ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PORTAL DE COLABORACIÓN CON CAPACIDADES SEMÁNTICAS

Realizado por
ELENA LOZANO ROSCH

Dirigido Por
DIEGO R. LÓPEZ
Y
MANUEL VALENCIA

Departamento
DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Sevilla, Septiembre de 2007

Índice general

Índice de figuras

Capítulo 1

Introducción

El **Portal de Colaboración con Capacidades Semánticas** es una aplicación web que trata de establecer un punto de unión entre los académicos y los investigadores con los desarrolladores de Software Libre. Pretende ser un portal de fomento de desarrollos libres para la comunidad de **RedIRIS**[1] y se espera que este proyecto se lance durante el próximo curso académico bajo la iniciativa del grupo de trabajo dentro de la comunidad de RedIRIS, dedicado al estudio y desarrollo del software libre: IRIS-Libre.

Pretende, también, solucionar un problema existente entre los investigadores y académicos con conocimientos no muy elevados de informática: si requieren una aplicación de algún tipo, necesitarían aprender a programar ellos mismos, lo cual no es viable en muchos de los casos.

En el caso de que un investigador quisiera una aplicación libre, daría de alta el requerimiento describiendo los requisitos funcionales y técnicos de la misma (Esto es lo que posteriormente llamaremos *Idea* en el modelo de datos.(5.3)).

Los investigadores o docentes registrados podrán "votar a favor" los proyectos e ideas que crean interesantes o que prefieran que hagan antes los desarrolladores. Además, pueden establecerse como

[1] RedIRIS es la red académica y de investigación nacional, patrocinada por el Plan Nacional de I+D+I y que está gestionada por Red.es

persona de contacto a la hora de que los desarrolladores les puedan preguntar dudas acerca de aspectos técnicos del mismo.

Gracias a esta aplicación, los desarrolladores que quieran colaborar en proyectos de Software Libre tendrán la posibilidad de hacerlo sin tener que inventar ellos una idea, pudiendo elegir alguna de las que ya estén propuestas.
Una vez que un desarrollador elige un proyecto, éste se creará en la Forja[2] y estará listo para empezar a trabajar en él.
Este tipo de software ya existe en organizaciones como la NASA, la cual tiene un pequeño programa para incentivar la unión entre investigadores y desarrolladores.
Como complemento a esta idea se han utilizado tecnologías relacionadas con la Web Semántica, las cuales analizaremos con detenimiento en el capítulo 2.

El motivo por el cual se han utilizado estas tecnologías es realizar un tratamiento de los datos distinto al usual. En vez de tratar con la sintaxis de las palabras, se trata con la **semántica,** es decir, el *significado* de las mismas. Ésto permite realizar unas búsquedas y un tratamiento de la información mucho más eficiente y real.

Gracias a las diversas ontologías, vocabularios y esquemas existentes en el ámbito de la Web Semántica es posible no solo representar la información de una forma mucho más coherente, sino que podemos utilizar las propiedades que éstas tienen para conseguir hacer referencia a otros datos, consiguiendo de esta forma que se de más significado a la relación y sea un aspecto más a la hora de analizar los datos de forma semántica.

En este documento podremos ver, en primer lugar en qué consiste la Web Semántica (Capítulo 2) y cuáles son sus objetivos.Tras esto pasaremos a analizar qué es una Ontología (Capítulo 3), cuáles

[2]La *Forja de Conocimiento Libre de la Comunidad RedIRIS* (http://www.forja.rediris.es) tiene como objetivo principal fomentar los desarrollos de software libre en la comunidad RedIRIS así como servir de soporte a iniciativas de interés en el entorno académico-científico relacionadas con el conocimiento libre.

son sus aspectos fundamentales y profundizaremos además en las tecnologías y lenguajes que han sido utilizados en este proyecto.

El siguiente punto que se analiza es la herramienta utilizada para optimizar el desarrollo de la Aplicación Web que es la base estructural del proyecto: Symfony (Capítulo 4).

Posteriormente entraremos de lleno en la descripción y el análisis de cómo han sido estructurados e implementados los aspectos analizados en los capítulos anteriores en el proyecto: Modelo de datos concretos, formas de instalación de los elementos necesarios para su funcionamiento, uso de las ontologías y su aplicación, etc.

Por último, en el capítulo 6, nos encontramos con las conclusiones a las que se han llegado tras la realización de este proyecto y el futuro que se le pretende dar al mismo.

Capítulo 2

Web Semántica

La Web Semántica es una evolución de la World Wide Web que la extiende de forma en que el contenido de las páginas web no está expresado sólo en lenguaje natural, si no que está formateado de tal forma que ciertos motores y agentes software son capaces de leerlo e interpretarlo.

Una de las causas que hizo posible el surgimiento de la Web Semántica fueron los problemas existentes en la especificación HyperText Markup Language (HTML) con respecto a la pocas posibilidades para categorizar los elementos que configuran el texto mas allá de las típicas funciones estructurales de otros lenguajes. Con HTML podíamos visualizar los datos, pero no podíamos relacionarlos con otros, obteniendo una sobrecarga y heterogeneidad de la información.
Por ésto, el objetivo de esta evolución era el poder compartir, buscar, relacionar e integrar los datos y contenidos de la Web de una forma mucho más coherente y eficiente a la hora de realizar interacciones con los mismos.

La forma de representar los datos que surgió de esta idea consiste en dar a los *metadatos* (datos que describen los recursos web) ciertas propiedades que describen su contenido, significado y relación con otros metadatos.

También se decidió que para que hubiera capacidad de interacción entre los motores que evaluaban la información semántica, la representación de los datos debía ser dada de manera formal, es decir,

debía cumplir unas especificaciones que hubieran sido establecidas previamente. En nuestro caso, el W3C (*World Wide Web Consortium, Consorcio Internacional que produce estándares para la World Wide Web*) se está encargando de estandarizar casi todos los formatos que se han desarrollado a partir de entonces.

Gracias a la Web Semántica:

- Es posible definir la información de una forma más precisa.
- Se ha dotado a la Web de mayor significado.
- Se ha mejorando la efectividad de las búsquedas realizadas.
- Se permite compartir, procesar, realizar deducciones lógicas y razonamientos y transferir la información de una forma sencilla.
- Aumenta la usabilidad y el aprovechamiento de los recursos Web

Esta web extendida y basada en el significado, se apoya en lenguajes que resuelven los problemas de una web carente de semántica. Estos lenguajes, denominados *ontologías*, los analizaremos en el siguiente capítulo. (3).

Capítulo 3

Ontologías

3.1. Introducción

En la informática se llama ontología a la fórmula exhaustiva y rigurosa del esquema conceptual de un dominio de forma que se facilite con ésta, la relación de la información entre los sistemas software existentes.

El concepto de ontología no es sólo utilizado en la Web Semántica; se extiende a más campos de la informática como la inteligencia artificial o la ingeniería del software, pero su base es la misma en todos los campos; tener la capacidad de describir:

- Objetos a bajo nivel.
- Clases, colecciones y tipos de objetos.
- Atributos, propiedades y características de los objetos que representan.
- Relaciones entre distintos objetos que definen.
- Eventos que hacen que las relaciones cambien entre los objetos que representan.

Por este motivo las ontologías están relacionadas con vocabularios fijos que describen y definen la información. Debido a que realizar un sólo vocabulario que englobe todo el dominio de la infor-

mación y sea capaz de definirlo correctamente es inviable, se han creado esquemas específicos para distintos dominios de información que son mucho más eficientes a la hora de tratar la información.

Los estándares más conocidos en la actualidad son *Resource Description Framework* (RDF) y *Ontology Web Language* (OWL). Éstos dos estándares ayudan a convertir la Web en una infraestructura global en la que es posible compartir, y reutilizar datos y documentos entre diferentes tipos de usuarios.

- Resource Description Framework (RDF), en español Marco de Descripción de Recursos, proporciona información descriptiva simple sobre los recursos que se encuentran en la Web. Éste lenguaje de descripción es el que se ha utilizado en este proyecto, por lo que se describirá más ampliamente en un apartado posterior.

- Ontology Web Language (OWL), es un lenguaje de marcado que se usa para definir temas o vocabularios específicos en los que asociar esos recursos a la información y semántica que contienen.

  Está diseñado para ser utilizado por aplicaciones que necesitan procesar el contenido de la información.

  Está basado en un lenguaje anterior denominado DAML + OIL. Éste proviene de la conjunción de *DARPA*[1]*Agent Markup Language* (*DAML*, en español *lenguaje DARPA de marcado*), y de *Ontology Interchange Language* (*OIL*, en español *Lenguaje de Intercambio de Ontologías*).

  En la actualidad OWL es una *recomendación* de W3C, ésto significa que OWL está en la etapa final del proceso de ratificación el W3C para llegar a ser un estándar.

  OWL es una de las principales tecnologías que implementan la Web Semántica debido al ámbito tan amplio en el que trabaja y la gran capacidad de definición que tiene.

Ambas ontologías necesitan, para su desarrollo una *estructura* que les permita ser reconocidas y

[1]DARPA, *the Defense Advanced Research Projects Agency*, es una agencia del Departamento de Defensa de los Estados Unidos responsable del desarrollo de nuevas tecnologías para uso militar. DARPA fue responsable por ejemplo, de desarrollar la precursora de Internet: ARPANET.

tratadas en todos los sistemas y motores que haya en la World Wide Web. Esta *estructura* o lenguaje básico que hace posible las ontologías es el eXtensible Markup Language o más comunmente conocido como XML (*XML* es, en español, *lenguaje de marcas extensible*).

El XML es un *metalenguaje*, es decir, un lenguaje usado para hacer referencia a otros lenguajes, que es extensible debido a su estructura basada en *tags* o *etiquetas*. Lo desarrolló el W3C.

Lo que ha convertido a XML en pionero en las aplicaciones de la Web Semántica, es la simplicidad, eficiencia y versatilidad que lo caracterizan.

Si quisiéramos, por ejemplo, describir un libro con atributos como título o autor, podríamos hacer un XML como el siguiente:

```
1  <libro>
2      <titulo>Las intermitencias de la muerte</titulo>
3      <autor>Jose Saramago</autor>
4  </libro>
```

Ésto parece (y es) bastante sencillo, pero no debemos olvidar dos puntos importantes:

- Hay que seguir el estándar y cumplir las normas de creación de XML establecidas por el W3C al hacer nuestros propios XML, para así hacer posible que la información pueda ser tratada por los distintos motores (ya sean navegadores o librerías de los distintos lenguajes).
- Aunque podamos hacer modelos tan sencillos como la definición básica de un libro, es posible también realizar modelos y estructuras mucho más complejos modificando la forma de definir la información.

Lo que diferencia a XML de las principales ontologías que estamos estudiando es la estandarización de las *etiquetas* o *tags* que se utilizan, puesto que tanto en RDF como en OWL el metalenguaje utilizado está aprobado por el W3C y debe estar especificado y regulado para que se garantice la

capacidad de interacción de la información entre distintos sistemas software.

Podemos decir, por tanto, que XML es una de las principales herramientas en la Web Semántica debido a que:

- Es una tecnología sencilla, que se hace compleja en el caso de ser necesario gracias a su extensibilidad. Prueba de ello son las distintas tecnologías que han ido surgiendo a partir de él:
  - **XSL:** Lenguaje Extensible de Hojas de Estilo.
  - **XPath:** Lenguaje de Rutas XML.
  - **XLink:** Lenguaje de Enlace XML.
  - **XPointer:** Lenguaje de Direccionamiento XML.
  - **XQL:** Lenguaje de Consulta XML.
- No sólo sirve para aplicarlo en Internet, sino que se propone como estándar para el intercambio de información de una forma estructurada para todo tipo de plataformas, ya sean bases de datos, aplicaciones de escritorio, editores de texto, etc.
- Guarda similitudes con HTML, por lo que es algo cercano a la mayoría de los desarrolladores.
- Es fiable y seguro en la forma que tiene de intercambiar la información.
- Es modular, lo cual lo hace mucho más legible.
- Tiene además algunas extensiones que son a día de hoy, tecnologías muy utilizadas .

3.2. RDF

Como se dijo anteriormente, RDF es la ontología utilizada para definir el modelo de datos semántico de este proyecto. A continuación pasaremos a estudiarla en profundidad.

3.2.1. Definiciones y conceptos básicos

RDF son las siglas de **Resource Description Framework**. Éste es un estándar cuyo objetivo es describir los recursos de manera que éstos sean una infraestructura global que permita compartir y reutilizar los datos y documentos entre distintos tipos de usuarios.

RDF proporciona información descriptiva sobre los recursos y es una base para procesar metadatos, dando capacidad de interacción a las aplicaciones que lo usan, facilitando así el procesamiento automatizado de los recursos web.

Las posibles aplicaciones que tiene son:

- **Recuperación de recursos:** Da más prestaciones a los motores de búsqueda.
- **Catalogación:** Se utiliza para describir el contenido y las relaciones de contenido disponibles en un sitio Web, una página Web o una biblioteca digital particular.
- **Agentes de software inteligentes:** Facilitan el intercambio y para compartir conocimiento.
- **Calificación:** Se utiliza para calificar el contenido de elementos de la Web, no sólo visualizarlo.
- **Descripción de colecciones:** Se utiliza de tal forma que la descripción de colecciones de páginas se representan con un documento lógico individual.
- **Descripción de derechos:** Es usado para describir los derechos de propiedad intelectual de las páginas web.
- **Decripción de privacidad:** Puede expresar las preferencias de privacidad de un usuario, así como las políticas de privacidad de un sitio Web.

Concretando, podríamos decir que el objetivo principal de RDF es definir un mecanismo que describa los recursos de una forma no concreta, es decir, que se especifique un esquema que sea posible utilizar en distintos dominios de aplicación.

Las **ventajas** de RDF frente a otra forma de representar la información son:

- La posibilidad de reutilización de las definiciones de los metadatos.
- Puede ser extensible de forma incremental, después de haber sido definido un esquema.
- La posibilidad de crear objetos que intercalen distintos tipos de metadatos para hacerlos más completos (Esto se consigue basándose en esquemas distintos: RDF, Friend Of a Friend (FOAF), Dublin Core (DC), OWL...)

3.2.2. Bases del Modelo RDF

Las características básicas de RDF son:

- Es un modelo orientado a objetos.
- Está compuesto de categorías, también denominados *schemas*, que se organizan de forma jerárquica.
- Pueden ser extensibles debido a la posibilidad de poder refinar el modelo con subcategorías.
- Representan propiedades designadas y valores de las mismas. En términos de la programación orientada a objetos, podemos definirlo como unos recursos que se corresponden con objetos y las propiedades con los objetos específicos y variables de una categoría.

Analizando este último punto un poco más a fondo, podríamos decir que las propiedades RDF pueden recordar a atributos de recursos y en este sentido se corresponden con los tradicionales pares de atributo-valor que se utilizan en otros lenguajes. Las propiedades RDF representan también la relación entre recursos y por lo tanto, un modelo RDF puede parecer un diagrama entidad- relación.[2]

[2]Un diagrama entidad-relación es la forma de representar un **modelo** entidad-relación, el cual es un concepto de modelado para bases de datos. Los objetos que pertenecen a la base de batos son entidades las cuales tienen unos atributos y se vinculan mediante relaciones.

El modelo de datos de RDF es una forma de sintaxis para representar expresiones en RDF. La representación del modelo de datos se usa para evaluar la equivalencia en significado entre distintas entidades. Dos expresiones RDF son equivalentes si y sólo si sus representaciones del modelo de datos son las mismas. Esta definición de equivalencia permite algunas variaciones sintácticas en expresiones sin alterar el significado.

El modelo de datos básico consiste en tres tipos de objetos, que constituyen la base sintáctica del lenguaje:

- **Propiedades:** Son los atributos, características o relaciones utilizadas para describir un recurso. Cada propiedad tiene un significado determinado, define cuáles son sus valores permitidos, los tipos de recursos que puede describir y sus relaciones con otras propiedades.
- **Recursos:** son todos los elementos que están descritos con expresiones RDF. (Ejemplos: Una página web completa, una parte específica de la misma, una colección completa de páginas, un libro impreso o cualquier otro elemento que no sea accesible vía web.)
  Éstos se identifican por Uniform Resource Identifiers (URIs), las cuales son una cadena corta de caracteres que identifica unívocamente un recurso. Está definido en el RFC 2396.
  Una URI sería, por ejemplo, *http://www.informatica.us.es/info/docs?pagina=3*
  Donde distinguimos varias partes:
  - Esquema: Nombre que identifica el protocolo de acceso al recurso. En nuestro caso, *http*. Los ":" son un separador y las dos barras son una indicación de raíz absoluta para la autoridad.
  - Autoridad: Elemento que identifica quién o qué es el autor del recurso, es decir, lo que en nuestro ejemplo es *www.informatica.us.es*
  - Ruta: Información que identifica la ruta jerárquica donde se encuentra el recurso o también la ruta por la cual será posible determinar la situación física real del mismo. En nuestro caso: */info/docs*

  - Consulta: Información con estructura no jerárquica, usualmente pares clave=valor que aportan más información para poder extraer los datos de una forma más eficiente. Este elemento se inicializa mediante el caracter *?*. Se puede concatenar más de una consulta mediante el símbolo *&*. En nuestro caso, *pagina=3*

- **Enunciados:** También llamados *RDF statement*. Son recursos específicos con una propiedad y un valor determinados.

  - Recursos: Sujeto de los enunciados.
  - Propiedad: Predicado de los enunciados.
  - Objeto: Valor de los enunciados.

Un ejemplo gráfico de esto es lo siguiente:

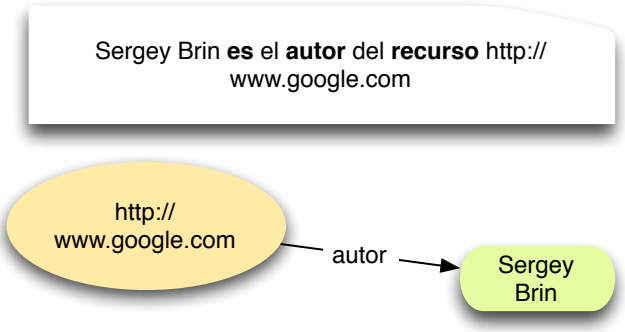

Figura 3.1: Elementos de un objeto en RDF (Ejemplo Básico).

En este ejemplo (figura 3.1), la información que queremos transmitir es que *Sergey Brin es el autor del recurso www.google.es*, donde el **sujeto o recurso** de la misma es *Sergey Brin*, el **predicado o propiedad** es la acción que éste realiza: *ser el autor* y donde el **objeto o valor** en sí es *google*.

Un segundo paso para representar más información que la simple oración anterior es el siguiente ejemplo (figura 3.2), en el cual la información que queremos añadir a lo que ya teníamos es el

correo electrónico de Sergey. Para ésto tenemos que el **objeto** *google* tiene como autor (este sería el **predicado**) a un elemento, el **sujeto**, cuyos atributos son nombre (*Sergey Brin*) y correo electrónico (*sergeybrin@x.y*).

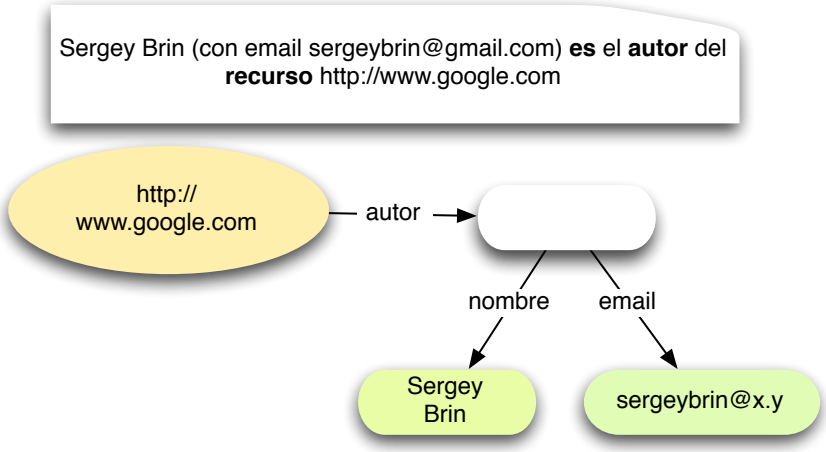

Figura 3.2: Elementos de un objeto en RDF (Ejemplo avanzado).

Aumentando el grado de complejidad, tenemos el siguiente ejemplo, que completa aún más el objeto que estamos intentando definir y representar en RDF.(figura 3.3)

Aquí, la información que queremos añadir es un elemento que sea capaz de identificar al sujeto de la acción. Tendríamos también que el **objeto** *google*, *tiene como autor* (este sería el **predicado**), a un elemento o **sujeto**, con identificador único *http://www.google.com/staffid/7777*, cuyos atributos son nombre (*Sergey Brin*) y correo electrónico (*sergeybrin@x.y*).

Avanzando un poco más en las bases del modelo, debemos hablar de otro elemento que es básico en RDF: Los *namespaces*. Como dijimos en el apartado de Ontologías (Sección 3.1), la sintaxis de RDF usa XML para crear e intercambiar metadatos. Pero no utiliza sólo esta útil herramienta, si no que hace uso de las *XML Namespace Facilities* que permiten que las propiedades y el propio esquema que la defina se asocien con precisión.

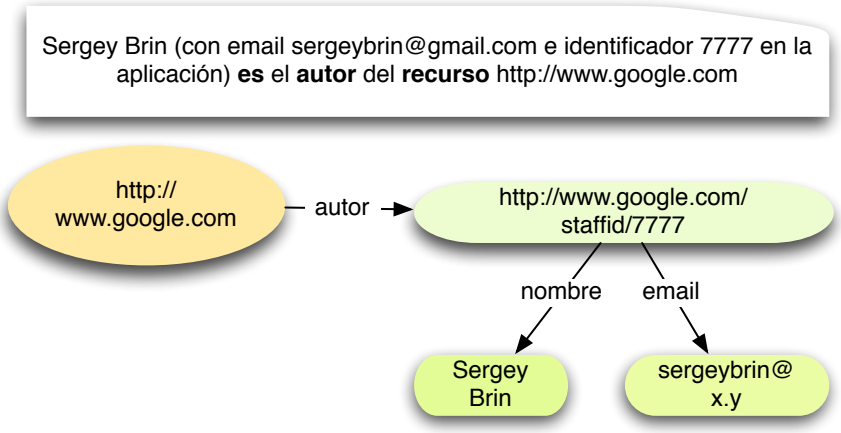

Figura 3.3: Elementos de un objeto en RDF (Ejemplo completo).

Namespaces Facilities

Las *Namespace Facilities* son uno de los aspectos más importantes en este lenguaje debido a que es el medio por el cual podemos organizar las clases y recursos dentro de un entorno, pudiendo así agruparlos por un modo jerárquico y lógico.

Un ejemplo de namespace sería el siguiente (figura 3.4):

```
1  <estacion_meteorologica>
2      <enero:temperatura_max>20</enero:temperatura_max>
3      <enero:temperatura_min>5<enero:temperatura_min>
4
5      <febrero:temperatura_max>23</febrero:temperatura_max>
6      <febrero:temperatura_min>12</febrero:temperatura_min>
7       ...
8  </estacion_meteorologica>
```

Figura 3.4: Ejemplo de namespaces.

En este ejemplo, si quisiéramos definir la temperatura máxima de los meses con tan sólo la eti-

queta *temperatura max* no podríamos diferenciar entre la temperatura de los distintos meses y habría que hacer más complejo el árbol jerárquico del XML, como vemos en la figura 3.5.

```
<estacion_meteorologica>
    <enero>
        <temperatura_max>20 </temperatura_max>
        <temperatura_min>5 </temperatura_min>
    <enero>
    <febrero>
        <temperatura_max>23 </temperatura_max>
        <temperatura_min>12 </temperatura_min>
    </febrero>
    ...

</estacion_meteorologica>
```

Figura 3.5: Ejemplo sin namespaces.

Profundizando un poco más en los namespaces, hay que ver otro aspecto que lo caracteriza, y es que los namespaces (en nuestro ejemplo `enero`) en realidad no son tan sencillos, puesto que cada uno está relacionado unívocamente con una URI (Ver apartado 3.2.2), que es el elemento que lo define.

En el caso del ejemplo anterior, el poner `estacion meteorologica` o `enero` no significaría nada si no fuera porque éstos están definidos con anterioridad en la cabecera del XML en el que se encuentren y se relacionan unívocamente con la URI que los identifica.

En la figura 3.6 podemos ver este aspecto basado en el ejemplo de la estación meteorológica.

Como podemos observar, esta forma de representar la información sería mucho más compleja, por no decir que daría pie a numerosos errores al escribir la información.

Para solventar este problema se debe determinar al principio del documento la especificación que tiene cada elemento de la forma `xmlns:alias=``Uri del alias"`.
Veamos, un ejemplo sobre ésto último para poder entender mejor el concepto. (figura 3.7)

```
<[http://estacion_meteorologica.es]>
    <[http://www.estacion_met.es/enero]:temp_max>20</enero:temp_max>
    <[http://www.estacion_met.es/enero]:temp_min>5</enero:temp_min>

    <[http://www.estacion_met.es/febrero]:temp_max>23</febrero:temp_max>
    <[http://www.estacion_met.es/febrero]:temp_min>12</febrero:temp_min>
    ...
</[http://estacion_meteorologica.es]>
```

Figura 3.6: Ejemplo de namespaces sin sustitución por alias.

```
    <?xml version="1.0"?> <xmlns:estacion_met="http://estacion_met.es" [...]>
```

Figura 3.7: Ejemplo de especificación de los namespaces.

Es muy importante tener en cuenta que los URIs sólo se utilizan para que el nombre sea único, no son enlaces, ni tienen que contener información, sin embargo, también los URIs sirven para acceder a recursos. Los *XML Namespaces* describen cómo se puede asociar una URI con cada etiqueta y atributo en un documento XML, si bien, para qué se utiliza el URI depende de la aplicación que lo lea. RDF, por ejemplo, lo usa para enlazar cada metadato a un archivo definiendo el tipo de ese metadato).

En resumen, los espacios de nombres o *namespaces* sirven para evitar las colisiones entre elementos del mismo nombre, y en general, para distinguir los distintos grupos de elementos en un mismo documento. Cada espacio de nombre o *namespace* se asocia con una URI, que sólo sirve como identificador único y no tiene por qué ser válida en cuanto a la ruta que defina.

3.2.3. Sintaxis de RDF

Hay dos tipos de sintaxis en RDF: la **serializada**, que expresa las capacidades totales del modelo de datos de una forma muy regular y la **abreviada**, que contiene términos adicionales que hacen que se pueda presentar de forma más compacta el modelo de datos. Ambas formas pueden mezclarse.A continuación podemos ver un ejemplo de las dos representaciones.(figura 3.8)

```
<!-- ************ Serializada ************ -->

<rdf:RDF>
  <rdf:Description about="http://www.etsii.us.es/user/id/333/documentacion">
    <d:Creator rdf:resource="http://www.etsii.us.es/user/id/333"
       p:Name="Elena Lozano"
       p:Email="elozano@us.es" />
  </rdf:Description>
</rdf:RDF>

<!-- ************ Abreviada: ************ -->

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
  xmlns:d="http://description.org//">
  <rdf:Description about="http://www.etsii.us.es/user/333/docs">
    <d:Creator>
      <rdf:Description about="http://www.etsii.us.es/user/333">
        <rdf:type resource="http://www.etsii.us.es/schema/Person"/>
        <p:Name>Elena Lozano</p:Name>
        <p:Email>elozano@us.es</p:Email>
      </rdf:Description>
    </d:Creator>
  </rdf:Description>
</rdf:RDF>
```

Figura 3.8: Ejemplos de los tipos de sintaxis de RDF.

A continuación pasamos a describir cada una de las sintaxis.

Sintaxis Serializada Básica

Una sentencia o declaración RDF rara vez aparece sola; normalmente se darán juntas varias propiedades de un recurso. La sintaxis RDF XML se ha diseñado para dar cabida a esto agrupando múltiples sentencias para el mismo recurso en un elemento **Description**. El elemento Description denomina, en un atributo about, el recurso para el cual se aplica cada una de las sentencias. Si el recurso no existe todavía, es decir, no tiene aún un identificador de recursos, el elemento **Description** puede proporcionar el identificador para el recurso usando el atributoID.

La sintaxis serializada RDF básica tiene los elementos que podemos ver en la siguiente lista:

- **RDF:** <rdf:RDF>description* </rdf:RDF>
- **description:** <rdf:Description idAboutAttr? >propertyElt* </rdf:Description>
- **idAboutAttr:** idAttr |aboutAttr
- **aboutAttr:** about= URI-reference
- **idAttr:** ID= IDsymbol
- **propertyElt:** <propName >value </' propName >|<propName resourceAttr />
- **propName:** Qname
- **value:** description |string
- **resourceAttr:** resource= URI-reference
- **Qname:** NamespacePrefix : name
- **URI:-reference** string (una URI)
- **IDsymbol:** cualquier símbolo permitido en XML
- **name:** cualquier nombre permitido en XML
- **namespacePrefix:** cualquier espacio de nombre permitido en XML
- **string:** cualquier texto permitido en XML

Pasamos a continuación a explicar brevemente los elementos más relevantes de la misma:

- El elemento RDF es un simple envoltorio que marca los límites en un documento XML donde el contenido será definido como una instancia de modelo de datos RDF.

- El elemento **Description** contiene los elementos sobrantes que posibilitan la creación de sentencias en el objeto específico de la categoría del modelo. Éste puede entenderse como un lugar donde mantener la identificación de un recuso descrito. Como normalmente habrá más de una sentencia sobre un recurso el elemento Description proporciona una forma de dar un nombre concreto a varias sentencias de una vez.
  Cuando se especifica el atributo about con Description, la sentencia del elemento Description se referirá al recurso cuyo identificador determina el elemento about.

- El valor del atributo **about** se interpreta como una referencia en forma de URI. El identificador de recursos correspondiente se obtiene resolviendo la URI de forma absoluta. Si se incluye un fragmento del identificador en la URI, el identificador de recursos se refiere sólo al subcomponente del recurso que lo contiene (aboutAttr), recurso que se identificará a través del correspondiente fragmento identificador que contiene dicho recurso (idAttr). Los valores para cada atributo identificador (idAttr) no deben ser duplicados en un mismo documento. Es importante tener en cuenta que tanto el idAttr como about pueden especificarse en el elemento Description, pero no los dos juntos en el mismo elemento.

- Un único elemento Description puede contener más de un elemento **propertyElt** con el mismo nombre de propiedad. Cada uno de dichos propertyElt añaden un nuevo atributo al sujeto o predicado de la sentencia. Dentro del elemento propertyElt, el atributo resource especifica que otros recursos son el valor de esta propiedad; es decir, el objeto de la sentencia es otro recurso identificado por una URI.

- Las cadenas o **Strings** deben ser XML bien formados; el XML convencional contiene mecanismos de citación y disgregación que pueden usarse si la serie de caracteres (string) contiene secuencias de caracteres que no cumplen las reglas de buena formación. Por tanto, es necesario que las cadenas tengan una sintaxis adicional que especifique un contenido XML bien formado y que conlleve un marcado de tal forma que éste no sea interpretado por RDF.

- Los nombres de propiedad o **names** pueden asociarse con un esquema. Esto puede hacerse

cualificando los nombres de los elementos con un prefijo de namespace que asocie con claridad la definición de propiedad con el esquema RDF correspondiente.

Un ejemplo de todo ésto lo podemos ver en la figura 3.9.

```
1        <rdf:RDF>
2            <rdf:Description about="http://www.informatica.us.es/user/docs">
3                <d:Creator>Elena Lozano</d:Creator>
4            </rdf:Description>
5        </rdf:RDF>
```

Figura 3.9: Código explicativo acerca de la sintaxis básica de RDF.

En este código podemos ver cómo describimos el objeto *documentación*. El *namespace* **d** se refiere a un prefijo específico elegido por nosotros de esta expresión RDF. Está definido en una declaración XML del namespace del tipo `xmlns:d="http://description.org/documentation/"`.

Este tipo de declaración del *namespace* podría incluirse normalmente como un atributo XML en el elemento **rdf:RDF** pero también puede incluirse con un elemento Description específico o incluso una expresión propertyElt concreta. La URI del nombre del *namespace*, en la declaración del *namespace*, es un identificador unívoco para un esquema particular que la persona que define los metadatos utiliza para definir el uso de la propiedad (en este caso `Creator`).

Otros esquemas pueden definir igualmente la propiedad denominada `Creator` y las dos propiedades se diferenciarán gracias a sus identificadores de esquema (en este caso **d**). Nótese también que un esquema normalmente define también varias propiedades; esto permite que con una única declaración de namespace será suficiente para crear un amplio vocabulario de propiedades que podrán usarse.

El documento XML completo que contiene la descripción citada anteriormente es el que podemos ver en la figura 3.10.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:d="http://description.org/docs/">
    <rdf:Description about="http://www.informatica.us.es/user/docs">
        <d:Creator>Elena Lozano</d:Creator>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.10: Código con declaración de esquemas de RDF.

Aun así, RDF no es algo estricto que sólo permita hacer las cosas de una manera, sino que permite una gran diversidad de formas de representar la misma información dependiendo del enfoque que se quiera dar a los distintos esquemas. Ejemplo de ésto son los códigos que podemos observar a continuación (3.11 y 3.12), donde se describe la misma información que en el código anterior (3.10)

```
<?xml version="1.0"?>
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:d="http://description.org/docs/">
  <Description about="http://www.informatica.us.es/user/docs">
    <d:Creator>Elena Lozano</d:Creator>
  </Description>
</RDF>
```

Figura 3.11: Variante del código de la figura 3.10 (I).

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Description about="http://www.informatica.us.es/user/docs">
    <d:Creator xmlns:d="http://description.org/docs/">Elena Lozano</d:Creator>
  </Description>
</RDF>
```

Figura 3.12: Variante del código de la figura 3.10(II).

La forma de condensar el código que podemos ver en la figura 3.12 debe evitarse cuando la codificación RDF/XML se escribe a mano o se edita en un editor de texto plano. Aunque no son ambiguas, la posibilidad de error es mayor que si los prefijos explícitos se usan de manera consistente.

Nótese que un fragmento RDF/XML que se realiza para insertarse en otro documento debe declarar todos los *namespaces* que utiliza de tal forma que sea completamente independiente.

Sintaxis Compacta Básica (o abreviada)

Aunque la sintaxis serializada muestra la estructura de un modelo RDF de una forma más esclarecedora, normalmente es mejor utilizar una forma XML más compacta. Esto es posible gracias a la sintaxis abraviada de RDF, la cual permite a los documentos seguir definiciones de tipo de documento (también denominados *DTD, Document Type Definition*) de XML bien estructuradas que se interpretan como modelos RDF.

Existen tres maneras de abreviar la sintáxis básica serializada:

- **Utilizar propiedades no repetidas dentro de un elemento Description donde los valores de dichas propiedades son literales.**

  Tratando un ejemplo similar al de las figuras anteriores (figuras 3.12 y 3.11), podemos abreviarlo de la forma (figura 3.13):

```
1  <rdf:RDF>
2    <rdf:Description about="http://www.informatica.us.es/user/docs"
3                     d:Creator="Elena Lozano" />
4  </rdf:RDF>
```

Figura 3.13: Ejemplo de código RDF abreviado utilizando propiedades no repetidas dentro de un elemento Description.

  Como podemos observar, el elemento Description tiene como atributo XML a la propiedad Creator. Ésto permite que se pueda omitir la etiqueta final de Description debido a la sintaxis de elemento vacío de XML.

  Hay que tener en cuenta que a pesar de que estas dos expresiones RDF son equivalentes, los distintos motores de procesamiento las tratarán de forma diferente. En concreto, si ambas estuviesen embebidas en un documento HTML el comportamiento por defecto de un navegador

que no reconozca RDF será presentar los valores de las propiedades en el primer caso y en el segundo caso debería presentarse de forma no textual.

- **Trabajar sobre elementos Description.**

  Esta forma abreviada puede emplearse para sentencias específicas donde el objeto de la declaración es otro recurso y el valor de cualquier propiedad de éste son cadenas de caracteres. Ésta es una transformación parecida a la realizada en XML nombrando distintos atributos, es decir, las propiedades del recurso en el elemento Description anidado puede escribirse como atributos XML del elemento propertyElt en el que se comprende Description.

  Para esclarecer este último punto, podemos ver la figura 3.14:

```
<rdf:RDF>
    <rdf:Description about="http://www.informatica.us.es/user/333/docs">
        <s:Creator rdf:resource="http://www.informatica.us.es/user/333">
            <rdf:Description about="http://www.informatica.us.es/user/333"
                p:Name="Elena Lozano"
                p:Email="elozano@us.es"
            </rdf:Description>
        </s:Creator>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.14: Ejemplo de código RDF abreviado trabajando con elementos Description.

Esta forma de representar los datos pretende enfocarse en el hecho de que se están describiendo dos recursos por separado, por un lado la persona y por otro el documento. La única pega de esta representación es que no queda relativamente claro que el recurso documento está referenciando al recurso usuario. Para intentar solucionar este problema se puede refinar el código de la forma que podemos observar en la figura 3.15. Debe puntualizarse que para el motor que trate la información no habrá diferencia entre uno y otro método.

Utilizando esta segunda sintaxis abreviada, el elemento interior Description y las expresiones de las propiedades que contiene pueden escribirse como atributos del elemento `Creator` como reflejamos en la figura 3.13

Cuando usamos esta forma abreviada, el atributo about del elemento anidado Description se

```
<rdf:RDF>
    <rdf:Description about="http://www.informatica.us.es/user/333/docs">
        <s:Creator>
            <rdf:Description about="http://www.informatica.us.es/user/333">
                <p:Name>Elena Lozano</p:Name>
                <p:Email>elozano@us.es</p:Email>
            </rdf:Description>
        </s:Creator>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.15: Ejemplo de código RDF abreviado trabajando con elementos Description (Ejemplo refinado).

```
<rdf:RDF>
    <rdf:Description about="http://www.informatica.us.es/user/333/docs">
        <s:Creator rdf:resource="http://www.informatica.us.es/user/333"
            p:Name="Elena Lozano"
            p:Email="elozano@us.es"/>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.16: Ejemplo de código RDF abreviado trabajando con elementos Description, de forma que tienen sus atributos como parte del elemento Creator.

convierte en un atributo de resource en el elemento propertyElt, de igual forma que el recurso designado por el URI es en ambos casos el valor de la propiedad Creator. Es simplemente una cuestión de preferencia de la persona que asigna los metadatos. Todas producen los mismos modelos RDF desde el punto de vista del motor que los trate.

- **Utilizar un elemento Description que contenga una propiedad type.**

  En este caso, el tipo de recurso definido en el esquema que corresponde al valor de la propiedad type puede utilizarse directamente como un nombre de elemento. En el siguiente fragmento de código (3.17) vemos cómo podemos aplicar esta tercera forma de sintaxis básica abreviada. Su significado es el mismo que el de la figura 3.16.

Los elementos de la sintaxis compacta son los mismos que los de la serializada, exceptuando los que podemos ver a continuación. Los dos primeros sustituyen a los que ya existían en la serializada y los otros dos se añaden a todos los demás.

```
<rdf:RDF>
    <rdf:Description about="http://www.informatica.us.es/user/333/docs">
        <s:Creator>
            <s:Person about="http://www.informatica.us.es/user/333"
                <p:Name>Elena Lozano</p:Name>
            </s:Person>
        </s:Creator>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.17: Ejemplo de código RDF abreviado trabajando con la propiedad type del elemento Description.

- **description:** <rdf:Description idAboutAttr? propAttr* />|
  <rdf:Description idAboutAttr? >propAttr* >propertyElt* </rdf:Description>|
  typedNode
- **propertyElt:** <propName >value </' propName >|
  <propName resourceAttr? propAttr* />
- **propAttr:** propName= string
- **typedNode:** <typeName idAboutAttr? propAttr* />|
  <typeName idAboutAttr? propAttr* >property* </ typeName>

3.2.4. Esquema RDF (*RDF Schema o RDFs*)

Una vez que se tienen en cuenta las normas básicas de la sintaxis de RDF (tanto serializada como abreviada) es importante, además, normalizar la forma que tenemos de determinar los términos que se utilizarán en RDF. Ésto se debe a que el lenguaje natural, lejos de ser algo imparcial y con un significado unívoco, no permite ser claro y conciso a la hora de establecer unos enunciados que sean lógicos y entendibles por todos las personas que los usen.

Con esto nos referimos a que al escribir una frase en lenguaje natural utilizamos palabras que quieren transmitir un significado inequívoco. Ese significado es fundamental para entender el enun-

ciado y, en el caso de aplicaciones de RDF es crucial para establecer que el procesamiento correcto se de como se esperaba. Es muy importante que tanto el escritor como el lector de una sentencia entiendan el mismo significado para los términos utilizados.

Para ser lo más preciso posible, se ha definido el concepto de **Esquema RDF o RDFs**. Éste es la forma mediante la cual el significado de RDF se puede expresar, es decir, actúa como un diccionario que define los términos que se van a usar en una sentencia y le otorgará significados determinados.

Aunque existen una gran variedad de formas de esquemas, existe un estándar específico, el **esquema RDF** que engloba las características más comunes y que permite que las tareas al usar RDF estén más automatizadas. Extiende a RDF para incluir un amplio vocabulario con un significado adicional.

RDFs define ciertas clases que pueden representar casi cualquier elemento que deseemos, siendo descritas por recursos y propiedades tales como rdfs:Class, rdf:Resource y rdf:type, rdfs:subClassOf respectivamente.
Por tanto, una clase es cualquier recurso que tenga un tipo (rdf:type) como propiedad cuyo valor sea un recurso rdfs:Class. Un ejemplo de ello lo podemos ver en la figura 3.18

```
1  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
3    <rdf:Description rdf:ID="Documentation">
4      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
5    </rdf:Description>
6    <rdf:Description rdf:ID="DocumentSection">
7      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
8      <rdfs:subClassOf rdf:resource="#Documentation"/>
9    </rdf:Description>
10 </rdf:RDF>
```

Figura 3.18: Ejemplo de la declaración del RDFSchema.

Si nos fijamos bien en el código de la figura 3.18, podemos ver que además de las etiquetas *rdf:about* usamos las del tipo *rdf:ID*. Estas últimas nos permiten identificar al recurso que queremos

definir.

En RDFs las propiedades se describen usando la clase *rdfs:Property*, definiendo en las mismas los elementos *rdfs:domain, rdfs:range y rdfs:subPropertyOf.* Cabe destacar que clases distintas pueden tener la misma propiedad. En la siguiente lista podemos ver la descripción de los elementos de las propiedades del RDFSchema.

- **rdfs:range** indica que los valores de la propiedad son instancias de algún otro tipo de dato.
- **rdfs:domain** se usa para indicar que una propiedad es aplicada a la clase en concreto.
- **rdfs:subPropertyOf** describe la especialización entre dos propiedades. (Similar a la propiedad *subClassOf*)

Muchas veces el valor de una propiedad es algo que tiene información contextual adicional que se considera parte de dicho valor. Es decir, es necesario distinguir los valores de las propiedades (por ejemplo; unidades de medida, precios en distintas divisas, etc.) Para poder representar este tipo de información tenemos lo que se denomina *valor de propiedad cualificada*.

3.3. FOAF

3.3.1. Introducción a FOAF

Friend of a Friend, o comúnmente ***FOAF***, es un diccionario de características y de clases que usan tecnología de RDF. (De hecho es un esquema RDF (apartado 3.2.4).
Se construye usando las tecnologías semánticas que hemos visto en apartados anteriores y su objetivo principal es definir un diccionario o vocabulario que especifica qué términos básicos que describen a personas y las relaciones que tienen entre ellas: a quién conoce, con quién está relacionado y con quién no, cuáles son sus proyectos actuales y cuáles los pasados, etc. Todo, o casi todo, puede ser definido en un archivo FOAF.

FOAF fue diseñado para ser utilizado junto a otros diccionarios, esquemas u ontologías, pudiendo integrar los distintos lenguajes y esquemas semánticos con éste, consiguiendo una herramienta descriptiva bastante potente.
Nació como un esfuerzo de colaboración entre los desarrolladores semánticos de la Web, los cuales eligieron el nombre *FOAF, Friend of a Friend,* para reflejar la preocupación actual sobre las redes sociales.

La idea básica que da el W3C con este vocabulario es bastante simple: Si la gente publica la información en el formato de documento de FOAF, los procesadores podrán hacer uso de esos datos. Si los archivos publicados con formato FOAF contienen las referencias a otros documentos de la Web, la información referenciada estará indexada de una forma mucho más coherente: de **forma semántica**, no simplemente por enlaces directos por coincidencia de palabras, sino por el **significado** de las mismas.

Gracias al vocabulario FOAF, obtenemos una colección de términos básicos que se pueden utilizar para expresar información acerca de las personas y su relación con la interacción de la misma con la web: fotos, identificadores de mensajería instantánea, correos electrónicos, etc. Una muestra de ello pueden ser las categorías o clases *foaf:Person*, *foaf:Document*, *foaf:image*, junto a algunas características prácticas de esas clases, tales como *foaf:name*, *foaf:mbox*, *foaf:homepage*, etc., así como algunas clases útiles que relacionan a los miembros de estas categorías.
Por ejemplo, *foaf:depiction*, este relaciona una clase como un *foaf:Person*, a una *foaf:Imagen*.
Estos aspectos del vocabulario lo desarrollaremos más en profundidad en apartados posteriores.

Los archivos de *FOAF* son documentos de texto, normalmente en formato Unicode[3]. Se escriben en XML y adoptan las convenciones de RDF.

[3]Unicode es un estándar industrial cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático.

Basarse en estas tecnologías permite que sea más sencillo para que el software y los distintos motores de tratamiento de la información procesen los datos de una forma eficaz.

Los motivos por los cuales se eligió RDF como base para FOAF fueron, entre otros los que se exponen a continuación:

- Los elementos necesarios para describir a personas y sus interrelaciones con la Web serían muchos y muy complejos si no se apoyaran en una tecnología existente, por lo que decidieron que no era necesario *reinventar la rueda* de nuevo para hacer lo mismo.
- Uno de los aspectos que se le exigían a FOAF era que pudieran ser compatibles con otras ontologías, por lo que al incorporar RDF a sus bases, ésto estaba solucionado.
- RDF proporciona una especificación simple de comprender y de utilizar, por lo que al basarse en ella pretendían que el modelo FOAF también lo fuera.
- Diversos vocabularios, aparte de FOAF, se basan también en RDF, por lo que al estar relacionados por una misma base común, es posible relacionar también estos otros vocabularios entre ellos.

Un aspecto importante que se debe entender es que el vocabulario de FOAF no es un **estándar** en el sentido de la estandarización ISO[4] o en el W3C.
FOAF depende de estándares del W3C que son desarrollados y mejorados continuamente; concretamente XML, XML Namespaces y RDF.
Por tanto, todos los documentos de FOAF deben ser documentos bien formados en RDF/XML aunque el vocabulario de FOAF en sí mismo se pueda manejar de una forma abierta y flexible.

Aun así no hay que olvidar que la especificación que da el W3C con respecto al vocabulario FOAF es una forma de adoptar una sintaxis común para que los diversos motores que procesan la

[4]ISO es la *Organización Internacional para la Estandarización*, la cual es una organización internacional no gubernamental, compuesta por representantes de cada país, que produce unas normas (normas ISO) cuya finalidad es coordinar las normas nacionales para facilitar el comercio, el intercambio de información y contribuir con unos estándares comunes para el desarrollo y transferencia de tecnologás.

información sean capaces de reconocer y de trabajar con los datos definidos en este vocabulario.

Antes de entrar en aspectos técnicos o teóricos acerca de XML, FOAF y RDF, veamos un ejemplo de cómo es un archivo FOAF (figura 3.19):

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
        <foaf:name>Francisco Ruiz</foaf:name>
    </foaf:Person>

</rdf:RDF>
```

Figura 3.19: Ejemplo de documento FOAF: Person, Name.

En este ejemplo tan sencillo puede observarse como describimos a una persona (*foaf:Person*) cuyo nombre (*foaf:name*) es *Francisco Ruiz.*

El problema de describir a personas con FOAF de la manera que vimos en la figura anterior (figura 3.19) es que existen en el mundo cientos de miles de personas que comparten los mismos atributos, como el nombre, color de ojos, pelo, altura, intereses, etc. Para solucionar esto, los desarrolladores de FOAF buscaron un atributo propio de los usuarios de internet, y lo encontraron: el **correo electrónico**. Éste se representa en FOAF de la siguiente forma: (figura 3.20)

Aunque en estos ejemplos no veamos la capacidad de descripción de FOAF, en códigos más complejos obtenemos ventajas tales como la rápida y efectiva constitución de comunidades virtuales, la búsqueda de personas con los mismos intereses que nosotros, documentos escritos por otros, etc., y teniendo la tranquilidad, además de saber que lo que aún no esta implementado, puede hacerse: tan solo es cuestión de definir y extender el esquema, utilizarlo y motivar a otros a que lo hagan.

Ya existen proyectos en internet que utilizan FOAF como su base principal. Algunos de ellos son *FOAFCorp* (un diagrama de relaciones entre diversas personas.) o *foafnaut!* (permite una veloz

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
        <foaf:name>Francisco Ruiz</foaf:name>
        <foaf:mbox rdf:resource="francisco_r@empresa.com"/>
    </foaf:Person>

</rdf:RDF>
```

Figura 3.20: Ejemplo de documento FOAF: Mbox.

identificación de quiénes conocen y a quién conoce una determinada persona, ver fotografías donde se encuentren ambos, etc.)

3.3.2. Elementos característicos de FOAF

Interrelaciones entre foaf:Person:

Ésta es una de las características más interesantes que permite desarrollar FOAF. Es la forma de indicar una relación entre dos personas mediante FOAF y para ello se utiliza la propiedad *foaf:knows*. Un ejemplo de la forma de representar esta relación es el que podemos ver en la figura 3.21

Si analizamos este ejemplo podemos ver cómo el código FOAF describe a *Francisco Ruiz*, con correo electrónico *francisco.r@empresa.com* y que además, **conoce** a *Jaime Lorente*.
Como podemos observar esta forma de expresar relaciones es bastante simple y efectiva, permitiéndonos expresar correctamente y sin ambiguedades quien conoce a quién.

Pero esto tiene un problema: ¿Cómo podemos expresar la relación de una persona con otras cuando el número de éstas es elevado? y ¿cómo podemos además, identificarlas a todas ellas? Poner el nombre, el email y otras propiedades de todas las personas a las que conociéramos en nuestro documento FOAF sería demasiado extenso y redundante, por lo que hacemos uso del siguiente elemento

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
        <foaf:name>Francisco Ruiz</foaf:name>
        <foaf:mbox rdf:resource="francisco_r@empresa.com"/>
        <foaf:knows>
            <foaf:Person>
                <foaf:name>Jaime Lorente</foaf:name>
            </foaf:Person>
        </foaf:knows>
    </foaf:Person>

</rdf:RDF>
```

Figura 3.21: Ejemplo de documento FOAF: Knows.

característico de FOAF.

Indicar datos externos:

Para solventar el problema mencionado anteriormente hacemos uso del esquema RDF o RDFs(sección 3.2.4) mediante la propiedad *rdfs:seeAlso*. Para muestra de ésto, veámos el siguiente ejemplo de la figura 3.22. Con una simple propiedad hemos enlazado a la persona que estamos describiendo con otra con la que está relacionado sin necesidad de redundar datos.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
        <foaf:name>Francisco Ruiz</foaf:name>
        <foaf:mbox rdf:resource="francisco_r@empresa.com"/>
        <foaf:knows>
            <foaf:Person>
                <foaf:name>Jaime Lorente</foaf:name>
                <rdfs:seeAlso resource="http://ejemplo.org/jaimelorente.rdf" />
            </foaf:Person>
        </foaf:knows>
    </foaf:Person>
</rdf:RDF>
```

Figura 3.22: Ejemplo de documento FOAF: rdfs:seeAlso.

Forma de evitar el SPAM:

Esta característica es muy útil en los tiempos que corren debido a la cantidad de correos no deseados (SPAM) que inundan nuestras cuentas de correo. Para que los *spammers* y demás software malicioso no sean capaces de leer la dirección de correo que mostramos en nuestro documento FOAF existe una propiedad que nos ayuda a evitar este hecho: *foaf:mbox_sha1sum*.

Esta propiedad calcula la suma SHA de la dirección de correo que es única y que no puede ser revertida. Con este método somos capaces de identificarnos unívocamente con nuestra propia suma como antes hacíamos con *foaf:mbox*.Veamos un ejemplo en la figura 3.23.

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
        <foaf:name>Francisco Ruiz</foaf:name>
        <foaf:mbox_sha1sum>203243957adfe259096587eabc348957a</foaf:mbox_sha1sum>
        <foaf:knows>
            <foaf:Person>
                <foaf:name>Jaime Lorente</foaf:name>
                <rdfs:seeAlso resource="http://ejemplo.org/jaimelorente.rdf" />
            </foaf:Person>
        </foaf:knows>
    </foaf:Person>

</rdf:RDF>
```

Figura 3.23: Ejemplo de documento FOAF: Evitar el SPAM.

Forma de relacionar imágenes con el objeto al que representan:

Como expusimos anteriormente, el hecho de poder relacionar las imágenes con los objetos a los que representaban, es una de las características más utilizadas del vocabulario FOAF y un claro ejemplo de como puede interactuar esta especificación con otras distintas.

Para definir esta característica utilizamos la propiedad *foaf:depiction*, la cual describe el hecho de que un recurso (persona, agente u otro objeto) es el que se muestra en la imagen. También existe la propiedad foaf:depicts, que se utiliza a la inversa; para reflejar que tal imagen muestra un recurso determinado.

Para verlo más claro, veamos el ejemplo de la figura 3.24. En ella podemos observar cómo la persona *Francisco Ruiz* tiene una imagen que lo describe, y más abajo, vemos la descripción de la imagen, con un título, creador y a quien describe.

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person rdf:ID="franciscoRuiz">
        <foaf:name>Francisco Ruiz</foaf:name>
        <foaf:depiction rdf:resource="http://www.ejemplo.org/franciscoRuiz.jpg"/>
    </foaf:Person>


 <!-- ******* A la Inversa: foaf:depicts ******** -->
<foaf:Image rdf:about="http://www.ejemplo.org/franciscoRuiz.jpg">
    <dc:title>My favourite travel</dc:title>
    <foaf:depicts rdf:resource="#franciscoRuiz"/>
    <foaf:maker rdf:resource="#franciscoRuiz"/>
 </foaf:Image>
</rdf:RDF>
```

Figura 3.24: Ejemplo de documento FOAF: Image, depicts y depiction.

Tras haber explicado alguna de las características más importantes de FOAF, pasamos a explicar cada una de sus clases y propiedades.

3.3.3. Elementos del Vocabulario FOAF

- **Clase foaf:Agent**:
    - Se refiere a una persona, grupo, software u objeto que realizan algo determinado. Alguna de sus subclases más conocidas son *foaf:Person*, persona que realiza alguna acción o

foaf:group.

- Son del mismo tipo que *foaf:maker* y *foaf:member*
- Las propiedades que lo extienden son: *foaf:mbox, foaf:mbox_sha1sum, foaf:gender, foaf:jabberID, foaf:aimChatID, foaf:icqChatID, foaf:yahooChatID, foaf:msnChatID, foaf:weblog, foaf:tipjar, foaf:made, foaf:holdsAccount y foaf:birthday*

- **Clase foaf:Document**:
  - Se refiere al concepto usual de documento que tenemos en el lenguaje natural.
  - Son del mismo tipo que *foaf:homepage, foaf:weblog, foaf:tipjar, foaf:workplaceHomepage, foaf:workInfoHomepage, foaf:schoolHomepage, foaf:interest, foaf:publications, foaf:isPrimaryTopicOf, foaf:page y foaf:accountServiceHomepage*
  - Las propiedades que lo extienden son: *foaf:sha1, foaf:topic y foaf:primaryTopic*
- **Clase foaf:Group**:
  - Representa una colección de agentes individuales y puede él mismo desempeñar el papel de un *foaf:Agent*.
- **Clase foaf:Image**:
  - Se refiere a una imagen.
  - Son del mismo tipo que *foaf:img, foaf:depiction y foaf:thumbnail*
  - Las propiedades que lo extienden son: *foaf:depicts y foaf:thumbnail*
- **Clase foaf:OnlineAccount**:
  - Se refiere a un cuenta de mensajería online.
  - Es del mismo tipo que *foaf:holdsAccount*.
  - Las propiedades que lo extienden son: *foaf:accountServiceHomepage y foaf:accountName*
- **Clase foaf:OnlineChatAccount**:

- Se refiere a un cuenta de mensajería instantánea.
- Es del mismo tipo que *foaf:holdsAccount*.
- Las propiedades que lo extienden son: *foaf:accountServiceHomepage y foaf:accountName*

- **Clase foaf:OnlineEcommerceAccount**:
  - Se refiere a un cuenta de e-commerce online (Por ejemplo, de Amazon, Ebay...).
  - Las propiedades que lo extienden son: *foaf:accountServiceHomepage y foaf:accountName*
- **Clase foaf:OnlineGamingAccount**:
  - Se refiere a un cuenta de juegos en línea (Por ejemplo, EverQuest, Xbox Live...).
  - Es del mismo tipo que *foaf:holdsAccount*.
  - Las propiedades que lo extienden son: *foaf:accountServiceHomepage y foaf:accountName*
- **Clase foaf:Organization**:
  - Representa un tipo de *foaf:Agent* correspondiendo a instituciones sociales como compañias o sociedades.
- **Clase foaf:Person**:
  - Representa un tipo de *foaf:Agent* correspondiendo a una persona determinada.
  - Es del mismo tipo que *foaf:knows*.
  - Las propiedades que lo extienden son: *foaf:geekcode, foaf:firstName, foaf:surname, foaf:family_name, foaf:plan, foaf:img, foaf:myersBriggs, foaf:workplaceHomepage, foaf:workInfoHomepage,foaf:schoolHomepage, foaf:knows, foaf:interest, foaf:topic_interest, foaf:publications, foaf:currentProject y foaf:pastProject*
- **Clase foaf:PersonalProfileDocument**:
  - Representa el documento con el RDF o perfil personal de un *foaf:Person.*

- **Clase foaf:Project**:
  - Representa un proyecto, ya sea colectivo o individual.
- **Clase foaf:accountName**:
  - Representa el nombre asociado a una cuenta online.
  - Extiende a *foaf:OnlineAccount*.
- **Clase foaf:accountServiceHomepage**:
  - Indica la página inicial o homepage de una cuenta online.
  - Extiende a *foaf:OnlineAccount*.
- **Clase foaf:aimChatID**:
  - Indica una identificador de mensajería instantánea de AIM.
  - Extiende a *foaf:Agent*.
- **Clase foaf:icqChatID**:
  - Indica una identificador de mensajería instantánea de ICQ.
  - Extiende a *foaf:Agent*.
- **Clase foaf:msnChatID**:
  - Indica una identificador de mensajería instantánea de MSN.
  - Extiende a *foaf:Agent*.
- **Clase foaf:jabberID**:
  - Indica una identificador de mensajería instantánea de Jabber.
  - Extiende a *foaf:Agent*.
- **Clase foaf:yahooChatID**:

- Indica una identificador de mensajería instantánea de Yahoo.
- Extiende a *foaf:Agent*.

- **Clase foaf:based_near**:
  - Indica la localización geográfica de un elemento (persona, agente, etc.)
  - Extiende a *foaf:based_near*.
- **Clase foaf:name**:
  - Define el nombre completo de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:nick**:
  - Define el nick de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:title**:
  - Define el título de cortesía de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:homepage**:
  - Define la homepage o página web de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:mbox**:
  - Define el correo electrónico de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:mbox_sha1sum**:

- Define el correo electrónico en suma *sha1* de una persona.
- Extiende a *foaf:Person*.

- **Clase foaf:sha1**:
  - Define una suma *sha1* de un documento.
  - Extiende a *foaf:Document*.
- **Clase foaf:surname**:
  - Define los apellidos de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:firstName**:
  - Define el nombre de pila de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:givenname**:
  - Define el nombre de pila de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:family_name**:
  - Define el nombre de la famlia o apellido de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:weblog**:
  - Define la dirección del weblog de una persona.
  - Extiende a *foaf:Person*.
- **Clase foaf:workplaceHomepage**:

  - Define la dirección de la página web del lugar de trabajo de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:workInfoHomepage**:
  - Define la dirección de la página web de información del lugar de trabajo de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:schoolHomepage**:
  - Define la dirección de la página web del lugar de estudio de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:knows**:
  - Define la relación existente entre una persona y otra.
  - Extiende a *foaf:Person*.

- **Clase foaf:depiction**:
  - Define la imagen de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:currentProject**:
  - Define el proyecto actual en el que está colaborando una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:pastProject**:
  - Define el proyecto pasado en el que estaba colaborando una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:interest**:

- Define la página o dirección que contiene un interés que tiene una persona.
- Extiende a *foaf:Person*.

- **Clase foaf:gender**:
  - Define el género de un Agente.
  - Extiende a *foaf:Agent*.

- **Clase foaf:depicts**:
  - Define una imagen que representa algo.
  - Extiende a *foaf:Image*.

- **Clase foaf:img**:
  - Define una imagen que representa a una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:plan**:
  - Define un .plan de un documento. Mas información sobre los archivos .plan en http://www.rajivshah.com/Case_Studies/Finger/Finger.htm.

- **Clase foaf:topic_interest**:
  - Define el o los intereses que tiene una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:publications**:
  - Indica un enlace a las publicaciones de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:geekcode**:

  - Define el geekcode que tiene una persona. Mas información sobre este código en http://www.geekcode.com/geek.html.
  - Extiende a *foaf:Person*.

- **Clase foaf:myersBriggs**:
  - Define el código myersBriggs que tiene una persona. Mas información sobre este código en http://www.geocities.com/lifexplore/mbintro.htm.
  - Extiende a *foaf:Person*.

- **Clase foaf:phone**:
  - Define el teléfono de una persona.
  - Extiende a *foaf:Person*.

- **Clase foaf:fundedBy**:
  - Define la forma recolección de fondos de de una empresa, proyecto o persona.

- **Clase foaf:holdsAccount**:
  - Define la relación entre una cuenta online y el agente que la tiene.
  - Extiende a *foaf:Agent*.

- **Clase foaf:primaryTopic**:
  - Define el tema principal de un documento.
  - Extiende a *foaf:Document*.

- **Clase foaf:logo**:
  - Define el logo de algún elemento.

- **Clase foaf:made**:

- Define algo que haya hecho el agente al que pertenezca.
- Extiende a *foaf:Agent*.

- **Clase foaf:maker**:
  - Define el creador de lo que se esté describiendo.
- **Clase foaf:member**:
  - Indica el miembro de un grupo.
  - Extiende a *foaf:Group*.
  - Contiene a *foaf:Agent*.
- **Clase foaf:membershipClass**:
  - Define la clase de miembros de un grupo.
  - Extiende a *foaf:Group*.
- **Clase foaf:page**:
  - Define una página o documento sobre el elemento al que pertenezca.
- **Clase foaf:theme**:
  - Define una tema de un elemento al que pertenezca.
- **Clase foaf:thumbnail**:
  - Define la miniatura de una imagen.
  - Extiende a *foaf:Image*.
- **Clase foaf:topic**:
  - Define el tema de un documento.
  - Extiende a *foaf:Document*.

3.3.4. Ejemplo Ilustrativo

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"

    <foaf:Person rdf:nodeID="elenalr">
        <foaf:firstname>Elena</foaf:firstname>
        <foaf:surname>Lozano Rosch</foaf:surname>
        <foaf:mbox_sha1sum>9642c26da203ef143f884488d49194eb0747547c</foaf:mbox_sha1sum>
        <foaf:homepage rdf:resource="http://elena.weblog.es"/>
        <foaf:phone rdf:resource="tel:+0034234563923"/>
        <foaf:jabberID>xxx@jabber.org</foaf:jabberID>
        <foaf:based_near>
            <geo:Point>
                <geo:lat>52.6796</geo:lat>
                <geo:long>-7.8221</geo:long>
            </geo:Point>
        </foaf:based_near>
        <foaf:workplaceHomepage rdf:resource="http://www.cica.es"/>
        <foaf:schoolHomepage rdf:resource="http://www.etsii.us.es"/>
        <foaf:dateOfBirth>1985-04-07</foaf:dateOfBirth>
        <foaf:weblog rdf:resource="http://elllore.wordpress.com"/>
        <foaf:interest dc:title="foaf" rdf:resource="http://www.foaf.com"/>
        <foaf:holdsAccount>
            <foaf:OnlineChatAccount>
                <foaf:accountName>elenalozano</foaf:accountName>
                <foaf:accountServiceHomepage dc:title="www.gmail.com"
                    rdf:resource="http://www.gmail.com" />
            </foaf:OnlineChatAccount>
        </foaf:holdsAccount>
        <foaf:knows rdf:nodeID="danielGarcia"/>
    </foaf:Person>

    <foaf:Person rdf:nodeID="danielGarcia">
        <foaf:name>Daniel Garcia</foaf:name>
        <foaf:mbox_sha1sum>203243957adfe259096587eabc348957a</foaf:mbox_sha1sum>
    </foaf:Person>

</rdf:RDF>
```

Figura 3.25: Ejemplo ilustrativo de un documento FOAF.

3.4. DOAP

DOAP (*Describe Open Source Projects, descripción de un proyecto*) nació inspirado en FOAF para resolver los problemas existentes acerca de la necesidad de mantener la información de un proyecto software actualizada. Se quería un vocabulario que se pudiera utilizar en un documento XML para que la información fuera accesible para todo tipo de motores.

Al crearse DOAP como vocabulario, se establecieron una serie de requisitos que necesitaba cumplir. Éstos son los siguientes:

- Descripción de un proyecto software y de sus recursos asociados que pueden internacionalizarse, incluyendo los participantes del mismo y sus recursos de la Web.
- Herramientas básicas para permitir la creación y la consumición fáciles de tales descripciones.
- Capacidad de interacción con otros proyectos conocidos que trataran metadatos en la Web. (RSS, FOAF, Dublín Core)
- La capacidad de ampliar el vocabulario según las necesidades del desarrollador.

Las tareas que se pueden realizar con DOAP son las siguientes:

- Importación fácil de proyectos en sistemas software.
- Intercambio de datos entre distintos motores y sistemas software.
- Configuración automática para los recursos tales como CVS.
- Asistir las bases de los paquetes que empaquetan el software para los distribuidores

3.4.1. Tecnologías utilizadas en DOAP

Las tecnologías utilizadas en DOAP son las que se describen a continuación:

- **Conjunto de elementos de Dublin Core:** Esta librería de aplicaciones de metadatos es un sistema de quince definiciones semánticas descriptivas que pretenden transmitir un significado semántico a las mismas, siendo estas definiciones opcionales se pueden repetir y aparecer en cualquier orden.

  Está diseñado para proporcionar un vocabulario de características capaces de representar la información descriptiva básica sobre cualquier recurso, sin que importe el formato de origen, el área de especialización o el origen cultural.

- HTML y XML (ver la sección 3.1).

- Se basa en RDF, al igual que FOAF (sección 3.2)

Lo más importante, al igual que pasaba en el vocabulario FOAF, es que al basarse en otras tecnologías más maduras, permite que sea todo mucho más apropiado para que el software y los distintos motores de tratamiento de la información procesen los datos de una forma eficaz; además, se consigue con esto que los elementos necesarios para describir a personas y sus interrelaciones con la Web se reutilicen, sean compatibles con otras ontologías y que sea posible relacionar a DOAP con otros vocabularios (como por ejemplo con FOAF).

3.4.2. Elementos del Vocabulario DOAP

El vocabulario de DOAP se diseñó tras estudiar cuales eran los elementos que describían a un proyecto software en las principales plataformas de desarrollo de los mismos: Freshmeat, SourceForge, GNOME, KDE-apps.org, Advogato, etc. Una vez analizadas cada una de las plataformas se decidió que el vocabulario fuera el siguiente:

- **Clase Project**: es la clase que engloba todas las propiedades que se explican a continuación.

- **name:** es el nombre por el cual es conocido públicamente el proyecto.

- **shortname:** es el nombre corto del proyecto, normalmente es el utilizado para los nombres de los archivos del mismo.

- **homepage:** es la página principal del proyecto. Asociada unívocamente a un proyecto en particular.
- **created:** es la fecha de cuando el proyecto fue creado.
- **description:** es la descripción del proyecto.
- **shortdesc:** es la descripción corta del proyecto (8-9 palabras).
- **category:** es una URI que refleja la categoría asignada a un proyecto.
- **wiki:** es una URI del Wiki relacionado con un proyecto.
- **bug-database:** es una URI donde se reportan los *bugs* o errores no resueltos de un proyecto.
- **screenshots:** es una URI con las capturas de pantalla de un proyecto.
- **mailing-list:** es una URI de la lista de correo relacionada con un proyecto.
- **download-page:** es una URI donde se puede descargar un proyecto.
- **download-mirror:** es una URI del *mirror* donde se puede descargar el proyecto.
- **license:** es una URI de la licencia de un proyecto.
- **programming-language:** es el lenguaje en el que está implementado un proyecto.
- **os:** es el sistema operativo para el cual está limitado.
- **maintainer:** es una *foaf:Person* que se encarga de mantener el proyecto (Similar a un líder del mismo)
- **developer:** es una *foaf:Person* que se encarga de desarrollar el proyecto.
- **documenter:** es una *foaf:Person* que se encarga de la documentación del proyecto.
- **translator:** es una *foaf:Person* que se encarga de realizar las traducciones del proyecto.

- **helper:** es una *foaf:Person* que colabora con el proyecto de algún otro modo.
- **repository:** es el repositorio del proyecto.
- **release:** es una versión que describe el actual estado del software del proyecto en cuestión.

3.4.3. Ejemplo Ilustrativo:

```
<?xml version="1.0"?>
<Project
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://usefulinc.com/ns/doap#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:admin="http://webns.net/mvcb/">

     <name>Portal de Busquedas Semanticas</name>
     <shortname>PortalSemantico</shortname>
     <shortdesc>Portal de busquedas semanticas para favorecer el desarrollo
                de proyectos de Software Libre</shortdesc>
     <description>Portal de busquedas semanticas para favorecer el desarrollo
                de proyectos de Software Libre aplicando conceptos sobre
                ontologias.</description>
     <homepage rdf:resource="http://portal.semantico.SWL.com" />
     <wiki rdf:resource="http://portal.semantico.es/wiki" />
     <download-page rdf:resource="http://portal.semantico.es/download" />
     <download-mirror rdf:resource="http://portal.semantico.es/mirror" />
     <bug-database rdf:resource="http://portal.semantico.es/bugDB" />
     <category rdf:resource="http://cateorias.es/web+semantica" />
     <os></os> <!-Si este campo no esta relleno significa que es multiplataforma-->
     <programming-language>PHP</programming-language>
     <programming-language>RDF</programming-language>
     <programming-language>FOAF</programming-language>
     <license rdf:resource="http://usefulinc.com/doap/licenses/gpl" />
     <maintainer>
      <foaf:Person>
         <foaf:name>Elena Lozano Rosch</foaf:name>
         <foaf:homepage rdf:resource="http://elena.lozano.rosch"/>
         <foaf:mbox_sha1sum>04fd79682ee97fe0d90dd9e51100a141556e2d6a</foaf:mbox_sha1sum>
      </foaf:Person>
     </maintainer>
     <repository>
       <SVNRepository>
         <browse rdf:resource='http://forja.rediris.com/svn/portalSemantico' />
         <location rdf:resource='http://forja.rediris.com/svn/portalSemantico' />
       </SVNRepository>
     </repository>
    </Project>
```

Figura 3.26: Ejemplo ilustrativo de un documento DOAP.

3.5. Lenguajes de Recuperación

Los Lenguajes de Recuperación (también denominados *query languages*) son muy importantes para las ontologías debido a que son una herramienta que facilita la recuperación y organización de la información representada por diferentes vocabularios y esquemas. Especifican las normas que rigen la formulación de las consultas o *queries*, aportando precendencia, posibilidades de combinación y orden para que las búsquedas sean más eficientes.

Los lenguajes de recuperación pueden dividirse en dos grandes grupos, distinguiendo si su uso se enfoca a las bases de datos relacionales o a la recuperación de información.

Una consulta en estos lenguajes se compone de *términos* y *operadores*. Los términos serán las palabras que informan al sistema sobre lo que debe ejecutarse. Los operadores son los encargados de expresar las relaciones que mantienen entre sí los términos que definen. Existen cuatro tipos distintos:

- **Operadores lógicos:** Son los más utilizados, se basan en que los conceptos semánticos y pueden expresarse como relaciones entre conjuntos, por lo que pueden ejecutarse operaciones lógicas sobre ellos, dando resultado a otro conjunto. Algunos de estos operadores son la unión (`OR`), el producto (`AND`) y la negación (`NOT`).
- **Operadores posicionales:** Se basan en los booleanos, pero van más allá, considerando la posición de un término en relación con los otros.
- **Operadores de comparación:** Especifican cuál es el rango de búsquedas en el que hay que actuar. Los límites pueden regirse por operadores del tipo *mayor que* o *menor que*.
- **Operadores de truncamiento:** Se dan en el caso en el que sea necesario utilizar un término derivado (ya sea por sufijos, prefijos o por variantes léxicas) Son operadores (normalmente símbolos como *, $),que pueden sustituir a un carácter o a un conjunto de caracteres, situados a la izquierda, dentro o a la derecha del término en cuestión.

Algunos ejemplos de lenguajes de recuperación son los siguientes:

- **SPARQL**, *SPARQL Protocol and RDF Query Language.*
- **SeRQL** , *Sesame RDF Query Language*.
- **SQL**, *Structured Query Language* (utilizado para bases de datos relacionales).
- **XQuery**, diseñado para consultar colecciones de datos XML.

Debido a que el lenguaje de recuperación utilizado para este proyecto es el **SPARQL**, pasamos a analizarlo más detalladamente.

SPARQL

Este lenguaje de recuperación basado en RDF es una recomendación para crear un lenguaje de consulta dentro de la Web semántica. Su utilidad está en que puede representar y utilizar los resultados obtenidos en las búsquedas a través de una gran variedad de información.

Está compuesto por tres especificaciones diferencidas:

- *SPARQL Query Language*, que define su sintaxis.
- *SPARQL Protocol*, formato que devuelve las consultas a partir de un XML.
- *SPARQL Query XML Results Format*, el cual describe cómo acceder remotamente a datos o transmitir consultas.

SPARQL tiene unos elementos denominados *triples*, al igual que los de RDF, que están formados por un sujeto, un predicado y un objeto. Estos *triples* son los que describen un recurso y se utilizan en las consultas de este lenguaje.

Un ejemplo de un *triple* podría ser el que se ve en la figura 3.27.

```
1 <http://xmlns.com/foaf/0.1/> foaf:mbox "elozano@us.es".
```

Figura 3.27: *Triple* en SPARQL.

Donde la cadena escrita entre *menor que* y *mayor que* es la URI que identifica a un elemento. La cadena central (`foaf:mbox`) es la especificación de ese elemento con la notación `namespace:elemento`. La última parte del *triple* es el objeto o valor del elemento que estamos describiendo (siempre entre comillas).

Si quisiéramos buscar mas de un elemento con alguna de las características comunes, utilizaríamos variables de la forma `?elemento voc:nombre ?valor` , que nos permitirían operar con las mismas. Podemos ver un ejemplo de ésto en la figura 3.28.
Cuando ejecutemos una búsqueda con este *triple*, la variable `?email` nos permitiría obtener todo recurso con una propiedad `mbox`.

```
1 <http://xmlns.com/foaf/0.1/> foaf:mbox ?email
```

Figura 3.28: *Triple* en SPARQL con variables.

Para realizar una consulta en SPARQL hay que seguir el modelo que podemos ver en la figura (3.29).

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?email
3 FROM <http://lugar_donde_esta_el_documento_rdf/archivo.rdf>
4 WHERE { ?URI foaf:mbox ?email. }
```

Figura 3.29: Consulta en SPARQL.

En la primera línea tenemos la palabra clave `PREFIX`, la cual sirve para declarar *namespaces*.

Esto permite no tener que repetir constantemente un URI, ahorrando espacio y tiempo. En la segunda línea podemos ver la cláusula `SELECT` con una variable, que es la que se devolverá tras la consulta. Se puede poner más de una variable. En la tercera línea tenemos la palabra clave `FROM`, donde especificamos el lugar de donde tenemos que coger la información donde deseamos realizar la búsqueda. Y, por último, la cláusula `WHERE`, donde ponemos el o los *triples* que queramos utilizar en la consulta.

Capítulo 4

Symfony

4.1. Introducción

Symfony es un framework[1] para aplicaciones web en el lenguaje PHP (*PHP Hypertext Preprocessor*) que se caracteriza por:

- Sigue el patrón Modelo Vista Controlador o *MVC*. [2]
- Se puede instalar en todo tipo de plataformas. (*Multiplataforma*)
- Fácilmente extensible. (Usando PEAR[3], por ejemplo)
- Es independiente del gestor de base de datos.
- Tiene un framework de desarrollo de pruebas unitarias, un panel de depuración y un sistema de logs, entre otros. Todo esto nos da una forma de probar que el código funciona correctamente.

[1] Un framework es una tecnología que simplifica el desarrollo de una aplicación automatizando ciertos elementos comunes que se suceden en el proceso de desarrollo de la misma. Además simplifica el código siguiendo una estructura determinada que ha sido optimizada en sucesivos análisis y facilita la programación debido a su aspecto modular.

[2] El patrón MVC es un patrón de arquitectura de software que separa los datos de una aplicación (*modelo*), el interfaz de usuario (*vista*), y la lógica de control (*controlador*) en tres componentes distintos. Es un patrón muy utilizado en las aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el controlador es el Sistema de Gestión de Base de Datos y el modelo es el modelo de datos.

[3] PHP Extensión and Application Repository, o PEAR, es un entorno de desarrollo y sistema de distribución para componentes de código PHP. Contiene un numero bastante elevado de librerías de código PHP que permiten hacer ciertas tareas de manera más rápida y eficiente reutilizando código escrito previamente por otras personas.

- Automatiza características comunes de los proyectos web, por lo que no hace falta *reinventar la rueda* de aspectos triviales, pero laboriosos de los mismos.
- Usa PHP5, la primera versión de PHP con soporte sólido para Programación Orientada a Objetos.
- Usa ORM (*Object relational mapping*). [4]
- Usa YAML para los archivos de configuración del modelo. [5]
- Su filosofía de trabajo se basa en RAD. [6]
- Es compatible con la mayoría de gestores de bases de datos (MySQL, PostgreSQL, Oracle, etc.)
- Para la configuración del mismo se utiliza la línea de comando, automatizando bastantes acciones rutinarias.
- Se pueden realizar cambios de la configuración sin necesidad de reiniciar el servidor.

Symfony nació de una investigación en la que se concluyó que no existía ninguna herramienta de Software Libre disponibles para el desarrollo de aplicaciones web con PHP. Una empresa francesa de desarrollo de aplicaciones web, *Sensio (http://www.sensio.com/)*, fue la que desarrolló el proyecto. En un principio el framework se desarrolló para utilizarlo en proyectos internos de la empresa, pero al cabo del tiempo, se publicó bajo licencia MIT.

[4] El ORM es un interfaz de programación que traduce los datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos. En la práctica ésto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional, lo cual posibilita el uso de las características propias de la orientación a objetos como pueden ser la herencia y polimorfismo.

[5] El YAML es un lenguaje que describe los datos con estructura similar a xml, pero con sintaxis más sencilla.

[6] El RAD o *Rapid application development*, es un método que comprende el desarrollo interactivo, la construcción de prototipos y el uso de herramientas que aumenten la productividad en el desarrollo del software (denominadas herramientas CASE, *Computer Aided Software Engineering*) para un mejor desarrollo de las aplicaciones, mejorando también la usabilidad, utilidad y la rapidez de ejecución de las mismas.

Aunque Symfony es un framework realmente útil, no lo es para todos los tipos de proyectos. Symfony *no* es adecuado para los proyectos con un acceso simple a la base de datos, donde no sea importante el rendimiento o con un numero no muy elevado de páginas web (entre cinco y diez páginas).

Por el contrario, *es* adecuado para aplicaciones complejas, con mucha lógica de negocio, donde el rendimiento sea algo importante para la aplicación y sea necesaria una actualización y extensión del código a medida que se vaya necesitando.

4.2. Creación de un proyecto en Symfony

Para crear la estructura de un proyecto en Symfony hay que seguir los pasos siguientes:

- **Crear proyecto:** Crear una carpeta con el nombre del proyecto y desde la misma ejecutar: `symfony init-project nombreDeMiProyecto`.
  Ésto crea varios directorios: apps/, batch/, cache/, config/, data/, doc/, lib/, log/, plugins/, test/ y web/.

- **Inicializar Aplicación**: (Backend, Frontend, etc.). Está formada por uno o más módulos. Para crear una aplicación situarse dentro de la carpeta de symfony y ejecutar
  `symfony init-app nombreDeMiApp`.
  Ésto crea varios directorios dentro de apps/: config/, il8n/, lib/, modules/ y templates/. También crea un archivo: miApp.php (Éste es el control fontral de producción de la nueva aplicación).

- **Inicializar Módulos:** Representan a una página web o a un conjunto de páginas que tienen un mismo objetivo. Los módulos almacenan acciones. Para crear un módulo hay que ejecutar
  `symfony init-module miApp miModulo`
  Ésto crea varios directorios: actions/, config/, lib/, templates/ y validate/.
  También crea tres archivos: uno relacionado con las pruebas unitarios en test/, otro denominado `config/actions.class.php`, el cual redirije la acción a la página de bienvenida del

módulo y `templates/indexSuccess.php`, que está vacío en un principio.

- **Crear Acciones:** Representan las operaciones que pueden realizarse en un módulo (añadir, mostrar, actualizar). Éstas se deben incluir en el archivo `actions.class.php` del módulo que deseemos modificar.

4.3. Modelo en Capas de Symfony: Modelo

Symfony utiliza *Propel* [7] como ORM [8] y *Creole*[9] como la capa de abstracción de bases de datos. Siguiendo esta estructura, los pasos a seguir para el desarrollo de esta capa son los siguientes:

- Describir el esquema relacional de la base de datos que vayamos a necesitar.Ésto debe hacerse creando un documento en formato YAML antes de generar el las clases del modelo de datos, en la ruta `config/schema.yml`.

- Configurar correctamente todos los datos necesarios para la conexión con la base de datos en el archivo `config/databases.yml`

- Ejecutar en la consola, dentro del directorio del proyecto Symfony sobre el que estemos trabajando, el comando:
  `symfony propel-build-model`,
  el cual convierte a xml el .yml y genera el ORM.

- Ejecutar en la consola, el comando:
  `symfony propel-build-sql`,

[7] Propel es un framework ORM para PHP5. Su objetivo es permitir el acceso a la base de datos usando un conjunto de objetos gracias a una librería ya implementada, abstrayendo la programación de todo contacto directo con la base de datos.

[8] El ORM es un interfaz de programación que traduce los datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos como pueden ser la herencia y polimorfismo.

[9] Creole es una forma de abstracción de la base de datos para PHP5. Crea un código más portable y limpio con un interfaz sencillo que permite trabajar con conceptos de programación orientada a objetos

que con la información generada en el paso anterior, genera el código SQL necesario para la creación de la base de datos.

- Ejecutar en la consola, el comando:
  `symfony propel-insert-sql`,
  el cual ejecuta el SQL generado y crea la base de datos.

- Ejecutar en la consola, el comando:
  `symfony propel-generate-crud miApp miModulo miModulo`,
  que genera código común a todas las aplicaciones web con respecto a la base de datos automáticamente desde el modelo.
  Crea unas clases *peer* que son las que permiten acceder a la información de la base de datos.
  Cabe destacar que estas clases pueden redefinirse de una forma sencilla si es necesario hacer más compleja la interacción con la base de datos.

La capa del modelo es la más compleja de Symfony debido a que la manipulación de los datos no es algo trivial y, por si eso fuera poco, hay que tener en cuenta la seguridad necesaria de los datos en las aplicaciones web. Aun así es casi la más importante de las capas.

4.4. Modelo en Capas de Symfony: Vista

La capa de la vista se encarga de generar las páginas que se muestran como resultado de las acciones definidas en el archivo `action.class.php` del modelo que estemos generando.

En Symfony, la vista se compone de varias partes, cuyo objetivo es ser lo más modular y modificable posible. Esto se debe a que en un proyecto de gran envergadura, las personas que se encargan de la vista no suelen tener grandes conceptos de programación y están especializados sólo en el diseño.

La vista es una combinación de una plantilla (template) descrita en un simple archivo `.php` y un archivo de configuración que nos indica cómo se va ajustar la plantilla a los demás elementos. La plantilla suele ser básicamente código HTML y un poco de PHP, el cual se encarga de llamar a las acciones definidas en el archivo `action.class.php` y de hacer uso de los *helpers*, pequeños trozos de código que nos automatizan y simplifican las tareas más triviales.

Un ejemplo de ésto son las dos formas de representar lo mismo que podemos ver en la figura 4.1. Los *helpers* pueden parecer algo enrevesados, pero una vez entendidos hacen mucho más rápido el desarrollo, puesto que evitan al programador tener que redundar código: sólo escribe la información realmente necesaria.

```
1 <!-- Forma de representar un elemento Option mediante un helper -->
2 <?php echo options_for_select(array(
3   '1' => 'Luis',
4   '2' => 'Pedro',
5   '3' => 'Mar'
6 ), 4) ?>
7
8 <!-- Forma de representar un elemento Option en HTML -->
9     <option value="1">Luis</option>
10    <option value="2">Pedro</option>
11    <option value="3">Mar</option>
```

Figura 4.1: Estructura de la vista en Symfony: *helpers*.

El estructura básica de la vista de un módulo en Symfony es la que podemos ver en la figura 4.2. Esta se basa, como ya dijimos, en separar la vista en dos partes: la relacionada con la acción (en el directorio `mimodulo/config`) y la relacionada con el resultado. (en el directorio `mimodulo/templates`)

Resumiendo un poco todo lo que hemos hablado de la vista, podemos concluir que existen numerosas herramientas y utilidades para manipular la capa correspondiente a la presentación. (*Plantillas, layouts, helpers* y otros elementos más avanzados que no hemos nombrado aquí). Además todas estas

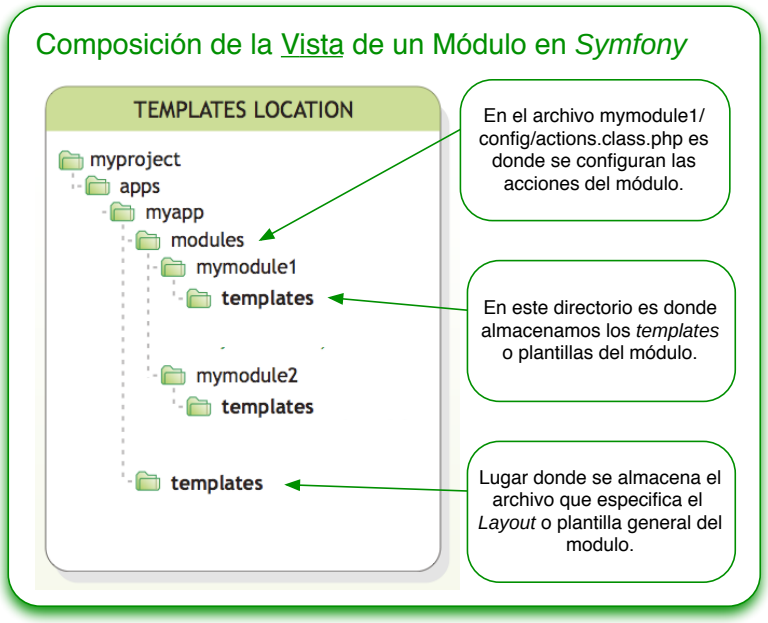

Figura 4.2: Estructura de la vista en Symfony: *Módulos*.

herramientas son configurables desde archivos `.yml`, lo cual permite configurar toda la vista de una sola vez.

4.5. Modelo en Capas de Symfony: Controlador

La capa del controlador se puede dividir en dos partes diferenciadas:

- El **controlador frontal** o *Front Controller*, el cual es el único punto de entrada a la aplicación. Éste carga la configuración y determina la acción al ejecutarse.

  El Front Controller es un patrón de diseño [10] que consiste en que un elemento del sistema (el Front Controller) maneje todas las peticiones que le llegan a una aplicación web y decida qué se debe hacer con ellas.

  En Symfony, las tareas del controlador frontal son las siguientes:

[10] Un patrón de diseño es una solución a un problema de diseño no trivial que es *efectiva*, debido a que ya se resolvió el problema satisfactoriamente en ocasiones anteriores y *reutilizable*, porque se puede aplicar a diferentes problemas de diseño en distintas circunstancias)

- Definir las constantes del núcleo de la aplicación.
- Localizar la librería de Symfony
- Cargar e inicializar las clases del núcleo del framework.
- Cargar la configuración.
- Decodificar la URL de la petición para determinar la acción a ejecutar y los parámetros de la petición.
- Si la acción no existe, redireccionar a la acción del error 404.
- Activar los filtros (por ejemplo, si la petición necesita autenticación).
- Ejecutar los filtros en una primera pasada.
- Ejecutar la acción y producir la vista.
- Ejecutar los filtros en una segunda pasada.
- Mostrar la respuesta.

- Las **acciones**, que son las que contienen la lógica de la aplicación. Verifican la integridad de las peticiones y preparan los datos requeridos por la capa de presentación.

Para que estos dos aspectos interactúen entre ellos existen varios objetos como son `sf_request`, `sf_response` y `sf_session`. Éstos dan acceso a los parámetros de la petición, las cabeceras de las respuestas y a los datos persistentes del usuario.

Otra de las características de esta capa es la existencia de *filtros*, trozos de código ejecutados para cada petición, antes o después de una acción. Ejemplos de éstos son los filtros de seguridad y validación. Además, en Symfony puedes crear filtros propios, aunque normalmente hay pluggins y extensiones muy completas para multitud de necesidades en la programación.

4.6. Estructura de Directorios en Symfony

Estructura del directorio raíz en Symfony

La descripción de la estructura del directorio raíz que genera Symfony es la siguiente:

- `apps/`: Contiene un directorio por cada aplicación del proyecto (normalmente se definen los denominandos *frontend* y *backend* para la parte pública y la parte de gestión respectivamente).
- `batch/`: Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes (denominados *batch processes*).
- `cache/`: Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de cache utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados.
- `config/`: Almacena la configuración general del proyecto.
- `data/`: En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de SQLite.
- `doc/`: Contiene la documentación del proyecto, formada por los archivos propios del usuario y por la documentación generada por PHPdoc.
- `lib/`: Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio `model/` guarda el modelo de objetos del proyecto.
- `log/`: Guarda todos los archivos de log generados por Symfony. También se puede utilizar para guardar los logs del servidor web, de la base de datos o de cualquier otro componente del proyecto. Symfony crea un archivo de log por cada aplicación y por cada entorno.
- `plugins/`: Almacena las extensiones o *plugins* instalados en la aplicación.
- `test/`: Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el framework de pruebas de Symfony. Cuando se crea un proyecto, Symfony crea algunos pruebas básicas.

- `web/`: La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio.

Para una mayor comprensión de cada uno de los aspectos estructurales comentados, véase la figura 4.3.

Estructura de una aplicación en Symfony

A continuación se incluye una lista en la que se explica cuál es el papel de cada uno de los elementos de la estructura de una aplicación en Symfony.

- `config/` Contiene los archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del *framework*. También es posible redefinir en este directorio los parámetros por defecto si es necesario.
- `i18n/` Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz . La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría.
- `lib/` Contiene las clases y librerías utilizadas exclusivamente por la aplicación.
- `modules/` Almacena los módulos que definen las características de la aplicación.
- `templates/` Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado `layout.php`, que es el diseño principal con el que se muestran las plantillas de los módulos.

Para una mayor comprensión de cada uno de los aspectos estructurales comentados, véase la figura 4.3.

Estructura de un módulo en Symfony

La descripción de la estructura de un módulo en Symfony es la siguiente:

- `actions/` Normalmente contiene un único archivo llamado `actions.class.php` y que corresponde a la clase que almacena todas las acciones del módulo.
  También es posible crear un archivo diferente para cada acción del módulo.
- `config/` Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo.
- `lib/` Almacena las clases y librerías utilizadas exclusivamente por el módulo.
- `templates/` Contiene las plantillas correspondientes a las acciones del módulo.
  Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada `indexSuccess.php`.
- `validate/` Contiene archivos de configuración relacionados con la validación de formularios.

Para una mayor comprensión de cada uno de los aspectos estructurales comentados, véase la figura 4.3.

Estructura del directorio Web en Symfony

La estructura del directorio Web en Symfony es la siguiente:

- `css/` Contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css).
- `images/` Contiene las imágenes del sitio con formato .jpg, .png o .gif.
- `js/` Contiene los archivos de JavaScript con extensión .js.
- `uploads/` Se almacenan los archivos subidos por los usuarios.
  Aunque normalmente este directorio contiene imágenes, no se debe confundir con el directorio que almacena las imágenes del sitio `(images/)`. Esta distinción permite sincronizar los servidores de desarrollo y de producción sin afectar a las imágenes subidas por los usuarios.

Para una mayor comprensión de cada uno de los aspectos estructurales comentados, véase la figura 4.3.

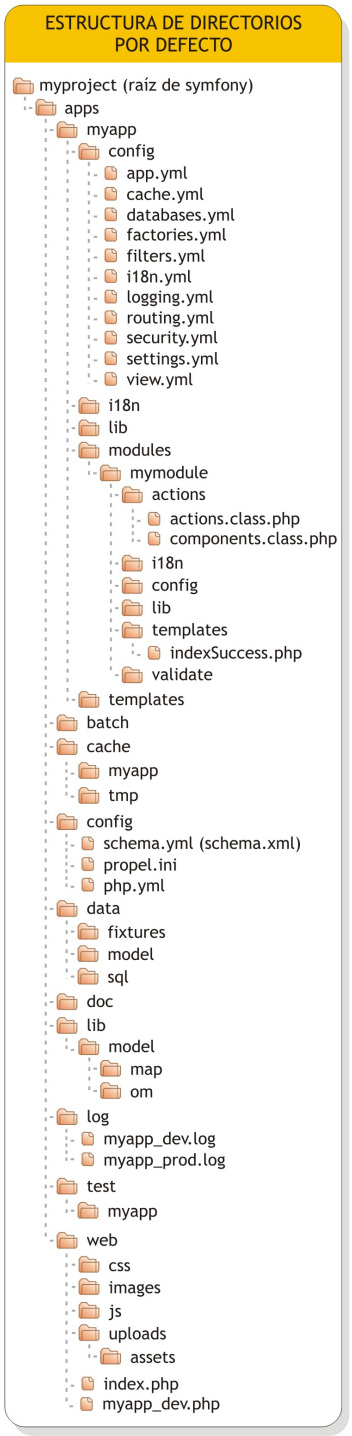

Figura 4.3: Estructura de directorios en Symfony.

Capítulo 5

Portal de colaboración con capacidades semánticas

5.1. Introducción

Una vez explicadas las tecnologías utilizadas en este proyecto, ya tenemos una idea más clara sobre ciertos términos que nos permitirán conocer más a fondo cuál es la estructura del proyecto.

En este capítulo veremos, en primer lugar, cuáles son los requerimientos necesarios para instalar la aplicación web y la explicación paso a paso de cada uno de ellos.

En otra sección veremos cuál es el dominio de la aplicación y su modelo de datos, es decir, cuáles son los elementos que la forman y qué funciones pueden realizar éstos.

Posteriormente veremos la estructura del portal desde dos puntos de vista distintos: por un lado la estructura que tiene la aplicación web como tal respecto a Symfony y por otro la especificación de la misma en cuanto a las ontologías utilizadas.

5.2. Requerimientos de la aplicación

Los requerimientos necesarios para ejecutar la aplicación son los siguientes:

- Servidor MySQL.
- Servidor Apache.
- Módulos de PHP5 y MySQL instalados en el Servidor Apache.
- Librerías Redland, Raptor, Redland Bindings, Rasqal y RAP.

- Symfony.

En los siguientes apartados se describe cada uno de ellos detalladamente y la forma de configurarlos para poder ejecutar la aplicación web.

5.2.1. Servidor MySQL

Es un sistema de gestión de bases de datos relacionales[1], que funciona de forma separada del servidor Web.

Es multihilo y multiusuario y además tiene implementadas características como Triggers, Vistas, Soporte para Unicode, Cursores, etc.

Esta disponible para una gran variedad de sistemas operativos: BSD, FreeBSD, GNU/Linux, Mac OS X, NetBSD, OpenBSD, Solaris, SunOS y las distintas versiones de Windows, entre muchos otros. Además es Open Source, por lo que es posible modificarlo y usarlo libremente.

Para instalarlo hay que ir a la página oficial: www.mysql.com y seguir los pasos que allí se exponen dependiendo del sistema operativo que se tenga instalado.

5.2.2. Servidor Apache

El servidor HTTP Apache es un servidor HTTP de código abierto multiplataforma.
Un servidor web es una aplicación que implementa el protocolo HTTP, el usado en las transacciones de información en la web. Se encarga de esperar peticiones HTTP para responderlas con la información que sea necesaria.

La *Apache Software Foundation* es la encargada de desarrollar este software dentro de su proyecto HTTP Server (httpd).

Para instalar un Servidor Apache, hay que bajarse la aplicación de la página http://httpd.apache.org/ y seguir los pasos que allí se exponen.

Un requisito indispensable para que la aplicación funcione es tener instalados los módulos de PHP5 y MySQL en el servidor Apache.

[1] Una base de datos relacional es la que es capaz de almacenar los datos en tablas separadas en vez de almacenarlo todo por igual en una misma localización. La ventaja de este tipo de base de datos es que consigue una mayor velocidad de búsqueda y almacenaje, a la vez que una mayor flexibilidad.

5.2.3. Librerías Redland, Raptor, Redland Bindings, Rascal y RAP

Las librerías utilizadas en la aplicación para la parte relacionada con las ontologías son las siguientes:

- **Redland** es un conjunto de librerías, las cuales son Software Libre, que dan soporte a RDF. Está orientada a objetos e implementada en C. Es capaz de manipular grafos RDF, URIs y otros elementos de RDF, de almacenar en distintos tipos de bases de datos la información y de realizar búsquedas semánticas sobre datos con formato RDF, utilizando para éstas búsquedas tecnologías como SPARQL o RDQL (ver apartado de lenguajes de recuperación: 3.5), según la preferencia del desarrollador.
  Aunque está programada en C tiene distintas extensiones, denominadas *bindings*, que permiten utilizar esta librería en otros lenguajes tales como Java, Perl, Python, PHP, Ruby, etc.

  Las librerías que componen la API[2] de REDLAND son las siguientes:

  - Raptor RDF Parser Toolkit: librería que permite serializar y formatear la información en RDF.
  - Rasqal RDF Query Library: permite ejecutar consultas RDF.
  - Redland RDF Library: La librería de Redland en sí misma. Necesita de las anteriores para poder funcionar correctamente.
  - Redland Language Bindings: Permiten que Redland funcione en C#, Java, Obj-C, Perl, PHP, Python, Ruby y Tcl.

- **RAP**: (*RDF API for PHP*) Es otra librería de RDF para PHP. Aunque abarca menos funcionalidades que Redland, tiene implementada una colección de funciones específicas para crear un documento FOAF que han sido de gran utilidad en este proyecto.

La instalación de ambas librerías no es trivial, por lo que a continuación se detalla cómo debe hacerse.

[2]Una API (*Application Programming Interface, Interfaz de Programación de Aplicaciones*) es el conjunto de funciones y procedimientos que ofrece una *biblioteca* para ser utilizado por otro software como una capa de abstracción.

Instalación de Redland:

(Pasos de la instalación descriptos para un Sistema Operativo tipo UNIX)

- Bajar el tar.gz de redland y el de los bindings de http://download.librdf.org/.Descomprimirlos
- Renombrar la carpeta de *Redland 2.X.X* a *Redland*.
- Dentro de la carpeta *Redland* (`cd Redland`): ejecutar en este orden en la consola:
  `./configure`
  `make`
  `make check`
- En el caso de que no hubiera ningún error hasta este paso, hacer `make install`.
- Ejecutar después los siguientes comandos en esa misma carpeta:
  `./configure --with-php` (podría ser `--with-ruby` o `--with --X` siendo X el lenguaje correspondiente al paquete que necesites.)
  `make`
  `make check`
  Y, si no hay ningún error, `make install`
- Un error común en este punto de la instalación es el siguiente:
  `unix.h: No such file or directory`.
  Para solucionarlo hay que buscar dónde se encuentra el archivo
  (`find dir_donde_tengamos_el_servidor_apache -name 'unix.h'`).
  Con la ruta resultante modificamos el archivo *makefile* de la carpeta del binding del lenguaje que estemos utilizando cambiando `AM_CPFLAGS` por
  `AM_CPPFLAGS = -I/dir_donde_encontramos_el_archivo_unix.h`
- Para que al realizar el `make check` de los pasos anteriores no de error, hay que modificar el archivo `test.php` que se encuentra dentro de los bindings, de forma que hay que dejarlo tal y como vemos en las figuras 5.1 y 5.2.

```
<?php
global $REDLAND_LOADED__;
if ($REDLAND_LOADED__) return;
if (!extension_loaded("redland")) {
  /* PHP 4.3 provides PHP_SHLIB_PREFIX and PHP_SHLIB_SUFFIX */
  if (!defined('PHP_SHLIB_SUFFIX')) {
    define('PHP_SHLIB_SUFFIX', strtoupper(substr(PHP_OS, 0,3)) == 'WIN' ? 'dll' : 'so');
  }
  if (!defined('PHP_SHLIB_PREFIX')) {
    define('PHP_SHLIB_PREFIX',PHP_SHLIB_SUFFIX == 'dll' ? 'php_' : '');
  }
  if (!dl(PHP_SHLIB_PREFIX . "redland" . "." . PHP_SHLIB_SUFFIX )){
    die('no redland?');
    exit;
  }
}
$REDLAND_LOADED__ = true;
$world=librdf_php_get_world();
$storage=librdf_new_storage($world,'hashes','dummy',"new=yes,hash-type='memo y'");
$model=librdf_new_model($world,$storage,'');
$new_uri=librdf_new_uri($world,'-');
$parser2=librdf_new_parser($world,'raptor','application/rdf+xml',NULL);
$uri=librdf_new_uri($world,'file:../data/dc.rdf');
librdf_parser_parse_into_model($parser2,$uri,$uri,$model);
librdf_free_uri($uri);
librdf_free_parser($parser2);
$nulluri = librdf_new_uri ($world, '-');
$query = librdf_new_query($world, 'sparql', $nulluri,
    "PREFIX dc: <http://purl.org/dc/elements/1.1/> SELECT  ?a ?c ?d " +
    "WHERE { ?a dc:title ?c . OPTIONAL { ?a dc:related ?d } }", $nulluri);
$results=librdf_model_query_execute($model, $query);
$count=1;
while($results && !librdf_query_results_finished($results)) {
  for ($i=0; $i < librdf_query_results_get_bindings_count($results); $i++)
  {
    $val=librdf_query_results_get_binding_value($results, $i);
    if ($val)
      $nval=librdf_node_to_string($val);
    else
      $nval='(unbound)';
    print "  ".librdf_query_results_get_binding_name($results, $i)."=".$nval."\n";
  }
  print "}\n";
  librdf_query_results_next($results);
  $count++;
}
if ($results)
  print "Returned $count results\n";
$results=null;
$results=librdf_model_query_execute($model, $query);
if ($results) {
    $format_uri=librdf_new_uri($world,
        "http://www.w3.org/TR/2004/WD-rdf-sparql-XMLres-20041221/");
    $str=librdf_query_results_to_string($results, $format_uri, $nulluri);
    print "Query results serialized to an XML string size ".strlen($str)." bytes\n";
```

Figura 5.1: Contenido de test.php: parte I

```
} else
    print "Query results couldn't be serialized to an XML string\n";
$serializer2=librdf_new_serializer($world,'rdfxml',NULL,NULL);
$base=librdf_new_uri($world,'http://example.org/base.rdf');
librdf_serializer_serialize_model_to_file($serializer2,'./test-out.rdf',$base ,$model);
librdf_free_serializer($serializer2);
librdf_free_uri($base);
librdf_free_model($model);
librdf_free_storage($storage);
?>
```

Figura 5.2: Contenido de test.php: parte II

Instalación de RAP:

Para instalar RAP, la mejor manera es la siguiente:

- Bajarse la última versión de la página http://sourceforge.net/projects/rdfapi-php/
- Descomprimir los archivos
- Incluir la API de RDF en los archivos PHP que necesites de la forma:
  `define('RDFAPI_INCLUDE_DIR', 'dir_donde_esta_la_api_de_RDF/api');`
  `include(RDFAPI_INCLUDE_DIR . 'RDFAPI.php');`

Existe otra forma de instalar RAP y es haciendo uso de la herramienta *PEAR*(*PHP Extension and Application Repository*), un repositorio de librerías y paquetes para PHP que funciona mediante consola. Para instalarlo sólo hace falta ejecutar el comando `pear install XML_FOAF` e incluir también la API de RDF en los archivos PHP que lo requieran.

5.2.4. Symfony

Para instalar Symfony (véase el apartado 4) existen tres formas distintas:

Mediante paquetes *PEAR*

Esta forma es la más sencilla y fácil de actualizar. Los pasos a seguir son los siguientes:

- Ejecutar en consola el comando
  `>pear channel-discover pear.symfony-project.com`
- Ejecutar: `>pear install symfony/symfony`

Para comprobar que tenemos instalado todo correctamente podemos ejecutar en consola el comando `>symfony -V`, el cual nos mostrará la versión actual de Symfony.

Mediante repositorio SVN

Este tipo de instalación se recomienda para los programadores de PHP avanzados y es el método con el que pueden obtener los últimos parches, pueden añadir sus propias características al framework y pueden colaborar con el proyecto en sí mismo.

Los comandos que deben ejecutarse son los siguientes:

```
>mkdir /dir_que_se_desee/
>cd /dir_que_se_desee/
>svn checkout http://svn.symfony-project.com/tags/RELEASE_1_0_0/ .
```

Para comprobar que tenemos instalado todo correctamente podemos ejecutar en consola el comando `>php /ruta/a/symfony/data/bin/symfony -V`

Mediante descarga directa:

La última forma de instalar Symfony es bajar directamente el paquete de PEAR (http://pear.symfony-project.com/get/symfony-1.0.0.tgz) y descomprimirlo en algún directorio. El resultado de esta instalación es el mismo que si se instalara mediante el repositorio Subversion.

5.3. Modelo de la Aplicación

Existen tres elementos básicos que conforman el modelo de la aplicación:

Usuarios

Los usuarios son las personas que utilizan la aplicación. Hay tres tipos: Administradores, usuarios sin registrar y usuarios registrados, donde cada uno podrá acceder a distintas funcionalidades.

Los administradores podrán acceder a todas las funcionalidades del sistema, mientras que los usuarios sin registrar sólo podran visualizar la información. No podrán añadir ideas ni colaborar con otras ideas o proyectos.

Los usuarios registrados, en cambio, podrán decidir de qué tipo son según sus intereses: *investigador* o *desarrollador*. La función del *investigador* sería dar ideas para realizar un proyecto y pedir colaboración a otros docentes y a los desarrolladores para realizar la idea que propone. Los *desarrolladores* podrán también dar ideas, aunque su función principal será colaborar activamente en el

desarrollo del proyecto.

Los datos que se almacenarán acerca de los usuarios registrados es la siguiente:

- Nombre.
- Apellidos.
- Nombre de usuario o *nick*.
- Correo electrónico.
- Imagen que lo identifique (opcional)
- Página Web personal (opcional)
- Weblog (opcional)
- Página de su lugar de trabajo o estudio (opcional)
- Identificador de Jabber (opcional)

Aparte de esta información, almacenaremos el perfil del usuario, el cual estará formado por:

- Tipo de **plataforma** (sistema operativo: Unix, Windows...) de los proyectos e ideas para la que desea colaborar.
- **Tecnología** (PHP, Java, C...) de los proyectos e ideas para la que desea colaborar.
- Tipo de **licencia** de las ideas y proyectos en los que quiere estar implicados.
- **Temática** de las ideas que propone o de los proyectos en los que quiere colaborar

Ideas

Una idea es una *propuesta* de un proyecto a desarrollar, la pueden dar tanto usuarios docentes/investigadores como desarrolladores.

Los datos de una idea que almacenaremos serán su **título**, la **descripción** de la misma y el **usuario** registrado que la ha propuesto.

Habrá, además, una clasificación parecida a la del perfil de los usuarios que contiene las características que el usuario que ha propuesto la idea quiere para la misma. Esta clasificación estará compuesta del tipo de **plataforma** para la que se desarrollará la idea, la **tecnología** o lenguaje en la que se realizará, la **licencia** y la **temática** que tendrá.

Proyectos

Un proyecto es lo que conocemos como un proyecto de software. Se crea una vez que se han puesto de acuerdo desarrolladores e investigadores para desarrollar una idea que ya haya sido propuesta.

La información que almacenaremos del proyecto serán:

- Nombre.
- Nombre corto. El que se utilizará para los ficheros del mismo.
- Descripción.
- Descripción corta. (Ocho o nueve palabras)
- Página principal o *homepage*
- Wiki
- Página de descarga.
- Mirror de descarga.
- Base de datos de errores
- Plataforma para la que está desarrollado (Si es multiplataforma, no se rellena)
- Licencia
- Lenguaje en el que está desarrollado
- Personas que colaboran en el proyecto: Colaborardor, desarrollador, creador (el de la idea), líder del proyecto.
- Repositorio: tipo, URL donde se encuentra.

También tendrá, al igual que las ideas, una clasificación con las características del proyecto: **plataforma**, **tecnología**, **licencia** y **temática**.

5.4. Estructura de la aplicación desde el punto de vista de Symfony

En esta sección se analiza cuál es la estructura de la aplicación desde el punto de vista de Symfony, es decir, cuáles son las clases que la forman (apartado 5.4.1) y cómo está organizada la base de datos.(apartado 5.4.2)

5.4.1. Estructura clases

La estructura de clases de la aplicación según la nomenclatura que utiliza Symfony es la siguiente:

Tiene dos aplicaciones a las cuales está permitido el acceso mediante un proceso de autenticación:

- `Frontend`: En ella pueden entrar todos los usuarios, estén o no autenticados, aunque ciertas opciones sólo estarán disponibles para cierto tipo de usuarios.
- `Backend`: Es la parte de la aplicación web que utilizarían los administradores del sitio, pudiendo añadir, modificar y editar todo lo que hay en el portal: borrar usuarios o ideas, editar un perfil, añadir otro proyecto, etc.

-

Dentro del *frontend* podemos distinguir distintos módulos, los cuales, como pudimos ver en el apartado 4, delimitan una funcionalidad o conjunto de funcionalidades que tienen un mismo objetivo.

La organización en módulos de la aplicación es la siguiente:

- Módulo **searches**: Se encarga de realizar todos los tipos de búsquedas (de usuarios, proyectos e ideas), haciendo uso de las funciones auxiliares y las librerías Redland y RAP (Esto se podrá ver más detalladamente en el apartado dedicado a ello, el 5.5).
- Módulo **commonPages**: Tiene implementadas las funciones comunes a todos los tipos de usuarios: crea la página de bienvenida, genera las noticias y búsquedas semánticas adecuadas para cada usuario.
- Módulo **subscribers**: Es el encargado de añadir, borrar, editar y mostrar toda la información relativa a los usuarios que exista en la base de datos. Trata también con los proyectos e ideas asignadas a cada usuario.
- Módulo **subInterests**:Se encarga del tratamiento del perfil de los usuarios: lo edita, añade, modifica y visualiza.
- Módulo **ideas**: Se encarga de añadir, borrar, editar y mostrar toda la información relativa a los ideas que existen en la base de datos. Permite añadir usuarios a una idea.
- Módulo **ideasInterests**:Se encarga del tratamiento de las características del perfil de las ideas: lo edita, añade, muestra y modifica .
- Módulo **projects**:Es el encargado de añadir, borrar, editar y mostrar toda la información relativa a los proyectos que exista en la base de datos. Además trata con los intereses y otros elementos del perfil de éstos. Permite añadir usuarios a cada proyecto.
- Módulo **projInterests**:Se encarga del tratamiento de las características del perfil de los proyectos.

5.4.2. Estructura BD

La base de datos está estructurada en tablas, las cuales analizamos a continuación:

- Tabla **ideas**: En ella almacenamos los datos relativos al registro de una idea.
- Tabla **ideasInterest**: Contiene las características del perfil de las ideas.
- Tabla **projects**: En ella se almacena el registro de todos los proyectos.
- Tabla **projInterest**: Guarda las características que tiene un proyecto.
- Tabla **subscribers**: Es donde guardamos la información de los usuarios del sistema.
- Tabla **subsInterest**: En ella almacenamos las características del perfil de los usuarios.

A continuación podemos ver la base de datos generada mediante Symfony: figuras 5.3, 5.4 y 5.5.

Campo	Tipo
id	int(11)
creator	int(11)
name	varchar(255)
description	text
created_at	datetime
updated_at	datetime

Campo	Tipo
id	int(11)
idIdea	int(11)
technology	varchar(255)
platform	varchar(255)
license	varchar(255)
topic	varchar(255)

Figura 5.3: Tablas *ideas* y *subsInterest* de la base de datos

Campo	Tipo
id	int(11)
name	varchar(255)
shortname	varchar(255)
shortdesc	varchar(255)
description	text
homepage	varchar(255)
wiki	varchar(255)
downloadpage	varchar(255)
downloadmirror	varchar(255)
bugdatabase	varchar(255)
programmingLanguages	varchar(255)
os	varchar(255)
license	varchar(255)
repositoryLocation	varchar(255)
repositoryBrowse	varchar(255)
maintainer	int(11)
developer	int(11)
helper	int(11)
creator	int(11)
created_at	datetime
updated_at	datetime

Campo	Tipo
id	int(11)
idProject	int(11)
technology	varchar(255)
platform	varchar(255)
license	varchar(255)
topic	varchar(255)

Figura 5.4: Tablas *projects* y *projInterest* de la base de datos

Campo	Tipo
id	int(11)
nickname	varchar(255)
password	varchar(255)
name	varchar(255)
surname	varchar(255)
salt	varchar(32)
sha1_password	varchar(40)
email	varchar(255)
imgsrc	varchar(255)
schoolhomepage	varchar(255)
workplacehomepage	varchar(255)
homepage	varchar(255)
weblog	varchar(255)
jabberId	varchar(255)
created_at	datetime
updated_at	datetime
type	varchar(20)

Campo	Tipo
id	int(11)
idSubscriber	int(11)
technology	varchar(255)
platform	varchar(255)
license	varchar(255)
topic	varchar(255)

Figura 5.5: Tabla *subscribers* y *subsInterest* de la base de datos

5.5. Estructura de la aplicación desde el punto de vista de las ontologías utilizadas

Uno de los principales aspectos para generar una aplicación que hiciera uso de tecnologías semánticas es representar los datos del modelo y la información que queramos relacionar semánticamente con un vocabulario específico, ya sea RDF o una extensión de éste.

Otro aspecto que es muy importante es la forma en que se realizan las búsquedas de forma semántica, la forma de recorrer la información y obtener un resultado que se ajuste a lo que se haya requerido.

En los siguientes apartados veremos cómo se ha ajustado el modelo de datos a distintos tipos de representaciones ontológicas y también cuál es el mecanismo o motor de búsqueda que hemos utilizado para realizar las consultas, de forma semántica.

5.5.1. Representación Semántica del Modelo de Datos

Para representar los datos de la aplicación de forma que pudiera ser interpretado como un documento RDF, hemos utilizado dos vocabularios específicos: FOAF (apartado 3.3) y DOAP (apartado 3.4).

Se ha hecho uso de FOAF para describir a los usuarios de la aplicación y de DOAP para describir tanto las ideas como los proyectos.

Modelo de Usuarios en FOAF

El primer caso se ha realizado haciendo uso de la librería RAP (apartado 5.2.3), a cuyas funciones llamamos cuando es necesario crear un documento FOAF de una persona.

La creación de un documento FOAF para una persona mediante RAP se realizaría de la forma que podemos ver en la figura 5.6.

En ella podemos ver cómo incluimos los archivos de la librería RAP mediante la sentencia `require_once 'XML/FOAF.php';` y creamos un nuevo FOAF de una persona `$foaf = new XML_FOAF();`, añadiéndole los distintos datos que queramos añadirle de un usuario.

El modelo que se generaría a partir de este código sería el código en RDF reflejado en la figura 5.7, el cual utilizamos en la aplicación para representar a todos los usuarios.

```php
require_once 'XML/FOAF.php';

$foaf = new XML_FOAF();
$foaf->newAgent('person');
$foaf->addNick($subs->getNickname());
$foaf->setFirstName($subs->getFirstname());
$foaf->setSurname($subs->getSurname());
$name=$subs->getFirstname() + " " + $subs->getSurname();
$foaf->setName($name);
$foaf->addHomepage($subs->getHomepage());
$foaf->addWeblog($subs->getWeblog());
$foaf->addDepiction($subs->getImgsrc());
$foaf->addJabberID($subs->getJabberId());
$foaf->addWorkplaceHomepage($subs->getWorkplacehomepage());
$foaf->addSchoolHomepage($subs->getSchoolhomepage());
```

Figura 5.6: Forma de generar un documento FOAF de un usuario

```xml
<rdf:RDF
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <foaf:Person>
        <foaf:nick> nombre de usuario </foaf:nick>
        <foaf:firstname> nombre </foaf:firstname>
        <foaf:surname> apellidos </foaf:surname>
        <foaf:mbox>correo electronico</foaf:mbox>
        <foaf:name> nombre completo </foaf:name>
        <foaf:homepage> pagina personal </foaf:homepage>
        <foaf:weblog> weblog </foaf:weblog>
        <foaf:depiction> url de la imagen </foaf:depiction>
        <foaf:jabberId> identificador de jabber </foaf:jabberId>
        <foaf:workplaceHomepage> pagina web del lugar de trabajo </foaf:workplaceHomepage>
        <foaf:schoolHomepage> pagina web del lugar de estudio </foaf:schoolHomepage>
    </foaf:Person>
</rdf:RDF>
```

Figura 5.7: Documento FOAF de un usuario

Modelo de Ideas y Proyectos en DOAP

El segundo caso, el de crear ideas y proyectos, no era tan sencillo como el anterior. No existía ninguna librería como RAP para DOAP y se tuvo que implementar un Servicio Web que generase el documento. Este Servicio Web, basado en tecnología REST (*Representational State Transfer*)[3] se encarga de, dados los datos de una idea o proyecto, generar el documento DOAP de los mismos.

La implementación de la función que genera el documento DOAP de una idea o un proyecto es la que podemos ver en las figuras 5.8 y 5.9.

```
<?php
/* rdfWS.php  -
 *  Servicio Web basado en tecnologia REST.
 *  Objetivo: Pasamos datos parametros y con eso nos debe devolver un xml-doap.
 *  http://xxxxx/xxxxx/rdfWS.php?isproject=''&isidea=''&data=''
 *  Parametos de entrada: isproject, isidea, data
 */
$isProject = $_GET["isproject"];
$isIdea = $_GET["isidea"];

if($isProject && data!=null){
    createADoap('project');
}else if($isIdea && data!=null){
    createADoap('idea');
}
/*************** Creacion del XML  de DOAP*****************/
function createADoap($type){
    if(strcmp($type,'project')==0)
    {
        $doap = generateRDFheader();
        $p .= doapHeader();
        $p .= addName($p,    $data[$i]['name']);
        $p .= addShortName($p, $data[$i]['shortname']);
        $p .= addHomepage($p, $data[$i]['homepage']);
        $p .= addWiki($p, $data[$i]['wiki']);
        $p .= addDownloadPage($p,    $data[$i]['downloadpage']);
        $p .= addDownloadMirror($p, $data[$i]['downloadmirror']);
        $p .= addShortdesc($p, $data[$i]['shortdesc']);
        $p .= addDesc($p, $data[$i]['description']);
        $p .= addMantainer($p,  $data[$i]['mantainer']);
        $p .= addCreator($p, $data[$i]['creator']);
        $p .= addHelper($p, $data[$i]['helper']);
        $p .= addDeveloper($p,$data[$i]['developer']);
        $p .= addRepository($p,$data[$i]['repbrowse'],$data[$i]['replocation']);
        [...]
```

Figura 5.8: Forma de generar un documento DOAP de una idea o proyecto (parte II)

[3]La *Transferencia de Estado Representacional* o REST es una técnica de arquitectura software que se utiliza para describir cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de otros protocolos basados en patrones de intercambio de mensajes.

```
    [...]
        $p .= addLicense($p,  $data[$i]['license'] );
        $p .= addProgrammingLanguage($p,  $data[$i]['progLangs']);
        $p .= addBugDB($p,  $data[$i]['bugdb']);
        $p .= addOS($p, $data[$i]['os']);
        $p .= doapFooter($p);
        $doap.=$p.generateRDFfooter();
        header('Content-type: text/xml');
        echo $doap;
    }
    else if(strcmp($type,'idea')==0)
    {
        $doap = generateRDFheader();
        $id.= doapHeader();
        $id.= addName($data[$i]['name']);
        $id.= addDesc($data[$i]['description']);
        $id.= addCreator($data[$i]['creator']);
        $id.= doapFooter();
        $doap.= $id;
        $doap.=generateRDFfooter();
        header('Content-type: text/xml');
        echo $doap;
    }
}
```

Figura 5.9: Forma de generar un documento DOAP de una idea o proyecto (parte I)

Tras llamar a este servicio habremos obtenido el modelo de la idea o proyecto que se haya requerido representado en un documento DOAP, de la forma que podemos ver en las figuras 5.10 y 5.11.

```
<Project
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://usefulinc.com/ns/doap#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    <name> nombre </name>
     <description> descripcion </description>
     <creator>
      <foaf:Person>
        <rdfs:seeAlso resource="http://uri_del_rdf_de_la_persona" />
      </foaf:Person>
     </creator>
</Project>
```

Figura 5.10: Documento DOAP de una idea

```
<Project
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://usefulinc.com/ns/doap#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
     <name> nombre </name>
     <shortname>nombre corto </shortname>
     <shortdesc> descripcion corta </shortdesc>
     <description> descripcion </description>
     <homepage rdf:resource="http://url_de_la_pagina_principal" />
     <wiki rdf:resource="http://url_del_wiki" />
     <download-page rdf:resource="http://url_de_la_pagina_de_descarga" />
     <download-mirror rdf:resource="http://url_del_mirror" />
     <bug-database rdf:resource="http://url_de_la_base_de_datos" />
     <os> plataforma </os>
     <programming-language> lenguaje de programacion</programming-language>
     <license rdf:resource="URI de la licencia" />
     <maintainer>
      <foaf:Person>
        <rdfs:seeAlso resource="http://uri_del_rdf_de_la_persona" />
      </foaf:Person>
     </maintainer>
     <developer>
      <foaf:Person>
        <rdfs:seeAlso resource="http://uri_del_rdf_de_la_persona" />
      </foaf:Person>
     </developer>
     <helper>
      <foaf:Person>
        <rdfs:seeAlso resource="http://uri_del_rdf_de_la_persona" />
      </foaf:Person>
     </helper>
     <repository>
       <SVNRepository>
         <browse rdf:resource='http://forja.rediris.com/svn/proyecto' />
         <location rdf:resource='http://forja.rediris.com/svn/proyecto' />
       </SVNRepository>
     </repository>
</Project>
```

Figura 5.11: Documento DOAP de un proyecto

5.5.2. Estructura e implementación de las búsquedas semánticas

Para realizar las búsquedas semánticas se ha hecho uso de la librería **Redland** (apartado 5.2.3). El funcionamiento de esta librería es bastante complejo y varía para cada caso concreto de búsqueda; por este motivo se analizará paso a paso la estructura de una de las consultas más sencilla de las que se realizan en el proyecto, con el fin de entender mejor cómo funciona.

El caso del ejemplo es una consulta en la cual se busca a un usuario según los datos que se hayan introducido como parámetros de búsqueda.

1. Generamos el Interfaz Redland:

```
$world=librdf_php_get_world();
```

2. Generamos el lugar de almacenaje temporal con el que vamos a trabajar durante la búsqueda:

```
$storage=librdf_new_storage($world,'hashes','dummy',"new=yes,hash-type='memory'");}
```

3. Inicializamos un modelo vacío del tipo *rdf-sparql* (Puesto q vamos a utilizar RDF como vocabulario y SPARQL como lenguaje de recuperación) dentro del $world.

```
$model = librdf_new_model($world,$storage,"http://www.w3.org/TR/2004/WD-rdf-sparql-"+
    "XMLres-20041221/");
```

4. Creamos un *parser* Raptor que sea capaz de interpretar los datos.

```
$parser=librdf_new_parser($world,'raptor','application/rdf+xml',NULL);
```

5. Inicializamos la URI donde encontramos el RDF concreto que tiene la información correspondiente a los usuarios en formato RDF-FOAF .

```
$uri=librdf_new_uri($world,'http://uri/del/documento/rdf/foaf/de/los/usuarios');
```

6. Generamos la URI necesaria del formato del modelo de rdf que quiero como resultado

```
$format_uri=librdf_new_uri($world, "http://www.w3.org/TR/2004/WD-rdf-sparql-XMLres-"+
            "20041221/");
```

7. Introducimos los datos debidamente interpretados por el *parser* en el modelo.

```
librdf_parser_parse_into_model($parser,$uri,$uri,$model);
```

8. Inicializamos una URI nula que es necesaria para la consulta que vamos a hacer. La inicializamos a nula porque queremos recorrer todos los enunciados del modelo.

```
$nulluri = librdf_new_uri ($world, '-');
```

9. Generamos la consulta concreta para este caso. Primero cogemos los datos que necesitamos del formulario de búsqueda para saber el tipo de búsqueda que hay, después llamamos a la función *getUsersQueries*, la cual nos da la consulta adecuada según si la búsqueda es por apellidos, nombre o por otro parámetro que se haya definido.

```
$type= $this->getRequestParameter("nameuworker");
$string= $this->getRequestParameter("inputuworker");
$q =  getUsersQueries($type, $string);}
$query = librdf_new_query($world, 'sparql', $uri, $q, $nulluri);
```

10. Una vez definidas las consulta, la ejecuto en el modelo.

```
$results=librdf_model_query_execute($model, $query);
```

11. Recorremos los resultados nodo a nodo, cogiendo la información necesaria y mostrándola por pantalla.

```
$this->arraySubs = array();
while (librdf_query_results_finished($results)1) {
    for ($i=0; $i < librdf_query_results_get_bindings_count($results); $i++)
    {
        $val=librdf_query_results_get_binding_value($results, $i);
        if ($val)
        {
            $nval=librdf_node_to_string($val);
            $this->subscribers = SubscribersPeer::retrieveByEMAIL($nval);
            $this->arraySubs[] = $this->subscribers;
        }
    }
    librdf_query_results_next($results);
}
```

12. Una vez realizadas todas las operaciones con Redland, liberamos los recursos utilizados.

```
librdf_free_uri($format_uri);
librdf_free_uri($uri);
librdf_free_uri($nulluri);
librdf_free_model($model);
librdf_free_storage($storage);
```

5.5.3. Mapa Web y capturas de pantalla de la aplicación.

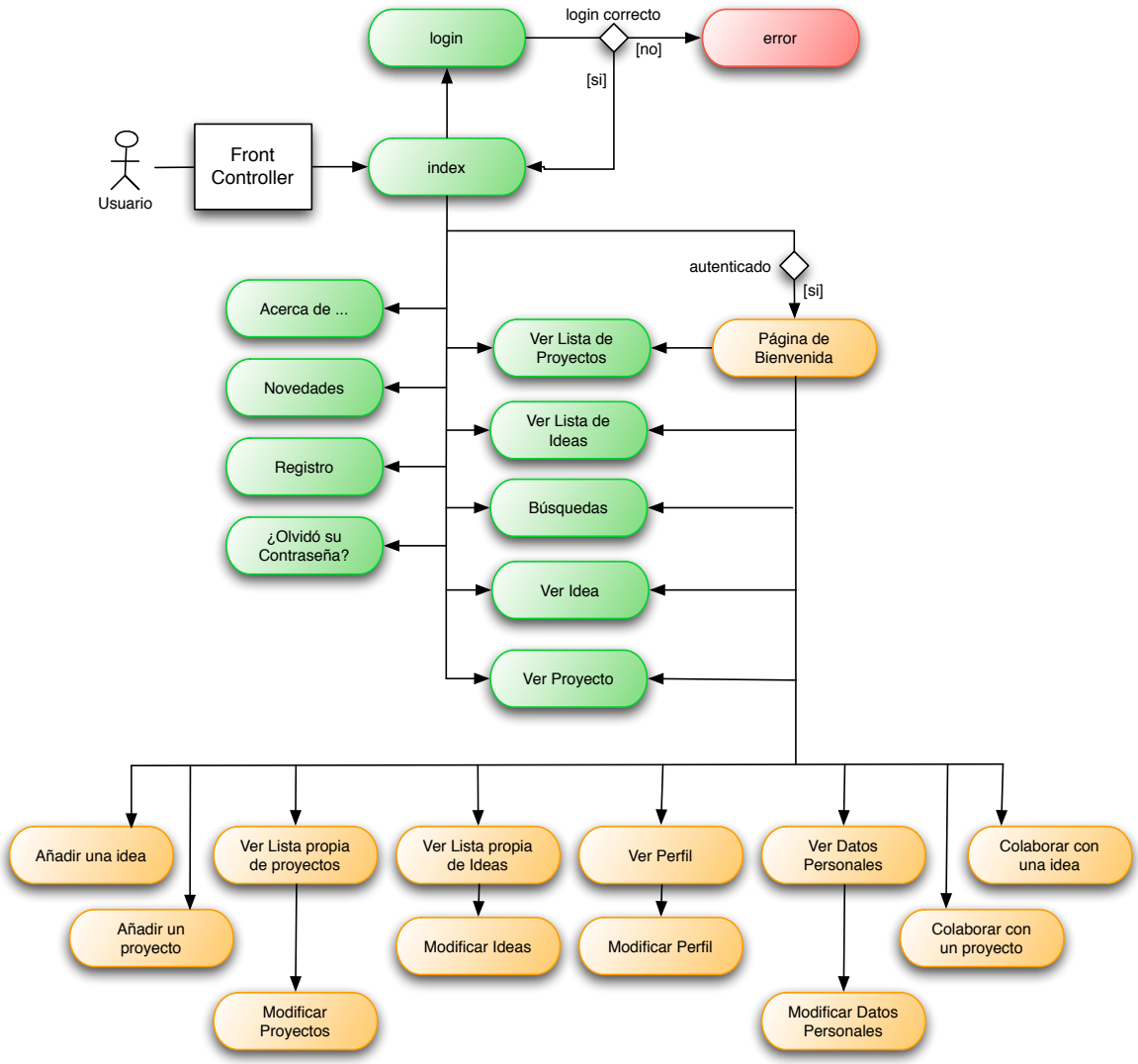

Figura 5.12: Mapa Web de la aplicación con respecto a un usuario de la misma.

Página de inicio

Figura 5.13: Página de inicio del portal.

Página de búsquedas

Figura 5.14: Página de búsquedas de la aplicación.

Página de bienvenida

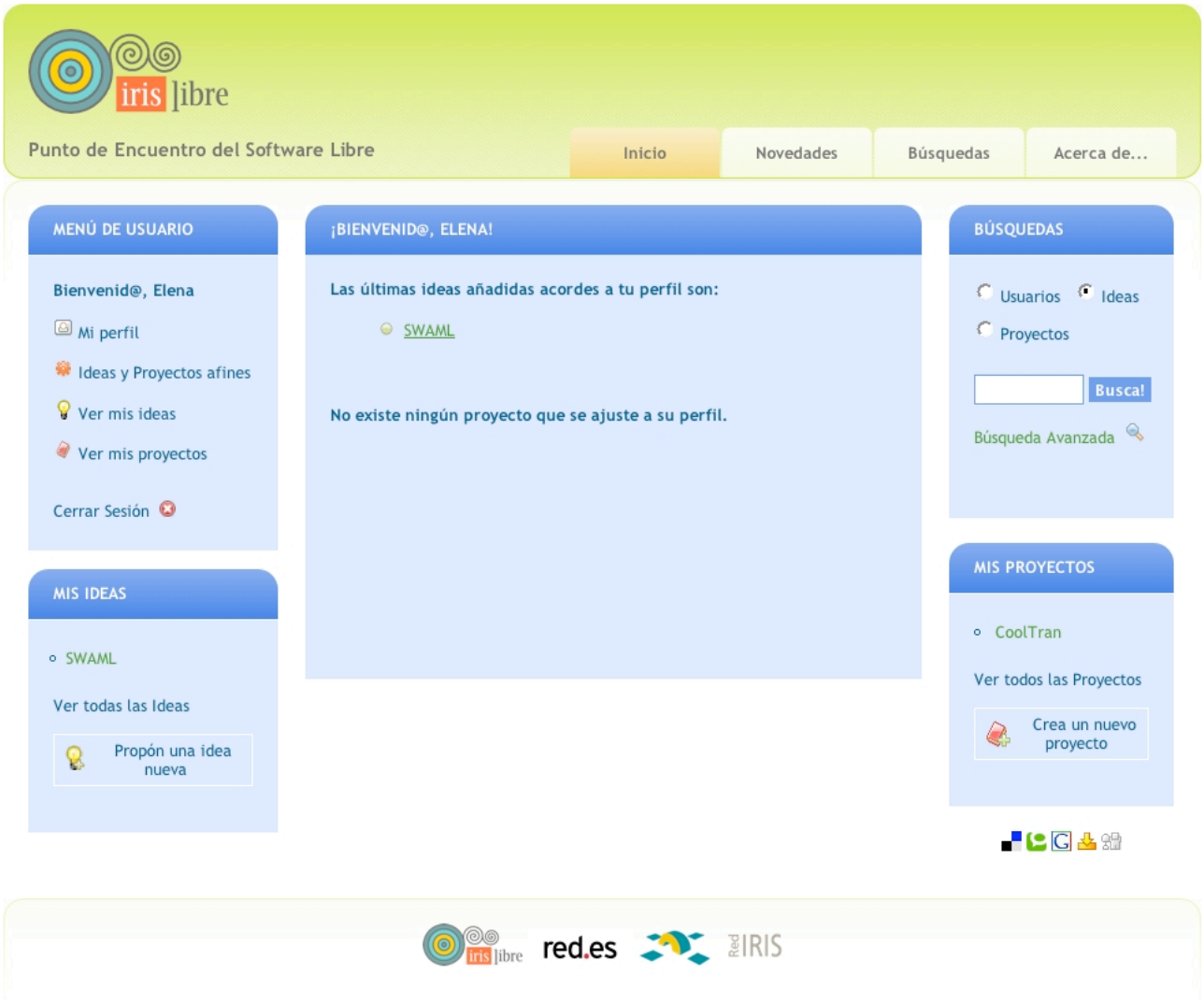

Figura 5.15: Página de bienvenida tras la autenticación de los usuarios.

Página del Perfil de un Usuario

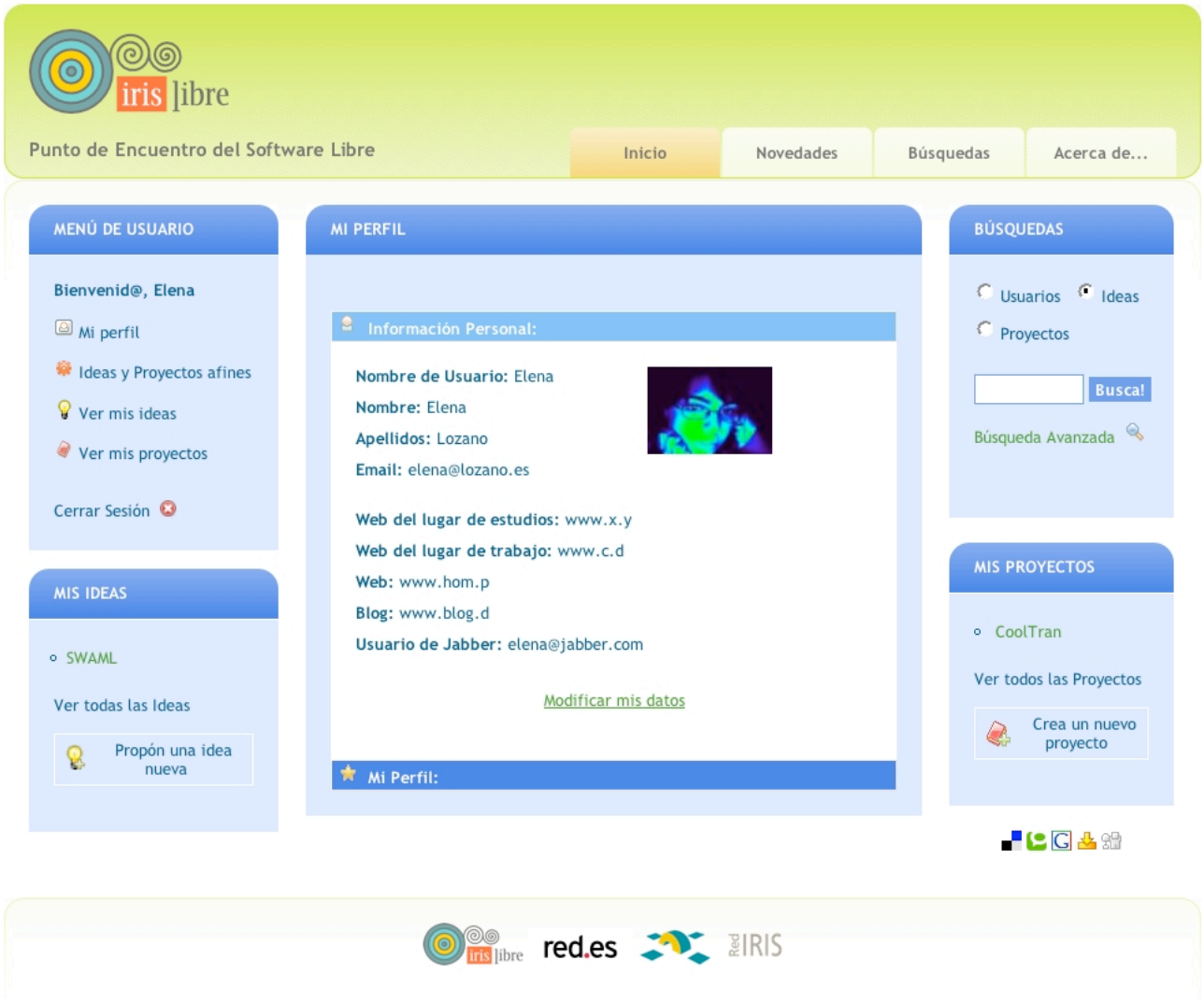

Figura 5.16: Página del Perfil de un Usuario.

Página descriptiva de una idea

Figura 5.17: Mapa Web de la aplicación con respecto a un usuario de la misma.

Página descriptiva de un proyecto

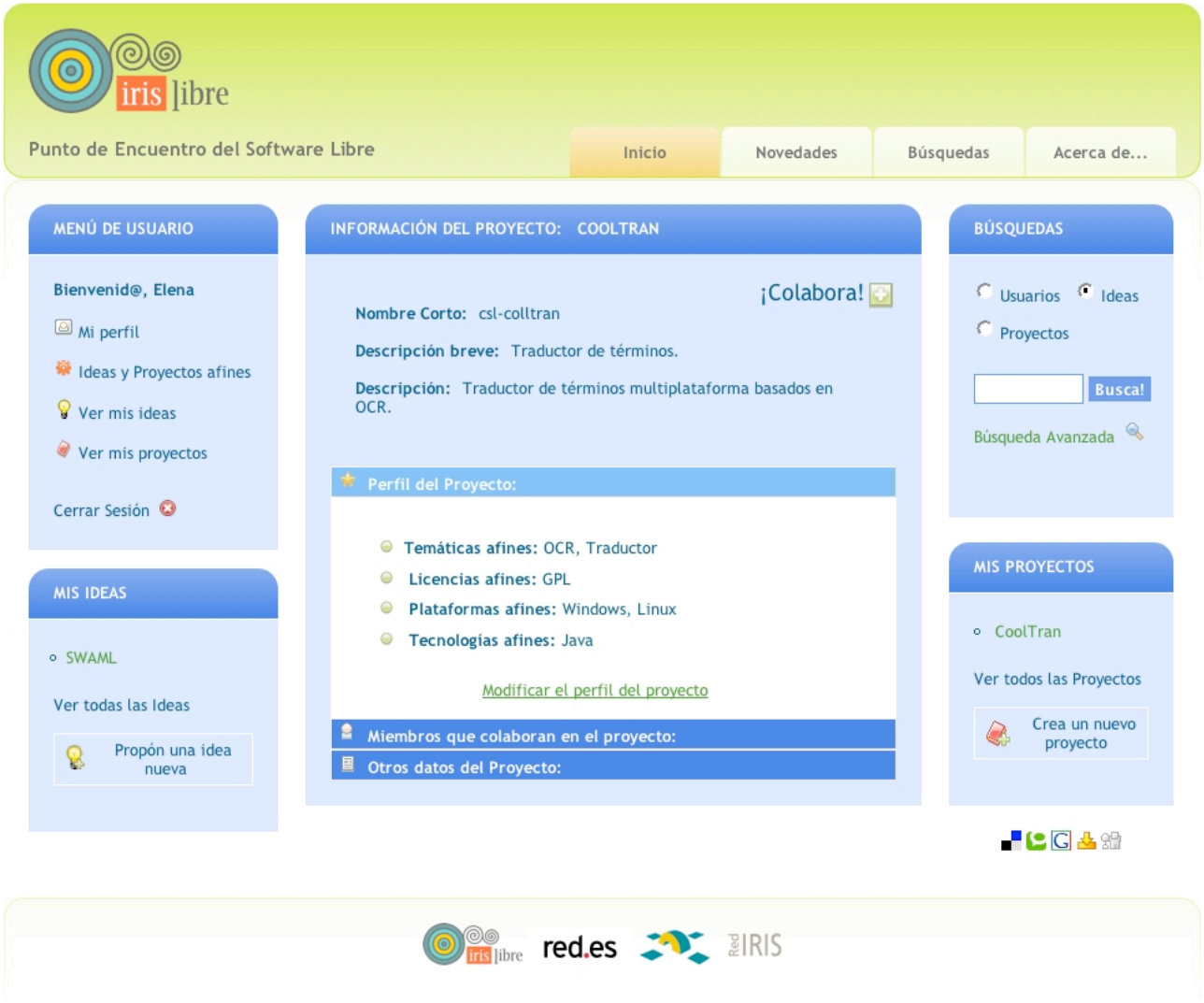

Figura 5.18: Mapa Web de la aplicación con respecto a un usuario de la misma.

Capítulo 6

Conclusiones

Este proyecto, cuyo ambicioso objetivo pretende establecer un punto de convergencia entre investigadores, docentes y desarrolladores, ha sido un elemento decisivo a la hora de hacerme ver el esfuerzo y la madurez necesaria para realizar una tarea de esta envergadura.

Un primer aspecto observado al realizar este proyecto, ha sido la necesidad de documentarse e investigar antes de realizar ninguna implementación. Si bien es importante llevar a la práctica los conceptos asimilados, más importante es realizar un estudio exhaustivo de las tecnologías que se van a utilizar, para así poder tomar las decisiones correctas cuando sea necesario.

Utilizar un *framework* para desarrollar los aspectos básicos de la aplicación web me ha permitido, además de conocer una nueva tecnología, ver las ventajas de utilizar una herramienta que ya ha sido testada por otros desarrolladores o usuarios, y lo fundamental que es poder aprovechar la base de conocimiento adquirida por otras personas por medio de herramientas y aplicaciones libres.
Todo ello me ha llevado a estar más motivada aún para cumplir los objetivos de esta aplicación, la cual favorecerá en un futuro a que existan más herramientas como la que he podido emplear.

Haber aplicado en el proyecto ontologías y vocabularios como RDF o FOAF me ha servido para entender la situación actual de la Web Semántica.

Tras profundizar en su estudio y en las distintas alternativas que se ofrecen, se me hace evidente que, aunque es una tecnología que puede optimizar enormemente el rendimiento y la eficacia de la recuperación y representación de los recursos, está infravalorada y muy poco desarrollada.

Las librerías que existen y el material de apoyo es muy reducido, y además, de las pocas herramientas que nos encontramos, casi todas están desarrolladas para solucionar el caso particular que su creador necesitaba resolver, por lo que muchas veces hay que recurrir a varias herramientas e incluso crearse unas propias. Un ejemplo de ésto lo podemos ver en el proyecto: *RAP* no implementaba todas las funcionalidades que se requerían, por lo que tuve que utilizar *Redland* y posteriormente imple-

mentar un Servicio Web propio que completara los requisitos de la aplicación.
Ahondando más en este tema, cabe destacar la problemática de las librerías utilizadas; muchas de ellas están en deshuso, sin desarrolladores que las mantengan y con un grado mínimo de abstracción, siendo muy poco funcionales.

Estos aspectos son escollos a superar por la Web Semántica, ya que al no haber herramientas que faciliten la tarea de desarrollar, muy pocos se atreven a utilizarla.

Todos estos conflictos son de difícil solución a corto plazo, puesto que es prácticamente imposible añadir motores semánticos a todas las aplicaciones existentes en la Web y menos aún añadir metadatos a toda la información que podemos encontrar en ella. Creo que la principal meta a alcanzar sería crear una herramienta eficaz, que fuera práctica y que no cayera en los errores que tienen las librerías y herramientas que existen actualmente.

Por lo que he podido investigar, muchas empresas están adecuando su software a tecnologías semánticas, pero en el ámbito de lo privado, por lo que no hay muchos desarrollos libres que sean eficaces a los cuales podamos acceder.
Es por esto, que el futuro de la Web Semántica dependerá en gran parte del éxito de las implementaciones que se hagan y de la difusión que se les de a las herramientas que se utilicen.

Es en consecuencia, de gran importancia, seguir avanzando en los procesos de estandarización de vocabularios y esquemas específicos para distintos ámbitos.

Debido a las carencias existentes en la Web actual, entre otras, la baja precisión de los motores de búsqueda; podemos ver que ésta necesita una reforma para hacer posible que la información sea representada e indexada de manera lógica y coherente. Por ésto pienso que la tecnología semántica es la que, en un futuro, va a llevar a cabo esta reforma.

Otro aspecto que me gustaría destacar es el desarrollo profesional que he obtenido realizando este proyecto, gracias al aprendizaje de multitud de aspectos que antes no imaginaba que fueran a ser tan importantes. El más obvio de ellos es el hecho de aprender nuevas tecnologías.

Aun así, lo que quizás vea más importante es la capacidad de **abstracción** y la **seguridad** que se adquiere tras enfrentarse a problemas que no tienen una solución aparente y que al final, tras muchos intentos y esfuerzo se es capaz de superar. Sinceramente, pienso que este aspecto ha sido el que más me ha hecho madurar intelectual y personalmente, sobre todo porque ahora se que puedo ser capaz de enfrentarme a cualquier tecnología desconocida y conseguir entenderla y utilizarla.

Como ya comenté en la introducción, esta aplicación aspira a fomentar los desarrollos libres dentro de la comunidad de *RedIRIS*, por lo que se quiere lanzar bajo la iniciativa de *IrisLibre* para el próximo curso académico. Por este motivo, en cuanto al futuro del proyecto, se pretende seguir im-

plementando y mejorando la aplicación, extendiéndola en tres ámbitos básicos:

- Modificar la forma de autenticación utilizando PAPI (http://papi.rediris.es/). [1]
- Conseguir que el portal interactúe con la Forja de Rediris, generando de forma automática los proyectos y todos los elementos necesarios que éstos requieran.
- Optimizar la eficiencia de las búsquedas mejorando los algoritmos de las mismas.

Espero que este proyecto consiga además de potenciar el papel que la Web Semántica tiene en la actualidad, el ser útil para el colectivo al que va dirigido y colabore en la difusión del desarrollo del Software Libre y de lo valores que éste implica. Todo ello conlleva un mayor esfuerzo por parte de los desarrolladores e investigadores, ya que aún queda mucho camino por recorrer.

[1] PAPI es un sistema desarrollado por RedIRIS que provee un control de acceso a recursos de información electrónica restringidos. Intenta mantener la autenticación como un aspecto local de la organización a la que pertenezca el usuario, mientras que da a los proveedores de información total control sobre los recursos que ofertan.

Bibliografía

[The Definitive Guide to symfony] Fabien Potencier, Fransois Zaninotto

The Definitive Guide to symfony, Apress.

[Practical RDF] Shelley Powers.

Practical RDF, Oreilly & Associates, July 2003

[Apache] THE APACHE SOFTWARE FOUNDATION

www.apache.org

[HTML] HTML 4.0 Specification, Raggett, Le Hors, Jacobs eds, World Wide Web Consortium Recommendation;

http://www.w3.org/TR/REC-html40/

[DTD] Extensible Markup Language (XML) 1.0, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Fransois YergeauRaggett, World Wide Web Consortium Recommendation;

http://www.w3.org/TR/REC-xml/dt-doctype

[ISO10646] ISO/IEC 10646. The applicable version of this standard is defined in the XML specification [XML].

http://www.w3.org/XML/

[Namespaces] Namespaces] Namespaces in XML; Bray, Hollander, Layman eds, World Wide Web Consortium Recommendation;

http://www.w3.org/TR/1999/REC-xml-names-19990114.

[PAPI] PAPI; http://papi.rediris.es/

[PEAR] PHP Extension and Application Resource; http://pear.php.net/

[PHP] PHP Hypertext Preprocesor; http://www.php.net/

[RDFSchema] Resource Description Framework (RDF) Schemas; Brickley, Guha, Layman eds., World Wide Web Consortium Working Draft; http://www.w3.org/TR/rdf-schema/

[RedIris] RED ESPAÑOLA DE I+D

www.rediris.es

[REST] SOAP VERSION 1.2 PART1: MESSAGING FRAMEWORK

http://www.w3.org/TR/soap12-part1/

[RFC2396] Uniform Resource Identifiers (URI): Generic Syntax

http://rfc.net/rfc2396.html

[SPARQL] SPARQL Query Language for RDF; Eric Prud'hommeaux, Andy Seabornet; http://www.w3.org/TR/rdf-sparql-query/

[Symfony] Symfony] Symfony, an open-source PHP web framework

http://www.symfony-project.com

[URI] Uniform Resource Identifiers (URI): Generic Syntax; Berners-Lee, Fielding, Masinter, Internet Draft Standard August, 1998; RFC2396.

http://www.w3.org/TR/uri-clarification/

[XML] Extensible Markup Language (XML) 1.0; World Wide Web Consortium Recommendation;

http://www.w3.org/TR/REC-xml.

[XMLinHTML] XML in HTML Meeting Report; Connolly, Wood eds.; World Wide Web Consortium Note;

http://www.w3.org/TR/NOTE-xh.