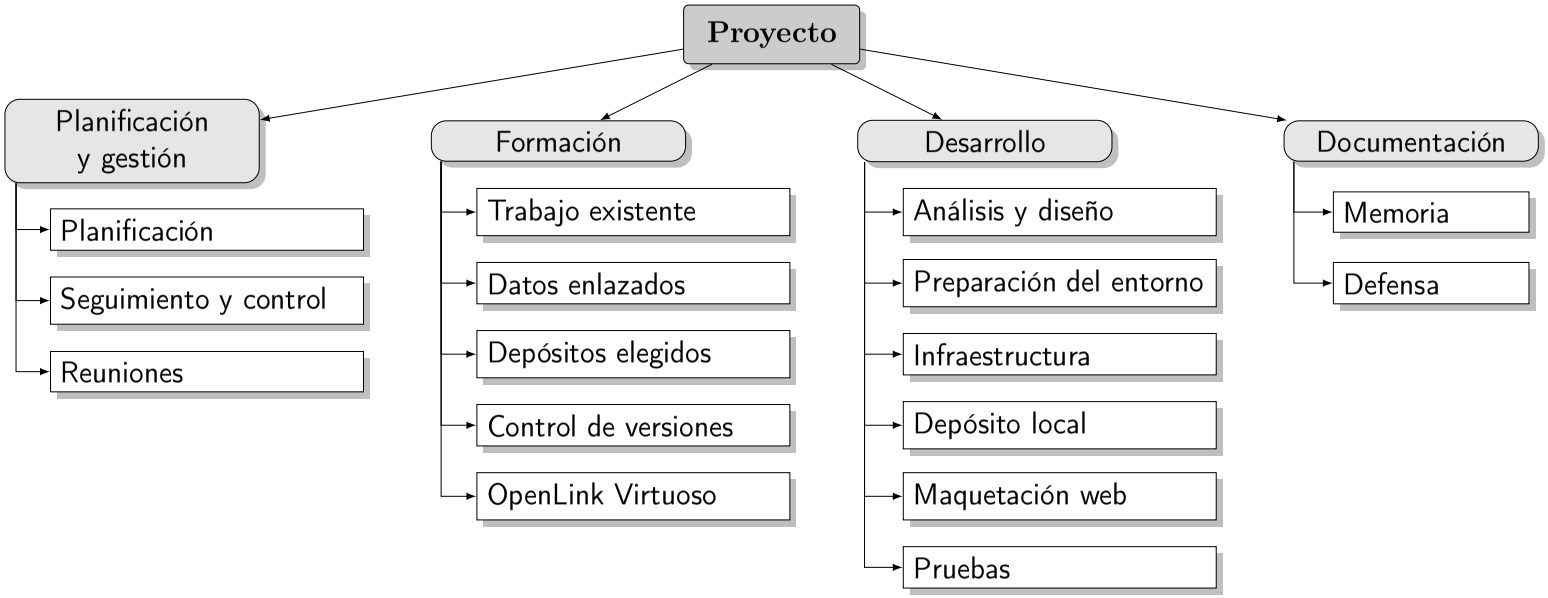


Figura 2.1: Estructura de descomposición del trabajo

2.5.2. Actividades

Las actividades a realizar para la consecución del alcance mínimo propuesto se estiman en un total de 340 horas de trabajo, y se distribuyen de la manera en la que se indica en el cuadro 2.2.

Tarea	T. estimado
Planificación y gestión del proyecto	**50 h.**
Planificación inicial	10 h.
Seguimiento y control	20 h.
Replanificaciones	10 h.
Reuniones	10 h.
Formación	**70 h.**
Formación general en datos enlazados	40 h.
Formación en el dominio elegido	10 h.
Formación en control de versiones	10 h.
Búsqueda y análisis del trabajo existente	10 h.
Desarrollo	**150 h.**
Análisis y diseño	20 h.
Preparación del entorno	10 h.
Programación de la infraestructura	100 h.
Maquetación web	10 h.
Pruebas	10 h.
Documentación	**70 h.**
Búsqueda y lectura de modelos	10 h.
Manual de usuario	10 h.
Resto de la memoria	30 h.
Revisión y corrección de errores	10 h.
Defensa	10 h.

Cuadro 2.2: Coste temporal estimado para el alcance mínimo

Asimismo, se plantean las tareas asociadas a la línea de ampliación propuesta que se detallan en el cuadro 2.3 y que suman un total de 100 horas. Su realización se decidirá tras el análisis de la situación que se realice el día del hito correspondiente.

Tarea	T. estimado
Formación en OpenLink Virtuoso	30 h.
Federación del depósito local	50 h.
Pruebas	10 h.
Documentación asociada	10 h.

Cuadro 2.3: Coste temporal estimado para la ampliación del alcance

2.5.3. Diagrama de Gantt

En la figura 2.2 se muestra el diagrama de Gantt de este proyecto, mediante el cual se pueden visualizar gráficamente los plazos de dedicación previstos para las diferentes actividades a lo largo de la vida del proyecto.

2.6. Calidad

Para asegurar la calidad de los entregables, se plantean dos líneas de actuación:

1. En en lo relativo al sistema, se ha fijado un hito (el 21 de marzo) para la presentación de un prototipo funcional que sirva como muestra. Basándose en las críticas recibidas, se elaborará una lista exhaustiva con los detalles y características que debe tener la versión final del programa.

2. En cuanto a la documentación, se ha planificado una tarea consistente en una lectura y revisión en profundidad de la totalidad de la memoria a entregar.

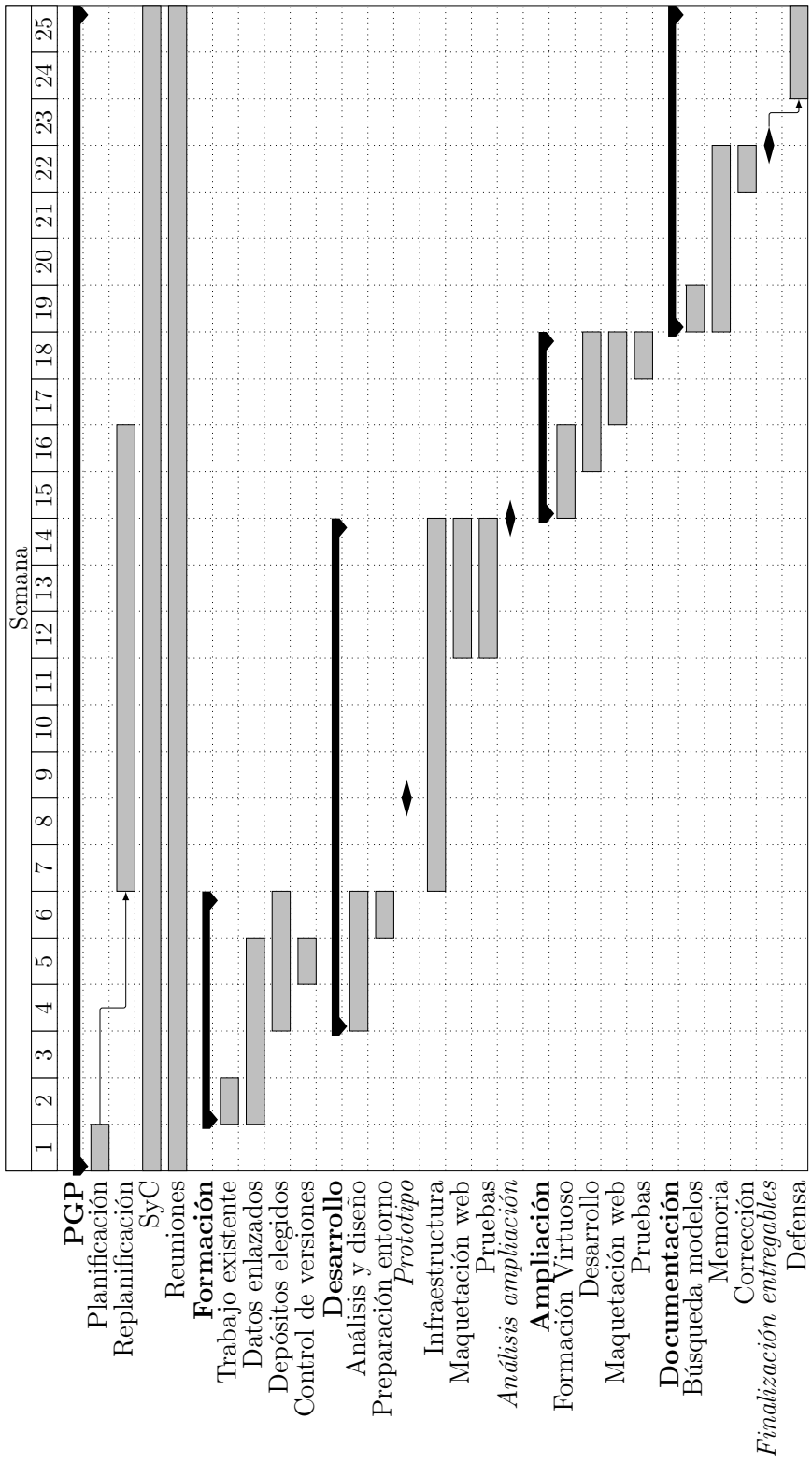


Figura 2.2: Diagrama de Gantt

2.7. Comunicaciones

Seguidamente se traza el plan de comunicaciones, que consta de la identificación de los interesados en el proyecto, los métodos que se prevén para la comunicación entre estos, y la metodología que se utilizará para garantizar el correcto desarrollo del proyecto.

2.7.1. Interesados

Nombre	**Rol**	**Correo electrónico**
Alberto Calvo García	Desarrollador	acalvo028@ikasle.ehu.es
Arantza Illarramendi Echave	Directora	a.illarramendi@ehu.es

2.7.2. Métodos de comunicación

Se establecen dos siguientes canales para la comunicación entre las personas implicadas en el proyecto:

1. Conversación en persona
2. Correo electrónico

Cada canal será utilizado dependiendo de la urgencia, de la necesidad de información y del tipo de la misma.

2.7.3. Seguimiento y control

Se celebrarán reuniones de seguimiento cada una o dos semanas en las que el desarrollador informará del estado del proyecto, para decidir y dirigir el rumbo y los detalles del mismo.

2.8. Riesgos

A continuación se listan los riesgos a los que se enfrenta el proyecto, así como la clasificación de los mismos según su probabilidad y el impacto que ocasionarían en la consecución satisfactoria de los objetivos del proyecto.

Retrasos

Probabilidad alta

Impacto bajo-medio, dependiendo de los días de retraso de los que se trate

Prevención ajustarse al plan establecido tanto como fuese posible

Planificación inicial inadecuada

Probabilidad alta

Impacto medio

Prevención imposible; si se diese el caso, convendría replanificar los aspectos que no sean adecuados en el plan original

Diseño inadecuado

Probabilidad media

Impacto alto

Prevención asegurar la validez del diseño antes de comenzar la implementación

Requisitos definidos incorrectamente

Probabilidad media

Impacto alto

Prevención tratar de detectar cuanto antes dichos errores, para actuar en consecuencia con el mínimo coste posible

Pérdida de información

Probabilidad muy baja

Impacto alto

Prevención utilización de sistemas para el almacenamiento automático de todo el sistema de información en la nube

Baja por enfermedad

Probabilidad baja

Impacto medio, dependiendo de los días de baja

Prevención imposible

Capítulo 3

Análisis

3.1. Contexto tecnológico

A continuación se introducen brevemente conceptos básicos del campo de los datos enlazados que se utilizarán recurrentemente a lo largo de la memoria.

3.1.1. Web semántica

La Web semántica[25] (del inglés *semantic web*) es un conjunto de actividades desarrolladas en el seno del *World Wide Web Consortium*[78] tendente a la creación de tecnologías para publicar datos legibles por aplicaciones informáticas. Se basa en la idea de añadir metadatos semánticos y ontológicos a la *World Wide Web* (tales como la descripción del contenido, el significado y la relación de los datos), para hacer posible su evaluación automática y ampliar así la interoperabilidad entre los sistemas informáticos.

Dentro de este marco nace la propuesta de datos enlazados[26] (*linked data* en inglés).

3.1.2. Datos enlazados

En 2006 Tim Berners-Lee, padre de la web, definió en un artículo del W3C[2] los pilares en los que se basa la propuesta, y años más tarde, en 2009, lo

presentó públicamente en el congreso TED[3].

La idea consiste en ofrecer un único mecanismo estándar de acceso a todos los datos publicados, en vez de que cada servicio ofrezca su propia interfaz de acceso (API) y devuelva los resultados en un formato propio. Así, un único estándar abierto de publicación y acceso a los datos hace posible enlazar datos de diferentes depósitos, acceder a ellos utilizando exploradores genéricos de datos, y automatizar su tratamiento utilizando herramientas informáticas interoperables.

Para animar a utilizar buenas prácticas a la hora de publicar datos, Tim Berners-Lee definió la siguiente clasificación de los mismos según su calidad. Cada escalón acumula las características de todos sus anteriores además de las suyas propias:

★ Datos disponibles en la web, en cualquier formato pero con una licencia libre o abierta

★★ Datos disponibles en un formato legible por una máquina[1]

★★★ Datos disponibles en un formato no propietario[2]

★★★★ Uso de los estándares del W3C (RDF y SPARQL) para identificar los objetos

★★★★★ Enlaces con depósitos externos de datos para interconectarlos y descubrir nueva información

3.1.3. RDF

RDF[17] (del inglés *Resource Description Framework*) es un estándar del W3C para la codificación del conocimiento. Modela las declaraciones de los recursos en *triples* con la forma "sujeto – predicado – objeto", cuyo significado es:

sujeto recurso, aquello que se está describiendo

predicado propiedad o relación que se desea establecer acerca del recurso

objeto valor de la propiedad o recurso distinto con el que se establece la relación

[1] por ejemplo, una tabla de datos en un archivo XLS en vez de en una imagen JPG
[2] por ejemplo, CSV en vez de XLS

Por ejemplo, en la figura 3.1 se representan los cinco triples que podrían modelar la información *Juan García conoce a Fulanito y Menganito*. El primero de ellos indica que la instancia *http://www.example.com/Juan_García* es una persona. El segundo, que su nombre de pila es *Juan*. El tercero, que sus apellidos son *García Pérez*. Y los dos últimos, que conoce tanto a *Fulanito* como a *Menganito*.

```
1 <http://www.example.com/Juan_García> <http://www.w3.org
      /1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/
      Person>
2 <http://www.example.com/Juan_García> <http://xmlns.com/foaf/0.1/
      givenName> "Juan"
3 <http://www.example.com/Juan_García> <http://xmlns.com/foaf/0.1/
      familyName> "García Pérez"
4 <http://www.example.com/Juan_García> <http://xmlns.com/foaf/0.1/
      knows> <http://www.example.com/Fulanito>
5 <http://www.example.com/Juan_García> <http://xmlns.com/foaf/0.1/
      knows> <http://www.example.com/Menganito>
```

Figura 3.1: Ejemplo de cinco triples de RDF

Dichos triples se pueden codificar utilizando diferentes notaciones, siendo la *normativa* XML. Aún así, en esta memoria, cuando se tienen que representar triples, se hace utilizando la notación N3 que está diseñada para facilitar la lectura a las personas. Así, los cinco triples anteriormente definidos pasarían a representarse de la forma en la que se muestra en la figura 3.2.

```
1 @prefix midominio: <http://www.example.com/> .
2 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3
4 midominio:Juan_García a                 foaf:Person ;
5                        foaf:givenName   "Juan" ;
6                        foaf:familyName  "García Pérez" ;
7                        foaf:knows       midominio:Fulanito ,
8                                         midominio:Menganito .
```

Figura 3.2: Ejemplo de la sintaxis de la notación N3

3.1.4. SPARQL

SPARQL[8] (del inglés *SPARQL Protocol and RDF Query Language*) es un lenguaje estandarizado por el W3C para la consulta de grafos RDF.

En la figura 3.3 se muestra una consulta de ejemplo que, en caso de ejecutarse sobre un depósito que contenga los triples de las figuras 3.1/3.2, devolvería lo que se muestra en la figura 3.4.

```
1 PREFIX midominio: <http://www.example.com/>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3
4 SELECT ?Conocidos WHERE {
5     midominio:Juan_García foaf:knows ?Conocidos .
6 }
```

Figura 3.3: Ejemplo de la sintaxis de SPARQL

Conocidos
http://www.example.com/Fulanito
http://www.example.com/Menganito

Figura 3.4: Ejemplo de resultado de una consulta SPARQL

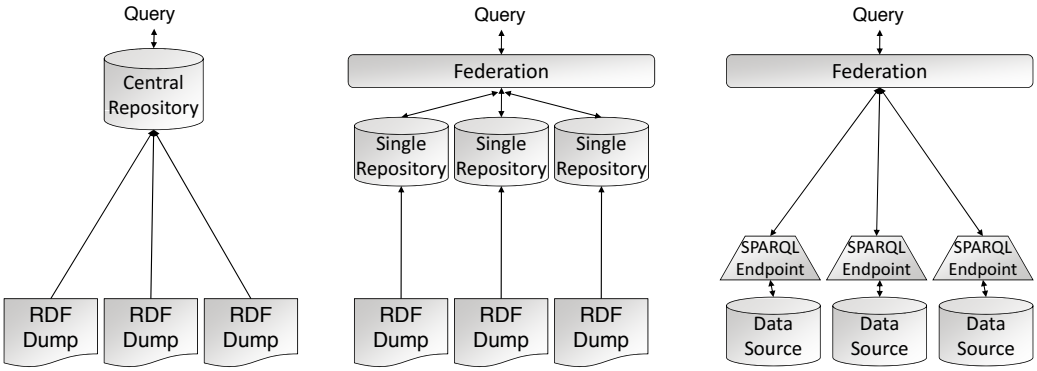
3.1.5. Depósito de triples

Un depósito de triples (del inglés *triplestore*) es una base de datos diseñada y optimizada para almacenar y recuperar triples. Dichas operaciones se realizan mediante un lenguaje de consulta como SPARQL y normalmente también se da la opción de importar o exportar datos en RDF u otros formatos.

3.2. Trabajo existente

La federación de depósitos en el campo de los datos enlazados puede abordarse a partir de tres modelos diferentes, los cuales se muestran en la figura 3.5.

En el artículo [12] se hace una interesante comparativa de los tiempos de carga y de respuesta entre configuraciones de los tres tipos. Como era de esperar, se obtiene que el modelo consistente en el volcado de todos los depósitos en uno central y la consulta directa a este (figura 3.5.a) ofrece mejores tiempos de respuesta a la hora de consultar, aunque el coste de carga y puesta en marcha



(a) Repositorio central (b) Múltiples repositorios individuales (c) Federación de *SPARQL endpoints*

Figura 3.5: Distintos modelos para la federación de depósitos de datos enlazados

es sustancial. Por otro lado, el modelo completamente distribuido consistente en la consulta a través de los *SPARQL endpoints* de cada depósito de datos (figura 3.5.c) supone eliminar por completo el coste de carga inicial de datos, pero a cambio arroja unos tiempos de respuesta muy superiores a la hora de realizar las consultas. Por último, el modelo de la figura 3.5.b supone en términos temporales una solución a caballo entre las dos anteriores. En el sistema a desarrollar en este proyecto, se decidió optar por la configuración representada por la figura 3.5.c, ya que dota al sistema de una flexibilidad y dinamicidad que el resto de modelos no pueden ofrecer.

En cuanto a los desarrollos realizados hasta el momento, destaca DARQ[22] que, a pesar de que en la actualidad haya dejado de desarrollarse, fue una de las infraestructuras pioneras en este terreno. Cuenta con un sistema de optimización de consultas que se vale de datos estadísticos sobre los diferentes depósitos para reescribir las consultas. Asimismo, ofrece un sistema en modo experimental para definir manualmente *mappings*, es decir, relaciones entre términos de diferentes ontologías. Del mismo modo podemos citar a FedX[24], infraestructura posterior a DARQ que puso especial atención en minimizar las peticiones a los depósitos para reducir las latencias en la medida de lo posible, así como evitar el preprocesamiento previo de los depósitos para poder crear federaciones *al vuelo*.

La principal diferencia con respecto al sistema desarrollado en este proyecto es que, mientras DARQ y FedX son infraestructuras generales para la federación de depósitos de datos enlazados, `litτera` es un sistema accesible desde la web que abstrae al usuario de las particularidades de los depósitos en uso y le permite realizar acciones adicionales con los resultados obtenidos, como su

revisión e inclusión en un nuevo depósito de datos. Las consultas se realizan a través una interfaz única y basada en formularios que, a diferencia de aquellas herramientas, no requiere de conocimientos de SPARQL por parte del usuario. Además, la interfaz web hace posible acceder al sistema desde cualquier dispositivo y sin la necesidad de instalaciones previas.

Por otro lado, y ya que en este proyecto se trabaja en la construcción de un depósito de datos enlazados con la información que van revisando los propios usuarios del sistema, merece la pena mencionar la existencia de las wikis semánticas. Estos sistemas permiten combinar la redacción tradicional de textos en lenguaje natural con la definición de datos semánticos de modo que puedan ser consultados, tratados o exportados. De entre las alternativas existentes, la plataforma de este tipo más utilizada es Semantic MediaWiki[44], aunque existen otras opciones quizá más completas como DataWiki[42]

Aunque para otros desarrollos puedan ser herramientas a tener en cuenta, para este proyecto se desestimó su uso ya que, en el caso de Semantic MediaWiki, los datos especificados no se guardan en depósitos de triples sino en bases de datos relacionales, y no admite SPARQL como lenguaje de consulta[11]. DataWiki, sin embargo, ofrece opciones más avanzadas que las proporcionadas por Semantic MediaWiki, pero el alto coste económico de su licencia (a partir de los 1900 euros) la descarta por completo para el proyecto que nos ocupa.

3.3. Dominio

Al seleccionar el dominio sobre el que hacer las pruebas y desarrollar la infraestructura, se valoraron diferentes alternativas. Aunque hay dominios muy interesantes a los que un sistema de estas características podría aportar mucho valor, tales como la biología, la química u otras ciencias en las que se estén constantemente descubriendo propiedades o características de elementos, su implantación requeriría de un experto en dicho dominio, extremo del que no se disponía. Es por ello que se decidió tomar el dominio de la literatura, que aunque su aplicación práctica en principio pudiera ser menor en lo que a utilidad para la investigación se refiere, trata conceptos que son fácilmente comprensibles, pudiendo prescindir de expertos en el campo.

Una vez elegido el dominio sobre el que actuar, se procedió a buscar y analizar depósitos de datos enlazados, poniendo especial atención en las bibliotecas nacionales de los distintos países, con el requisito de que ofreciesen *SPARQL*

endpoints para acceder a sus datos. Se seleccionó la DBpedia[35] y las bibliotecas nacionales española[30], británica[29], sueca[32] y húngara[31]. A continuación se describen brevemente las características principales de cada una de ellas.

DBpedia La DBpedia[35] es un proyecto auspiciado por la Universidad de Leipzig[39], la Universidad Libre de Berlín[40] y la compañía OpenLink Software[33] para la extracción de datos de Wikipedia y la construcción de un depósito abierto de datos enlazados. Aunque, al igual que la Wikipedia, es de carácter general, resulta muy interesante debido por un lado, a la calidad de sus datos (consecuencia de haber sido redactados por personas humanas), y por otro lado a la ontología utilizada para la categorización de sus obras que, como se explicará más adelante, abre un abanico de posibilidades a la hora de consultar los datos. Como depósito de almacenamiento para los triples utiliza OpenLink Virtuoso.

BNE La Biblioteca Nacional de España[30] (en adelante BNE por sus siglas en castellano), tiene sus datos publicados bajo los estándares de los datos enlazados, almacenados al igual que la DBpedia mediante Virtuoso.

BL La Biblioteca Británica[29] (en adelante BL por sus siglas en inglés) es la mayor biblioteca de investigación del mundo. Dispone de un proyecto donde mantienen los datos derivados de la *British National Bibliography* (libros y artículos publicados o distribuidos en el Reino Unido e Irlanda desde 1950) en formato de datos enlazados, originalmente mediante una plataforma denominada Talis[37], y en la actualidad mediante un desarrollo de la empresa TSO[38].

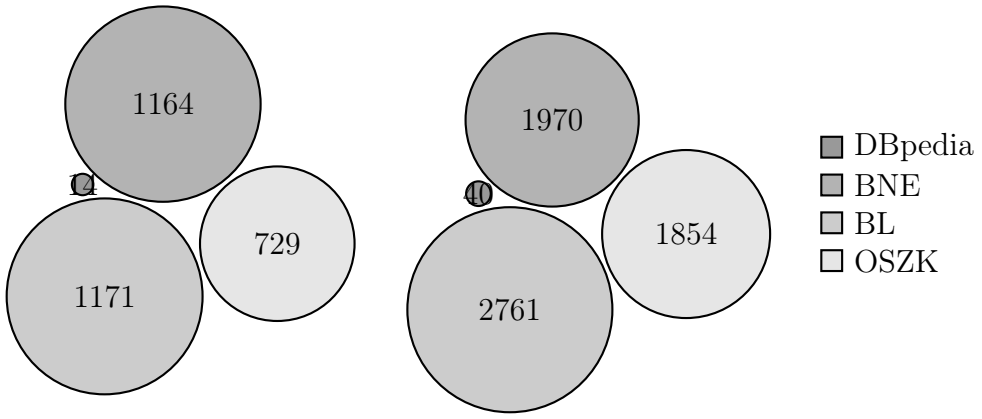
KB La Biblioteca Nacional de Suecia[32] (en adelante KB por sus siglas en sueco) disponía también de su catálogo publicado como datos enlazados. Sin embargo, en la recta final de este proyecto dejó de estar utilizable el *SPARQL endpoint* asociado al servicio. Es por ello que a lo largo de esta memoria en ocasiones se ofrecen datos sobre el depósito (aquellos que fueron recabados antes de la caída) y otras veces no. La decisión de mantener las referencias y los datos recabados en vez de obviar por completo el depósito se ha tomado porque se tiene la esperanza de que la no disponibilidad sea transitoria y en un futuro cercano vuelva a estar disponible el servicio, haciendo que la información y el análisis que aquí se recoge sea relevante de nuevo.

OSZK La Biblioteca Nacional Széchényi[31] (en adelante OSZK por sus siglas en húngaro), es una de las dos bibliotecas nacionales de Hungría. Aunque los datos publicados pueden considerarse enlazados, lo cierto es que en comparación con el resto de depósitos hay muchísimas instancias sin enlaces salientes, quedando por tanto completamente aisladas e inalcanzables desde el resto de depósitos.

En el cuadro 3.1 y en la figura 3.6 se muestran, numérica y gráficamente, el número de instancias almacenadas en cada depósito de datos. Esta información, sin embargo, hay que tomarla con cautela y en ningún caso juzgar la calidad o idoneidad de los depósitos basándose en ella.

	Escritores	**Obras**
DBpedia	13 743	40 016
BNE	1 163 764	1 969 526
BL	1 171 185	2 761 412
OSZK	728 834	1 853 697

Cuadro 3.1: Número de instancias almacenadas en cada depósito de datos



(a) escritores (b) obras

Figura 3.6: Número de instancias (en miles) almacenadas en cada depósito

De hecho, ni siquiera se compara exactamente lo mismo, ya que mientras que en la DBpedia los escritores y las obras escritas están clasificadas como tales, en las bibliotecas nacionales las instancias no están categorizadas. Así,

se toman como escritores a todas las personas definidas en dichos depósitos (aunque no tenga por qué ser así), mientras que se consideran obras escritas todas las obras almacenadas (aunque pudieran ser, por ejemplo, piezas musicales).

En el cuadro 3.2 se recoge de manera pormenorizada el vocabulario utilizado por cada uno de los depósitos para almacenar la información básica de los escritores y de las obras. Merece la pena realizar un pequeño análisis en torno a esto para comprender mejor el reto al que nos enfrentamos al federar depósitos que, aunque aparentemente ofrecen datos similares, lo hacen de manera muy dispar.

Tal es el caso de las fechas de nacimiento y fallecimiento, ya que de los cinco depósitos analizados, se presentan cuatro alternativas diferentes. Por ejemplo, la DBpedia dispone de dos campos diferenciados en el los que se especifica la fecha completa (*1970-01-01*), mientras que la KB y OSZK proporcionan solo el año (*1970*), y la BNE un solo campo en el que se especifican los dos años separados por un guión (*1970-2038*). Por último, la BL no define atributos con esta información.

Algo similar, aunque no tan dispar, ocurre con los nombres de los autores, ya que tres depósitos proporcionan de forma diferenciada el nombre completo, el nombre de pila, y el apellido, mientras que los otros dos solo ofrecen el nombre completo, delegando en el usuario el tratamiento del string para la distinción entre estos dos datos.

También puede llamar la atención la ontología utilizada por la BNE. Se trata de un modelo llamado *Requerimientos funcionales para registros bibliográficos*[14] (FRBR por sus siglas en inglés), diseñado por la *Federación Internacional de Asociaciones e Instituciones Bibliotecarias*[36] (IFLA) para el establecimiento de un marco compartido sobre la información que un registro bibliográfico debe proporcionar. Sin querer entrar a valorar más en profundidad este modelo y las características y deficiencias observadas en su implementación, simplemente merece la pena apuntar que no ha aportado ninguna ventaja ni facilidad a la hora de acometer los objetivos de este proyecto.

En un escenario ideal, el análisis que aquí se ha realizado podría obtenerse de manera semiautomática ya que, igual que se definen relaciones de equivalencia entre instancias de distintos depósitos, se puede (y se debe) establecer relaciones similares entre los términos de las diferentes ontologías. Sin embargo, estas relaciones son por desgracia muy escasas, no existiendo en la mayoría de los casos.

	DBpedia	**BNE**	**KB**	**BL**	**OSZK**
escritor					
tipo	dbpedia-owl:Writer	ifla-frbr:C1005	foaf:Person	foaf:Person	foaf:Person
nombre entero	dbpprop:name	ifla-frbr:P3039	foaf:name	rdfs:label	foaf:name
nombre de pila	foaf:givenName	-	-	foaf:givenName	foaf:givenName
apellido	foaf:surname	-	-	foaf:familyName	foaf:familyName
nacimiento	dbpedia-owl:birthDate	ifla-frbr:P3040	dbpprop:birthYear	-	dbpprop:birthYear
fallecimiento	dbpedia-owl:deathDate		dbpprop:deathYear		dbpprop:deathYear
obras					
tipo	dbpedia-owl:WrittenWork	ifla-frbr:C1001	bibo:Book	bibo:Book	bibo:Document
título	foaf:name	ifla-frbr:P3001	dc:title	dcterms:title	dcterms:title
escritor	dbpedia-owl:author	ifla-frbr:P2009	dc:creator	dcterms:creator	dcterms:creator

Cuadro 3.2: Parte de la ontología utilizada por cada depósito de datos para representar escritores y obras

Ante esta situación, se planteó la posibilidad de definir manualmente esas relaciones para su posterior tratamiento automatizado por parte del sistema. Aun así, eso conlleva nuevos problemas ya que, como puede observarse de nuevo en el cuadro 3.2, hay términos que se utilizan con diferente sentido semántico en diferentes depósitos. Tal es el caso, por ejemplo, de *foaf:name*, que en dos depósitos (KB y OSZK) se utiliza para definir el nombre completo de la persona, mientras que en la DBpedia se usa para el título de las obras. Por lo tanto, definir manualmente relaciones sobre términos de ontologías que se han utilizado sin tener en cuenta este tipo de cuestiones añaden nuevos problemas y complejidades al escenario que se está tratando.

Es por ello que pronto se vio que la opción más conveniente sería definir unos *mappings* para cada depósito de datos en los que se indiquen qué término de qué ontología se utiliza para cada uno de los significados semánticos de los que hará uso la infraestructura.

Por otro lado, a la hora de definir los términos a utilizar por el depósito de datos local que se iba a construir, hubo que aplicar todo este conocimiento para tomar en cada caso la elección más coherente y correcta. El resultado se muestra en el cuadro 3.3.

	litτera
escritor	
tipo	dbpedia-owl:Writer
nombre entero	foaf:name
nombre de pila	foaf:givenName
apellido	foaf:familyName
nacimiento	dbpprop:birthYear
fallecimiento	dbpprop:deathYear
obras	
tipo	dbpedia-owl:WrittenWork
título	dcterms:title
escritor	dcterms:creator

Cuadro 3.3: Ontología utilizada por litτera

Para definir el nombre de pila y el apellido se utilizaron los términos *givenName* y *familyName* de FOAF[9] debido a la recomendación de su propia especificación, que prefiere dichos términos frente a otros más arcaicos como *surname*, *firstName* o *lastName*.

Algo similar ocurrió con la disyuntiva entre utilizar vocabulario de DC o DCTERMS para especificar el título y escritor de las obras, ya que en los depósitos analizados se utilizaban ambas sin aparente diferencia. Aun así, de acuerdo con su propia documentación[10], DC data de antes de la aparición de RDF y por tanto no está bien definida respecto al estándar, mientras que DCTERMS se creó precisamente para resolver estos problemas. DC, por tanto, pervive únicamente para mantener la retrocompatibilidad con los datos antiguos ya definidos y no debería de utilizarse en nuevas definiciones. Es por ello que se optó por DCTERMS.

No hay muchos más comentarios que hacer con respecto a la elección del resto de términos, a excepción del tipo de obras, que en la tabla se ha especificado el más general de todos aunque, como se comentará más adelante, se ha definido una jerarquía propia para poder aplicar algoritmos de una complejidad mayor.

3.4. Arquitectura

A grandes rasgos, el sistema puede dividirse en los cuatro grandes módulos que se muestran en la figura 3.7.

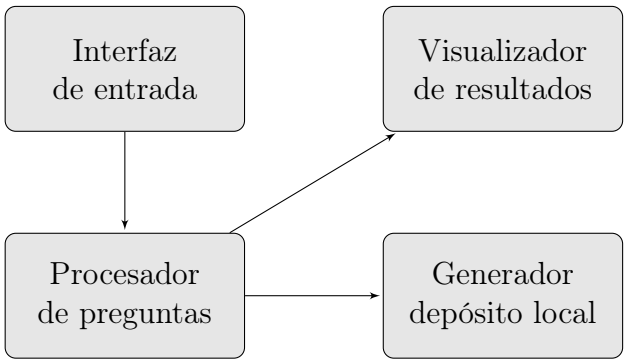


Figura 3.7: Módulos que forman la aplicación

El módulo *Procesador de preguntas*, cuenta a su vez con una arquitectura basada en el modelo *Mediator-Wrapper*. En dicho modelo, se define un *mediator*

que se encarga de gestionar la lógica de negocio, y un número indeterminado de *wrappers* (tantos como recursos diferentes a los que se quiera acceder), que se especializan en el acceso a cada uno de los distintos recursos.

En este caso, sin embargo, se deseaba dotar al sistema de una forma dinámica de añadir y eliminar depósitos de datos, y la creación de un nuevo *wrapper* por cada depósito se antojaba demasiado compleja y engorrosa de cara a la administración futura del sistema. Por ello, lo que finalmente se planteó y llevó a cabo fue un modelo en el que los *wrappers* de cada depósito son virtuales o implícitos y se agrupan en dos categorías (depósitos categorizados y no categorizados, en referencia a si tienen jerarquizadas o no las obras que almacenan).

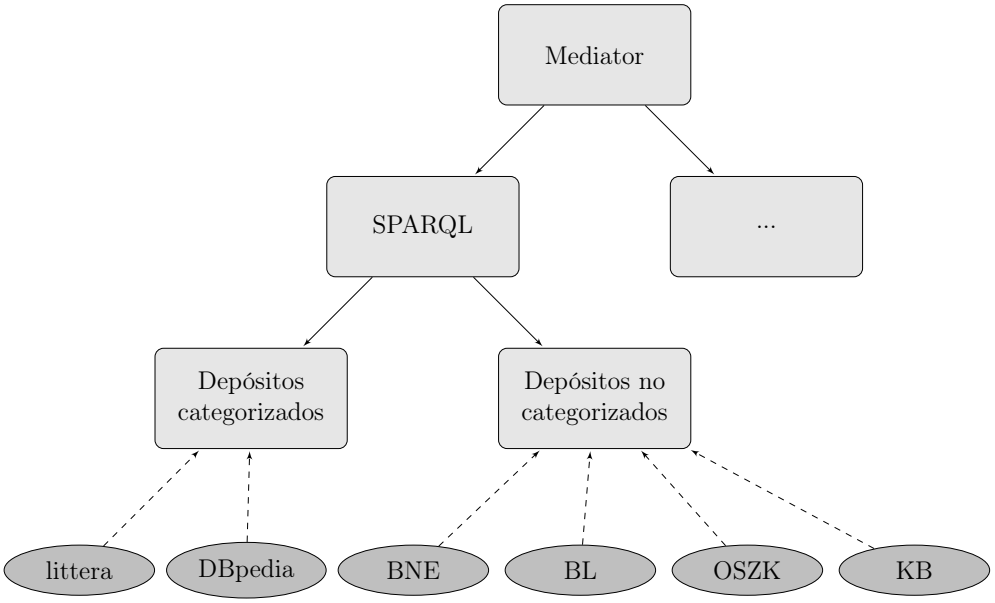


Figura 3.8: Adaptación de la arquitectura *Mediator-Wrapper* utilizada en `litτera`

Este modelo, que se representa gráficamente en la figura 3.8, proporciona lo mejor de los dos mundos ya que por un lado aisla completamente las especificidades de SPARQL, permitiendo de forma muy sencilla programar nuevos *wrappers* que accedan a depósitos almacenados en cualquier plataforma (depósitos accesibles mediante lenguajes de consulta propios y diferentes a SPARQL, bases de datos relacionales, etc.); y por otro da la opción de añadir y eliminar depósitos accesibles a través de *SPARQL endpoints* de forma sencilla, ya que el sistema se encarga de generar dinámicamente el *wrapper* correspondiente a cada depósito, evitando que el administrador tenga que programar nada.

3.5. Tecnología

A continuación se enumeran y describen los diferentes lenguajes de programación, bibliotecas, *frameworks*, programas y herramientas que se han utilizado a lo largo de la vida del proyecto. En los casos en los que existen alternativas, se incluye una reflexión y justificación de los motivos que llevaron a la elección en cuestión.

3.5.1. PHP

PHP[58] (del inglés *PHP Hypertext Pre-processor*) es un lenguaje de programación interpretado diseñado para el desarrollo web de contenido dinámico. Su sintaxis está influenciada por C aunque, a diferencia de este, su tipado de datos es dinámico. Asimismo, a partir de la versión 5.0 cuenta con las características necesarias para considerarlo orientado a objetos.

PHP está ampliamente extendido, es multiplataforma y viene instalado en prácticamente cualquier servidor web, por lo que lo se convertía de entrada en una buena opción para lograr la interoperabilidad de la aplicación a desarrollar. Por otro lado, ser un lenguaje interpretado (lo cual permite un prototipado y desarrollo final muy fluido), ser software libre, y la experiencia previa del desarrollador con este lenguaje, hicieron que la balanza se inclinase claramente del lado de PHP frente a otras alternativas como Java.

libcurl

libcurl[18] es una biblioteca de PHP que permite conectarse y comunicarse con diferentes tipos de servidores y protocolos (http, https, ftp, gopher, telnet, dict, file y ldap). También admite certificados HTTPS, HTTP, POST, HTTP PUT, subidas mediante FTP, subidas basadas en formularios HTTP, proxies, cookies, y autenticación mediante usuario y contraseña.

Es, por tanto, una herramienta perfecta para realizar la comunicación entre `litτera` y los depósitos de datos externos a los que se quiere conectar.

SimpleTest

SimpleTest[51] es el *framework* utilizado para la realización de pruebas unitarias (*unit tests*) en PHP.

Aun siendo menos conocido que PHPUnit[55], se prefirió sobre este porque, a pesar de ser simple, cuenta con características avanzadas como la simulación de objetos o la automatización de procesos como la pulsación sobre enlaces o el envío de formularios que resultan interesantes a la hora de probar los elementos dinámicos de los distintos módulos implementados.

3.5.2. Apache

Apache[71] es un servidor web HTTP de código abierto ampliamente conocido y utilizado. Es multiplataforma, aunque es especialmente utilizado en sistemas de tipo Unix. Cuenta, por supuesto, con integración con PHP y, de hecho, el trío Apache+PHP+MySQL es la tecnología más utilizada para el desarrollo de páginas web dinámicas. Sin embargo, en el caso que nos ocupa prescindimos de MySQL, ya que la aplicación no requiere de ninguna base de datos relacional.

3.5.3. HTML5 y CSS3

HTML (del inglés *HyperText Markup Language*) es el lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir y traducir la estructura y la información en forma de texto, así como para complementarla con elementos multimedia; mientras que CSS (del inglés *Cascading Style Sheets*) es el lenguaje de hojas de estilo usado para describir su presentación semántica (aspecto y formato).

HTML5 y CSS3 son las últimas versiones de estos dos lenguajes que, si bien no son aún estándares estables, todos los navegadores modernos implementan en mayor o menor medida las ventajas, facilidades y mejoras que aportan estas nuevas versiones.

Bootstrap

Bootstrap[50] es un *framework front-end* desarrollado y liberado por Twitter que permite agilizar y profesionalizar la creación de interfaces web con CSS y JavaScript. Los diseños creados con esta herramienta son simples, limpios e intuitivos, y soporta nativamente la adaptación de la interfaz al tamaño del dispositivo con el que se está visualizando (*responsive design*).

Existen otras alternativas como Foundation[52], HTML5 Boilerplate[53] o Blueprint[49]. Sin embargo, se ha preferido Bootstrap por el estilo por defecto que soporta (mucho más atractivo visualmente que el resto de opciones), la compatibilidad (soporte para Internet Explorer 8 que, por ejemplo, Foundation 4 no tiene), su fama (que hace que exista una comunidad que crea elementos complementarios muy interesantes, como el que se expone seguidamente) y su facilidad de uso y nula curva de aprendizaje.

Bootstrap switch

Bootstrap switch[19] es una extensión para Bootstrap que permite mostrar las casillas de verificación (*checkboxes*) de una manera visualmente atractiva, como si se tratase de un interruptor que se apaga o se enciende al desplazarse hacia los lados.

3.5.4. JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas mediante la manipulación del DOM (*Document Object Model*). Se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

JSON

JSON[20] (del inglés *JavaScript Object Notation*) es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML, convirtiéndolo en un formato ligero para el intercambio de datos. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX.

Se ha decidido utilizar este formato para almacenar la configuración de los depósito de datos porque, tal y como se puede ver en el ejemplo de la figura 3.9, su sintaxis es muy simple y fácilmente entendible, haciendo que la definición de nuevos depósitos no suponga ningún problema extra. Asimismo, muchos lenguajes de programación incluyen nativamente métodos para tratar datos en JSON (como es el caso de PHP a partir de su versión 5.2), con lo que la elección también supuso facilidades a la hora de procesar y tratar los datos en cuestión.

```
1  {
2      "name": "DBpedia",
3      "endpoint": "http://dbpedia.org/sparql",
4      "vocab": [
5          { "type": "type_author", "word": "dbpedia-owl:Writer" },
6          { "type": "author_name", "word": "dbpprop:name" }
7      ],
8      "pref": [
9          { "prefix": "dbpprop", "iri": "http://dbpedia.org/
               property/" },
10         { "prefix": "dbpedia-owl", "iri": "http://dbpedia.org/
               ontology/" }
11     ]
12 }
```

Figura 3.9: Ejemplo de la sintaxis de JSON

jQuery

jQuery[54] es una biblioteca de JavaScript que simplifica la manera de tratar dinámicamente los documentos HTML, manipular el árbol DOM, manejar eventos, ejecutar animaciones e interactuar asíncronamente con el servidor web mediante AJAX.

A lo largo de los últimos años, jQuery ha ido ganando popularidad hasta convertirse en un estándar de facto. Existe un gran ecosistema de *plugins* y complementos para esta biblioteca, así como gran cantidad de información, documentación y tutoriales que hacen secillo su aprendizaje.

tablesorter

tablesorter[54] es un *plugin* para jQuery que permite ordenar cuadros dinámicamente por cada una de sus columnas. La versión utilizada es un *fork* del original de 2008[21] que aporta numerosas mejoras y, sobre todo, corrección de errores.

jQuery UI

jQuery UI es una biblioteca de componentes para jQuery al que le añaden un conjunto de *plugins*, *widgets* y efectos visuales para la creación de aplicaciones web.

Dichos componentes son independientes entre sí y, en caso de no necesitar alguno de ellos, puede evitarse su carga. Así, en el proyecto el único componente que se carga y se utiliza es el de efectos, que proporciona efectos visuales tales como el resaltado de elementos o las transiciones de color.

3.5.5. OpenLink Virtuoso

OpenLink Virtuoso[43] es la edición libre (licencia GPLv2[27]) de Virtuoso Universal Server[45], un *servidor universal* que combina funcionalidades de distintos sistemas de gestión de bases de datos. De todas las funcionalidades que ofrece, en este proyecto se utilizan únicamente aquellas relacionadas con el campo de los datos enlazados.

Destacan las siguientes características:

Depósito de triples ofrece uno de los sistemas más conocidos y utilizados en este área debido a sus tiempos de respuesta, que están muy por encima del resto de alternativas

Lenguaje de consulta permite realizar consultas en SPARQL 1.0, e insertar, eliminar y actualizar los triples en SPARUL

SPARQL endpoint genera automáticamente una interfaz web pública que permite a cualquiera realizar consultas SPARQL al depósito local

WebDAV

WebDAV (del inglés *Web-based Distributed Authoring and Versioning*) es un conjunto de extensiones del protocolo HTTP que hacen posible la edición y gestión colaborativa de archivos en servidores remotos.

Virtuoso ofrece numerosas APIs y métodos para insertar datos en el depósito de triples. Por ejemplo, se le pueden dar permisos de escritura al *SPARQL endpoint* y utilizar para la inserción de datos el mismo método que para su consulta. Sin embargo, esto no resulta ser una buena idea teniendo en cuenta que por su naturaleza los *SPARQL endpoints* son públicos, por lo que abordar el problema de esa forma conllevaría que cualquier persona pudiese vaciar el depósito sin más. Otra opción son las conocidas APIs Jena[6] y Sesame[7], que presentan dos problemas: por un lado están hechas para el lenguaje de programación Java, cuando ya se ha tomado la decisión de utilizar PHP como tecnología de desarrollo, y por otro son tan completas y potentes como complejas de utilizar, siendo una solución desmedida si lo único que se pretende es insertar unas decenas de triples con cada petición. Llegamos, pues, a una tercera opción: WebDAV.

WebDAV permite el envío de datos mediante peticiones HTTP, lo cual garantiza la compatibilidad con cualquier lenguaje de programación que soporte la biblioteca libcurl (requisito ya existente para la realización de consultas a depósitos de datos externos, como se ha comentado anteriormente). Además, garantiza la autentificación mediante usuario y contraseña, evitando el escollo por el que se descartó la inserción mediante el *SPARQL endpoint*. De esa forma, lo único que habría que hacer es la petición HTTP para crear un archivo con los datos a insertar, y a continuación ordenar a Virtuoso su carga. Sin embargo, Virtuoso dispone de un directorio especial llamado *rdf_sink* que permite ahorrar este segundo paso, ya que cualquier archivo depositado en él se carga automáticamente sin necesidad de ordenarlo explícitamente.

Así pues, se consideró que la forma óptima en este caso para insertar datos en Virtuoso es hacerlo mediante WebDAV, ejecutando la orden cURL que aparece en la primera línea de la figura 3.10 (sustituyendo, obviamente,

```
1 curl -d "INSERT { ...triples... }" -u "test:123456" -H "Content-
    Type: application/sparql-query" http://localhost:8890/DAV/
    home/test/rdf_sink/xx
2 curl -X DELETE -u "test:123456" http://localhost:8890/DAV/home/
    test/rdf_sink/xx
```

Figura 3.10: Inserción de triples en Virtuoso mediante WebDAV

`...triples...` por una lista formada por un número indeterminado de triples, y `test` y `123456` por los datos del usuario creado siguiendo lo expuesto en el anexo B.2.1). Seguidamente conviene ejecutar la orden que se muestra en la segunda línea de la misma figura, ya que permite a Virtuoso liberar espacio innecesario.

3.5.6. Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de un producto. Aunque esta tarea puede realizarse de forma manual, es interesante el uso de sistemas de control de versiones (SVC), que son las herramientas que facilitan esta gestión almacenando los elementos y guardando un registro histórico de los cambios efectuados para poder, si fuese necesario, revertir los cambios, o desarrollar diferentes ramas del producto en paralelo.

git

Git[73] es un moderno software de control de versiones diseñado por Linus Torvalds que pone especial atención en el soporte a desarrollos no lineales. A diferencia de otros sistemas, la creación de nuevas ramas y su reunificación son operaciones muy ágiles, con lo que se promueve su uso.

Existen otras alternativas como Concurrent Versions System[72], Apache Subversion[75] o Mercurial[74], siendo las dos primeras utilizadas principalmente en desarrollos que heredan sistemas antiguos, nada propicias si no existen ese tipo de ataduras. La elección Git frente a Mercurial, sin embargo, vino condicionada por la popularidad de Git y su uso en el entorno.

Bitbucket

Bitbucket es un servicio de alojamiento basado en web para proyectos que utilicen como sistema de control de versiones Mercurial o Git.

Se prefirió Bitbucket[69] frente a GitHub[70] porque este último solo permite de forma gratuita crear repositorios públicos y visibles por todo el mundo, mientras que Bitbucket permite la creación de cuantos repositorios privados se desee, siempre que no pertenezcan más de cinco usuarios a ellos.

3.5.7. Herramientas de desarrollo

Geany

Geany es un editor de texto pequeño y ligero con características básicas de los entornos de desarrollo integrado (IDEs) como el coloreado, plegado y autocompletado del código.

Cuenta, además, con diferentes plugins; entre ellos GeanyVC, que proporciona una interfaz gráfica integrada en el editor para acceder a las funciones más frecuentes del sistema de control de versiones.

La elección de Geany en vez de entornos de desarrollo integrado completos como Eclipse[56] o NetBeans[57] se dio porque, debido a la baja potencia de procesamiento del equipo desde el que iba a trabajar, se decidió primar la velocidad del editor frente a las características y funcionalidades avanzadas que proporcionaban dichos IDEs.

Herramientas para desarrolladores de Google Chrome

Las *Herramientas para desarrolladores* son un conjunto de utilidades integradas en el navegador web Google Chrome que permiten, entre otras cosas, ver el estado del árbol DOM en cada momento, ver los errores de JavaScript y ejecutar sentencias desde su consola, o visualizar diferentes estadísticas y datos que permiten tener constancia del desempeño de la web a lo largo de la interacción ella, con el objetivo de depurarla.

3.5.8. Documentación

LaTeX

LaTeX es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. Su principal ventaja frente a procesadores de texto como *Microsoft Word* o *LibreOffice Writer* es que separa nítidamente el contenido del documento de su formato, animando así a crear documentos bien estructurados a los que luego se les puede aplicar fácilmente en bloque las propiedades que se deseen.

Entre otras cosas, cuenta con la numeración automática de figuras y cuadros, confección de índices, referencias cruzadas, gestión de la bibliografía y optimización del documento PDF resultante. Hace, por tanto, que los documentos de varias decenas de páginas sean mucho más manejables, tanto a la hora de escribirlos (facilidades en la numeración y las referencias), como a la de consumirlos (estructura, metadatos y enlaces).

Una vez tomada la decisión de redactar el presente documento en LaTeX, se tomó como referencia la plantilla elaborada por Iñaki Silanes en 2007 y publicada en el sitio web del grupo ITSAS[16]. Sin embargo, pronto se apreciaron pequeños errores y carencias en la misma, consecuencia natural de llevar cinco años sin actualizarse. Tras realizar, a lo largo del proceso de redacción del documento, numerosos cambios en la plantilla, se terminó sin apenas nada del código fuente original, por lo que en términos prácticos podría considerarse que se ha utilizado una plantilla propia.

ShareLaTeX

ShareLaTeX[1] en un editor de LaTeX en la nube que facilita la edición de documentos en este lenguaje. Al acceder a él a través del navegador web, permite trabajar sobre los documentos en cualquier momento y desde cualquier dispositivo con conexión a Internet, con independencia de la plataforma.

Adicionalmente, dispone de las siguientes características:

- Visualización simultánea del código fuente del documento y el PDF resultante.
- Soporte para la modularización y un amplio número de paquetes.
- Repetidas compilaciones automáticas internas para resolver las referencias y generar los índices y la bibliografía, haciendo que el usuario solo tenga que pedir su compilación una sola vez para obtener el documento completo; todo esto sin comprometer la velocidad de compilación.
- Ocultación de los archivos auxiliares generados automáticamente y procesamiento de los *logs* para mostrarlos de manera simple y enlazada con el código fuente.
- Coloreado de sintaxis y corrector ortográfico en más de sesenta idiomas (entre ellos castellano, euskera e inglés).
- Opción de compartir el documento y colaborar en su redacción con otro usuario.

Se consideró que todo esto lo convierte en una alternativa superior a entornos de escritorio convencionales como pudieran ser Kile[46] o TeXworks[48].

Capítulo 4

Desarrollo del sistema

En este capítulo se exponen los casos de uso planteados, algunos de los algoritmos utilizados y el modelo de clases confeccionado para el desarrollo del sistema.

4.1. Casos de uso

El diagrama de casos de uso de la figura 4.1 sintetiza la interacción de los diferentes actores (usuario final y administrador) con el sistema. Seguidamente se detallan las características y el flujo de eventos de los que consta cada uno.

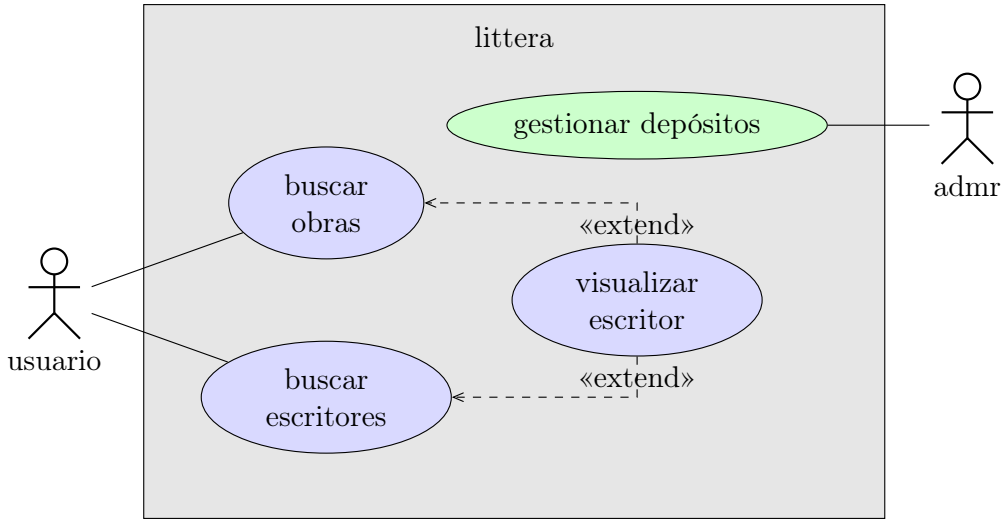


Figura 4.1: Diagrama de casos de uso

4.1.1. Buscar escritores

Al usuario se le presenta un formulario (figura 4.2) en el que puede ir definiendo grupos y parámetros de búsqueda hasta confeccionar una consulta tan compleja como desee.

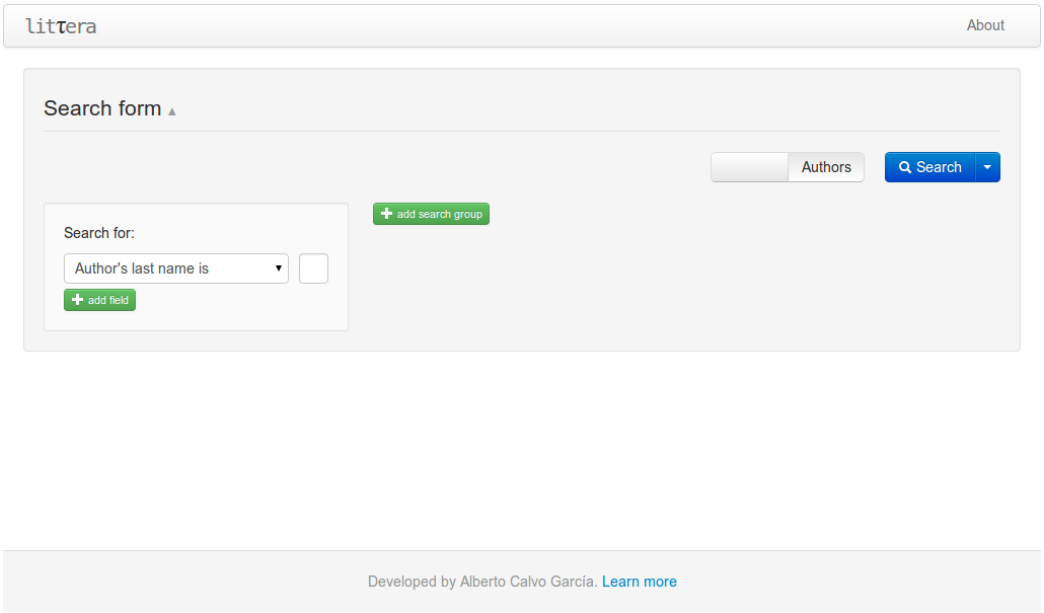


Figura 4.2: Página principal de `litτera`

Para cada grupo de búsqueda se puede elegir si lo que se desea es que los resultados que se muestren cumplan todas y cada una de las condiciones definidas en dicho grupo (*match ALL terms*), o lo que se quiere es que cumplan al menos una (*match ANY term*). Asimismo, se puede seleccionar entre el cumplimiento de al menos un grupo de búsqueda (*match ANY group*) o de todos ellos (*match ALL groups*).

Por ejemplo, supongamos que estamos buscando a *Cervantes* pero no recordamos su nombre ni su segundo apellido. Dependiendo el depósito de datos sobre el que hagamos esa búsqueda tan genérica, podemos obtener cientos de resultados (concretamente 214 en el caso de la Biblioteca Nacional de España), lo cual dificulta encontrar al autor que estamos buscando. Sin embargo, en el caso de que sí que recordemos su novela más importante, *el Quijote*, podemos refinar la búsqueda indicando a `litτera` que encuentre aquellos escritores llamados *Cervantes* que tengan un libro que contenga la palabra *Quijote*.

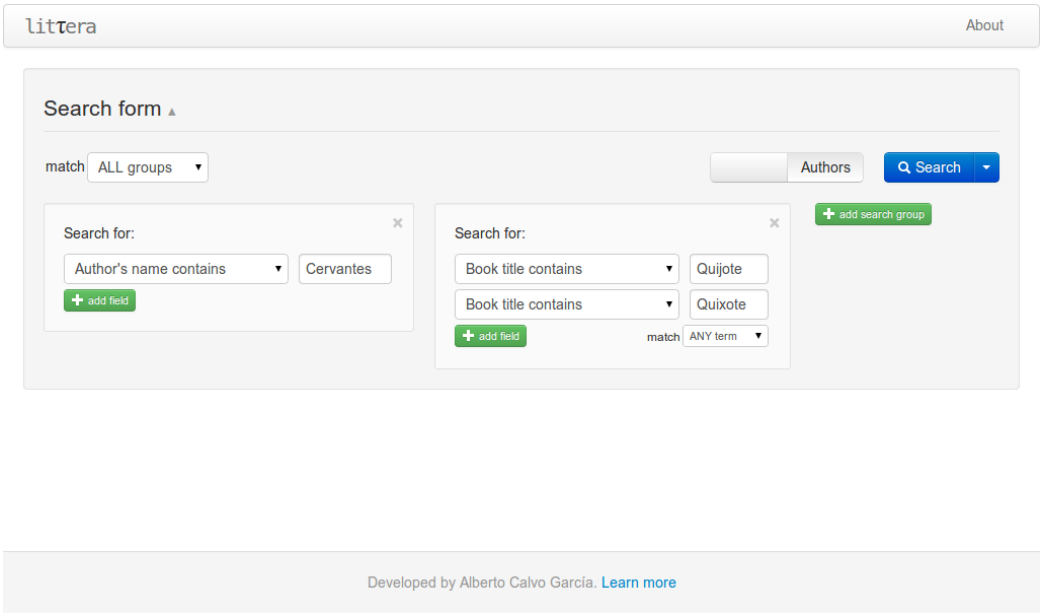


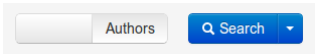
Figura 4.3: Formulario de búsqueda relleno con una consulta de ejemplo

Esta última búsqueda, sin embargo, puede que no devuelva los resultados esperados en todos los depósitos de datos, ya que muchos de ellos, al ser internacionales, almacenan sus datos en inglés. En vez de hacer dos consultas, la primera con *Quijote* (en castellano), y la segunda con *Quixote*, en inglés, se puede aprovechar la funcionalidad de `littera` y escribir una búsqueda con dos grupos, el primero de ellos que contenga un solo campo de búsqueda indicando que el nombre del autor debe contener la palabra *Cervantes*, y el segundo indicando que debe tener un libro que se contenga o bien la palabra *Quijote*, o bien *Quixote*, y que lo que se desea es que se cumplan las condiciones de ambos grupos de búsqueda. Dicha búsqueda quedaría tal y como se muestra en la figura 4.3 y, frente a los 214 resultados obtenidos en la Biblioteca Nacional de España en caso de buscar únicamente por *Cervantes*, en este caso se obtendría un único resultado: Miguel de Cervantes Saavedra.

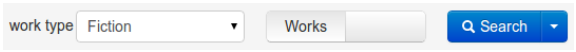
Cabe destacar que las búsquedas no están restringidas a un solo escritor. Así, se pueden definir búsquedas como *autores que se apelliden Cervantes o Shakespeare*, *autores que se apelliden Brown o que hayan escrito sobre Da Vinci*, etc.

Una vez terminado de definir los criterios de búsqueda, se puede seleccionar dónde ejecutarla: en el depósito propio de `littera` (si lo hubiera), o en cualquier otro depósito de datos de entre los definidos. Esto se realiza a través del botón de la parte superior derecha del formulario (figura 4.4.a),

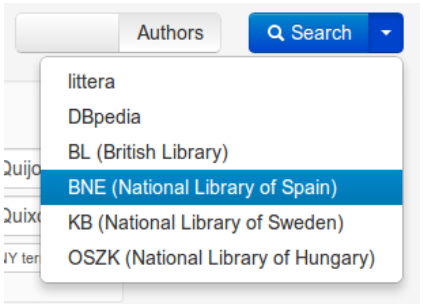
que está acompañado de un desplegable en el que se muestran todos los depósitos de datos definidos (figura 4.4.c). En caso de seleccionar uno de ellos, la consulta se enviará a dicho depósito, mientras que si no se selecciona ninguno (pulsando directamente sobre el botón *Search*), la consulta se realiza en el depósito local en caso de que este esté definido, y si no lo está se selecciona uno al azar de entre los disponibles.



(a) vista por defecto



(b) búsqueda de obras



(c) lista de depósitos

Figura 4.4: Detalles del formulario de búsqueda

En caso de haber dejado campos en el formulario sin rellenar, se obligará a hacerlo antes de poder enviar los datos. Si no, se continuará con el flujo normal de eventos.

Una vez enviada la petición y obtenida la respuesta del depósito de datos, pueden darse cuatro casos:

1. Que el depósito de datos no contenga ningún escritor que satisfaga el criterio de búsqueda.
2. Que existan hasta cincuenta escritores.
3. Que existan más de cincuenta escritores.
4. Que el depósito de datos no pueda resolver la consulta en un tiempo prudencial.

En el primer caso (figura 4.5) se mostrará un aviso indicando lo sucedido y dando al usuario dos opciones: o bien realizar una búsqueda menos restrictiva, o bien hacer exactamente la misma consulta pero en otro depósito de datos diferente.

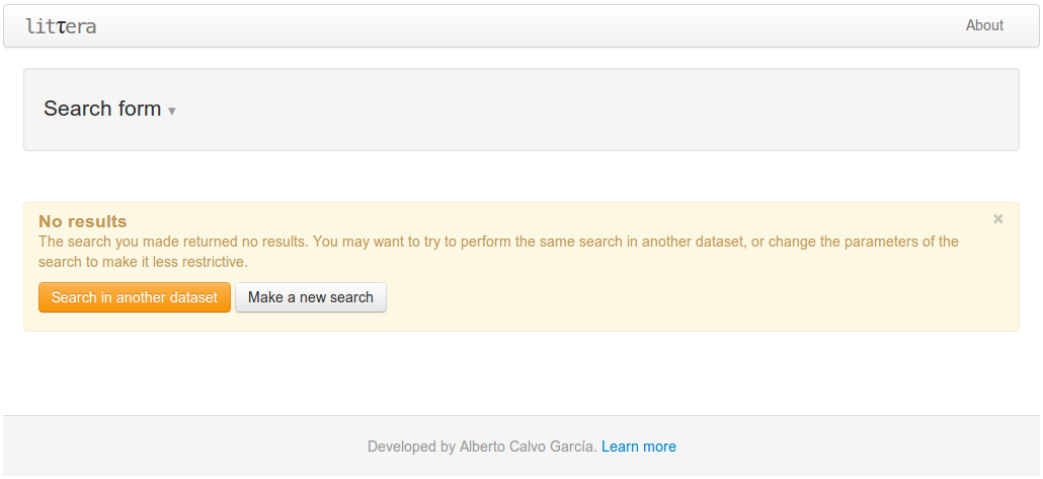
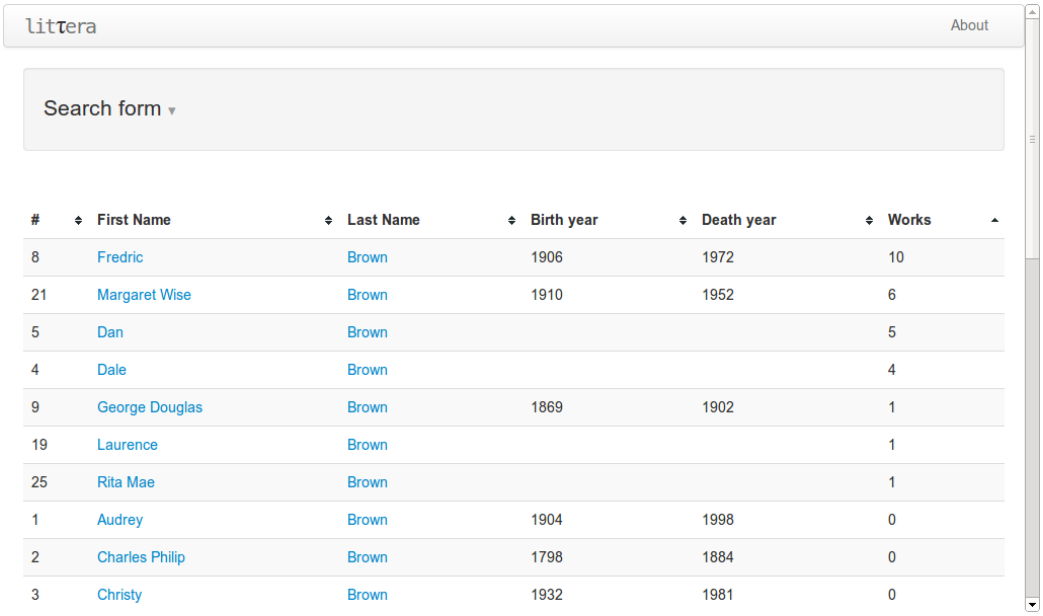


Figura 4.5: Vista en caso de que el depósito no devuelva ningún resultado



#	First Name	Last Name	Birth year	Death year	Works
8	Fredric	Brown	1906	1972	10
21	Margaret Wise	Brown	1910	1952	6
5	Dan	Brown			5
4	Dale	Brown			4
9	George Douglas	Brown	1869	1902	1
19	Laurence	Brown			1
25	Rita Mae	Brown			1
1	Audrey	Brown	1904	1998	0
2	Charles Philip	Brown	1798	1884	0
3	Christy	Brown	1932	1981	0

Figura 4.6: Vista de hasta cincuenta escritores

En el segundo caso (figura 4.6) se mostrará un cuadro con los siguientes datos de cada escritor: nombre de pila, apellido(s), año de nacimiento y fallecimiento (si fuese el caso), y número de obras de él que hay registradas en el depósito. El cuadro se ordena automáticamente por este último campo, aunque el usuario puede ordenarlos dinámicamente por el que quiera pulsando sobre el título de la columna. Pinchando sobre cualquiera de las filas del cuadro se accede a la página de visualización del escritor que se detalla más

adelante.

En el tercer caso, aquel en el que se muestran más de cincuenta autores, el comportamiento será igual que en el caso anterior excepto por la última columna (la relativa a las obras de cada autor), que no se mostrará (figura 4.7). Los resultados no vendrán ordenados por ningún campo, aunque el usuario puede seguir haciéndolo de la misma forma.

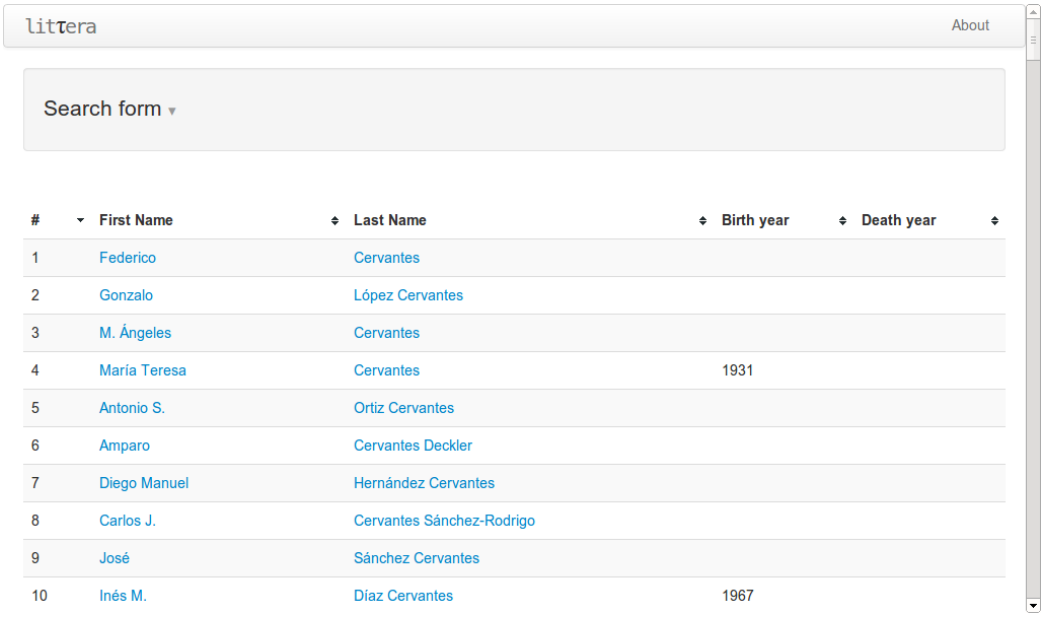


Figura 4.7: Vista de más de cincuenta escritores

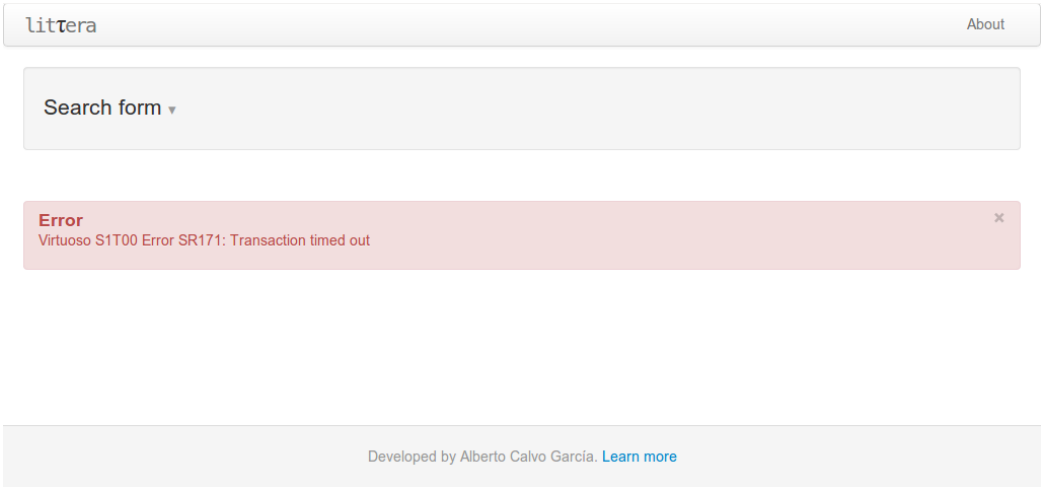


Figura 4.8: Vista en caso de que el depósito no pueda procesar la consulta

En el último caso, si el depósito de datos por el motivo que fuera no pudiese procesar la consulta recibida (bien porque esté fuera de servicio o porque sea demasiado compleja), mostrará un mensaje de error (figura 4.8) y el usuario podrá volver al formulario de búsqueda para realizar una nueva consulta en el mismo o en otro depósito de datos.

4.1.2. Buscar obras

Al usuario se le presentará exactamente el mismo formulario que para buscar escritores. Sin embargo, activando la opción correspondiente podrá seleccionar la búsqueda obras en vez de la de escritores.

Al hacerlo, aparecerá un nuevo desplegable en el que poder seleccionar el tipo de obra que se desea buscar (figura 4.4.b). Dichos tipos estarán ordenados alfabéticamente, mezclándose las diferentes ontologías y sin hacer distinción de a cuál pertenece cada uno de los tipos.

El usuario siempre recibirá el resultado que más se ajuste a su consulta; pero en este punto pueden distinguirse claramente tres casos:

1. Que el usuario seleccione un depósito que no tiene categorizadas las obras.
2. Que el usuario seleccione un tipo de obra que contempla la ontología utilizada por el depósito seleccionado.
3. Que el usuario seleccione un tipo de obra que contempla una ontología de otro depósito de datos.

En el primer caso se obviará la selección del usuario y se mostrarán todas las obras almacenadas en el depósito de datos que cumplan los criterios de búsqueda.

En el segundo se mostrarán las obras que sean del tipo seleccionado y además cumplan el criterio de búsqueda.

En el tercero, el más interesante, se aplicará el algoritmo descrito en la sección 4.2.1 para transformar el término en una unión y/o intersección de términos contemplados en la ontología del depósito en el que se está realizando la búsqueda.

Una vez construida, enviada la consulta y obtenida la respuesta del depósito de datos, podrán darse tres casos, todos ellos análogos a los del caso de uso anterior:

1. Que el depósito de datos no contenga ninguna obra que satisfaga el criterio de búsqueda.
2. Que exista al menos una obra.
3. Que el depósito de datos no pueda resolver la consulta en un tiempo prudencial.

El primer y último caso son idénticos a lo que ocurre al buscar escritores (y que puede verse gráficamente en las figuras 4.5 y 4.8 respectivamente). Mientras que en el segundo se muestra un cuadro con el nombre de la obra y el nombre de pila y apellido(s) del autor (figura 4.9). Si se pulsa sobre el nombre del autor, se accede a la página de visualización del mismo.

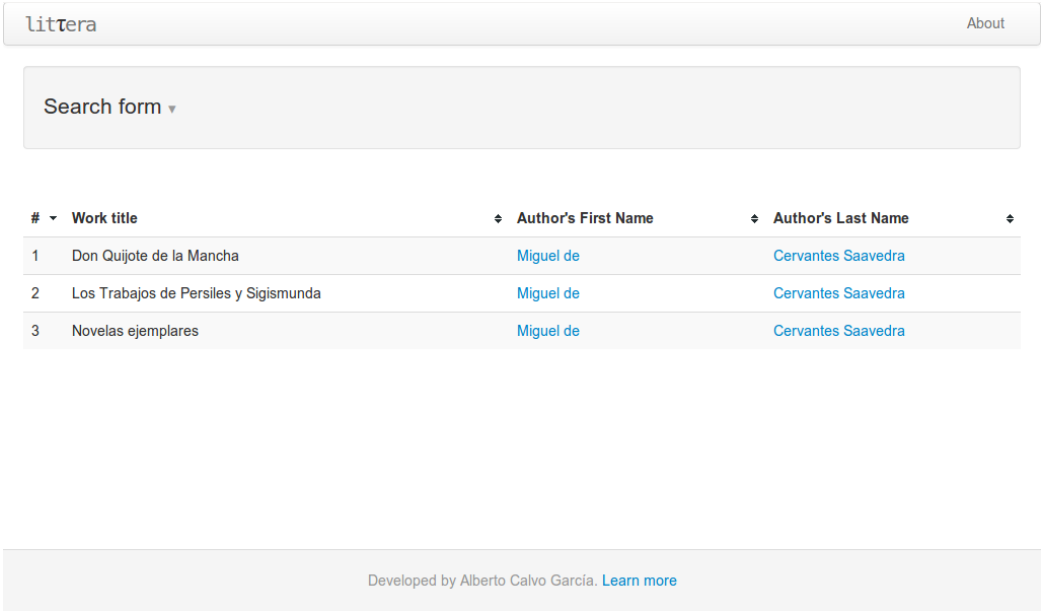


Figura 4.9: Vista de obras que cumplen los criterios de búsqueda

4.1.3. Visualizar escritor

Desde la lista de resultados de cualquiera de los dos casos de uso anteriores podrá accederse a la visualización del escritor, que constará principalmente de cuatro elementos diferenciados (figura 4.10):

1. Barra de pestañas, en la que la primera lleva el nombre del depósito de datos en uso, y la siguiente se llama *Search in other datasets*.

2. Información sobre la página, incluyendo el nombre completo del escritor que se está visualizando.
3. Botón verde nombrado *Add to littera's dataset*.
4. Listado de los documentos del escritor en cuestión registrados en el depósito actual.

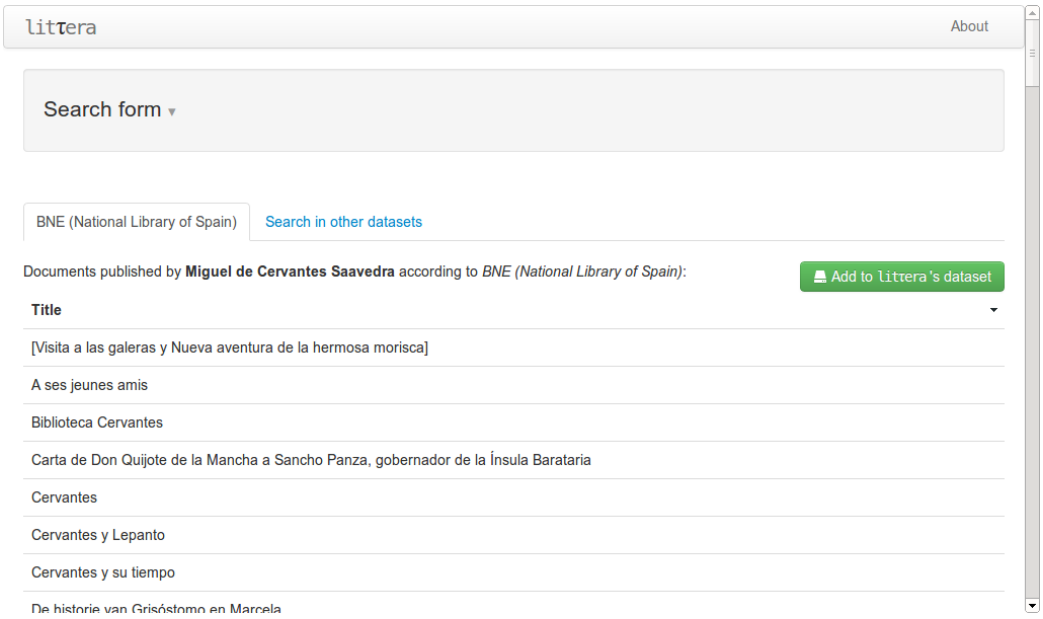


Figura 4.10: Vista de las obras de un escritor

Al pulsar sobre el enlace *Search in other datasets* de la barra de pestañas, la aplicación mostrará en pantalla otros depósitos en los que también existe el autor que se está consultando (figura 4.11). Así se permite alternar fácilmente entre los depósitos para ver qué información tiene almacenada cada uno de ellos sobre el escritor (figura 4.12).

Figura 4.11: Barra de pestañas con los depósitos disponibles

Por otro lado, al pulsar sobre el botón verde *Add to littera's dataset* se accede al formulario de inserción en el depósito local de littera. Al hacerlo, la aplicación toma como datos de partida los del depósito sobre el que se ha solicitado la inserción en littera. Por lo tanto, si se ha hecho sobre la vista de la figura 4.12, la aplicación mostrará la pantalla de la figura 4.13.

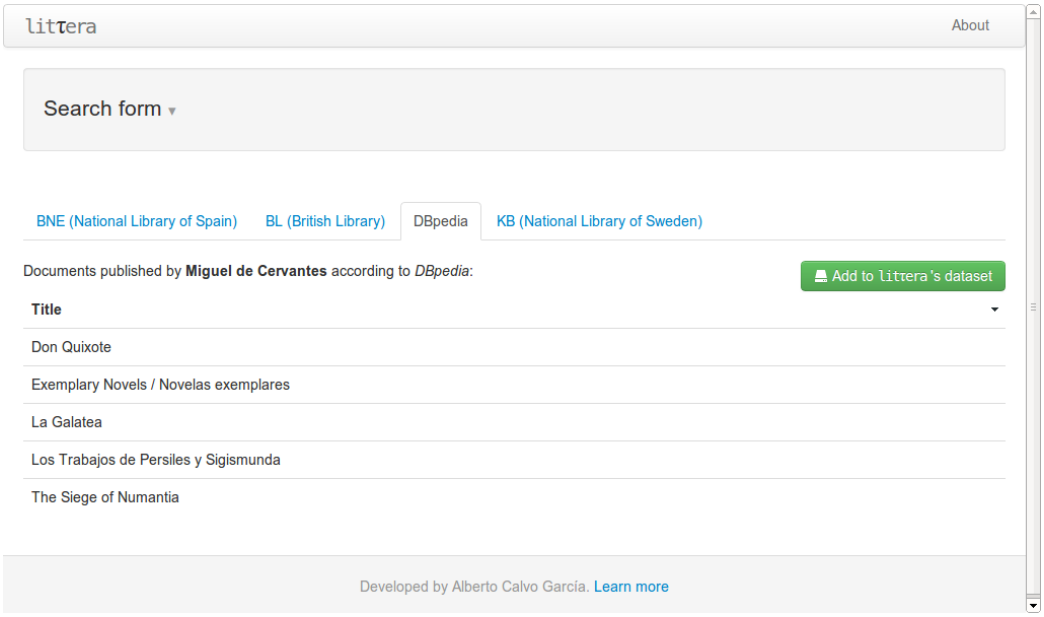


Figura 4.12: Vista del escritor de la figura 4.10 en otro depósito

Dichos datos son completamente configurables, pudiendo:

- Modificar el nombre y los apellidos del escritor, así como su año de nacimiento y fallecimiento.
- Modificar el título de cualquiera de las obras.
- Asignar un tipo a cada obra.
- Borrar obras existentes y añadir otras nuevas.

Una vez terminado de editar, basta con pulsar un botón para grabar los cambios en `littera` (figura 4.14).

La visualización de un escritor alojado en `littera` tendrá la particularidad de contar en la parte inferior con un campo en el que poder añadir nuevas obras, para mantener así el depósito actualizado con las nuevas publicaciones que escriba el autor (figura 4.15).

`littera` permite insertar nuevos contenidos relacionados con los escritores ya existentes en el depósito, con el fin de mantener los datos actualizados. Esto permite que, cuando un autor publica un nuevo trabajo, este pueda ser registrado en el depósito de `littera` fácilmente. Al escribir el título de una obra en dicho campo y pulsar sobre el botón, aparecerá un diálogo modal en el que se confirmará el título introducido, y se dará la opción de seleccionar

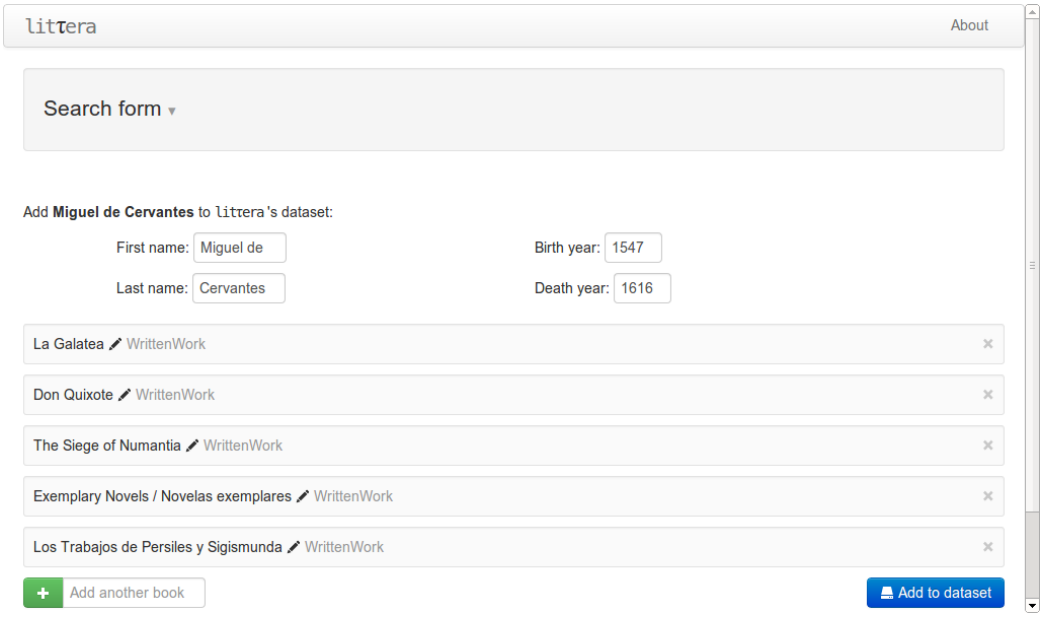


Figura 4.13: Formulario de inserción de un nuevo escritor

Figura 4.14: Aviso de inserción satisfactoria en el depósito local

el tipo de obra del que se trata (figura 4.16).

Figura 4.15: Detalle de la vista de un escritor almacenado en `littera`

Una vez confirmados los datos, se registrarán en el depósito y se mostrará instantáneamente en la lista de obras del escritor.

4.1.4. Gestionar depósitos

El administrador de `littera` podrá, de forma sencilla, retirar depósitos o definir otros nuevos modificando simples archivos de configuración como el de la figura B.8.

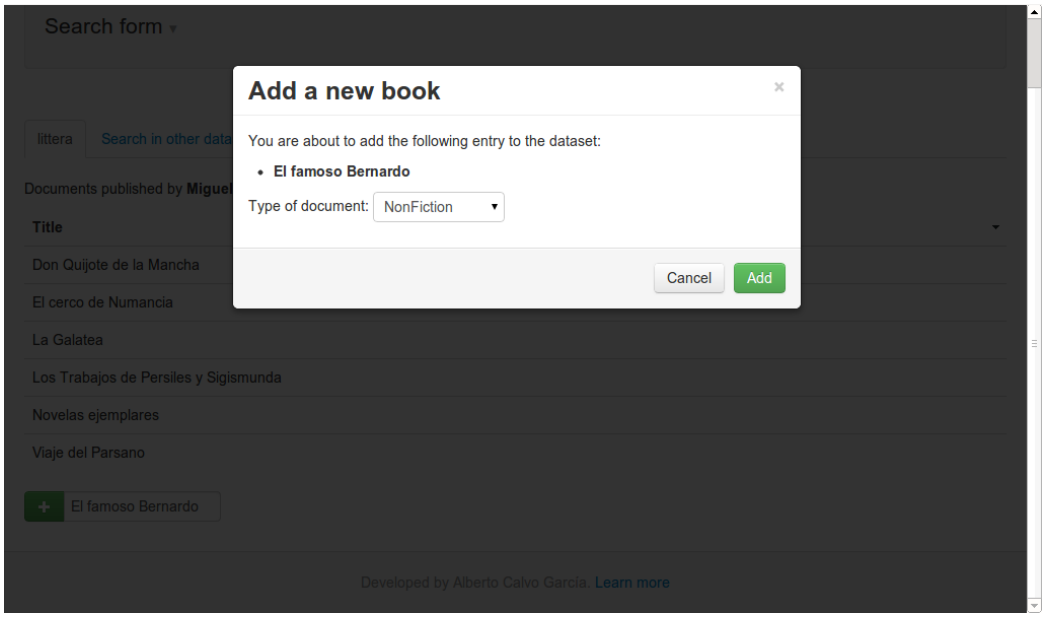


Figura 4.16: Diálogo modal para confirmar la inserción de un libro

4.2. Algoritmos

De entre los algoritmos utilizados para la implementación de `litτera` merecen especial atención dos: el primero de ellos relativo a la reescritura de consultas para la adecuación al depósito destino de la ontología utilizada, y el otro relativo a la búsqueda eficiente de equivalencias entre entidades de distintos depósitos de datos. A continuación se explica más detalladamente la problemática a la que cada uno de ellos hace frente, así como el algoritmo en el que se basa la solución y comentarios generales en torno a ellos.

4.2.1. Reescritura de consultas

Aprovechando que uno de los depósitos de datos del dominio (concretamente *DBpedia*) almacena las obras de forma jerarquizada, se planteó la posibilidad de que el depósito local de `litτera` hiciera lo propio con otra ontología diferente, abriendo así la puerta a que el sistema ofreciera la opción de selección de tipos de obras.

Para hacerlo transparente al usuario, se decidió dar la opción de seleccionar cualquier tipo de entre los existentes, independientemente de la ontología a la que pertenezciera el término. Así se añade un nivel de complejidad muy

```
1  //Returns an adequate query for onto_target,
2  //produced by a rewriting of Q with the least loss of information
3  QUERY_SELECTION(Q, ontoSource, ontoTarget) return Query
4   terms = DecomposeQuery(Q); // terms is the set of terms in Q
5   for each term in terms do
6   {
7     rewritingExpressions = REWRITE(term, ontoSource, ontoTarget);
8     //stores the term together with its adequate rewriting
          expressions
9     termsRewritings.add(term, rewritingExpressions);
10  }
11  //Constructs queries from the expressions obtained for each term
12  possibleQueries = ConstructQuery(Q, termsRewritings);
13  //Selects and returns the query that provides less loss of
       information
14  return LeastLoss(Q, possibleQueries);
15
16
17 //Constructs a queue of adequate expressions for term in
      onto_target
18 REWRITE(term, ontoSource, ontoTarget) return Queue<Expression>
19  resultQueue = new Queue();
20  traverseQueue = new Queue();
21  traverseQueue.add(term);
22  while  not traverseQueue.isEmpty() do
23  {
24    t = traverseQueue.remove();
25    if has_synonym(t, ontoTarget) then
26       resultQueue.add(map(t, ontoTarget));
27    else //t is a conflicting term
28       {         ceiling = directSuper(t);
29                 traverseQueue.enqueueAll(ceiling);
30                 floor = directSub(t);
31                 traverseQueue.enqueueAll(floor);       }
32  }
33  return resultQueue;
```

Figura 4.17: Algoritmo para la reescritura de consultas SPARQL

interesante, ya que en caso de seleccionar un término que no pertenezca a la ontología utilizada por el depósito en cuestión se requieren una serie de transformaciones para obtener una consulta válida.

Para resolverlo se ha utilizado como base el algoritmo diseñado por Ana I. Torre Bastida[79] para la conferencia *Flexible Query Answering Systems* de 2013[34] que se reproduce en la figura 4.17.

Se obviarán los detalles del algoritmo, ya que se explican detalladamente en el artículo original ([79]), pero sí que resulta interesante explicar los fundamentos en los que se basa.

El algoritmo reescribe consultas de SPARQL manejando relaciones o enlaces existentes entre las ontologías origen y destino. En el caso de que exista un enlace directo de equivalencia entre el término en cuestión de la ontología origen con otro de la de destino, la traducción es directa. Sin embargo, esto no ocurre la mayoría de las veces, y ahí es donde el algoritmo despliega su potencial, recorriendo la ontología hacia arriba y hacia abajo y construyendo así una alternativa que, utilizando los términos de la ontología destino, se acerque al término original de la ontología origen. En caso de que exista más de una posible reescritura, contempla el análisis de la pérdida de información de las diferentes alternativas y la selección la de menor pérdida.

Por lo tanto, y a modo de resumen, el algoritmo devuelve la consulta original que se le proporcione reescribiéndola y adaptándola al depósito destino, la mayoría de las veces mediante la unión e intersección de términos pertenecientes a la ontología destino.

Para probarlo, se ha diseñado la ontología que se muestra en tono rosado en la figura 4.18. En dicho gráfico, las flechas de trazo continuo representan relaciones *rdfs:subClassOf* mientras que las discontinuas indican relaciones *owl:equivalentClass*. Por otro lado, los nodos coloreados en verde muestran la ontología utilizada por la DBpedia. Tal y como puede apreciarse, se ha intentado abarcar el mayor número de casos posibles para poder, una vez implementado, comprobar el comportamiento del algoritmo ante las diferentes situaciones.

Por ejemplo, en caso de pedir obras de tipo `littera-owl:Fiction` a la DBpedia, el algoritmo detecta que es un término de una ontología no contemplada en el depósito destino y comienza con el proceso de reescritura. Una vez comprobado que no existe ninguna equivalencia directa con otro término de la ontología destino, inicia dos caminos: la unión de sus subclases (en este caso, `littera-owl:Fantasy` $\cup$ `littera-owl:ScienceFiction` $\cup$ `littera-owl:Mistery` $\cup$ `littera-owl:Humor`), y la intersección de sus superclases con las subclases de estas (en este caso, `littera-owl:Book` $\cap$ `littera-owl:NonFiction`). Aplica el mismo procedimiento para traducir los nuevos términos de ambas ramas hasta llegar a términos contemplados en la ontología destino. En este caso, puede observarse cómo la primera rama resultará infructuosa, mientras que en la segunda hallará `dbpedia-owl:Book` y el algoritmo concluirá resolviendo que la traducción más acertada de `littera-owl:Fiction` es `dbpedia-owl:Book`.

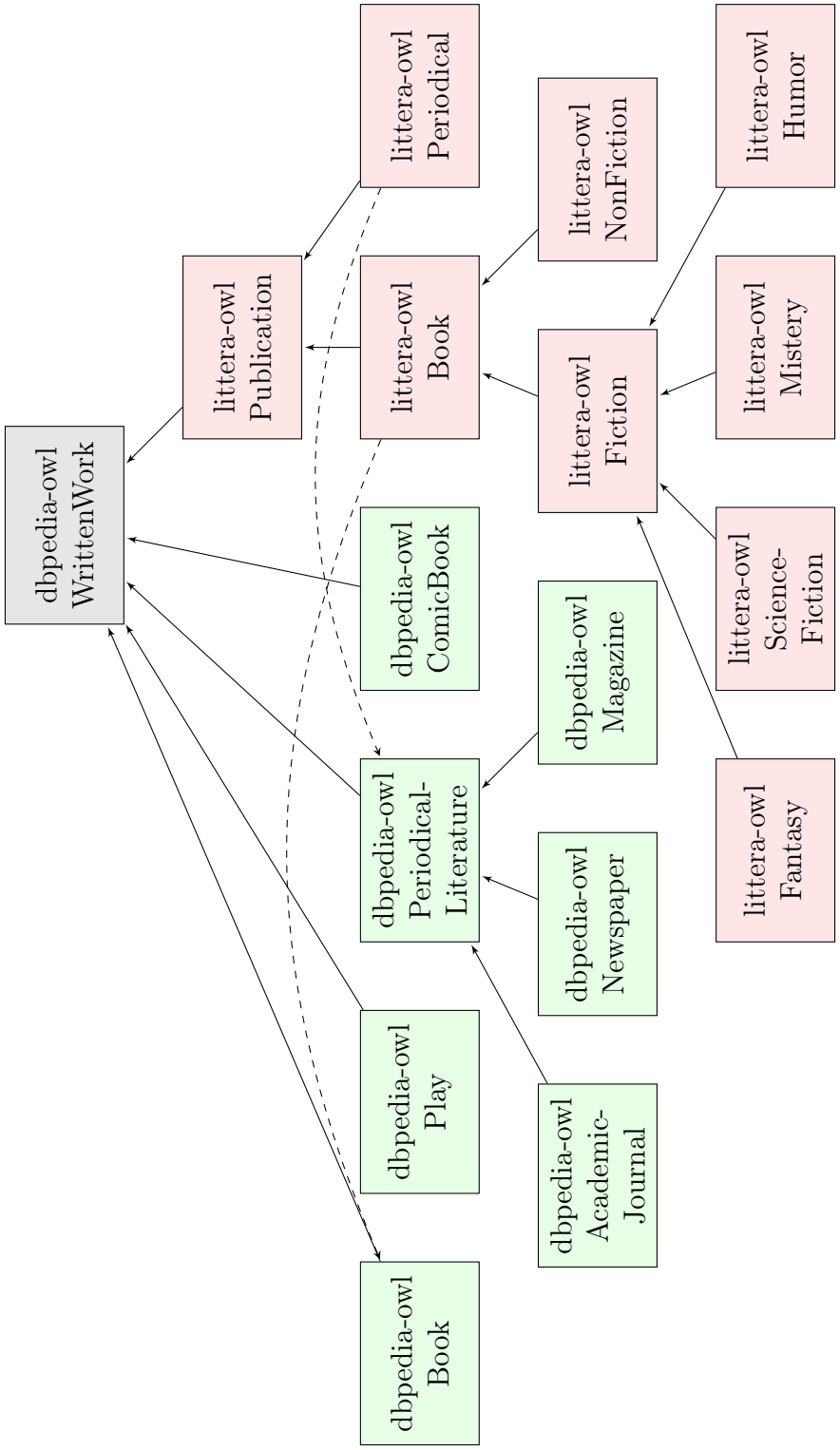


Figura 4.18: Ontología utilizada para categorizar las obras escritas

Otro ejemplo algo más complejo puede ser el de la traducción de dbpedia-owl:Play a la ontología contemplada por litτera. En este caso, y una vez verificada la falta de equivalencias directas, el algoritmo tratará de iniciar las dos ramas pero, al comprobar que el término no tiene ninguna subclase, solamente podrá partir de una de ellas: dbpedia-owl:WrittenWork ∩ (littera-owl:Publication ∪ dbpedia-owl:Book ∪ dbpedia-owl:PeriodicalLiterature ∪ dbpedia-owl:ComicBook). Los dos primeros términos están contemplados en la ontología utilizada por litτera; a diferencia de los tres últimos, que requieren seguir aplicando el algoritmo para ser reescritos. La reescritura de dbpedia-owl:Book y dbpedia-owl:PeriodicalLiterature es inmediata, ya que tienen definidas sendas equivalencias directas con littera-owl:Book y littera-owl:Periodical respectivamente. El término dbpedia-owl:Play, sin embargo, resulta imposible de transformar, ya que no tiene subclases y su única superclase es el término ya analizado. Por lo tanto, el algoritmo concluirá resolviendo que la traducción más acertada de dbpedia-owl:Play es dbpedia-owl:WrittenWork ∩ (littera-owl:Book ∪ littera-owl:Periodical ∪ littera-owl:Publication).

4.2.2. Obtención de equivalencias

La obtención de los equivalentes de una instancia en otros depósitos de datos es una tarea algo más compleja de lo que en principio podría parecer. Por un lado, hay que tener en cuenta que los enlaces no son bidireccionales, pudiendo ocurrir que un depósito solo tenga enlaces salientes pero ninguno entrante. Por otro lado, los depósitos del conjunto no tienen por qué estar conectados directamente entre sí, pudiendo darse el caso de que la única relación entre dos de ellos sea el enlace saliente hacia un tercero común de fuera del conjunto. Y por último y no menos importante, que hay que optimizar las peticiones que se hagan, tanto en número como en complejidad, ya que cada una de ellas supone unas ciertas décimas de segundo que, multiplicándose por el número de peticiones totales que hay que realizar para obtener las equivalencias, pueden llegar a suponer un tiempo importante.

Por lo tanto el problema radica en, dado un conjunto de depósitos de datos y una instancia que pertenezca a uno de esos depósitos, encontrar la instancia equivalente a la dada en cada uno de los depósitos del conjunto de la forma más eficiente posible.

El algoritmo diseñado para resolverlo se muestra en la figura 4.19 y se fundamenta en la creación de una lista propia para cada depósito en el que ir

1. Obtener una lista `e` con la relación de todos los depósitos definidos excepto aquel al que pertenece `autor`
2. Crear una lista `f` vacía en la que se almacenará el resultado final
3. Obtener las equivalencias directas de `autor` y, junto con `autor`, almacenarlo tantas veces como elementos haya en la lista `e` en una nueva lista `s`
4. Mientras haya elementos en `s`, recorrer la lista de depósitos `e`, ejecutando para cada uno:
   - 4.1. Si no hay ninguna URI en la lista asociada de `s`, continuar con la siguiente iteración
   - 4.2. Si alguna de las URIs en la lista `s` asociada al depósito pertenece al propio depósito, o alguna de las entidades almacenadas en el depósito tiene una equivalencia directa definida con cualquiera de las URIs de la lista `s` asociada al depósito
     - 4.2.1. Insertar la URI en cuestión en la lista `f`
     - 4.2.2. Obtener las equivalencias directas de dicha URI e insertarlas en cada sublista de `s`
     - 4.2.3. Eliminar el depósito de la lista `e`, así como su `s` asociada
   - 4.3. Si no, vaciar la lista `s` asociada al depósito

Figura 4.19: Algoritmo para la obtención de equivalencias indirectas

almacenando todas las equivalencias obtenidas hasta el momento y que no se han comprobado aún en el depósito en cuestión. Esto permite realizar consultas agregadas (que son en términos temporales mucho menos costosas que consultas fraccionadas con los mismos datos) y que las consultas sucesivas sean a su vez más simples y rápidas (lo cual no sería posible de hacer si no se llevase el registro de las instancias que se han comprobado en cada depósito).

Por último, solo queda reseñar que es importante que la condición expresada en el paso 4.2. del algoritmo se evalúe de forma perezosa (es decir, que la segunda condición se evalúe si y solo si la primera ha resultado falsa), ya que en caso contrario se estaría haciendo una petición adicional absolutamente innecesaria. Afortunadamente la mayoría de los lenguajes de programación (incluido el elegido para realizar la implementación en este caso) actúan así, pero si alguno no lo hiciera, convendría modificar ligeramente el algoritmo para forzar esa evaluación en dos pasos.

Podemos ilustrar el comportamiento del algoritmo con el ejemplo de la figura 4.20. En dicha figura, los nodos D_i representan los depósitos del conjunto

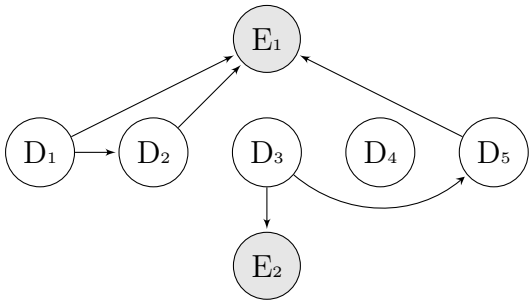


Figura 4.20: Ejemplo para ilustrar el algoritmo de equivalencias

sobre el que se quiere realizar la búsqueda, mientras que los marcados como E_i indican depósitos externos. Por otro lado, las flechas representan enlaces de equivalencia (`owl:sameAs`) entre las instancias que se están buscando.

Si la búsqueda comienza por una instancia del depósito D_1, el algoritmo empezará obteniendo sus equivalencias directas (E_1 y D_2) para a continuación comenzar a iterar con el resto de la lista de depósitos del conjunto. La instancia de D_2 será descubierta sin necesidad de ninguna consulta, ya que está presente en la lista de equivalencias confeccionada hasta el momento. En cualquier caso sí que se deberán consultar las equivalencias directas definidas en el depósito por si hubiese alguna no descubierta hasta el momento (cosa que, en este caso, no ocurre). La búsqueda en D_3 y D_4 no reportará por ahora ningún resultado, ya que no comparten enlaces directos con D_1, D_2 y E_1; al contrario que D_5, que será descubierto por su enlace saliente a E_1. En una segunda vuelta, se descubrirá D_3 por su enlace saliente a D_5, mientras que se concluirá que D_4 no contiene entre sus instancias ninguna equivalente a la que se está buscando.

Por lo tanto, el resultado se obtendrá tras nueve consultas, realizadas en el siguiente orden: D_1, D_2, D_3, D_4, D_5, D_5, D_3, D_3, D_4. De ellas, cinco serían comprobaciones de enlaces salientes a alguno de los depósitos obtenidos hasta el momento (independientemente de que pertenezcan o no al conjunto de búsqueda) y las otras cuatro peticiones solicitando los enlaces directos que tiene definidos el depósito en cuestión para la instancia encontrada.

Como puede apreciarse, el número de consultas necesarias varía dependiendo del orden en el que se evalúen los depósitos del conjunto. Así, tratar D_5 antes que D_3 conlleva ahorrarse dos consultas (orden de ejecución: D_1, D_2, D_5, D_5, D_3, D_3, D_4), mientras que tratar D_4 antes que D_3 conlleva ejecutar una consulta adicional (orden de ejecución: D_1, D_2, D_4, D_3, D_5, D_5, D_4, D_3, D_3, D_4). Aunque una vez vistos los enlaces puede resultar obvio el orden óptimo

para tratar los depósitos, hay que tener en cuenta que precisamente lo que se desconoce al inicio del problema son estos enlaces, por lo que no se dispone de información para seleccionar un orden u otro.

4.3. Clases

En la figura 4.21 se encuentra una versión simplificada del diagrama de clases que permite visualizar de forma global la composición del sistema así como las relaciones entre las diferentes clases. Por motivos de espacio y legibilidad se obvian los atributos y métodos concretos de cada una. Es por ello que a continuación se hace una descripción general de cada clase adjuntándose ahí ya toda la información (tanto gráfica como textual) relativa a las mismas, con el ánimo de que esta forma de presentación sea más cómoda para el lector.

4.3.1. Clase Endpoint

La clase *Endpoint* (figura 4.22) representa los depósitos de datos definidos por el administrador.

Como atributos tiene toda la información almacenada en el archivo de configuración correspondiente, y para inicializarla existen dos opciones: o bien pasar a su constructor el archivo de configuración procesado o bien utilizar sus *setters* correspondientes. En caso de elegir la primera opción, el método privado *isWellFormed* se encarga de validar la corrección de los datos proporcionados.

Además de los *setters* y *getters*, la clase ofrece el método *belongs* que comprueba si un recurso pertenece al dominio, y el método *belongsWorkType* que comprueba si el término que se pasa como argumento pertenece a la ontología utilizada por el depósito para la categorización de obras.

4.3.2. Clase Local_Dataset

La clase *Local_ Dataset* (figura 4.23) representa el depósito propio de `litτera`, en caso de haberlo.

Como atributos tiene el *endpoint* correspondiente, así como los datos de acceso de un usuario existente con permisos de escritura en el depósito en

Figura 4.21: Simplificación del diagrama de clases

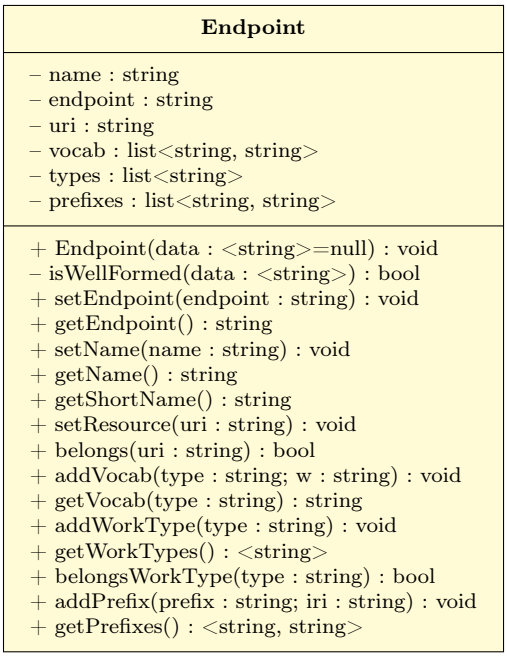


Figura 4.22: Atributos y métodos de la clase Endpoint

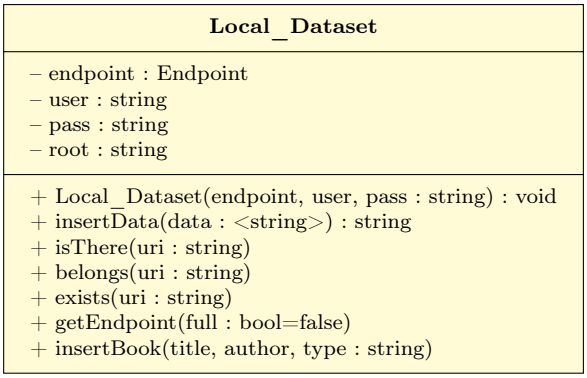


Figura 4.23: Atributos y métodos de la clase Local_Dataset

cuestión. Asimismo, se almacena el atributo auxiliar *root* que facilita algunos de los métodos de la clase.

Entre sus métodos destaca *insertData* (que recibe y trata la petición de inserción de un nuevo escritor con diferentes obras) e *insertBook* (que inserta un nueva obra de un escritor ya existente en el depósito local). Por otro lado se ofrece un método para obtener el *Endpoint* correspondiente al depósito, y varios métodos auxiliares como *belongs* (que comprueba la pertenencia de una URI al dominio), *exists* (que además de comprobar su pertenencia certifica

su existencia en el depósito), o *isThere* (que comprueba la pertenencia de algún equivalente a la instancia dada).

4.3.3. Clase Endpoint_Manager

La clase *Endpoint_Manager* (figura 4.24) está creada siguiendo el patrón de diseño *singleton*. Este patrón garantiza la existencia de una única instancia de la clase mediante la definición de métodos públicos estáticos y un constructor privado, lo cual permite que sea accesible en todo momento y desde cualquier otra clase de la aplicación.

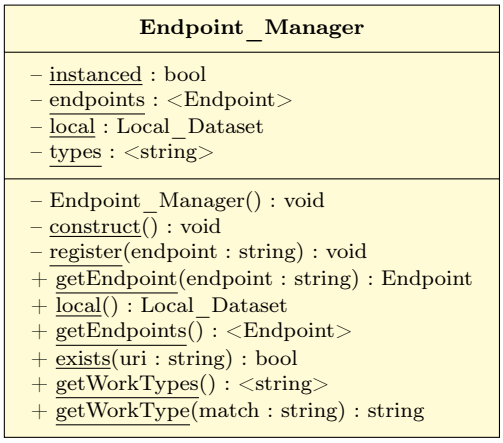


Figura 4.24: Atributos y métodos de la clase Endpoint_Manager

Al instanciarse carga los depósitos de datos definidos (desechando aquellos que por alguna razón sean incorrectos) y los pone accesible a través del método *getEndpoints*. Del mismo modo, detecta si se ha definido un depósito local y, en caso de haberlo hecho, lo ofrece mediante el método *local*.

El resto de métodos son meras facilidades para hacer consultas globales a todos los depósitos disponibles. Por ejemplo, *getEndpoint* devuelve el *Endpoint* correspondiente al identificador dado. Este tipo de tareas pueden realizarse obteniendo la lista completa de depósitos (*getEndpoints*) y comprobando a cuál de ellos pertenece el identificador en cuestión. Sin embargo, al ser una tarea muy común, se ha preferido definir este tipo de métodos.

4.3.4. Clase Author

La clase *Author* (figura 4.25) representa un escritor en un depósito de datos.

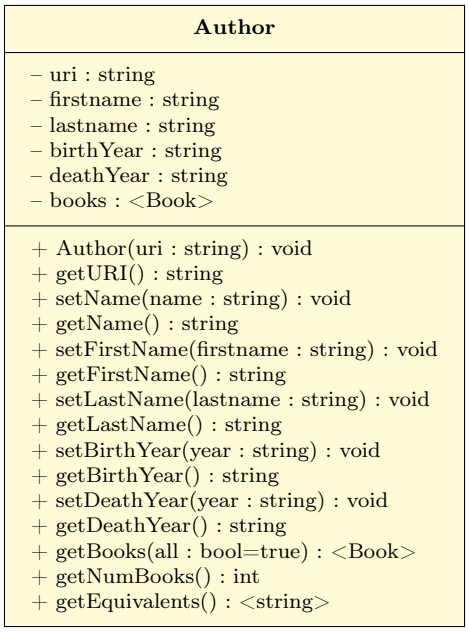


Figura 4.25: Atributos y métodos de la clase Author

Tiene como atributos la URI, el nombre, el apellido, el año de nacimiento, el año de fallecimiento y la lista de obras escritas de acuerdo al depósito de datos en cuestión. En cuanto a los métodos, más allá de los *getters* y *setters*, destaca *getEquivalents*, que obtiene las instancias enlazadas directa o indirectamente con el escritor en el resto de depósitos.

4.3.5. Clase Book

La clase *Book* (figura 4.26) representa una obra en un depósito de datos.

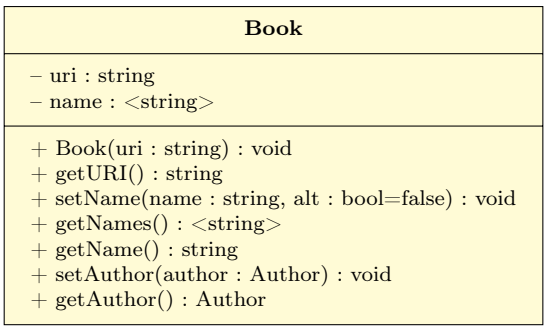


Figura 4.26: Atributos y métodos de la clase Book

Dispone de la opción de establecer títulos alternativos por si en el depósito de datos así estuviese definido.

4.3.6. Clase Criteria

La clase *Criteria* (figura 4.27) se utiliza para representar criterios de búsqueda, procesándolos y validándolos.

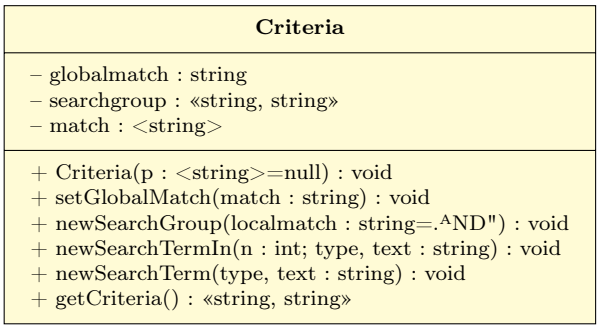


Figura 4.27: Atributos y métodos de la clase Criteria

Cuenta con varios métodos para establecer los diferentes grupos de búsqueda y campos en cada grupo, y otro (*getCriteria*) para obtener el resultado final, que previamente valida y asegura la corrección de los datos.

4.3.7. Clase Mediator

La clase *Mediator* (figura 4.28) unifica la comunicación con todos los depósitos de datos. Se encarga de comprobar el destino de la petición y delegarla en el *wrapper* correspondiente, procesando los datos si fuera necesario. Así, se hace posible la abstracción de los detalles de la implementación de cada depósito, proporcionando una interfaz unificada.

Dispone de los mismos métodos que la interfaz *Wrapper*, por lo que se detallarán en dicho apartado.

4.3.8. Interfaz Wrapper

La interfaz *Wrapper* (figura 4.29) define los métodos mínimos con los que deben contar los *wrappers* para ser considerados como tales.

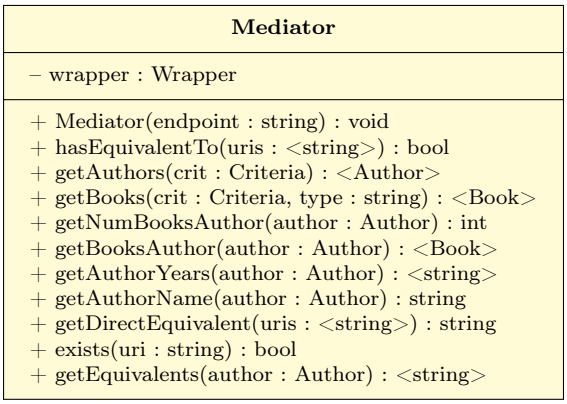


Figura 4.28: Atributos y métodos de la clase Mediator

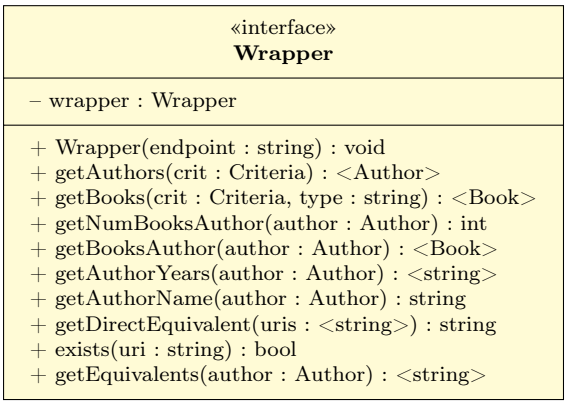


Figura 4.29: Atributos y métodos de la clase Wrapper

La funcionalidad de la mayoría de los métodos puede deducirse fácilmente de su propio nombre, cabiendo mencionar, si acaso, *getDirectEquivalent* que, dada una lista de identificadores de entidades devuelve (si lo hubiera) la entidad del depósito con el que alguno de ellos está directamente enlazado.

4.3.9. Clase Sparql_Wrapper

La clase abstracta *Sparql_ Wrapper* (figura 4.30) implementa los métodos comunes válidos para cualquier depósito de datos accesible a través del lenguaje de consultas SPARQL.

Define como abstracto el método *getBooks*, ya que su implementación dependerá de si cada depósito en concreto almacena las obras de forma jerarquizada o no. Por otro lado, cabe destacar el método *getEquivalents*, definido para

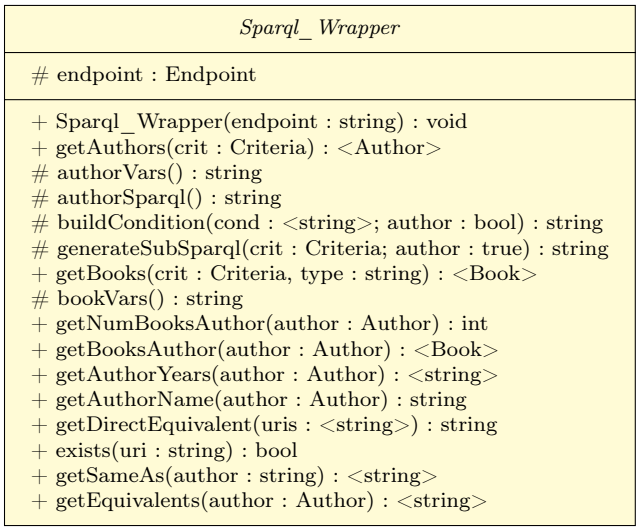


Figura 4.30: Atributos y métodos de la clase Sparql_Wrapper

implementar el algoritmo expuesto en la sección 4.2.2.

4.3.10. Clase Uncat_Sparql_Wrapper

La clase *Uncat_ Sparql_ Wrapper* (figura 4.30) extiende *Sparql_ Wrapper* para soportar depósitos de datos que no almacenen las obras de forma jerarquizada.

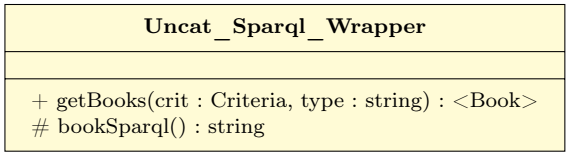


Figura 4.31: Atributos y métodos de la clase Uncat_Sparql_Wrapper

4.3.11. Clase Cat_Sparql_Wrapper

La clase *Cat_ Sparql_ Wrapper* (figura 4.30) extiende *Sparql_ Wrapper* para soportar depósitos de datos que sí almacenen las obras de forma jerarquizada.

Para ello, define varios de métodos protegidos que hacen posible la implementación del algoritmo expuesto en la sección 4.2.1.

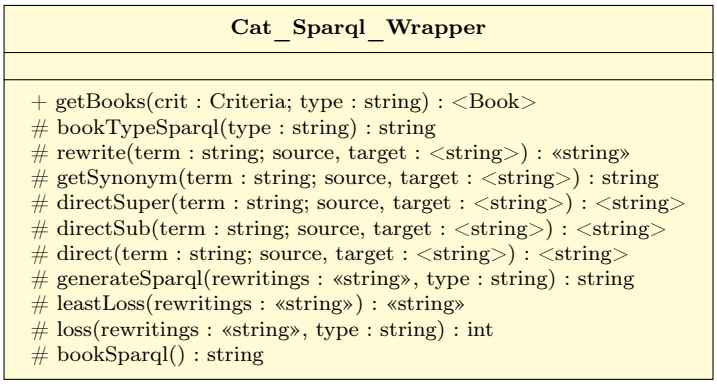


Figura 4.32: Atributos y métodos de la clase Cat_Sparql_Wrapper

4.3.12. Clase Sparql

La clase *Sparql* (figura 4.33) proporciona acceso de bajo nivel y universal a cualquier depósito de datos enlazados accesible mediante un *SPARQL endpoint*.

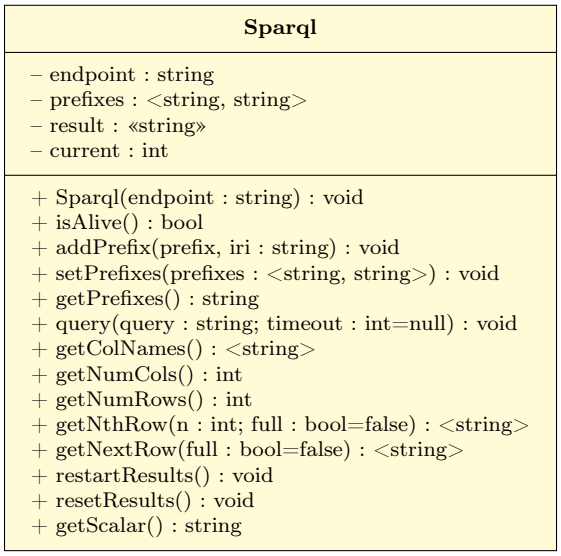


Figura 4.33: Atributos y métodos de la clase Sparql

Su método principal es *query* que, una vez instanciada la clase con la URL del *SPARQL endpoint* sobre el que actuar, permite realizar consultas en SPARQL, indicando opcionalmente su periodo de expiración o *timeout*. Para hacer más cómodas las series de consultas, existen varios métodos (*addPrefix* y *setPrefixes*) que permiten definir globalmente prefijos, y no tener así que

especificarlos en cada consulta. Esto último, en todo caso, es completamente opcional.

Por otro lado existen una serie de métodos con los que acceder cómodamente a los resultados de la consulta. Entre ellos el más relevante es *getNextRow*, que va devolviendo en cada llamada una fila del resultado hasta agotarlas todas. De la misma manera existe *getNthRow* que proporciona acceso directo a la enésima fila del conjunto resultado. *restartResults* permite volver acceder desde el principio a los resultados mediante el método *getNextRow*, y *resetResults* limpia los datos de la consulta anterior en los casos en los que sea interesante asegurarse que no se puede volver a acceder a ellos a menos que se repita la consulta. *getNumCols* y *getNumRows* devuelven respectivamente el número de columnas y filas que tiene el conjunto resultado, mientras que *getColNames* proporciona los nombres de las columnas (especialmente útil en caso de haber utilizado comodines en la consulta). Por último, *isAlive* comprueba la existencia y disponibilidad del *SPARQL endpoint* solicitado ejecutando una sencilla consulta, y *getScalar* proporciona un acceso rápido al dato en caso de estar completamente seguros de que el resultado de la consulta consta de una única fila y columna (y fallando en caso de no ser así).

Cabe recalcar que esta clase es completamente independiente al dominio elegido para desarrollar la infraestructura y que, por tanto, puede integrarse como biblioteca en cualquier desarrollo en PHP relacionado con SPARQL.

4.4. Diagramas de secuencia

En las figuras 4.34, 4.35 y 4.36 se muestran los diagramas de secuencia de los casos de uso definidos en la sección 4.1.

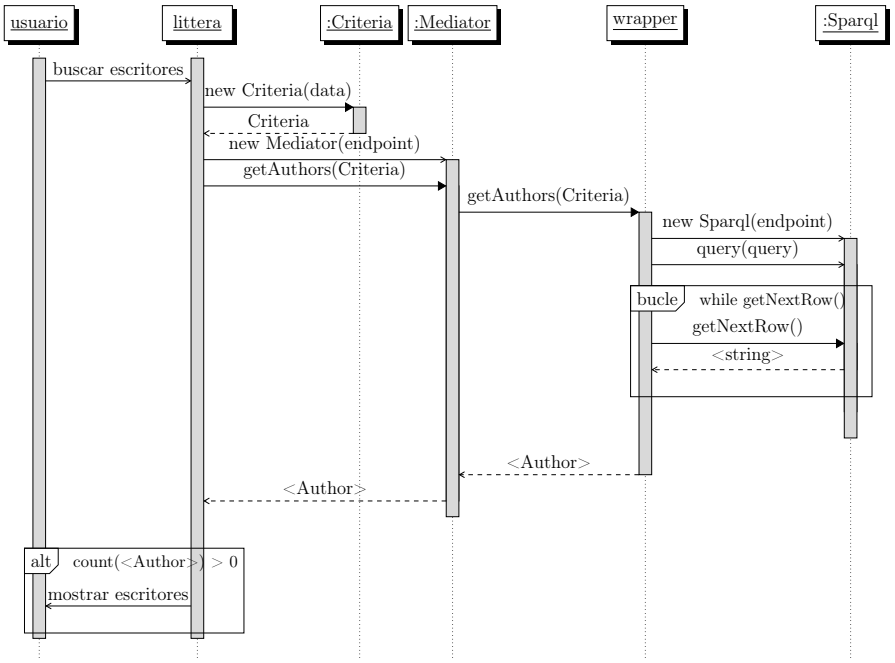


Figura 4.34: Diagrama de secuencia del caso de uso *buscar escritores*

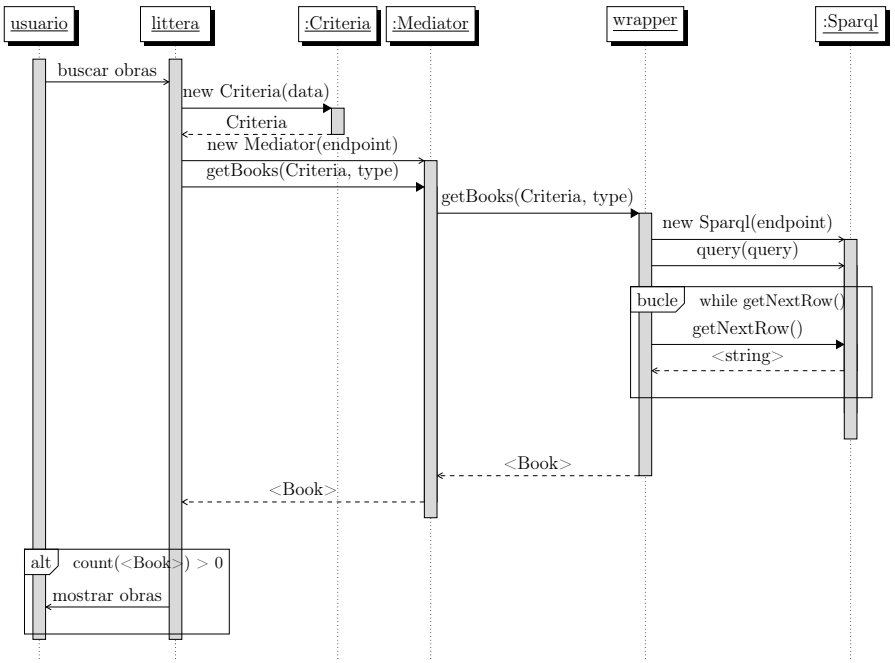


Figura 4.35: Diagrama de secuencia del caso de uso *buscar obras*

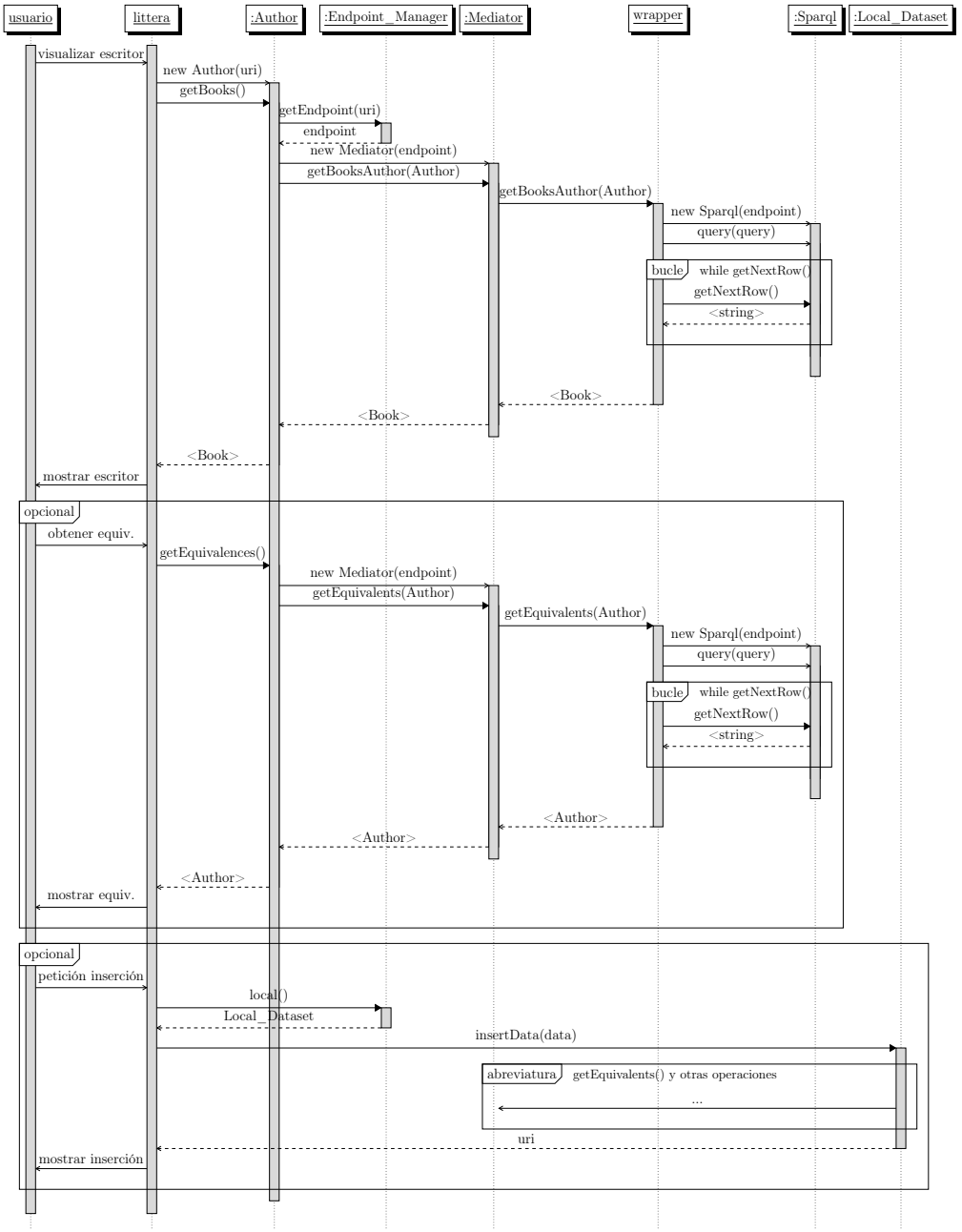


Figura 4.36: Diagrama de secuencia del caso de uso *visualizar escritor*

Capítulo 5

Implementación

En este capítulo se incluyen diversos detalles de la implementación del sistema, tales como el modelo de desarrollo utilizado, el sistema de pruebas usado para cerciorar su correcto funcionamiento, los requisitos mínimos necesarios para su uso y los resultados empíricos obtenidos con el sistema.

5.1. Modelo de desarrollo

Para desarrollar el sistema se ha utilizado un modelo iterativo y creciente[5] consistente en el desarrollo en pequeñas ciclos continuos e incrementales. Esto ha permitido ir modelando la dirección y los detalles del desarrollo de acuerdo con la experiencia adquirida y las observaciones hechas en los ciclos anteriores, tanto en lo relativo a su desarrollo como a su uso y comportamiento.

Por otro lado se ha utilizado un sistema de control de versiones que ha hecho posible un desarrollo más ágil, aislando las características experimentales y respaldando con históricos reversibles los cambios realizados a lo largo del periodo de implementación.

5.2. Pruebas

Para realizar las pruebas se han utilizado dos estrategias diferenciadas: para las clases de más bajo nivel se han diseñado *suites* de pruebas unitarias (*unit*

tests), mientras que las de más alto nivel se han probado directamente en el sistema.

Se decidió actuar así porque las clases de bajo nivel son más estables y menos propensas al cambio de funcionalidad e interfaz a lo largo de la vida del desarrollo. El tiempo invertido en un principio en definir nítidamente los resultados y errores esperados en cada caso y hacer las pruebas unitarias correspondientes permitieron asegurar el correcto funcionamiento inicial de dichas clases. Asimismo, hicieron posible las refactorizaciones a posteriori del código, que de no haber contado con dichas pruebas no se hubiesen realizado ante el miedo de cambiar involuntariamente la respuesta ante algún caso sobre el que una entidad de nivel superior dependiese.

Análogamente, se decidió no programar pruebas unitarias para las clases de nivel superior porque estas son más propensas a cambios, además de que son más fáciles de probar en vivo a través de la propia aplicación.

Más allá del correcto funcionamiento de la aplicación, se probaron también diferentes configuraciones a la estándar utilizada durante el desarrollo (que constaba de Apache HTTP Server 2.2.22, PHP 5.4.9, libcurl 7.29.0 y OpenLink Virtuoso 06.01.3127 en el lado del servidor y la última versión de Google Chrome en el lado del cliente) para obtener los requisitos mínimos necesarios para utilizar el sistema, obteniendo los resultados que se exponen en la siguiente sección.

5.3. Requisitos

Los requisitos mínimos en el lado del servidor para ejecutar `litτera` con todas sus funcionalidades son los que siguen:

- Servidor web
- PHP versión 5.3 o superior
- Biblioteca libcurl para PHP
- OpenLink Virtuoso

En los casos en los que no se indica el número de versión necesaria, no se conocen incompatibilidades que requieran el uso de una u otra versión en concreto; mientras que en los que sí que se indican, son siempre versiones *mínimas*, pudiendo elegir cualquiera más reciente que la expresada.

Cabe destacar que *no* es requisito que OpenLink Virtuoso se encuentre instalado en la misma máquina que el servidor web en el que se ejecuta la aplicación, pudiendo estos dos componentes estar situados distintos lugares del planeta siempre y cuando estén conectados a Internet (aunque, como es natural, la latencia en las consultas y peticiones será mayor que si se encontrasen en la misma máquina). De la misma forma, es interesante destacar que no existen requisitos en cuanto a la potencia de la máquina (o máquinas) que ejecuten el sistema, más allá, claro está, de los que impongan las herramientas mencionadas anteriormente, los cuales los cumple sin ningún problema cualquier ordenador personal de la actualidad.

5.4. Compatibilidad

En el lado del cliente, para acceder a la aplicación y hacer uso de toda su funcionalidad, solo hace falta un navegador web de cualquiera de los tipos que se enumeran a continuación, siempre que esté conectado a Internet y tenga activada la ejecución de código JavaScript:

- Google Chrome[63]
- Mozilla Firefox[65] versión 4.0 o superior
- Internet Explorer[64] versión 9.0 o superior
- Safari[67]
- Opera[66]
- Navegadores web basados en los motores de renderizado WebKit[61] (p. ej. Epiphany[62]), Blink[59] (p. ej. Opera 15[41]) y Gecko[60] (p. ej. SeaMonkey[68])

Es interesante hacer un pequeño análisis sobre lo que supone en términos cuantitativos esta compatibilidad. En la figura 5.1 se representa cómo se repartió la cuota de mercado de cada navegador durante el mes de junio de 2013 de acuerdo a los datos de StatCounter[4].

Según dichos datos, la aplicación es totalmente compatible con los navegadores que representan el 89,06 % de cuota de mercado. Por otro lado, se ofrece un soporte parcial a Internet Explorer 8 (8,04 % de cuota) en el que la aplicación también es funcional aunque se pueden apreciar diferencias cosméticas y visuales con respecto a los navegadores totalmente soportados.

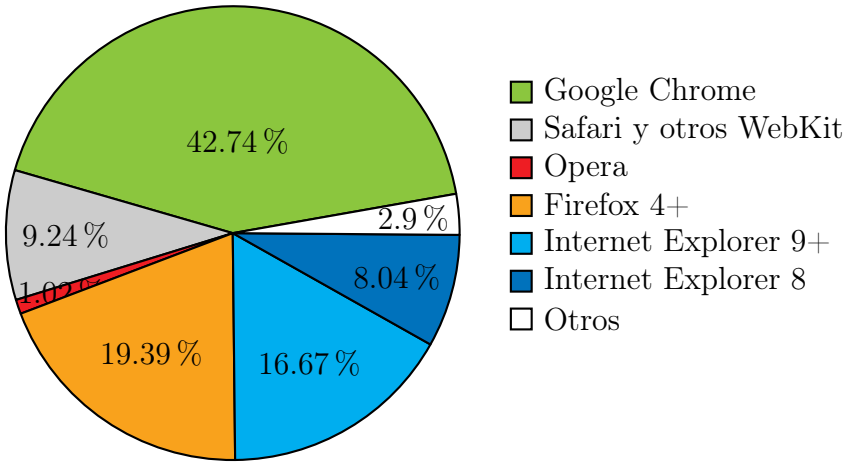


Figura 5.1: Cuota de mercado de los navegadores web durante junio de 2013

El 2,9 % de cuota restante no suponen necesariamente navegadores no soportados, sino casos minoritarios que no se han analizado y de los que se desconoce, por tanto, su nivel de compatibilidad con la aplicación. En cualquier caso, se ha puesto especial atención en la escritura de código HTML válido y estándar, por lo que cualquier navegador presente o futuro que respete los estándares y los interprete adecuadamente debería ser compatible.

5.5. Resultados

A continuación se presentan los costes en términos temporales que suponen diferentes consultas y acciones en cada uno de los depósitos de datos que se han probado. Cada dato que se presenta en esta sección se ha obtenido realizando cinco ejecuciones y tomando la mediana de dichos resultados. Se omiten los relativos a la Biblioteca Nacional de Suecia (KB) porque, como se ha explicado anteriormente, en el tramo final de este proyecto ha estado inaccesible.

Dicho esto, podemos proceder a presentar los datos del cuadro 5.1. En él se expresan los tiempos de respuesta derivados de la realización de tres consultas simples: la primera de ellas la búsqueda exacta del apellido de un escritor (opción *Author's last name is* de la aplicación), la segunda la búsqueda por el comienzo de un apellido (opción *Author's last name starts with*), y la tercera una búsqueda libre en el nombre compreto del escritor (opción *Author's name contains*).

	litτera	**DBpedia**	**BNE**	**OSZK**	**BL**
exacto	0,2 s.	0,3 s.	0,1 s.	0,4 s.	*timeout*
comienzo	0,4 s.	0,4 s.	21,8 s.	5,9 s.	*timeout*
libre	1,1 s.	1,3 s.	6,8 s.	*timeout*	*timeout*

Cuadro 5.1: Tiempos respuesta para diferentes búsquedas simples

Podría esperarse que los tiempos fuesen incrementales entre las consultas, ya que la búsqueda exacta no hace uso de expresiones regulares ni filtros que puedan ralentizarla, y la búsqueda por el comienzo del apellido puede aprovechar índices y ordenaciones que tenga construidos el depósito de triples. Aunque en la mayoría de los casos es así, destaca la Biblioteca Nacional de España (BNE) como excepción, ya que en dicho depósito la segunda operación es muchísimo más costosa de lo que cabría esperar.

También puede sorprender la similitud entre los resultados del depósito propio de litτera frente a la DBpedia ya que, estando el primero almacenado localmente y conteniendo muy pocos triples, cabría esperar que sus tiempos de respuesta fuesen notablemente inferiores a los de la DBpedia. Esto seguramente se deba a dos factores: el principal de ellos es que el sistema se ha probado ejecutándose en un *netbook* de muy bajas prestaciones lo cual ralentiza sensiblemente las operaciones, y por otro lado, y aunque no se tengan datos concretos que respalden esta afirmación, la reciente actualización en la DBpedia de OpenLink Virtuoso a su versión 7.0 ha conllevado unas mejoras sustanciales en el rendimiento general del depósito.

litτera	**DBpedia**	**BNE**	**OSZK**	**BL**
0,9 s.	1,2 s.	1,4 s.	1,3 s.	1,4 s.

Cuadro 5.2: Tiempos respuesta para la visualización de un escritor

En el cuadro 5.2 se recogen los tiempos aproximados de respuesta para la visualización de un escritor cualquiera en cada uno de los depósitos de datos. Puede apreciarse cómo no hay grandes diferencias entre ellos, ni siquiera con la Biblioteca Británica (BL), que arrojaba tan malos resultados a la hora de buscar los escritores.

Por último, solo queda mencionar el tiempo de inserción de un escritor junto con unas decenas de obras, que se sitúa en torno a los 3,4 segundos.

Capítulo 6

Conclusiones y líneas futuras

En este último capítulo se resumen las características y funcionalidades del sistema desarrollado, se hace una breve conclusión final sobre el área de investigación en el que se ha trabajado, y se presentan las líneas principales sobre las que se puede continuar trabajando para extender el proyecto.

6.1. Conclusiones

En este proyecto se ha desarrollado un sistema que cuenta con las siguientes características:

- Federación de varios depósitos de datos enlazados del campo de la literatura.
- Interfaz web visualmente moderna y compatible con los navegadores más utilizados.
- Reescritura de consultas y adecuación automática a la ontología destino.
- Construcción colaborativa de un nuevo depósito de datos abiertos y enlzados.

El sistema permite a los usuarios:

- Utilizar un formulario de búsqueda enriquecido para poder realizar búsquedas complejas sin la necesidad de tener conocimientos de SPARQL.

- Consultar diferentes tipos de datos almacenados en los depósitos (en este caso, escritores y obras).
- Buscar de forma sencilla instancias equivalentes en otros depósitos de datos para poder alternar entre ellos y obtener así más información.
- Revisar e insertar datos en un depósito propio del sistema y poder consultarlos depués.

Por otro lado, `litτera` proporciona al administrador una gran flexibilidad a la hora de manejar los depósitos de datos de los que hace uso la aplicación, ya que estos pueden añadirse o retirarse del sistema editando simples archivos de configuración y sin la necesidad de modificar nada de código. Además, su arquitectura *Mediator-Wrapper* deja la puerta abierta a incluir de forma sencilla otros depósitos de datos enlazados que *no* hagan uso de SPARQL como lenguaje de consulta.

Todo esto ha sido posible gracias al uso de diversas tecnologías, entre las que destacan:

- PHP como lenguaje del lado del servidor, cuya popularidad y extendida presencia en servidores web permite que `litτera` pueda ser instalado en un gran número de entornos.
- Bootstrap para la construcción de la interfaz visualmente moderna y atractiva.
- jQuery, JavaScript y AJAX para dotar al sistema de elementos dinámicos que ofrecen fluidez y mejoran la experiencia de usuario al utilizar el sistema.
- JSON para la definición y el intercambio ligero de datos.
- OpenLink Virtuoso como plataforma para el almacenamiento y consulta eficiente de triples.
- WebDAV como método seguro para la autentificación y posterior manipulación de triples en OpenLink Virtuoso.

Aunque el dominio elegido para realizar el análisis, la implementación y las pruebas correspondientes haya sido el de la literatura, las técnicas utilizadas para el desarrollo del sistema no son particulares de este dominio, por lo que pueden ser fácilmente extrapolables a otros casos de uso.

Asimismo, como ya se ha comentado, se ha propuesto un sistema para la creación colaborativa de depósitos de datos que combina la extracción automatizada de datos de depósitos de terceros con la revisión humana para

asegurar su corrección. Este aspecto es especialmente relevante, ya que si el campo de los datos enlazados "cojea", no es porque no existan aplicaciones o herramientas que permitan sacar partido a los datos almacenados en los distintos depósitos, sino por los propios depósitos de datos que, habiendo sido generados automáticamente, demasiadas veces están faltos de los enlaces o las definiciones necesarias para que se pueda construir una nueva generación de aplicaciones mucho más avanzadas e interesantes.

6.2. Líneas futuras

litτera es un proyecto con potencial para expandirse a otros dominios y ser utilizado en producción para la investigación o consulta pública. Para ello, lo primero que debería desarrollarse es un sistema de usuarios que regule la inserción de datos mediante un sistema de meritocracia, por un lado para evitar el vandalismo, y por otro para asegurar el rigor y la calidad de los datos.

También podría desarrollarse, y este sería un objetivo mucho menos ambicioso que el anterior, un sistema de cacheo que refrescase regularmente el estado de los depósitos de datos. Esto permitiría retirar temporalmente aquellos depósitos que estén caídos o innaccesibles, cosa que ocurre con cierta frecuencia, evitando que el usuario intente buscar en ellos y reciba un desagradable mensaje de error.

Dependiendo del dominio sobre el que se esté actuando, puede resultar interesante aprovechar el potencial que ofrece la arquitectura de litτera para federar depósitos de datos que no sean accesibles mediante *SPARQL endpoints*. Tal es el caso, por ejemplo, de Freebase[28], que ofrece un lenguaje de consulta llamado MQL y que podría ser integrado en el sistema programando únicamente su *wrapper* correspondiente.

Por último, y más allá del sistema desarrollado, existe dentro del campo de los datos enlazados un amplio área en el que poder seguir trabajando, investigando, e intentando aportar soluciones.

Capítulo 7

Bibliografía

7.1. Libros

[13] Tom Heath y Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 2011.

7.2. Artículos

[12] Peter Haase, Tobias Mathäß y Michael Zille. "An Evaluation of Approaches to Federated Query Processing over Linked Data". En: *I-SEMANTICS* (2010).

[22] Bastian Quilitz y Ulf Leser. "Querying Distributed RDF Data Sources with SPARQL". En: *Proceedings of the 5th European Semantic Web Conference (ESWC2008)* (2008).

[24] Andreas Schwarte y col. "FedX: Optimization Techniques for Federated Query Processing on Linked Data". En: *The 10th International Semantic Web Conference* (2011).

[79] Ana I. Torre-Bastida y col. "Query Rewriting for an Incremental Search in Heterogeneous Linked Data Sources". En: *Flexible Query Answering Systems (FQAS)* (Aceptado).

7.3. Referencias en Internet

[1] *Aplicación ShareLaTeX, editor online de LaTeX.* URL: https://www.sharelatex.com/.

[2] *Artículo de Tim Berners-Lee sobre el uso y la publicación de datos enlazados abiertos.* URL: http://www.w3.org/DesignIssues/LinkedData.html.

[3] *Conferencia de Tim Berners-Lee en el congreso TED de 2009, acerca del papel de los datos enlazados en el futuro de la web.* URL: http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html.

[4] *Cuota de mercado de los navegadores web durante junio de 2013 de acuerdo a StatCounter.* URL: http://gs.statcounter.com/#browser_version_partially_combined-ww-monthly-201306-201306-bar.

[5] *Desarrollo iterativo y creciente: información en la Wikipedia.* URL: http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente.

[6] *Documentación de Virtuoso Jena Provider.* URL: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtJenaProvider.

[7] *Documentación de Virtuoso Sesame Provider.* URL: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSesame2Provider.

[8] *Especificación del lenguaje de consulta SPARQL.* URL: http://www.w3.org/TR/rdf-sparql-query/.

[9] *Especificación del vocabulario FOAF.* URL: http://xmlns.com/foaf/0.1/.

[10] *Explicación de la diferencia entre DC y DCTERMS en la wiki de Dublin Core.* URL: http://wiki.dublincore.org/index.php/FAQ/DC_and_DCTERMS_Namespaces.

[11] *Extracto del manual de Semantic MediaWiki en el que se explica la forma de almacenamiento interna así como el lenguaje de consulta a utilizar.* URL: http://semantic-mediawiki.org/wiki/FAQ#How_does_SMW_store_its_data.3F.

[14] *Informe de los Requisitos Funcionales de los Registros Bibliográficos.* URL: http://www.ifla.org/files/assets/cataloguing/frbr/frbr-es.pdf.

[15] *Instalación de los binarios precompilados de OpenLink Virtuoso en Windows*. URL: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSUsageWindows.

[16] *Plantilla para PFCs en LaTeXpublicada por ITSAS (grupo de software libre de la UPV/EHU) en 2007*. URL: http://itsas.ehu.es/workgroups/plantillas_proyecto_fin_de_carrera.

[17] *Portal dedicado a RDF dentro del sitio web del W3C*. URL: http://www.w3.org/RDF/.

[18] *Página de la librería* libcurl *para diferentes lenguajes*. URL: http://curl.haxx.se/libcurl/.

[19] *Página web de Bootstrap switch*. URL: http://www.larentis.eu/switch/.

[20] *Página web de JSON*. URL: http://www.json.org/.

[21] *Página web de tablesorter,* plugin *para jQuery*. URL: http://tablesorter.com/docs/.

[23] *Repositorio GitHub del* fork *de tablesorter*. URL: https://github.com/Mottie/tablesorter.

[25] *Sección dedicada a la web semántica dentro del sitio web del W3C*. URL: http://www.w3.org/standards/semanticweb/.

[26] *Sección dedicada a los datos enlazados dentro del sitio web del W3C*. URL: http://www.w3.org/standards/semanticweb/data.

[27] *Segunda versión de la Licencia Pública General de GNU (GPLv2)*. URL: http://www.gnu.org/licenses/gpl-2.0.html.

[28] *Sitio web de la base de conocimientos colaborativa Freebase*. URL: http://www.freebase.com/.

[29] *Sitio web de la Biblioteca Británica*. URL: http://www.bl.uk/.

[30] *Sitio web de la Biblioteca Nacional de España*. URL: http://www.bne.es/.

[31] *Sitio web de la Biblioteca Nacional de Hungría*. URL: http://www.oszk.hu/.

[32] *Sitio web de la Biblioteca Nacional de Suecia*. URL: http://es.wikipedia.org/wiki/Biblioteca_Nacional_de_Suecia.

[33] *Sitio web de la compañía OpenLink Software*. URL: http://www.openlinksw.com/.

[34] *Sitio web de la conferencia* Flexible Query Answering Systems *de 2013*. URL: http://idbis.ugr.es/fqas2013/.

[35] *Sitio web de la DBpedia*. URL: http://dbpedia.org/.

[36] *Sitio web de la Federación Internacional de Asociaciones e Instituciones Bibliotecarias*. URL: http://www.ifla.org/.

[37] *Sitio web de la plataforma Talis*. URL: http://n2.talis.com/.

[38] *Sitio web de la plataforma TSO*. URL: http://www.tso.co.uk/.

[39] *Sitio web de la Universidad de Leipzig*. URL: http://www.zv.uni-leipzig.de/.
[40] *Sitio web de la Universidad Libre de Berlín*. URL: http://www.fu-berlin.de/.
[41] *Sitio web de la versión en desarrollo de Opera*. URL: http://www.opera.com/next.
[42] *Sitio web de la wiki semántica DataWiki*. URL: http://diqa-pm.com/en/DataWiki.
[43] *Sitio web de OpenLink Virtuoso*. URL: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/.
[44] *Sitio web de Semantic MediaWiki*. URL: http://semantic-mediawiki.org/.
[45] *Sitio web de Virtuoso Universal Server*. URL: http://virtuoso.openlinksw.com/.
[46] *Sitio web del editor de LaTeXpara el entorno de escritorio KDE*. URL: http://kile.sourceforge.net/.
[47] *Sitio web del editor Geany*. URL: http://www.geany.org/.
[48] *Sitio web del editor multiplataforma de LaTeX*. URL: http://www.tug.org/texworks/.
[49] *Sitio web del* framework *Blueprint*. URL: http://www.blueprintcss.org/.
[50] *Sitio web del* framework *Bootstrap*. URL: http://twitter.github.io/bootstrap/.
[51] *Sitio web del* framework SimpleTest *para PHP*. URL: http://www.simpletest.org/.
[52] *Sitio web del* framework *Foundation*. URL: http://foundation.zurb.com/.
[53] *Sitio web del* framework *HTML5 Boilerplate*. URL: http://html5boilerplate.com/.
[54] *Sitio web del* framework *jQuery*. URL: http://jquery.com/.
[55] *Sitio web del* framework *PHPUnit*. URL: http://phpunit.de/.
[56] *Sitio web del IDE Eclipse*. URL: http://www.eclipse.org/.
[57] *Sitio web del IDE NetBeans*. URL: http://netbeans.org/.
[58] *Sitio web del lenguaje de programación PHP*. URL: http://www.php.net/.
[59] *Sitio web del motor de renderizado Blink*. URL: http://www.chromium.org/blink.
[60] *Sitio web del motor de renderizado Gecko*. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Gecko.
[61] *Sitio web del motor de renderizado WebKit*. URL: http://www.webkit.org/.

[62] *Sitio web del navegador Epiphany (también conocido como GNOME Web)*. URL: https://projects.gnome.org/epiphany/.
[63] *Sitio web del navegador Google Chrome*. URL: http://www.google.com/chrome/.
[64] *Sitio web del navegador Internet Explorer*. URL: http://windows.microsoft.com/ie.
[65] *Sitio web del navegador Mozilla Firefox*. URL: http://www.mozilla.org/firefox/.
[66] *Sitio web del navegador Opera*. URL: http://www.opera.com/.
[67] *Sitio web del navegador Safari*. URL: http://www.apple.com/safari/.
[68] *Sitio web del navegador SeaMonkey*. URL: http://www.seamonkey-project.org/.
[69] *Sitio web del servicio de alojamiento Bitbucket*. URL: https://bitbucket.org/.
[70] *Sitio web del servicio de alojamiento GitHub*. URL: https://github.com/.
[71] *Sitio web del servidor HTTP Apache*. URL: http://httpd.apache.org/.
[72] *Sitio web del sistema de control de versiones CVS*. URL: http://www.cvshome.org/.
[73] *Sitio web del sistema de control de versiones Git*. URL: http://git-scm.com/.
[74] *Sitio web del sistema de control de versiones Mercurial*. URL: http://mercurial.selenic.com/.
[75] *Sitio web del sistema de control de versiones Subversion*. URL: http://subversion.apache.org/.
[76] *Sitio web del sistema operativo Microsoft Windows*. URL: http://windows.microsoft.com/.
[77] *Sitio web del sistema operativo para escritorios Ubuntu Linux*. URL: http://www.ubuntu.com/desktop.
[78] *Sitio web del World Wide Web Consortium (W3C)*. URL: http://www.w3.org/.

Apéndice A

Seguimiento y control

Este apéndice recoge y reflexiona sobre los sucesos acontecidos y los cambios que se han dado a lo largo de la vida del proyecto con respecto a lo inicialmente planificado en el documento de objetivos.

A.1. Objetivos

Se marcó como objetivo principal el desarrollo de una aplicación flexible y escalable que permitiera realizar consultas que repercutiesen sobre distintos depósitos de datos enlazados, objetivo que se ha cumplido con el desarrollo de `litτera`. Más adelante, en el apartado relativo al *alcance* se enumeran y detallan las características concretas de dicha aplicación, y su comparación con las previsiones iniciales.

Por otro lado, se marcaron como objetivos indirectos la familiarización del desarrollador con el campo de los datos enlazados y la adquisición de experiencia en el desarrollo web, en los procedimientos ingenieriles para el desarrollo de software, y en la edición de documentos con LaTeX.

La competencia (o al menos familiarización) resultante en el campo de los datos enlazados puede apreciarse en los análisis realizados para el desarrollo del sistema y que se recogen en el capítulo 3 de esta memoria.

En cuanto a la adquisición de experiencia en el desarrollo web se ha materializado no solo en la profundización de PHP (lenguaje que se conocía de antemano), sino también en el aprendizaje desde cero de bibliotecas y *frameworks* como jQuery y Bootstrap.

En el terreno de los procedimientos ingenieriles para el desarrollo de software, se han llevado a cabo diversas acciones que se consideran buenas prácticas, tales como la elaboración de pruebas unitarias (*unit tests*) o el uso de sistemas modernos de control de versiones.

Por último, la pericia adquirida en la edición de documentos con LaTeX puede apreciarse en esta misma memoria, teniendo en cuenta que al inicio del proyecto no se tenía ningún conocimiento de dicho sistema.

A.2. Alcance

En cuanto al alcance, se fijó como requisito mínimo la obtención e integración de los resultados de diferentes depósitos de datos, aspecto que se ha cumplido; al igual que la elaboración de una interfaz web compatible con los navegadores web más utilizados en la actualidad (la compatibilidad concreta con cada navegador y los porcentajes de mercado que eso supone puede consultarse en la sección 5.4). Por último se fijó la necesidad de federar al menos dos depósitos de datos, aspecto que se ha cumplido ya que se ha trabajado con cinco depósitos externos; pero lo que es más importante es que se ha hecho posible la adición dinámica de depósitos por parte del administrador sin tocar nada de código, lo cual dota al sistema de una gran flexibilidad.

Por otro lado, originalmente se planteaba como línea de ampliación el volcado de un depósito en una base de datos NoSQL local para su posterior acceso desde la aplicación. Al final esto se llevó un paso más allá y lo que se ha implementado ha sido un depósito local desde cero que puede poblarse desde la propia aplicación web, con lo que no solo se ha trabajado el acceso a los datos sino también las técnicas de inserción. Además, tener control total sobre el depósito que se estaba construyendo abrió un abanico de opciones que ha hecho posible, entre otras cosas, la implementación del algoritmo de reescritura de consultas para la búsqueda de diferentes tipos de obras.

A.3. Planificación temporal

En esta sección se evalúa el cumplimiento de los hitos marcados al inicio del proyecto y se analiza el coste temporal real que han tenido las diversas actividades realizadas.

A.3.1. Hitos

Para comenzar, cabe destacar que, aunque inicialmente se había planificado realizar el desarrollo en dos fases (diferenciando la parte de la infraestructura básica de la del depósito almacenado localmente), finalmente se decidió plantearlo como un todo, realizando un desarrollo incremental, pero sin distinguir ni poner líneas divisorias entre las dos fases originales. Es por ello que el hito del 2 de mayo en el que se esperaba finalizar el desarrollo del sistema y analizar la situación de cara a comenzar la ampliación del alcance careció de sentido.

El resto de hitos, sin embargo, se mantuvieron intactos, y su grado de cumplimiento se expone a continuación:

- El primero, que preveía la presentación de un prototipo funcional el 21 de marzo sobre el que recibir críticas y proponer cambios, se antojó demasiado ambicioso. Para entonces sí que se disponía de una versión funcional de la aplicación, pero se encontraba en una etapa muy temprana de desarrollo y no podía considerarse en ningún caso el prototipo que originalmente se había planificado.
- El siguiente hito (obviando, como se ha explicado, el divisorio entre las dos fases originales), era la terminación de la implementación y el inicio del proceso de documentación, fechado para el 3 de junio. Dicho hito se cumplió a rajatabla, ya que el 31 de mayo se recibió el visto bueno final de la aplicación, dándose así por terminada, y el 3 de junio comenzaron las tareas de redacción de la memoria.
- El hito relativo a la finalización de todos los entregables, sin embargo, no pudo cumplirse, ya que para dicha fecha (30 de junio) se tuvo una primera versión acabada de la memoria, pero faltaba aún el proceso de revisión para darla por finalizada. A pesar de todo, el margen de quince días con el que se contaba hizo que este pequeño retraso no supusiera mayor problema. La finalización de todos los entregables finalmente se logró completar el 9 de julio.

A.3.2. Actividades

En cuanto a las desviaciones temporales en las actividades, en el cuadro A.1 se recogen los costes temporales originalmente previstos frente a los reales.

Cabe hacer al respecto los siguientes comentarios:

Tarea	T. estimado	T. real
Planificación y gestión del proyecto	**50 h.**	**29.5 h.**
Planificación inicial	10 h.	10 h.
Seguimiento y control	20 h.	12.5 h.
Replanificaciones	10 h.	0 h.
Reuniones	10 h.	7 h.
Formación	**100 h.**	**95,5 h.**
Formación general en datos enlazados	40 h.	25 h.
Formación en el dominio elegido	10 h.	30 h.
Formación en control de versiones	10 h.	2 h.
Búsqueda y análisis del trabajo existente	10 h.	8,5 h.
Formación en OpenLink Virtuoso	30 h.	8,5 h.
Análisis y formación en otras tecnologías	-	21,5 h.
Desarrollo	**210 h.**	**203,5 h.**
Análisis y diseño	20 h.	31.5 h.
Preparación del entorno	10 h.	10,5 h.
Programación de la infraestructura	100 h.	61 h.
Federación del depósito local	50 h.	38 h.
Maquetación web	10 h.	25 h.
Elementos dinámicos	-	25,5 h.
Pruebas	10+10 h.	12 h.
Documentación	**80 h.**	**99 h.**
Búsqueda y lectura de modelos	10 h.	4,5 h.
Manual de usuario	10 h.	11 h.
Resto de la memoria	40 h.	61.5 h.
Revisión y corrección de errores	10 h.	12 h.
Defensa	10 h.	*10 h.*

Cuadro A.1: Coste temporal estimado frente al coste real

- A pesar de los cambios que se fueron dando a lo largo del proyecto, no se vio la necesidad de hacer una nueva planificación, por lo que no se consumió ninguna de las horas previstas para esa tarea.
- *Formación en el dominio elegido* incluye la búsqueda de depósitos de datos enlazados, el análisis de sus características, etc. En ocasiones era difícil dilucidar si lo que se estaba haciendo era un análisis/formación en ellos, o una formación general en los datos enlazados. Dicho de otra forma, hubo multitud de aspectos generales de los datos enlazados que se aprendieron mediante la formación y el análisis de los depósitos de datos del dominio. Por ello, más que razonar cada elemento de forma independiente, conviene combinarlos y ver cómo las 50 horas inicialmente previstas (40+10) finalmente ascendieron a 55 (25+30), por lo que la estimación inicial fue en conjunto muy acertada.
- La *formación en control de versiones* ya se sabía de antemano que sería una tarea que consumiría poco tiempo. Sin embargo, al haber realizado todas las estimaciones en módulos de 10 horas, se decidió asignarle a la tarea un módulo completo, en vez de las 5-7 horas que realmente se creía que se iba a tardar. Sin embargo, finalmente el aprendizaje de su uso resultó trivial, consumiendo poco más de 2 horas. Si se hubiese sabido esto desde el principio, se habría obviado la tarea, agrupándola en alguna categoría más general.
- La formación en OpenLink Virtuoso consumió bastantes menos horas de las previstas (8,5 frente a 30) principalmente por la documentación existente (al menos para hacer las tareas básicas de instalación, configuración e inserción y recuperación de datos, que son los que se han trabajado en este proyecto), y la colaboración de Ana Torre, que ayudó resolviendo dudas sobre el uso y las características de la aplicación, y que en otro caso hubiera llevado a pequeños bloqueos que habrían terminado consumiendo unas cuantas horas más.
- No se previó una tarea de análisis y formación en otras tecnologías a utilizar, que como puede verse ha consumido la nada desdeñable cifra de 21,5 horas. En ese cómputo se incluye principalmente el análisis de tecnologías y el aprendizaje de jQuery y Bootstrap, así como unas pocas horas consumidas en el análisis y formación en *frameworks* PHP (que luego no repercutiría en el proyecto porque se descartó su uso).
- *Preparación del entorno* incluye la instalación y configuración de OpenLink Virtuoso que fue, de hecho, la subtarea más costosa dentro de esa categoría.

- Se han desarrollado diversos elementos dinámicos en la interfaz de la aplicación. Algunos de ellos son meramente cosméticos (como animaciones al insertar y eliminar elementos en el formulario de búsqueda), pero otros muchos son intrínsecos a la funcionalidad de la aplicación. En cualquier caso, se ha considerado interesante diferenciar el coste de dichos elementos. Así, el desarrollo de la infraestructura básica (código ejecutado en el servidor) se recoge dentro de *programación de la infraestructura*, mientras que las funcionalidades dinámicas (código ejecutado en el cliente) se recogen dentro del nuevo epígrafe *elementos dinámicos*. Por ello, para comparar el coste real con respecto al estimado, lo más justo es sumar ambas categorías, con lo que se obtiene que el desarrollo de la parte básica de la infraestructura ha llevado 86,5 horas frente a las 100 estimadas. Por último se debe destacar la ayuda de Gorka Maiztegi, quien resolviendo diversas dudas de programación ha hecho que se ahorren unas cuantas horas en este apartado.

- *Federación del depósito local* incluye tanto el soporte para la inserción y consulta de datos en un depósito propio de OpenLink Virtuoso como el algoritmo para la reescritura de consultas que hace posible hacer búsquedas utilizando ontologías que no pertenecen al depósito en cuestión. Los trabajos relativos a esta última característica se han agrupado dentro de esta categoría primero porque se planteó en la etapa final del proyecto, y luego porque ha sido posible solamente gracias a la presencia de dicho depósito, ya que en otro caso no se habría dispuesto de los depósitos y ontologías necesarias para hacer las pruebas correspondientes.

- Se estimó que las tareas de *maquetación web* podrían realizarse en 10 horas, ante lo cual ha habido una gran desviación. El uso de un *framework* CSS redujo mucho el tiempo invertido en esta tarea y aumentó enormemente la calidad resultante con respecto a si no se hubiese utilizado, pero dicha reducción en el coste no llegó hasta los límites que se habían previsto. Además, las tareas de maquetación asociadas a la federación del depósito local hicieron que la cifra aumentase aún más hasta situarse en las 25 horas.

- En cuanto a las *pruebas*, se previeron 10 para cada una de las dos fases, cuando finalmente solo se necesitaron unas 12 horas en total. La cifra de este apartado depende mucho de la forma de cómputo, ya que solo se tuvieron en cuenta las sesiones intermedias y finales de pruebas y no los pequeños intervalos que se realizaban según se iba desarrollando la infraestructura.

- No se ha imputado al proyecto el coste de confeccionar la plantilla que utiliza esta memoria, que ha supuesto un total de 16 horas.
- *Manual de usuario* incluye la redacción de los dos anexos de esta memoria: el manual de instalación y administración y el propio manual de usuario, aunque finalmente este último se excluyó del entregable final.
- Las tareas asociadas a la *defensa* del proyecto (preparación de la presentación, grabación del vídeo, ensayos y la propia defensa final) no se han llevado a cabo a la hora de entregar esta memoria, por lo que las 10 horas que se indican continúan siendo una estimación. Por eso están marcadas en cursiva en la columna relativa al tiempo real invertido.

A.4. Comunicaciones

Se celebraron las siguientes reuniones de seguimiento entre los interesados en el proyecto (el desarrollador Alberto y la directora Arantza). Todos las fechas que se indican corresponden al año 2013:

- **24 de enero** Inicio del proyecto.
- **31 de enero** Revisión de la planificación inicial.
- **18 de febrero** Selección de los depósitos iniciales y comentario de artículos.
- **25 de febrero** Primer análisis sobre las diferentes ontologías utilizadas, y comentarios generales.
- **7 de marzo** Definición de la arquitectura y de los módulos que forman la aplicación.
- **15 de marzo** Diseño de clases.
- **26 de marzo** Muestra de la primera versión funcional de la aplicación.
- **10 de abril** Aumento de la complejidad de las consultas mediante operadores SPARQL.
- **24 de abril** (asistencia de Jesús Bermúdez) Diálogo sobre las wikis semánticas y su encaje dentro del desarrollo.
- **8 de mayo** Propuesta de implementación de un algoritmo para la reescritura de consultas utilizando diferentes ontologías.

- **22 de mayo** Muestra del trabajo realizado, recibiendo unos últimos comentarios y propuestas de mejora.
- **31 de mayo** (asistencia de Jesús Bermúdez) Revisión de la versión completa de la aplicación, y recepción del visto bueno final para dar por concluida la implementación.
- **4 de julio** Corrección de la memoria.

A.5. Riesgos

A continuación se enumeran los riesgos que se especificaron en el plan inicial, comentando hasta qué punto sucedieron o se consiguieron evitar:

- **Retrasos** Se dieron importantes retrasos, sobre todo al inicio del proyecto, cuando la dedicación a él no era completa y el tiempo de adaptación al mismo y su dinámica llevó varias semanas. Sin embargo, compensándolo con una mayor dedicación a partir de entonces, se logró reconducir la situación y cumplir con los hitos originalmente fijados, tal y como se ha expuesto anteriormente.
- **Planificación inicial inadecuada** La planificación resultó bastante más acertada de lo que en un principio se esperaba, por lo que este riesgo no supuso ningún problema real.
- **Diseño inadecuado** Aunque el diseño fue variando a lo largo del desarrollo del proyecto, las necesidades fueron detectadas y atajadas a tiempo, sin darse ningún caso en el que se tomasen decisiones incorrectas y a la vez importantes que hipotecasen el desarrollo posterior.
- **Requisitos definidos incorrectamente** Al igual que en el caso anterior, los cambios en los requisitos fueron incrementales y no dramáticos, por lo que tampoco hubo mayores problemas derivados de este riesgo.
- **Pérdida de información** El sistema de copias de seguridad fue estricto (copias diarias tanto en el caso del código de la aplicación como en el de la memoria), por lo que, en caso de catástrofe, se habría perdido como máximo el trabajo de un día. En cualquier caso, la situación no se dio y no hubo que hacer uso de dichas copias.
- **Baja por enfermedad** Afortunadamente no hubo bajas por enfermedad de más de un día.

Apéndice B

Manual de instalación y administración

Este manual es un una guía de instalación y administración de la aplicación web litτera que cubre la instalación del entorno necesario y las configuraciones a realizar para su puesta en funcionamiento.

Como sistema operativo sobre el que realizar la explicación se ha elegido Ubuntu Linux[77] (cuya última versión a la hora de redactar este manual es la 13.04), aunque las herramientas que se exponen son multiplataforma y no hay ningún impedimento en instalarlas en otros sistemas operativos. Merece una mención aparte OpenLink Virtuoso[43], ya que su instalación en Windows[76] no es trivial, por lo que se proporciona una referencia donde se detalla paso a paso cómo hacerlo.

B.1. Preparación del entorno

litτera requiere de un servidor web (preferiblemente Apache HTTP Server[71]), PHP5[58] y la biblioteca libcurl[18] para funcionar. Adicionalmente, en caso de querer activar el soporte para el depósito local (lo cual no es estrictamente necesario para que la aplicación funcione), debe instalarse OpenLink Virtuoso[43].

Afortunadamente, todos esos paquetes están disponibles directamente en los repositorios de Ubuntu, por lo que para su instalación basta con ejecutar el comando que se muestra en la figura B.1. Durante la instalación de OpenLink Virtuoso se solicitará una contraseña para el usuario administrador. Como es

natural, se debe recordar esta contraseña para poder configurar el servidor más adelante.

```
sudo apt-get install apache2 php5 php5-curl virtuoso-opensource
```

Figura B.1: Comando para la instalación de los paquetes necesarios en Ubuntu

La instalación de Apache, PHP5 y libcurl en Windows es sencilla y, en el improbable caso de surgir algún problema, existe numerosa información en la red al respecto. Por otro lado, la instalación de OpenLink Virtuoso debe hacerse descargando los binarios precompilados para Windows y siguiendo atentamente las instrucciones detalladas en [15].

En caso de que la instalación de los paquetes no haya generado ningún error, el sistema estará ya en condiciones de ejecutar `litτera`. Si por el contrario ha ocurrido algo inesperado y se sospecha que los paquetes pueden no haberse instalado correctamente, puede comprobarse fácilmente de la forma que se expone a continuación.

Crear el fichero `/var/www/info.php` que contenga el código que se muestra en la figura B.2. Una vez hecho, escribir en tres ventanas de cualquier navegador web las URLs `http://localhost/`, `http://localhost/info.php` y `http://localhost:8890/sparql` y comprobar los siguientes extremos:

1. Que al acceder a la primera URL se muestra una página web titulada *It works!*, lo cual certifica la correcta instalación de Apache HTTP Server.
2. Que al acceder a la segunda URL se muestra una página web encabezada por el título *PHP Version* seguida de un número (que debería ser igual o superior a 5.3) y el logo de PHP, lo cual certifica la correcta instalación del intérprete de este lenguaje.
3. Que en dicha segunda página existe un apartado llamado *curl* cuya primera fila dice *cURL support enabled*, lo cual certifica la presencia de la biblioteca libcurl.
4. Que al acceder a la tercera URL se muestra una página web titulada *Virtuoso SPARQL Query Editor*, lo cual certifica la correcta instalación de OpenLink Virtuoso.

En caso de que alguna de las cuatro comprobaciones falle, se recomienda reiniciar la máquina y volver a repetir las comprobaciones.

```
1 <?php
2 phpinfo();
3 ?>
```

Figura B.2: Código para comprobar la instalación y configuración de PHP

B.2. Puesta en marcha

La puesta en marcha de `litτera` es inmediata ya que solo requiere colocar el código fuente en la carpeta correspondiente del servidor web utilizando, por ejemplo, el comando que se muestra en la figura B.3. Con esto `litτera` ya estará accesible y en funcionamiento, excepto la parte relativa al depósito local, que para ponerlo en marcha se deben seguir los pasos que se describen a continuación.

```
sudo mv littera/ /var/www/
```

Figura B.3: Comando para mover el código de `litτera` al servidor web

B.2.1. Configuración de OpenLink Virtuoso

Se necesita crear en OpenLink Virtuoso un nuevo usuario con permisos de escritura para la inserción de datos mediante WebDAV. Para ello:

1. Abrir la URL `http://localhost:8890/conductor/` a través de cualquier navegador web.
2. En el formulario de la izquierda, escribir *dba* junto con la contraseña elegida en el momento de la instalación de Virtuoso.
3. Navegar a través de los menús pinchando en *System Admin* → *User accounts* → *Create New Account*. Aparecerá en pantalla un formulario similar al que se muestra en la figura B.4.
4. Escribir en *User Login* cualquier nombre de usuario que no esté en uso, y en *Password* y *Confirm Password* una contraseña que sea segura.
5. En el desplegable *User type* seleccionar *SQL/ODBC and WebDAV* y marcar la casilla *create* que está próxima al campo *DAV Home Path*
6. En *Account Roles* seleccionar *SPARQL_ UPDATE* y pinchar sobre el botón >>.

7. Una vez que el formulario tenga un aspecto equivalente al que se muestra en la figura B.4, pulsar sobre *Save* para guardar los resultados.

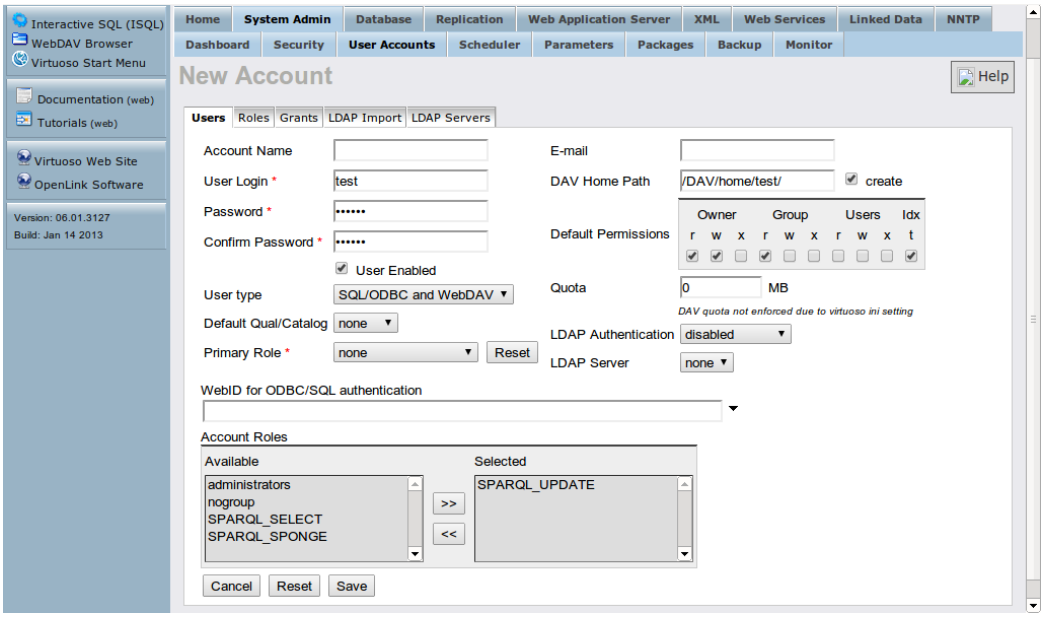


Figura B.4: Formulario para la creación de una nueva cuenta en Virtuoso

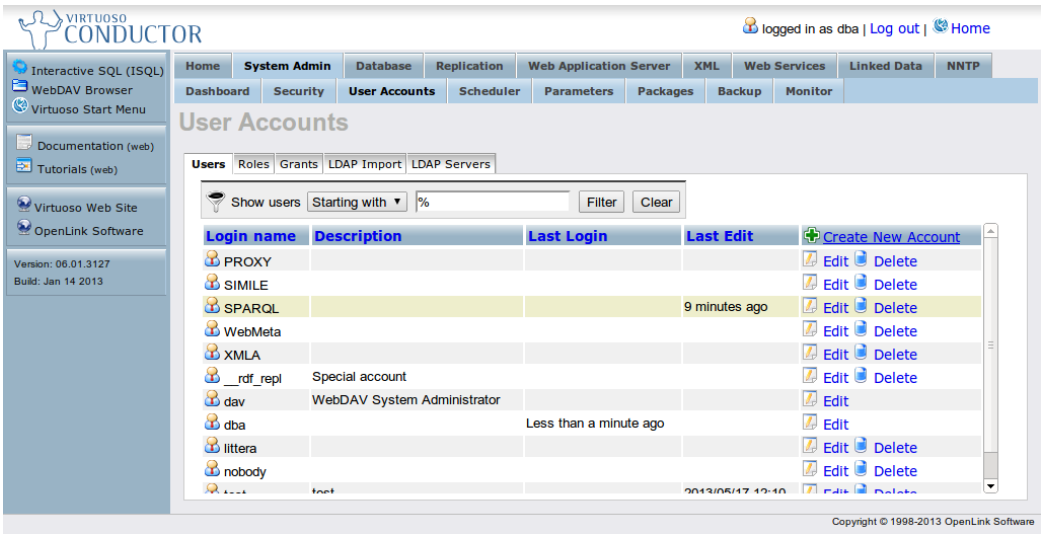


Figura B.5: Lista de usuarios definidos en OpenLink Virtuoso

Ahora se debe definir la ontología que se utilizará para categorizar los diferentes tipos de obras escritas. Para hacer esto, lo más cómodo es introducir la consulta SPARUL en el formulario de ejecución que proporciona el panel

de administración de Virtuoso. Sin embargo, y a pesar de estar identificados como administradores, se debe asignar temporalmente permisos de escritura a un usuario especial, ya que mientras tanto no se disponen de privilegios suficientes para insertar (o modificar) datos de esta forma.

Para ello, hay que localizar en la lista de usuarios (figura B.5) al usuario *SPARQL* y pulsar sobre su correspondiente botón *Edit*, lo cual desplegará una ventana análoga a la mostrada en la figura B.4. En ella se debe seleccionar *SPARQL_UPDATE* dentro de la columna izquierda de *Account Roles*, y pulsar sobre el botón >>. Seguidamente pinchar sobre *Save* para guardar los cambios.

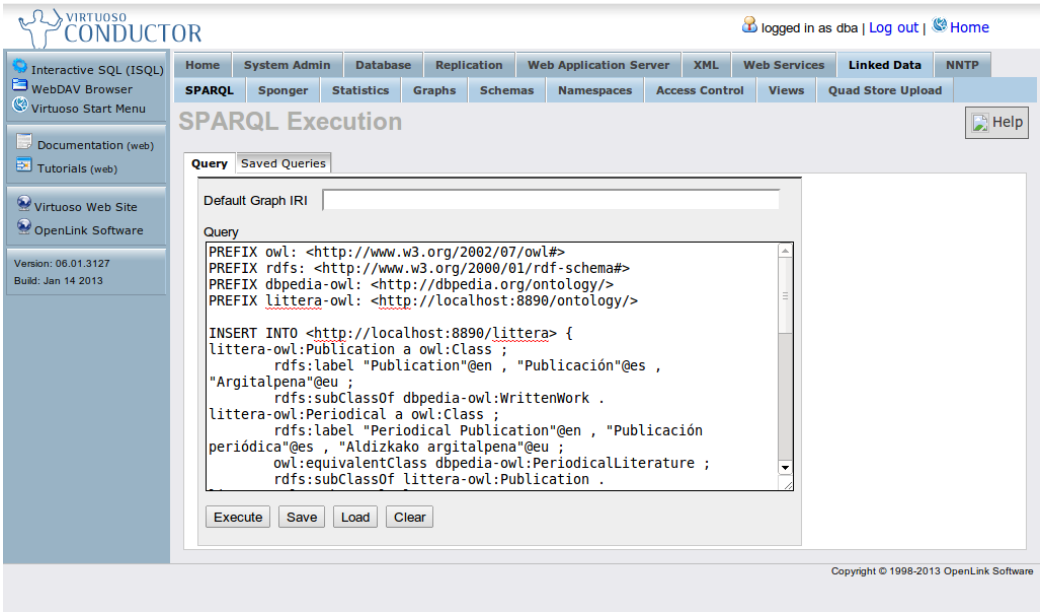


Figura B.6: Formulario para la ejecución de consultas SPARQL

Pulsar sobre la pestaña *Linked Data* e insertar en el campo de texto que aparece (figura B.6) la consulta SPARUL que cree la jerarquía deseada. En la figura B.7 se muestra el código que genera la utilizada por el autor en los ejemplos de la memoria, aunque el sistema no está limitado a la misma. Una vez relleno el formulario, pulsar sobre *Execute* para ejecutar la consulta y grabar los datos. Por último solo queda volver a retirar por cuestiones de seguridad los permisos de escritura realizando de nuevo lo expuesto en el párrafo anterior pero pulsando el botón << en vez de >>.

Una vez terminado el proceso, pinchar sobre *Log out* (en la esquina superior derecha) para cerrar la sesión y salir del panel de administración de OpenLink Virtuoso.

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
4  PREFIX littera-owl: <http://localhost:8890/ontology/>
5
6  INSERT INTO <http://localhost:8890/littera> {
7  littera-owl:Publication a owl:Class ;
8      rdfs:label "Publication"@en , "Publicación"@es , "Argitalpena
           "@eu ;
9      rdfs:subClassOf dbpedia-owl:WrittenWork .
10 littera-owl:Periodical a owl:Class ;
11     rdfs:label "Periodical Publication"@en , "Publicación perió
           dica"@es , "Aldizkako argitalpena"@eu ;
12     owl:equivalentClass dbpedia-owl:PeriodicalLiterature ;
13     rdfs:subClassOf littera-owl:Publication .
14 littera-owl:Book a owl:Class ;
15     rdfs:label "Book"@en , "Libro"@es , "Liburua"@eu ;
16     owl:equivalentClass dbpedia-owl:Book ;
17     rdfs:subClassOf littera-owl:Publication .
18 littera-owl:NonFiction a owl:Class ;
19     rdfs:label "Non-Fiction"@en , "Realidad"@es , "Errealitatea"
           @eu ;
20     rdfs:subClassOf littera-owl:Book .
21 littera-owl:Fiction a owl:Class ;
22     rdfs:label "Fiction"@en , "Ficción"@es , "Fikzioa"@eu ;
23     rdfs:subClassOf littera-owl:Book .
24 littera-owl:Fantasy a owl:Class ;
25     rdfs:label "Fantasy"@en , "Fantasía"@es , "Fantasia"@eu ;
26     rdfs:subClassOf littera-owl:Fiction .
27 littera-owl:ScienceFiction a owl:Class ;
28     rdfs:label "Science Fiction"@en , "Ciencia ficción"@es , "
           Zientzia-fikzioa"@eu ;
29     rdfs:subClassOf littera-owl:Fiction .
30 littera-owl:Mistery a owl:Class ;
31     rdfs:label "Mistery"@en , "Misterio"@es , "Misterioa"@eu ;
32     rdfs:subClassOf littera-owl:Fiction .
33 littera-owl:Humor a owl:Class ;
34     rdfs:label "Humor"@en , "Comedia"@es , "Komedia"@eu ;
35     rdfs:subClassOf littera-owl:Fiction .
36 }
```

Figura B.7: Código SPARQL para la creación de una ontología

B.2.2. Configuración de litτera

Crear en la carpeta /endpoints/ un archivo .json que contenga el nombre y los datos básicos del depósito local siguiendo los pasos descritos en la sección B.3, exactamente igual que si se tratase de un depósito externo.

Si es que se ha utilizado la ontología de la figura B.7, puede ser más rápido aprovechar el ejemplo littera.json.example que se proporciona, renombrándolo a *cualquier_cosa*.json y modificándolo a partir de ahí.

Una vez definido el depósito, abrir el archivo /config.php con cualquier editor de texto plano, quitar el comentario inicial (//) de las tres líneas de código existentes en él, y configurarlo de la siguiente manera:

1. Como endpoint especificar la misma URL que en el archivo de configuración .json que se acaba de definir.
2. Como user y pass, especificar el nombre de usuario y la contraseña elegidos al crear el usuario de OpenLink Virtuoso en la subsección anterior.

B.3. Adición de nuevos depósitos

litτera proporciona una forma sencilla y dinámica de añadir y eliminar depósitos de datos con los que interactuar. Este método se basa en la creación de un archivo de configuración para cada depósito, evitando la necesidad de escribir o modificar código. Dichos archivos deben colocarse en la carpeta /endpoints/, tener extensión .json, y contener la información que se expone a continuación. Cumplidos estos tres requisitos, litτera los detectará y cargará automáticamente.

Se proporciona un ejemplo, littera.json.example (que se reproduce en la figura B.8), sobre el que se recomienda aplicar los cambios oportunos en vez de empezar desde un archivo en blanco, ya que simplificará la tarea y minimizará los errores de sintaxis.

Los campos que debe contener cada archivo de configuración son los siguientes:

- name nombre del depósito de datos
- endpoint URL del *SPARQL endpoint*

- `resource` parte común de las URIs de las instancias que almacena
- `vocab` *mapping* del vocabulario utilizado para designar los términos que se enumeran a continuación:
  - `type_author` *obligatorio* tipo escritor
  - `author_name` *obligatorio* nombre completo del escritor
  - `author_first_name` *opcional* nombre de pila del escritor
  - `author_last_name` *opcional* apellido(s) del escritor
  - `author_birth` *opcional* año (o fecha completa) de nacimiento del escritor
  - `author_death` *opcional* año (o fecha completa) de fallecimiento del escritor
  - `author_birth_death` *opcional* año de nacimiento y fallecimiento del escritor; carente sentido si ya se han definido los dos términos anteriores
  - `type_book` *obligatorio* tipo más general, dentro de los trabajos escritos, con el que están categorizados los documentos del depósito
  - `book_author` *obligatorio* término que relaciona un documento con su autor
  - `book_name` *obligatorio* título del documento
- `work` ontología utilizada para categorización de los trabajos escritos en el depósito; en caso de no existir tal jerarquía, puede especificarse uno solo (el único término que se utilice)
- `pref` listado de prefijos utilizados en la definición de `vocab` y `pref`, indicando a cada uno de ellos qué IRI le corresponde.

```
{
    "name": "littera",
    "endpoint": "http://localhost:8890/sparql",
    "resource": "http://localhost/resource/",
    "vocab": [
        { "type": "type_author", "word": "dbpedia-owl:Writer" },
        { "type": "author_name", "word": "foaf:name" },
        { "type": "author_first_name", "word": "foaf:givenName" },
        { "type": "author_last_name", "word": "foaf:familyName" },
        { "type": "author_birth", "word": "dbpprop:birthYear" },
        { "type": "author_death", "word": "dbpprop:deathYear" },
        { "type": "type_book", "word": "dbpedia-owl:WrittenWork" },
        { "type": "book_author", "word": "dcterms:creator" },
        { "type": "book_name", "word": "dcterms:title" }
    ],
    "work": [
        { "type": "dbpedia-owl:WrittenWork" },
        { "type": "littera-owl:Publication" },
        { "type": "littera-owl:Periodical" },
        { "type": "littera-owl:Book" },
        { "type": "littera-owl:NonFiction" },
        { "type": "littera-owl:Fiction" },
        { "type": "littera-owl:Fantasy" },
        { "type": "littera-owl:ScienceFiction" },
        { "type": "littera-owl:Mistery" },
        { "type": "littera-owl:Humor" }
    ],
    "pref": [
        { "prefix": "dbpprop", "iri": "http://dbpedia.org/property/" },
        { "prefix": "dbpedia-owl", "iri": "http://dbpedia.org/ontology/" },
        { "prefix": "foaf", "iri": "http://xmlns.com/foaf/0.1/" },
        { "prefix": "dcterms", "iri": "http://purl.org/dc/terms/" },
        { "prefix": "littera-owl", "iri": "http://localhost:8890/ontology/" }
    ]
}
```

Figura B.8: Archivo de configuración de un depósito de datos