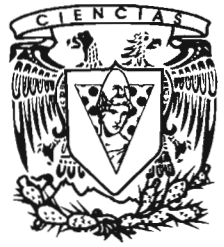
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

"AÑADIENDO INFORMACIÓN SEMÁNTICA A PAQUETES DE SOFTWARE."

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T A :
JORGE JESÚS SANTOS FIERRO



FACULTAD DECIENCIAS
UNAM

DIRECTOR DE TESIS:
DR. SERGIO RAJSBAUM GORODEZKY

2005

VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Añadiendo Información Semántica a Paquetes de Software."

realizado por Jorge Jesús Santos Fierro

con número de cuenta 09850551-9 , quien cubrió los créditos de la carrera de:

Licenciatura en Ciencias de la Computación.

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director Propietario	Dr. Sergio Rajsbaum Gorodezky
Propietario	M. en C. José de Jesús Galaviz Casas
Propietario	Lic. en C. C. Francisco Lorenzo Solsona Cruz
Suplente	Lic. en C. C. Dario Bahena Tapia
Suplente	Lic. en C. C. Karla Ramírez Pulido

Consejo Departamental de Matemáticas

Dr. Francisco Hernández Quiroz

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE

Agradecimientos

La conclusión de una carrera involucra la acción de tantas personas, que es imposible mencionarlas a todas.

Sin embargo, hay personas que tuvieron un papel sobresaliente en mi vida y haré un esfuerzo por olvidar a las menos posibles.

Quiero agradecer a mis padres, cuyo apoyo y amor constantes forman los cimientos de mi ser.

Quiero agradecer a Paola, mi amor, que siempre me da ánimos, me inspira y da sentido a mi vida.

Quiero agradecer a mis amigos, que lejos o cerca, están siempre conmigo, y yo con ellos.

Quiero agradecer a mi director de tesis, por todo lo que me ayudó y por el mucho tiempo que me dedicó del poco del que dispone.

Quiero agradecer a mis sinodales, los cuales hicieron un trabajo verdaderamente extraordinario.

Quiero agradecer a mis maestros, los cuales formaron mi mente y mi espíritu.

Quiero agradecer a los que fueron mis jefes y compañeros de trabajo, por todo lo que aprendí y por lo que crecimos juntos.

Quiero agradecer a la Universidad, esfuerzo y amor de generaciones, pilar México.

Quiero agradecer a México, por hacer todo ésto posible.

Finalmente, quiero agradecer a la vida, misterio que vale la pena vivir.

Añadiendo Información Semántica a Paquetes de Software

Jorge Jesús Santos Fierro

30 de agosto de 2005

Índice general

Índice de figuras

Capítulo 1

Resumen

El mundo del software libre ha producido una gran cantidad de aplicaciones útiles y de gran calidad. Existen tantas de ellas que el problema muchas veces recide en encontrar lo que necesitamos. Las distribuciones de GNU/Linux, en particular, frecuentemente vienen acompañadas de miles de paquetes de software. En el caso específico de Debian GNU/Linux, este tiene más de 10,000 paquetes de software y estos están clasificados por secciones. Esto hace difícil encontrar el paquete de software que requerimos. Por ejemplo, si queremos un lector de correo para el ambiente de escritorio GNOME, ¿buscamos en la sección de Internet?, ¿o en la de GNOME?, y una vez que seleccionamos la sección lo mejor que podemos hacer es realizar búsquedas de texto sobre los nombres y las descripciones de los paquetes.

Evidentemente es necesario buscar una mejor aproximación al problema.

Por otro lado, la Web Semántica fue diseñada específicamente para describir recursos, y hacer más fácil el intercambio de información de los mismos.

En este trabajo, haciendo uso de recursos ya existentes en lo posible, se elaboró un sistema basado en RDF que nos permitiera tener un directorio de software, especialmente enfocado a las distribuciones de GNU/Linux.

Al basar el sistema en RDF tenemos ventajas en cuanto al intercambio de información, y la extensibilidad de la misma. En cuanto al primer aspecto, ya existen directorios especializados, como el Code Zoo de O'Reilly[1]. Al usar RDF, se puede hacer uso de esta información y complementarla con nuestros propios datos, esto es posible debido a la extensibilidad que nos proporciona RDF, gracias a ésta podemos hacer nuevas afirmaciones acerca de objetos ya existentes, lo que nos permite, por ejemplo, dar información sobre la

[1] Para más información, consultar: `http://usefulinc.com/doap/news/contents/2005/08-03-codezoo/read`

categoría a la que pertenecen los proyectos mencionados en Code Zoo. Otra ventaja es que cualquier otro proyecto puede usar nuestra información y tomar o agregar lo necesario para sus fines.

Capítulo 2

Introducción

2.1. El problema

El mundo del software libre ha conseguido, a través de Internet, elaborar una gran cantidad de software, que en muchos casos resulta extraordinario. Se ha producido tanto, que muchas veces es difícil encontrar lo que se necesita y es una tarea titánica tratar de clasificarlo.

GNU/Linux es el sistema operativo libre más usado en el mundo. Normalmente, GNU/Linux es usado a través de una distribución, un conjunto de programas integrados y probados para facilitar la instalación y uso de los mismos. Las distribuciones de GNU/Linux heredan el problema del software libre: es difícil organizar y por lo tanto encontrar el software que necesitamos. Algunas distribuciones han tratado de solucionar (al menos parcialmente) este problema clasificando los paquetes (que contienen a los diversos programas) en secciones, pero esto es una pobre solución que pierde gran parte de su utilidad cuando nos encontramos un programa que puede pertenecer a dos o más secciones.

En este trabajo, usaremos la de distribución Debian GNU/Linux como referencia y plataforma de desarrollo.

2.1.1. Debian GNU/Linux

Debian GNU/Linux (de aquí en adelante lo llamaremos Debian) es un sistema operativo de computadora libre que usa Linux como su kernel.

Debian es elaborado por voluntarios de todo el mundo (el proyecto Debian), los cuales, entre otras labores, toman algún programa libre desarrollado por otros o por ellos mismos y lo procesan de tal manera que el resultado es un "paquete" que puede ser instalado fácilmente en el sistema operativo.

Al ser un proyecto de voluntarios, son éstos mismos los que deciden qué paquetes estarán disponibles en Debian: la primera versión de Debian, liberada en 1999, tenía 474 paquetes [19], actualmente cuenta con más de 10,000 paquetes, obviamente, encontrar un programa para alguna actividad específica se convierte en todo un reto.

Los paquetes de Debian están organizados en secciones, cada paquete indica en qué sección debe ubicarse. Por ejemplo, *Evolution*, un programa para leer correo electrónico, se encuentra en la categoría "Gnome", puesto que es un programa que pertenece a este ambiente de escritorio; pero si estoy buscando las diferentes opciones que ofrece Debian en cuanto a lectores de correo electrónico se refiere, tengo que hacerlo con alguna herramienta de búsqueda tratando de cazar alguna cadena relacionada con el tema (e.g. "mail client"), con alguna de las herramientas que ofrece Debian para buscar paquetes basándonos en sus descripciones. Desafortunadamente este método produce resultados poco satisfactorios; sin ir más lejos, la cadena "mail client" no aparece en la descripción de Evolution, por lo que este método no funciona para encontrarlo.

2.2. Algunas aproximaciones al problema

El problema descrito en la sección 2.1 ha sido atacado de diferentes maneras. La más sencilla es añadir descripciones textuales a los paquetes de software y depender de herramientas de búsqueda de texto para encontrar lo que necesitamos, como se ha mencionado anteriormente. Otra aproximación, la más flexible y prometedora de la que se tiene noticia hasta el momento, además de la propuesta en este trabajo, consiste en añadir etiquetas a los paquetes para asignarles ciertas propiedades, esta técnica es usada por el proyecto "Debian Package Tags".

2.2.1. Debian Package Tags

El sistema "Debian Package Tags"[1] (también conocido como debtags) permite asociar etiquetas (tags) o propiedades a los paquetes de Debian. Así, en la base de "tags" de este sistema tenemos que *Evolution* trabaja con correo electrónico (media::mail), es un cliente de red (role::client) y es parte de GNOME (suite::gnome), entre otros atributos, con lo que es posible buscar los clientes de correo y obtener *Evolution* entre los resultados. Este

[1] Se puede encontrar más información sobre este sistema en `http://debtags.alioth.debian.org/`

sistema permite además registrar *implicaciones*[2], así, sabemos que el tag "suite::gnome" implica el tag "suite", por lo que sabemos que *Evolution* es parte de un conjunto de paquetes integrados entre sí, aunque no se menciona específicamente en sus etiquetas.

El vocabulario de etiquetas, es decir, las etiquetas que se pueden aplicar a los paquetes, están definidas por un grupo de voluntarios y su correspondencia con cada paquete se define colaborativamente a través de una interfaz de web.

Las características de este sistema sugieren que el problema que está tratando de resolver es ideal para ser resuelto por las herramientas que proporciona la Web Semántica [28], ya que se trata de añadirle meta-información a los paquetes de Debian, de una manera distribuida.

2.3. Objetivo

El objetivo de esta tesis es proveer una infraestructura moderna, usando herraminetas estándar para añadir información semántica a paquetes de software. Como se mencionó en la sección 2.1, se trabajará con la distribución Debian GNU/Linux para la implementación del proyecto.

Para desarrollar la solución al problema se hará uso del lenguaje *Resource Description Language* [25] (Lenguaje de Descripción de Recursos, o RDF, por sus siglas en inglés), dado que éste es un lenguaje creado explícitamente para añadir información semántica a recursos de Internet (y los paquetes de software los podemos ver fácilmente como recursos de Internet) y ha sido utilizado con éxito en diversos proyectos.

Para desarrollar la infraestructura, nos basaremos en el proyecto DOAP[3] (Description of a Project), el cual proporciona infraestructura para añadir información semántica a proyectos de software libre haciendo uso de RDF, un modelo para representar afirmaciones pensado especialmente para el Web.

Partiendo del proyecto DOAP y haciendo uso de la información ya generada por diversos proyectos que proporcionan información sobre software libre, en este trabajo se elaboró un esquema de RDF para añadirle información semántica a los paquetes de Debian así como algunas herramientas para hacer uso de esta información.

[2] En otras palabras, que una categoría es parte de otra que la contiene propiamiente

[3] Se puede encontrar más información en: `http://usefulinc.com/doap`

2.4. Desarrollo

El trabajo desarrollado en esta tesis consistió en hacer uso de tecnologías existentes, como DOAP (basado en RDF) e información previamente disponible, como la de `freshmeat.net` para elaborar un prototipo de sistema de información para el Web que ayuda a encontrar los paquetes de software que necesitamos. Asimismo, este sistema nos permite modificar esta información, importarla y exportarla en RDF, lo cual nos permite utilizar los datos de diversas fuentes, combinarlos y proporcionarlos a otros sistemas a los cuales les sean de utilidad.

Adicionalmente, podemos mencionar que el sistema se desarrolló haciendo uso de la plataforma para aplicaciones web conocido como *Ruby on Rails*[4], el cual es un sistema basado en el lenguaje de programación Ruby, mismo que por sus características permitió desarrollar el sistema con gran flexibilidad y rapidez. Así mismo, se utilizó un conjuto de tecnologías que recientemente ha recibido el nombre de "Ajax"[5], las cuales ayudan en el desarrollo de aplicaciones web más interactivas, esto fue facilitado enormemente mediante el uso de Ruby on Rails y algunas de sus bibliotecas.

[4]Para más información consultar el sitio web: `http://www.rubyonrails.com`
[5]Para más información consultar la dirección: `http://en.wikipedia.org/wiki/AJAX`

Capítulo 3

RDF

3.1. Introducción

> It takes three legs to make a tri-pod or to make a table stand. It takes three wheels to make a ve-hicle called a tricycle. Every triangle has three corners, Every triangle has three sides, No more, no less. You don't have to guess. When it's three you can see it's a magic number.
>
> Three is a Magic Number, Bob Dorough

Día a día las computadoras se empeñan en hacernos difícil la vida, guardamos un archivo hace una semana y ahora no hay manera de encontrarlo, buscamos información sobre una persona en Google[1] y obtenemos la de todos sus pseudónimos y no la de la persona que estamos buscando.

¿Y las computadoras? ¿No se supone que deben hacernos la vida más fácil? Sí y, sin embargo, en casos como este, si no les ayudamos nosotros, son incapazes de ayudarnos a su vez. Desgraciadamente, no son lo suficientemente inteligentes para encontrar lo que realmente queremos cuando buscamos información sobre el correo de una persona específica, digamos, el de Juan Pérez, si buscamos su correo vamos a encontrar a ocenas de Juan Pérez pero si además pudiésemos especificar que nos estamos refiriendo al estudiante de Ciencias de la Computación en la Facultad de Ciencias de la UNAM, generación 1998, deberíamos acercarnos bastante al resultado deseado, el problema es que, para una computadora, todos estos datos no son mas que cadenas de caracteres sin ningún significado en particular. Es por

[1] Uno de los buscadores de internet más usados, se puede consultar en: `http://www.google.com/`

esto que se ha desarrollado el *Marco de Descripción de Recursos* (RDF por sus siglas en inglés: *Resource Description Framework*). Este es un lenguaje para representar información acerca de recursos (*resources*) en el Web. Así, si tenemos un recurso en el Web que identifica a Juan Pérez por medio de RDF podemos especificar que es (o fue) estudiante de la Facultad de Ciencias (otro recurso representado en el Web), en la carrera de Ciencias de la Computación (otro recurso), de la generación 1998 y, además, su dirección de correo es `juan@ciencias.unam.mx`. De esta manera, hasta una computadora sería capaz, de conocer el lugar donde puede encontrar la información y de comunicarle a quien lo necesite la dirección de correo electrónico de Juan Pérez, imagínense los beneficios, probablemente en el futuro me dejen de llegar mensajes de SPAM sobre medicamentos canadienses y me llegarán únicamente los que realmente me interesan, como los de los Rolex de 15 dólares.

Pero, ¿por qué usar RDF?, ¿porqué no nada más usamos alguna convención ad-hoc para cada problema?, como por ejemplo, un directorio de direcciones de correo electrónico basado en un formato arbitrario. Usar RDF nos da ventajas al haber herramientas ya creadas para su procesamiento, además, al ser un formato estándar, otras personas pueden usar estos datos en nuevas aplicaciones y "agregarlos" con otra información. De hecho, una de las principales ventajas de RDF es que permite el intercambio de información por medio de un formato estándar.

RDF define un modelo para expresar información acerca de recursos. Para poder identificar unívocamente de qué estamos hablando, RDF hace uso de Identificadores Uniformes de Recursos, URIs por sus siglas en inglés (*Uniform Resource Identifiers*), así podemos hacer afirmaciones como las de la Figura 3.1.

> La Persona identificada por http://example.org/people/jsf tiene como nombre Jorge Santos.
>
> Dicha Persona tiene como dirección de correo electrónico jsf@ciencias.unam.mx

Figura 3.1: Una afirmación

Esto se puede representar por medio de una gráfica dirigida, de hecho, un conjunto de afirmaciones (*statements*) de RDF *es* una gráfica dirigida. La Figura 3.2 ilustra la gráfica que la Figura 3.1 define.

Como se puede ver en la Figura 3.2, RDF hace uso de URIs para iden-

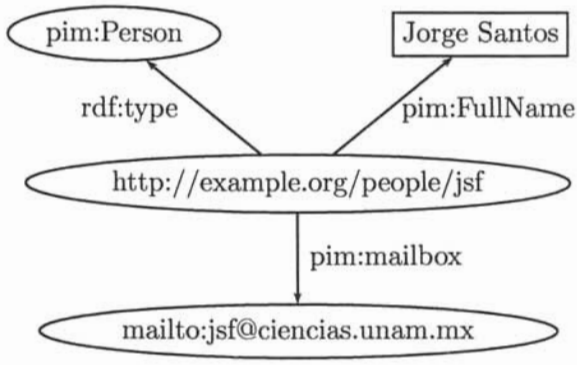


Figura 3.2: Una gráfica

tificar:

Recursos: En este caso personas, más específicamente, a mí, con el identificador `http://example.org/people/jsf`

Tipos de cosas: Por ejemplo, Personas, con `http://www.w3.org/2000/10/swap/pim/contact#Person`

Propiedades de dichas cosas: En este ejemplo, una dirección de correo: `http://www.w3.org/2000/10/swap/pim/contact#mailbox`

Valores de las propiedades: Como puede ser `mailto:jsf@ciencias.unam.mx`

Por lo regular los recursos se dibujan como un óvalo mientras que las literales se dibujan dentro de un rectángulo.

La razón por la que se utilizan URIs es porque éstas nos permiten identificar unívocamente un recurso, de este modo, si estamos hablando de `http://example.org/people/jsf` sabremos siempre que nos estamos refieriendo al mismo recurso (en este caso una persona). De manera similar, si siempre usamos el URI `http://www.w3.org/2000/10/swap/pim/contact#mailbox` para referirnos a direcciones de correo electrónico, sabremos siempre a qué nos referimos, si, por el contrario, a veces usáramos como identificador de la propiedad la palabra buzón y otras veces usáramos la palabra mailbox una computadora no tendría manera de darse cuenta de que en realidad nos estamos refiriendo a la misma propiedad.

Las especificaciones de RDF también describen una manera estándar de "serializar" gráficas RDF, es decir, de ponerlas por escrito, este formato es

RDF/XML [12]. La serialización correspondiete a la gráfica de la Figura 3.2 se ve como en la Figura 3.3.

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

  <contact:Person rdf:about="http://example.org/people/jsf">
    <contact:fullName>Jorge Santos</contact:fullName>
    <contact:mailbox rdf:resource="mailto:jsf@ciencias.unam.mx"/>
  </contact:Person>

</rdf:RDF>
```

Figura 3.3: Ejemplo de RDF/XML

Podemos ver que este documento de XML contiene los URIs de la gráficas, así como propiedades (fullName, mailbox) y sus respectivos valores (Jorge Santos, jsf@ciencias.unam.mx).

Antes de proseguir, es necesario explicar un poco el origen y fin de RDF. RDF es parte del resultado de una iniciativa del W3C[2], dicha iniciativa es el "Web Semántico" (Semantic Web), que tiene como objetivo extender el Web para darle a la información un significado bien definido, de tal manera que las computadoras puedan interoperar mejor entre sí y con las personas, para más información al respecto consultar [1] y [28].

3.2. Haciendo afirmaciones sobre recursos

RDF fue ideado para hacer afirmaciones sobre recursos Web, como pueden ser páginas Web. En esta sección veremos las ideas básicas por medio de las cuales RDF nos permite hacer esto.

3.2.1. Conceptos Básicos

Supongamos que queremos hacer una afirmación sencilla, como en el ejemplo de la Figura 3.1

En esta afirmación se han enfatizado ciertas partes:

[2]Para más información sobre el w3c consultar: `http://www.w3c.org`

http://example.org/index.html tiene un *creador* cuyo valor es *Jorge Santos*

Figura 3.4: Afirmación

1. *http://example.org/index.html* es el sujeto del que estamos hablando.
2. *creador* es la propiedad que estamos definiendo.
3. *Jorge Santos* es el valor de la propiedad definida.

RDF se basa en la idea de que las cosas descritas tienen propiedades, las cuales, a su vez, tienen valores. Así cualquier cosa que se exprese en RDF tendrá esta forma. La cosa que estamos describiendo es llamada el sujeto (*subject*), la propiedad que estamos especificando de este sujeto es el predicado (*predicate*) y el valor de dicho predicado es el objeto (*object*).

En la Figura 3.4 estas partes son:

- El *sujeto* es el URL http://example.org/index.html.
- El *predicado* es la palabra "creador".
- El *objeto* es la frase "Jorge Santos".

Para que una máquina pueda entender este tipo de enunciados necesitamos dos cosas:

- Un lenguaje para expresar las afirmaciones que sea procesable por computadoras.
- Identificadores para el sujeto, el predicado y el objeto, de tal manera que estos no se confundan con otros usados en el Web.

La segunda de estas necesidades, como ya se había comentado, queda cubierta por los URIs. Los URIs son una generalización del los URLs, los nombres que sirven para identificar recursos en el Web, los URIs, adicionalmente, pueden identificar cosas no accesibles en el Web, como pueden ser personas, cosas, conceptos, etc. Más especificamente, RDF usa algo conocido como una referencia URI (*URI reference*) o URIref, un URIref es un URI con un *identificador de fragmento* anexado al final, un ejemplo de URIref es `http://example.org/index.html#parte1`, el URI está conformado

por la cadena de caracteres `http://example.org/index.html` y el identificador de fragmento `parte1`, el carácter # se usa para separar el URI del identificador de fragmento.

Para representar las afirmaciones en un lenguaje que la computadora pueda entender, RDF hace uso de XML, XML fue diseñado para que cualquiera pudiese diseñar su propio formato y usar dicho formato para representar información. RDF define un formato particular de XML, llamado RDF/XML, para la representación de afirmaciones y el intercambio de las mismas. En la Figura 3.3 se puede ver un ejemplo de RDF/XML, se usaron etiquetas como `<contact:fullName>` y `<contact:mailbox>` para indicar propiedades del recurso que se estaba definiendo (`http://examle.org/people/jsf`), tales etiquetas permiten a programas con un conocimiento de las mismas interpretar el contenido del archivo RDF/XML. Tanto el contenido XML como las etiquetas (con algunas excpeciones) pueden contener caracteres UNICODE [6], por lo que contenido en muchos idiomas puede ser directamente representado. La definición de RDF/XML se da en [12].

3.2.2. El modelo RDF

Retomemos el concepto básico en el que se basa RDF. En la introducción mencionamos que RDF se basa en la idea de expresar conceptos simples acerca de recursos, esto se logra por medio de afirmaciones (o enunciados, en el presente trabajo se usarán estos dos términos indistintamente), las cuales consisten de un sujeto, un predicado y un objeto. En RDF, la afirmación en español:

> *http://example.org/index.html* tiene un *creador* cuyo valor es *Jorge Santos*

puede ser representada por un enunciado de RDF cuyas partes sean:

1. *http://example.org/index.html* es el sujeto del que estamos hablando.
2. *http://purl.org/dc/elements/1.1/creator* es la propiedad que estamos definiendo.
3. *http://example.org/staffid/1234* es el valor de la propiedad definida.

RDF modela las afirmaciones como nodos y arcos en una gráfica. El modelo basado en gráficas de RDF está definido en [15]. En esta notación un enunciado está representado por:

1. Un nodo para el sujeto.
2. Un nodo para el objeto.
3. Un arco para el predicado, dirigido del nodo del sujeto al nodo del objeto.

De este modo, la afirmación anterior tendría una representación gráfica como la de la Figura 3.5.

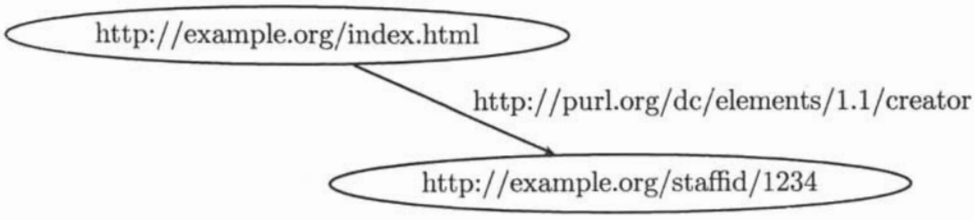


Figura 3.5: Un enunciado sencillo de RDF

Varias afirmaciones sobre un mismo recurso tienen una gráfica con varios arcos apuntando a otros tantos nodos desde el nodo correspondiente a dicho recurso. De este modo las afirmaciones:

> *http://example.org/index.html* tiene una *fecha-de-creación* cuyo valor es 21 de septiembre de 2005. *http://example.org/index.html* tiene un *idioma* cuyo valor es español.

Tendrían una gráfica (usando URIrefs adecuados para fecha-de-creación e idioma) como la que se ve en la Figura 3.6.

A veces no es conveniente dibujar las gráficas al hablar de enunciados de RDF, por lo que existe una notación llamada *tripletas* (triples). La Figura 3.6, podría representarse en tripletas de la siguiente manera:

```
<http://example.org/index.html>
    <http://purl.org/dc/elements/1.1/creator>
        <http://example.org/staffid/1234> .

<http://example.org/index.html>
    <http://example.org/terms/creation-date>
        "21 de septiembre de 2005" .

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

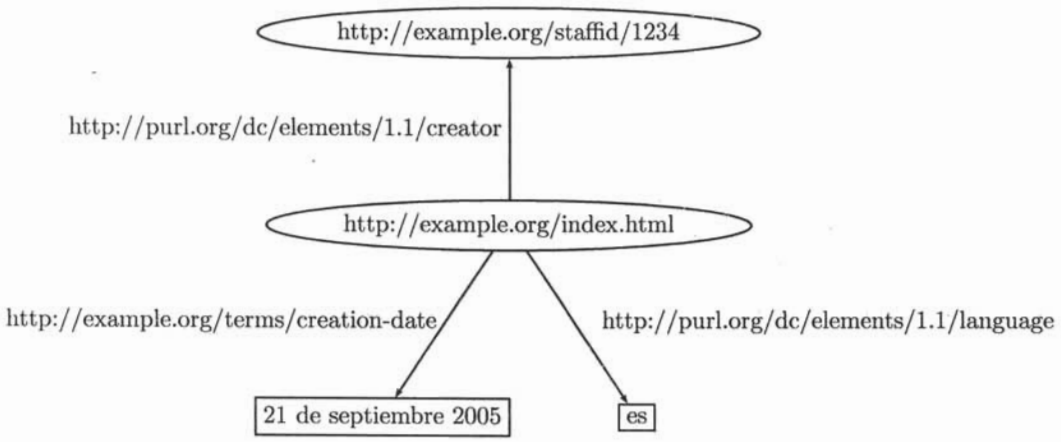


Figura 3.6: Varias afirmaciones en RDF

Cada tripleta corresponde a un solo arco de la gráfica, incluyendo los nodos de salida y entrada de la misma (el sujeto y objeto del enunciado, respectivamente). Esta representación requiere que se repita para cada una de las afirmaciones la representación del sujeto (en este caso `<http://example.org/index.html>`), a diferencia de la gráfica (o las afirmaciones en lenguaje natural). Esto es irrelevante, lo fundamental en RDF es el modelo basado en gráficas (en el sentido matemático de la acepción) de las afirmaciones. La notación usada para representar esta gráfica es secundaria.

Esta notación requiere que los URIrefs sean escritos en su totalidad, entre los caracteres de mayor qué y menor qué, por lo que las afirmaciones se pueden volver un poco largas. Para conveniencia se puede usar una manera abreviada de escribir referencias URI, en la cual se escribe un nombre de XML calificado (XML qualified name, QName) sin los caracteres menor qué y mayor qué como una abreviación para un URIref completo. Un QName contiene un prefijo que ha sido asignado a un URI que designa un espacio de nombres (namespace), seguido por un punto y coma, finalizando con un nombre local. El Uriref completo es formado a partir del QName y añadiendo el nombre local al URI del espacio de nombres asignado al prefijo. Por ejemplo, si tenemos un prefijo de QName `foo` asignado al URI de espacio de nombres `http://example.org/`, entondes el QName foo:bar es una abreviación para el URIref `http://example.org/bar`. Algunos prefijos

comunmente utilizados son:

- rdf:, para el URI de espacio de nombres `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- rdfs:, para el URI de espacio de nombres: `http://www.w3.org/2000/01/rdf-schema#`
- dc:, para el URI de espacio de nombres: `http://purl.org/dc/elements/1.1/`
- owl:, para el URI de espacio de nombres: `http://www.w3.org/2002/07/owl#`
- ex:, para el URI de espacio de nombres: `http://example.org/`
- xsd:, para el URI de espacio de nombres: `http://www.w3.org/2001/XMLSchema#`

Usando esta abreviación el conjunto previamente mencionado de tripletas queda de la siguiente manera:

```
ex:index.html  dc:creator              exstaff:1234 .

ex:index.html  exterms:creation-date   "21 de septiembre de 2005" .

ex:index.html  dc:language             "es" .
```

3.2.3. Valores de propiedades estructurados y nodos en blanco

Hay algunas ocasiones en que las propiedades de un recurso no se pueden expresar como sujeto, nombre y predicado fácilmente. En algunos casos, cuando una propiedad es más propiamente vista como una "agregación" de otras propiedades, se puede hacer uso de los llamados "nodos en blanco" (*blank nodes*).

Por ejemplo si queremos decir que mi nombre es Jorge Jesús Santos Fierro, podemos querer especificar que el nombre propio es Jorge Jesús, el apellido paterno es Santos y el materno Fierro.

En RDF esto se logra considerando el recurso a ser descrito, en este caso, el nombre completo, como un recurso y haciendo afirmaciones adicionales sobre este recurso. La gráfica de la Figura 3.7 muestra el resultado de hacer esto.

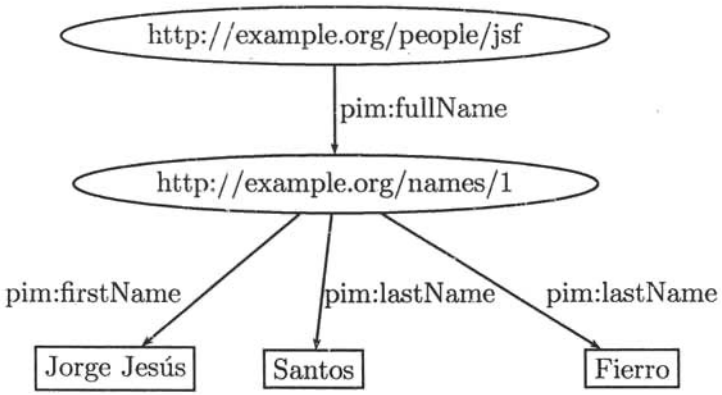


Figura 3.7: Nombre separado en distintos elementos

En la Figura 3.7 podemos notar como al nombre completo se le asignó un URI y a este se le agregaron propiedades que tienen como valores los componentes del mismo. Sin embargo esto puede llevar a generar una gran cantidad de URIs innecesarios, por lo que en RDF podemos hacer uso de "nodos en blanco" (*blank nodes*), que pueden ser sujetos de afirmaciones pero que no tienen asociado un URI específico. Haciendo uso de un nodo en blanco la Figura 3.7 se transformaría en la gráfica de la Figura 3.8.

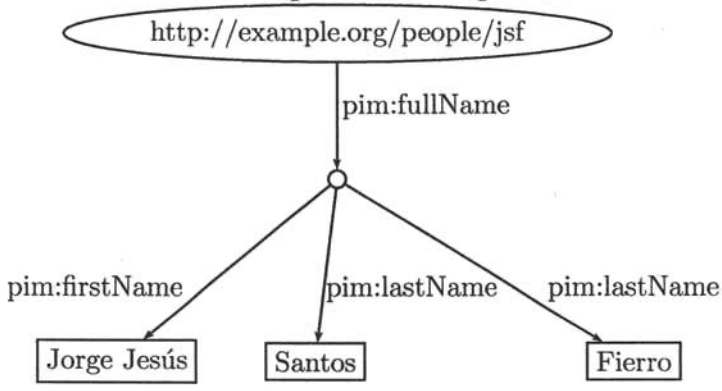


Figura 3.8: Nombre separado en distintos elementos, haciendo uso de un nodo en blanco

Otro caso en donde son útiles los nodos en blanco es cuando queremos decir, por ejemplo que Paola (identificada por `http://example.org/people/paola`) tiene tres hijos, cuyos nombres son Hugo y Luis y sus edades son 11 y 13 años, respectivamente, pero no nos interesa asignar URIs específicas

para los mismos, la gráfica de la Figura 3.9.

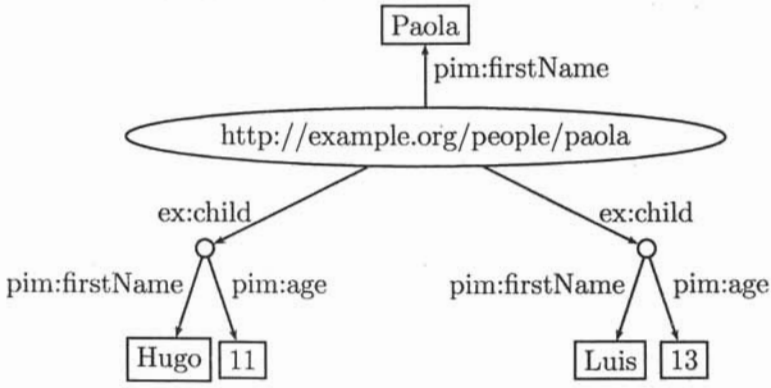


Figura 3.9: Especificando propiedades de un nodo en blanco

En este caso lo interesante es que estamos expresando información sobre los hijos de Paola sin darles un URI explícito.

Los nodos en blanco también pueden ser usados en casos en los que no podemos identificar adecuadamente a un recurso por medio de un URI, por ejemplo, si no hay un URI que me identifique, podemos usar un nodo en blanco para representarme y hacer afirmaciones que describan este nodo, es decir, a mi. Así, para representar las afirmaciones de la Figura 3.10 tendríamos un modelo como el de la Figura 3.11.

> Hay una persona que tiene como nombre Jorge Santos.
>
> Dicha persona tiene como dirección de correo electrónico jsf@ciencias.unam.mx

Figura 3.10: Afirmaciones sobre mi persona

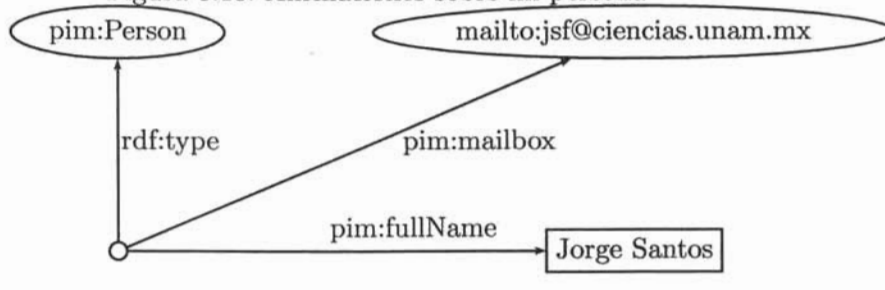


Figura 3.11: Gráfica para las afirmaciones sobre mi persona

En casos como este, podemos tomar propiedades como la de dirección de correo electrónico como identificadores del recurso, dado que la direeción de correo electrónico se asocia normalmente con una sola persona.

3.2.4. Literales con tipo

Hasta ahora, no hemos especificado el tipo de las literales, por ejemplo, en la Figura 3.12, se da la edad de una persona, pero no se especifica el tipo de la edad, la interpretación de la literal se deja al programa que esté trabajando con la información. No hay nada en el modelo que nos indique que la cadena de caracteres "26" deba ser interpretada como un entero, en vez de como el carácter "2" seguido del carácter "6". Esta información puede residir en los programas que procesen la gráfica de RDF con este dato, pero esto supone un problema para otros programas que no cuenten con esta información y que tal vez necesiten procesar la gráfica de RDF.

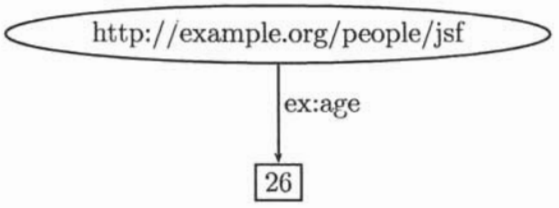


Figura 3.12: Literal sin tipo

Convencionalmente, en bases de datos y lenguajes de programación esto se soluciona asociando tipos a las literales usadas, en RDF se usan literales con tipo (*typed literals*) para solucionar el problema. De tal manera que si un programa ve la cadena de caracteres "26", éste pueda saber si nos estamos refiriendo a un número decimal, octal o a una cadena de caracteres.

Una literal con tipo de RDF se forma añadiendo una cadena con un URIref que identifica a un tipo de datos particular a la literal simple. De este modo, el valor representado por la literal con tipo es el valor que el tipo de datos especificado asocia con la cadena especificada como literal. Por ejemplo, podemos especificar mi edad por medio de la siguiente tripleta:

```
<http://example.org/staffid/1234>
    <http://example.org/terms/age>
        ''26''^^<http://www.w3.org/2001/XMLSchema#integer> .
```

Lo cual, representado por una gráfica, se puede ver en la Figura 3.13.

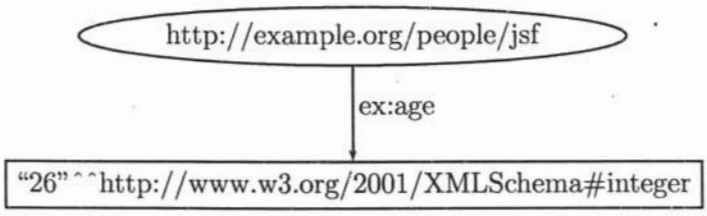


Figura 3.13: Literal con tipo

RDF no tiene un conjunto de tipos de datos predeterminado, en realidad, las literales con tipo de RDF simplemente proveen una manera explícita de indicar qué tipo de datos debe de ser usado para interpretar cierta literal. Adicionalmente, los tipos de datos usados en literales con tipo son definidas de manera externa a RDF e identificados por un URI (a excepción del tipo de datos interno con el URIref rdf:XMLLiteral ver [15]). Con frecuencia son usados los tipos de datos definidos en "XML Schema Part 2: Datatypes" [18]. El que los tipos de datos sean externos tiene la ventaja de que RDF puede representar información de diversas fuentes, sin necesidad de hacer conversiones entre éstas y un conjunto predefinido de tipos nativos.

3.3. RDF/XML

Como se discutió en la sección 3.2.1, el modelo que sigue RDF es el de una gráfica. Hay varias serializaciones posibles de un modelo RDF, como las Tripletas-N (*N-Ttriples*) definidas en el documento *RDF TestCases* [17]. Las especificaciones oficiales de RDF dan una "serialización" oficial de este modelo por medio de la Especificación de Sintaxis RDF/XML (*RDF/XML Syntax Specification* [12]).

3.3.1. Principios Básicos

Un ejemplo sencillo de serialización de una afirmación como la de la Figura 3.14 sería la que podemos ver en la Figura 3.15.

> *http://example.org/index.html* tiene una *creation-date* cuyo valor es *febrero 5, 2005*

Figura 3.14: Descripción de la fecha de creación de un recurso

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:exterms="http://example.org/terms/">

4.   <rdf:Description rdf:about="http://example.org/index.html">
5.       <exterms:creation-date>febrero 5, 2005</exterms:creation-date>
6.   </rdf:Description>

7. </rdf:RDF>
```

Figura 3.15: Serialización RDF/XML para la fecha de creación de un recurso

La linea 1 da la declaración XML (*XML declaraction*), que indica que el resto del archivo es código de XML, así como la versión de XML usada.

Las linea 2 comienza con un elemento `rdf:RDF`. Esto indica que el contenido que sigue representa RDF. Siguen declaraciones de espacio de nombres que serán usadas para para indicar los elementos que pertenecerán al espacio de nombres de RDF (`http://www.w3.org/1999/02/22-rdf-syntax-ns\#`) y otro espacio de nombres para los términos del vocabulario de una organización ficticia (`http://example.org/terms/`).

Las líneas 4 a 6 proveen el texto RDF/XML para la afirmación mostrada en la Figura 3.14. En este ejemplo estamos hablando de una página, el recurso (`http://example.org/index.html`) y estamos diciendo que tiene una propiedad (`exterms:creation-date`) cuyo valor es una fecha (febrero 5, 2005). En este fragmento de XML podemos notar que tanto el recurso como la propiedad son identificados por medio de un URL (en el caso de la propiead, el URIref es `http://example.org/terms/creation-date`, después de expander el prefijo `exterms`; mientras que el valor de la propiead, en este caso es una literal que representa una fecha.

Finalmente, la linea 7 provee la etiqueta que cierra el elemento RDF, en RDF no es indispensable proveer el elemento RDF si el contexto indentifica al texto XML com RDF/XML.

3.4. RDF Schema

RDF provee mecanismos para hacer aserciones acerca de recursos, sin embargo las comunidades que hacen estas descripciones también necesitan ponerse de acuerdo sobre las clases de recursos que se van a describir y las propiedades adecuadas para estas clases de recursos. RDF no provee

mecanismos para especificar clases de recursos y sus propiedades, estos mecanismos son provistos por RDF Schema (RDFS).

RDF Schema provee métodos para describir clases de recursos y sus propiedades por medio de *clases* similares a las usadas en lenguajes de programación orientados a objetos.

Los documentos de RDF Schema no son, en realidad, otra cosa que documentos de RDF, el cual hace uso de un vocabulario de RDF predeterminado, es decir, un conjunto especializado de recursos de RDF con significados particulares.

3.4.1. Classes

Las clases en RDF Schema son "tipos" de objetos que comparten ciertas características, de manera similar a como se definen las clases en Java y otros lenguajes de programación orientados a objetos. Las clases en RDFS son definidas haciendo uso de los recursos de RDF Schema `rdfs:Class` y `rdfs:Resource` y las propiedades `rdf:type` y `rdfs:subClassOf`.

Por ejemplo, supongamos que la organización denotada por el URI `http://ciencias.mx` quiere usar RDF para dar información sobre diferentes tipos de salones. En RDF Schema `http://ciencias.mx` en primer lugar necesitaría una clase para representar la categoría de cosas que son salones. Los recursos que pertenecen a la clase son llamados sus instancias. En este caso, la idea es que las instancias de esta clase sean recursos que son salones.

En RDF Schema, una clase es cualquier recurso con una propiedad `rdf:type` cuyo valor es el recurso `rdfs:Class`. De este modo, la clase de salones sería descrita asignando a la clase un URIref, por ejemplo, `cfs:Salón` (usando `cfs:` para denotar el espacio de nombres `http://ciencias.mx/schemas/facilities\#`, que es el prefijo para el vocabulario de `http://ciencias.mx`) y describiendo dicho recurso con una propiedad `rdf:type` cuyo valor es el recurso `rdfs:Class`. Esto es, `http://ciencias.mx` expresaría la siguiente afirmación de RDF:

```
cfs:Salón    rdf:type    rdfs:Class .
```

Ya con esta clase podemos afirmar que la instalación denotada por el URIref `cf:P201` (donde `cf:` denota el espacio de nombres `http://ciencias.mx/facilities\#`) es, de hecho, un salón:

```
cf:P201    rdf:type    cfs:Salón .
```

`rdfs:subClassOf` se usa para denotar clases especializadas a partir de clases más generales. En el caso de `http://ciencias.mx` tal vez sería deseable designar qué instalaciones son laboratorios, por ejemplo:

```
cf:S0     rdf:type     cfs:Laboratorio .
```

Asumiendo que antes definimos `cfs:Laboratorio` como una clase:

```
cfs:Laboratorio     rdf:type     rdfs:Class .
```

Además de esto, probablemete querríamos distinguir entre laboratorios de Biología y laboratorios de Física. Esto se puede hacer por medio de `rdfs:subClassOf`:

```
cfs:LaboratorioFísica      rdfs:subClassOf     cfs:Laboratorio .
cfs:LaboratorioBiología    rdfs:subClassOf     cfs:Laboratorio .
```

De este modo, podríamos decir que la instalación denotada por `cf:101` es un Laboratorio de Física, mientras que la denotada por `cf:201` es un Laboratorio de Biología:

```
cf:101     rdf:type     cfs:LaboratorioFísica .
cf:201     rdf:type     cfs:LaboratorioBiología .
```

De esto se desprende que tanto la instalación denotada por `cf:101` como la denotada por `cf:201` son `cfs:Laboratorios`. Aunque esto no se diga explícitamente en ninguna afirmación.

En la Figura 3.16 se muestra cómo se puede expresar la información de las expresiones antes mencionadas por medio de RDF/XML.

RDF/XML provee una manera de abreviar la descripción de recursos con una propiedad `rdf:type`. Esto se logra omitiendo dicha propiedad y nombrando al elemento que describe al recurso con el QName equivalente al valor de la propiedad `rdf:type` (una URIref que nombra a la clase). Haciendo uso de esta abreviación nuestro RDF/XML quedaría como se ve en la Figura 3.17.

3.4.2. Properties

Además de describir clases de objetos, los usuarios de RDF necesitan describir las propiedades de dichos objetos (por ejemplo, el número de bancas de un salón). En RDF Schema las propiedades se describen usando la clase

```
<?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:base="http://ciencias.mx/schemas/facilities">

<!-- Clases  -->

<rdf:Description rdf:ID="Salón">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="Laboratorio">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="LaboratorioFísica">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Laboratorio"/>
</rdf:Description>

<rdf:Description rdf:ID="LaboratorioBiología">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Laboratorio"/>
</rdf:Description>


<!-- Otros recursos  -->

<rdf:Description rdf:about="http://ciencias.mx/facilities/P201">
  <rdf:type rdf:resource="#Salón"/>
</rdf:Description>

<rdf:Description rdf:about="http://ciencias.mx/facilities/S0">
  <rdf:type rdf:resource="#Laboratorio"/>
</rdf:Description>

<rdf:Description rdf:about="http://ciencias.mx/facilities/101">
  <rdf:type rdf:resource="#LaboratorioFísica"/>
</rdf:Description>

<rdf:Description rdf:about="http://ciencias.mx/facilities/201">
  <rdf:type rdf:resource="#LaboratorioBiología"/>
</rdf:Description>

</rdf:RDF>
```

Figura 3.16: Serialización RDF/XML para los ejemplos sobre clases

```
<?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:cfs="http://ciencias.mx/schemas/facilities"
 xmlns:base="http://ciencias.mx/schemas/facilities">

<!-- Clases  -->

<rdfs:Class rdf:ID="Salón"/>

<rdfs:Class rdf:ID="Laboratorio"/>

<rdfs:Class rdf:ID="LaboratorioFísica"/>

<rdfs:Class rdf:ID="LaboratorioBiología"/>


<!-- Otros recursos  -->

<cfs:Salón rdf:about="http://ciencias.mx/facilities/P201"/>

<cfs:Laboratorio rdf:about="http://ciencias.mx/facilities/S0"/>

<cfs:LaboratorioFísica rdf:about="http://ciencias.mx/facilities/101"/>

<cfs:LaboratorioBiología rdf:about="http://ciencias.mx/facilities/201"/>

</rdf:RDF>
```

Figura 3.17: Serialización abreviada RDF/XML para los ejemplos sobre clases

rdf:Property, así como las propiedades de RDF Schema rdfs:domain, rdfs:range y rdfs:subPropertyOf.

Las propiedades en RDF son instancias de la clase rdf:Property. En nuestro ejemplo sobre el salón y el número de bancas para definir la propiedad noBancas, lo primero es asignar un URIref a la propiedad, digamos, cfs:noBancas y después describir este recurso con una propiedad rdf:type cuyo valor sea el objeto rdf:Property, como en las tripletas:

```
cfs:noBancas    rdf:type    rdfs:Property .
```

La propiedad rdfs:subPropertyOf nos permite especificar, de manera análoga a como se hace con las clases, qué propiedades son un tipo más específico de una propiedad general. Por ejemplo podemos especificar una propiedad crs:profesor que relacione un grupo con un profesor, esta propiedad, a su vez, puede tener como supropiedades crs:titular y crs:ayudante (donde crs: hace referencia a el espacio de nombres http://ciencias.mx/schemas/registroAcadmcio\#), las tripletas para estas afirmaciones se verían así:

```
crs:profesor    rdf:type              rdfs:Property .
crs:titular     rdf:type              rdfs:Property .
crs:ayudante    rdf:type              rdfs:Property .
crs:titular     rdfs:subPropertyOf    crs:profesor .
crs:ayudante    rdfs:subPropertyOf    crs:profesor .
```

RDF Schema también provee un mecanismo para especificar cómo se deben usar las propiedades en conjunto con las clases, las propiedades rdfs:domain y rdfs:range se usan con este fin. Por ejemplo, asumiendo que crs:Grupo y cps:Académico han sido definidos como clases de RDF Schema:

```
cfs:noBancas    rdfs:domain    cfs:Salón .
cfs:noBancas    rdfs:range     xsd:integer .
crs:profesor    rdfs:domain    crs:Grupo .
crs:profesor    rdfs:range     cps:Académico .
crs:ocupa       rdfs:domain    crs:Grupo .
crs:ocupa       rdfs:range     cfs:Salón .
```

Esto nos indica que la propiedad cfs:noBancas tiene como dominio recursos de la clase cfs:Salón y como rango literales del tipo xsd:integer. De manera similar las siguientes dos tripletas nos indican que la propiedad crs:titular tiene como dominio recursos de la clase crs:Grupo y como

rango recursos de la clase `cps:Académico` (donde `cps:` hace denota el espacio de nombres `http://ciencias.mx/schemas/personal#`). Finalmente, la propiedad `crs:ocupa` tiene como dominio `crs:Grupo` y como dominio `cfs:Salón`, por lo que esta propiedad tiene como intención indicar que un grupo ocupa un cierto salón.

Una propiedad en XML Schema puede ser subpropiedad de ninguna, una o varias propiedades. Todas las propiedades de RDF Schema `rdfs:domain` y `rdfs:range` que aplican a una propiedad también aplican a todas sus subpropiedades. De esta manera, las propiedades `crs:titular` y `crs:ayudante` tienen el mismo dominio y rango que `crs:profesor`, dada su relación de subpropiedades con respecto a `crs:profesor`.

Finalmente, podríamos tener como ejemplo de instancias de estas clases y propiedades las siguientes tripletas:

```
cf:P201     cfs:noBancas
            ''40''^^<http://www.w3.org/2001/XMLSchema#integer> .
cg:2035     crs:titular      http://ciencias.mx/staffid/234 .
cg:2035     crs:ayudante     http://ciencias.mx/staffid/978 .
cg:2035     crs:ocupa        cf:P201 .
```

Donde `cg:` denota el espacio de nombres `http://ciencias.mx/grupos`. La versión RDF/XML de las anteriores afirmaciones se puede ver en la Figura 3.18.

3.4.3. Interpretación de las declaraciones de RDF Schema

Anteriormente se mencionó que el sistema de tipos de RDF Schema es parecido a los sitemas de tipos de lenguajes de programación orientados a objetos. Es necesario dejar claro que aunque la comparación es válida existen diferencias importantes entre estos dos sitemas.

Una diferencia importante es que, en los lenguajes de programación orientados a objetos, normalmente los atributos se definen como pertenecientes a una clase, si una clase Alumno define un atributo nombre, este atributo pertence únicamente a esta clase, una clase Académico puede tener un atributo nombre, pero este atributo se considera diferente al de la clase Alumno; en términos de alcance, este atributo aplica solo a la clase en donde se define. En RDF Schema, por otro lado, las propiedades se definen externamente a las clases, esto quiere decir que, a menos de que se definan propiedades de dominio y rango, un atributo puede tener un recursos de cualquier clase en su dominio y su rango, es decir que, por omisión, tienen un alcance global. Además podemos definir varios dominios y rangos para una misma propiedad. Esto nos dá más flexibilidad, dado que no tenemos

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
<!ENTITY cps "http://ciencias.mx/schemas/personal#">
<!ENTITY cfs "http://ciencias.mx/schemas/facilities#">
<!ENTITY crs "http://ciencias.mx/schemas/registroAcadémico#">
<!ENTITY cst "http://ciencias.mx/staffid/">
<!ENTITY cf  "http://ciencias.mx/facilities/">
<!ENTITY cg  "http://ciencias.mx/grupos/">]>
<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:cps="&cps;"
 xmlns:cfs="&cfs;"
 xmlns:crs="&crs;"
 xmlns:base="http://ciencias.mx/facilities/schema">


<!-- Clases  -->
<rdfs:Class rdf:about="&cfs;Salón"/>
<rdfs:Class rdf:about="&crs;Grupo"/>
<rdfs:Class rdf:about="&cps;Académico"/>


<!-- Datatypes -->
<rdfs:Datatype rdf:about="&xsd;integer"/>


<!-- Propiedades -->
<rdfs:Property rdf:about="&cfs;noBancas">
  <rdfs:domain rdf:resource="&cfs;Salón"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdfs:Property>
<rdfs:Property rdf:about="&crs;profesor">
  <rdfs:domain rdf:resource="&crs;Grupo"/>
  <rdfs:range rdf:resource="&crs;Académico"/>
</rdfs:Property>
<rdfs:Property rdf:about="&crs;titular">
  <rdfs:subPropertyOf rdf:resource="&crs;profesor"/>
</rdfs:Property>


<!-- Otros recursos  -->
<cfs:Salón rdf:about="&cf;P201">
  <cfs:noBancas rdf:datatype="&xsd;integer">40</cfs:noBancas>
</cfs:Salón>
<crs:Grupo rdf:about="&cg;2035">
  <crs:titular rdf:resource="&cst;234"/>
  <crs:ayudante rdf:resource="&cst;978"/>
  <crs:ocupa rdf:resource="&cf;P201"/>
</crs:Grupo>

</rdf:RDF>
```

Figura 3.18: Fragmento de la serialización RDF/XML para los ejemplos sobre propiedades

que definir nuevas propiedades para cada clase que declaremos, pero debemos tener cuidado de no aplicar una propiedad a una clase para la cual no tiene sentido.

Otra consecuencia de la manera como se definen las propiedades en RDF Schema es que no podemos especificar un rango específico para un dominio específico, por ejemplo, podemos tener una propiedad `ex:padreDe`, para la cual desearíamos que cuando tuviese como dominio una instancia de la clase `ex:Humano`, tuviese a su vez como dominio un recurso de la clase `ex:Humano` y cuando la aplicásemos a un recurso de tipo `ex:Perro`, quisiéramos que el rango también perteneciese a la clase `ex:Perro`. Sin embargo, no es posible especificar este comportamiento en RDF Schema, *todos* los rangos definidos para una propiedad aplican para cualquier uso de la misma. Por esto, es importante definir con cuidado los dominios y rangos de las propiedades. Por otro lado, en algunos lenguajes de esquema más ricos, es posible definir este tipo de restricciones.

Otra diferencia importante entre RDF Schema y los lenguajes de programación orientados a objetos es que las descripciones de RDF Schema no son necesariamente *prescriptivas* como normalmente lo son las declaraciones de clases de los lenguajes de programación orientados a objetos. En un lenguaje de programación, si definimos una clase Persona, ésta puede tener un atributo domicilio, la cual puede tener como valor únicamente objetos de tipo Domicilio. Con esta definición es imposible crear una instancia de la clase Persona sin un atributo domicilio. Asimismo, por lo regular no podemos añadir atributos a una instancia cuya clase no los defina. Aún más, este atributo no puede tener como valor objetos que no sean del tipo Domicilio.

En RDF Schema, por otro lado, la información provista por el esquema es información *adicional* sobre recursos, RDF Schema no especifica el comportamiento que una aplicación que haga uso de esta información. Por ejemplo si tenemos un esquema de RDF el cual indica que las propiedades `ex:autor` tienen una propiedad `rdfs:range` cuyo valor es `ex:Persona`, esto es simplemente una afirmación de RDF diciendo que las afirmaciones de RDF que contenga una propiedad `ex:autor` tienen instancias de `ex:Persona` como valores.

La información provista por los esquemas puede ser usada de diferentes maneras. Una aplicación la puede usar para crear formas que reciban datos y se asegure de que los valores de la propiedad `ex:autor` sean recursos de la clase `ex:Persona`. En este sentido la aplicación interpretará la información de manera similar a como lo haría un lenguaje de programación. Otra aplicación podría comparar la clase del valor de una propiedad `ex:autor` y dar una advertencia al usuario en caso de que no sea del tipo `ex:Persona`. Tam-

bién es posible que una aplicación acepte instancias de una clase con menos propiedades de las que el esquema indica como posibles para las mismas o que acepte instancias con propiedades no especificadas en el esquema para la clase de dichas instancias.

El punto es que depende totalmente de la aplicación el uso que se le de a la información contenida en un esquema de RDF, pues siempre son `descriptivos`. Pueden ser `prescriptivos`, pero solo si la aplicación haciendo uso de la información así lo determina. Lo único que RDF Schema hace es proveer afirmaciones adicionales. Si esta información se encuentra en conflicto (para alguna definición de conflicto) con otras afirmaciones, depende de la aplicación determinarlo y actuar en consecuencia.

3.4.4. Otros lenguajes de esquema

RDF Schema provee un lenguaje muy básico para la descripción de vocabularios de RDF. Existen varios aspectos de la descripción de un vocabulario que no provee pero que podrían ser útiles. Algunos de estos aspectos son:

- Restricciones de cardinalidad, por ejemplo, que una persona tiene un solo padre biológico.
- Especificar que una cierta propiedad (p.e. `ex:hasAncestor`) es transitiva.
- Especificar que cierta propiedad es un identificador único para las instancias de una cierta clase.
- Especificar que dos clases (con URIrefs diferentes) representan, en realidad, a la misma clase.
- Especificar que dos instancias (con diferentes URIrefs) representan, en realidad, a la misma instancia.
- Especificar restricciones en el rango o cardinalidad de una propiedad que depende de la clase de recurso a la cual se aplica la propiedad. Por ejemplo, poder decir que un equipo de basquetbol tiene 5 jugadores, mientras que uno de futbol tiene 11.
- Especificar nuevas clases en términos de combinaciones (e.g. uniones e intersecciones) de otras clases, o bien decir que dos clases son disjuntas (i.e. no existen instancias que pertenezcan a ambas clases).

Estos aspectos, así como otros se pueden describir haciendo uso de lenguajes de ontologías como DAML+OIL[3][23] y OWL[4][24]. Ambos lenguajes están basados en RDF y RDF Schema. La idea de dichos lenguajes es proveer semántica adicional capaz de ser entendida por las computadoras, con la intención de que los modelos que describen se apeguen más a sus contrapartes en el mundo real. Aunque las características provistas por este tipo de lenguajes no son indispensables para crear aplicaciones de RDF útiles, su desarrollo es un tema de trabajo muy activo en el desarrollo del Web Semántico.

[3]Para más información sobre DAML+OIL ver: `http://www.daml.org/`
[4]Para más información sobre OWL ver: `http://www.w3.org/2004/OWL/`

Capítulo 4

DOAP

4.1. Objetivos

El proyecto "Descripción de un Proyecto" (Description of a Project, DOAP) es un proyecto para crear un vocabulario RDF/XML cuyo objetivo es describir proyectos de software libre creado por Edd Dumbill[8][9][10][11].

Además de desarrollar un esquema de RDF y ejemplos, el proyecto DOAP pretende proveer herramientas para que tanto el esquema como las instancias de éste puedan ser utilizados en todos los lenguajes de programación populares.

La idea del proyecto surgió de la necesidad de compartir información sobre proyectos de software libre de manera fácil y estándar.

En el mundo del software libre, cualquiera puede iniciar su propio proyecto. Esto trae como consecuencia tal abundancia de software que muchas veces es difícil encontrar lo que se necesita. Como consecuencia, han surgido directorios de software libre, el mejor conocido es, tal vez, *Freshmeat*, el cual es considerado ampliamente como el más completo; pero existen otros, frecuentemente orientados a un subconjunto más específico de software.

Esta situación lleva a que actualizar la información de un proyecto de software libre (e.g. cuando se libera una nueva versión) sea una labor ardua y propensa a fallas, ya que se debe de realizar independientemente en los diferentes registros, por lo cual es frecuente que la información en estos directorios se encuentre desactualizada.

Al tener un formato estándar de XML para representar la información sobre un proyecto de sotware libre, se abre la posibilidad de que los diferentes directorios de software libre hagan uso de éste para actualizar su información sin la mediación directa de personas. Lo único que se modificaría sería el

archivo de XML, y los directorios podrían obtener estos archivos de una ubicación conocida sin mayor intervención humana.

Desde un comienzo, se planteó como uno de los objetivos limitar la complejidad del vocabulario para que no resultara más oneroso el crear instancias del mismo que llevar a cabo la actualización manual de hoy en día.

Los requerimientos para la primera iteración del vocabulario incluyeron los siguientes:

- Limitar las metas para evitar complejidad innecesaria.
- Descripción en varios idiomas de un proyecto de software y sus recursos asociados, incluyendo participantes y recursos del Web.
- Herramientas básicas para la creación y consumo sencillo de dichas descripciones.
- Interoperabilidad con otros proyectos de metadatos basados en Web (RSS, FOAF, Dublin Core).
- Poder extender el vocabulario para propósitos específicos.

No se pretendió en esta primera iteración la descripción de versiones de software (software releases), los cuales podrían ser añadidos en trabajos subsecuentes. Adicionalmente, la información sobre planeación interna del proyecto, tal como asignación de tareas y objetivos del proyecto no fueron considerados como objetivos.

Para auxiliar en el diseño del esquema, se tomaron en cuenta los siguientes casos de uso para descripciones de proyectos incluyen:

- Fácil importación de proyectos a directorios de software.
- Intercambio de datos entre directorios de software.
- Configuración automática de recursos tales como repositorios de CVS o de errores (bug trackers).
- Ayudar a desarrolladores de paquetes que empaquetan software para distribuciones.

4.2. Tecnologías

A partir de estas metas, llega el momento de decidir qué tecnologías existentes usar, al pretender ser un formato de intercambio de información en Internet, procesable por computadoras, se nos presentan dos opciones naturales: XML y RDF. Al decidir que tecnología usar no debemos olvidar que queremos tener una especificación lo suficientemente precisa para evitar ambigüedades.

El uso de RDF presenta algunas limitaciones: un documento de XML bien formado[1] no es necesariamente un documento de RDF, por lo que es un poco más complejo crear un documento de RDF que uno de XML. Además, para beneficiarnos de las ventajas de RDF necesitamos herramientas especiales. Por otro lado, aunque XML define perfectamente la sintaxis de un documento, no nos dice nada acerca de la semántica de sus documentos. RDF Schema (y su hermano mayor OWL, W3C Ontology Language) permite especificar, por ejemplo, que el desarrollador principal de un proyecto es una subclase del término "creator" del vocabulario Dublin Core. En contraste, un documento de XML llano no brinda información explícita a un programa que no trate de manera particular el espacio de nombres de DOAP, aún cuando tuviese un esquema de XML que especificase la sintaxis adecuada.

Finalmente, en XML existe el problema de mezclar espacios de nombres. Si tenemos dos vocabularios arbitrarios de diferentes espacios de nombres, estos no se pueden mezclar arbitrariamente, se tiene que especificar explícitamente un nuevo vocabulario en términos de los otros dos. Por otro lado, RDF tiene una solución bien especificada, por lo que si se pretende mezclar DOAP con otros espacios de nombres, RDF es una mejor opción.

Finalmente Edd Dumbill se decidió por RDF, ya que este lenguaje está diseñado explícitamente para expresar metadatos. No se dejó de lado, sin embargo, la percepción de que RDF es más complejo y se procuró que los documentos DOAP pudiesen ser procesados con herramientas de XML normales, para lo cual se diseñó un esquema de XML.

Para propósitos de proceso automatizado, se usó RDF Schema para especificar el vocabulario DOAP. Fue complementado con prosa lo más posible. Esta aproximación fue tomada por la especificación FOAF [5] y ha tenido bastante éxito.

[1] En el sentido del estándar de XML [4].

4.3. Trabajos Relacionados

Durante el desarrollo de DOAP fue importante tomar en cuenta el trabajo existente y cómo se relaciona con este proyecto. Se tomaron en cuenta los siguientes proyectos:

Exportación en XML de Freshmeat: El registro de software Freshmeat.net exporta todos sus datos en un archivo de XML, actualizado diariamente[2]. También provee un DTD para el formato de XML usado[3]. Leigh Dodds ha hecho algo de trabajo para transformar este archivo a datos en términos de FOAF[4].

Open Source Metadata Framework: Este proyecto se enfoca en metadatos para la documentación de proyectos de software libre, por lo que comparte algunas metas importantes con DOAP. Se usa ampliamente como parte del Proyecto de Catalogación de Documentación Abierta ScrollKeeper (ScrollKeeper Open Documentation Cataloging Project).

PRJ Project Vocabulary: Este vocabulario, creado por Danny Ayers, está enfocado a la administración de proyectos en general, sin importar el dominio de estos.

CPAN2FOAF: CPAN (the Comprehensive Perl Archive Network), un enorme repositorio de software escrito en Perl. Dan Brickley ha estado trabajando en transformar información sobre los autores de módulos en CPAN a FOAF/RDF.

Description of a Software Project: Este es el comienzo de un vocabulario parecido a DOAP, creado por Max Völkel.

RPMFind: Este servicio de localización de software usa descripciones de RDF de software empaquetado en el formato RPM. Los metadatos son muy detallados en cuanto a cada "release" del software.

Revisando las propiedades que se definen en cada uno de estos proyectos se llegó a una lista de propiedades que debería tener el proyecto DOAP.

---

[2] Ver `http://download.freshmeat.net/backend/`. Nótese que los archivos con terminación `.rdf` son, en realidad archivos XML

[3] Ibidem

[4] Ver `http://rdfweb.org/pipermail/rdfweb-dev/2004-January/012482.html`

4.4. Indentificación Unívoca de Proyectos

Un problema que fue necesario resolver es el de cómo identificar unívocamente a los proyectos. El identificador del proyecto debía ser global y único para que fuese posible compartir las descripciones DOAP en el Web. Esto se contraponía al objetivo de que DOAP fuese descentralizado. Las descripciones pueden ser creadas y distribuidas sin registrarse en un "Web site" particular.

En el Web, una forma común de identificar globalmente un objeto es darle un URI. Como todo proyecto de software tiene una página web en el Web, parecía razonable usar la URI de esta página web como la característica identificadora de un proyecto. La otra opción era usar el nombre del proyecto como identificador. El problema de usar el nombre es la falta de una autoridad a la cual apelar en el caso de duplicados. Puede suceder que un proyecto escoja un nombre ya existente. En tales casos, hay confusión y el conflicto puede no tener una conclusión satisfactoria. En el caso de URIs (esto es, URLs) de páginas web, la autoridad global del sistema DNS asegura que no haya conflictos de nombres.

El usar URLs de páginas web tiene una desventaja obvia. En un mundo ideal los URLs no cambian [2]. En el mundo real cambian continuamente. El administrador de un proyecto puede cambiar el servidor anfitrión donde está disponible la página del proyecto. Un proyecto pude cambiar de administrador y tener éste otros recursos. O, simplemente, puede que el sitio web sea reorganizado. Claramente, se qusiera evitar que todas las descripciones DOAP afectadas sean invalidadas si algo así sucede.

Este problema se solucionó añadiendo una propiedad de página vieja. Un proyecto puede tener más de una de estas propiedades, las cuales pueden ser añadidas siempre que el páginas web cambia. Se puede entonces considerar a las páginas web viejas como un identificador adicional.

Al hacer de DOAP un proyecto descentralizado y global se encontraron otras complicaciones, además de la identificación única de objetos. El rango de valores que las propiedades pueden tomar deben también ser predecibles para que se pueda hacer uso de la colección global de información DOAP. Podemos tomar como ejemplo la propiedad de la licencia del software.

4.5. Limitando las propiedades de los valores

Las personas pueden darse cuenta fácilmente de que no hay diferencia lguna en el significado de GPL2, GNU General Public License, Version 2,

e, inclusive, http://www.gnu.org/licenses/gpl.html. Las computadoras no pueden hacer esto. La solución usual a un problema como éste, inspirada por las bases de datos, es usar un conjunto de códigos previamente acordados o abreviaturas para cada una de las licencias. Adicionalmente, se necesita un mecanismo de extensión para lo casos en que un nueva licencia es usada.

Aquí es donde un aspecto interesante del uso de RDF/XML entra en juego. En RDF/XML, una propiedad puede tomar dos tipos de valores: un recurso, identificado por un URI, o una cadena de caracteres literal. Estas literales pueden tener un tipo, de tal manera que se puede definir una enumeración de W3C XML Schema para controlar los posibles valores [16]. La propiedad *licence* puede en este caso ser uno de, por ejemplo, GPL, BSD, Apache, etc. Si es *"Other"*, es decir, otra, se puede añadir un campo de texto extra para describir la nueva licencia.

La desventaja de tomar esta aproximación es que se les añade una carga adicional a aquellos que procesan un archivo DOAP. Tienen que tomar en cuenta la presencia de un esquema extra e importar la pesada maquinaria requerida para hacer validación de W3C XML Schema. Aún entonces, lo único que se obtiene es una cadena opaca que debe ser aumentada con información extra (e.g. una liga al texto de la licencia), si ha de ser de utilidad a un observador humano que no conozca el significado de la cadena. Usar una enumeración también crea una sobrecarga adicional para los desarrolladores del vocabulario DOAP, ya que ahora existe un esquema extra que desarrollar y distribuir.

Al usar un recurso en vez de una cadena literal, se gana en flexibilidad. Podemos ahora designar a la licencia GPL como un URI en un espacio de nombres en el cual se tenga control, por ejemplo, `http://example.org/doap/licenses/GPL`. Adicionalmente se puede poner una página con información adicional sobre la licencia en esa dirección, incluyendo su texto completo. Como una cortesía adicional, se puede publicar una lista completa de las licencias soportadas. Esto no añade ninguna sobrecarga al procesador DOAP. Se puede facilitar la labor de los procesadores si se crea un archivo ubicado en el URL `.../doap/licences/` que contenga una lista completa de las licencias que pueda ser manipulada automáticamente y aumentada con etiquetas, descripciones de las licencias y otra información útil.

Usar un URL para representar las licencias tiene dos desventajas. La primera es que resulta más engorroso escribir el URL completo de la licencia que, simplemente, la cadena GPL, esto puede ser simplificado usando abreviaturas en la sintaxis o con la ayuda de herramientas de edición.

La segunda desventaja se refiere al control del URL, pues puede perderse y aunque no se requiere que el URL especifique una ubicación de web exis-

tente para que el espacio de nombres sea válido esto puede causar confusión.

Es importante hacer notar que el usar recursos para definir propiedades no siempre es la mejor solución. Se debe tener en cuenta en cada caso si es necesaria la flexibilidad y extensibilidad de usar recursos o si es mejor usar una cadena de caracteres fijos dado que esto último es más sencillo.

En el caso de Creative Commons[5], se usaron URIs para designar la licencia de recursos electrónicos y dicho método ha resultado bastante exitoso.

4.6. La estructura de un archivo DOAP

Al usar RDF y su serialización en XML tendremos archivos como el que se ve en la Figura 4.1.

En esta figura podemos ver una descripción mínima del proyecto DOAP mismo. De este ejemplo podemos notar varias cosas:

- El elemento raíz es `<Project>`, dado que la especificación de febrero de 2004 de la sintaxis de RDF permite omitir el contenedor `<rdf:RDF>` si escribimos la descripción con un solo nodo.
- Se utiliza el vocabulario FOAF[6], para describir a las personas.
- El espacio de nombres de DOAF es `http://usefulinc.com/ns/doap#`.
- El atributo estándar `xml:lang` denota el idioma de propiedades textuales.

Las clases del proyecto DOAP son las siguientes:

Project El elemento principal del proyecto.

Version Una instancia de software liberado.

Repository Un repositorio de software.

4.7. Clase `Project`

La clase *Project* es la principal clase de DOAP. Cada proyecto es identificado unívocamente por su página principal, así como sus páginas anteriores,

[5]Se puede encontrar más información acerca de Creative Commons en la dirección `http://creativecommons.org/`

[6]Se puede encontrar más información sobre FOAF en `http://www.foaf-project.org/`

```
<Project xmlns="http://usefulinc.com/ns/doap#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">

        <name>DOAP</name>
        <homepage rdf:resource="http://usefulinc.com/doap" />
        <created>2004-05-04</created>
        <shortdesc xml:lang="en">
            Tools and vocabulary for describing community-based
            software projects.
        </shortdesc>
        <description xml:lang="en">
            DOAP (Description of a Project) is an RDF
            vocabulary and associated set of tools for
            describing community-based software projects.
            It is intended to be an interchange vocabulary for
            software directory sites, and to allow the
            decentralized expression of involvement in a project.
        </description>
        <maintainer>
            <foaf:Person>
                <foaf:name>Edd Dumbill</foaf:name>
                <foaf:homepage
                 rdf:resource="http://usefulinc.com/edd" />
            </foaf:Person>
        </maintainer>
</Project>
```

Figura 4.1: Ejemplo de archivo DOAP: Parte de la descripción del proyecto DOAP.

como vimos en la sección 4.4. La propiedad *homepage* es la única "obligatoria"[7] para esta clase.

Sus atributos son:

name: El nombre del proyecto.

shortname: El nombre corto del proyecto, normalmente usado en nombres de archivo.

homepage: URI del *homepage*, asociada únicamente con este proyecto.

old-homepage: URI de una *homepage* anterior, asociada únicamente con este proyecto.

created: La fecha de creación del proyecto en formato: AAAA-MM-DD

description: Una descripción completa del proyecto, puede tener varios enunciados.

shortdesc: Una descripción corta del proyecto, suele consistir de ocho o nueve palabras.

category: Una URI de una categoría asignada al proyecto.

wiki: URI de un wiki asociado a este proyecto.

bug-database: URI de una base de datos de errores o una dirección de correo a donde se puedan mandar reportes de errores.

screenshots: URI de una página con imágenes del proyecto.

mailing-list: URI de una lista de correos asociada al proyecto.

programming-language: Lenguaje de programación con el que está implementado el proyecto o con el que se supone se va a usar este proyecto.

os: Sistema operativo en el que se puede usar el proyecto (omitir si el proyecto no es específico de un sistema operativo).

license: URI de la licencia bajo la cual está el proyecto.

[7] En realidad no hay ninguna propiedad obligatoria, *homepage* es obligatoria en el sentido de que un recurso de la clase *Project* no es de mucha utilidad sin esta propiedad, dado que es la propiedad que identifica el proyecto.

download-page: URI del lugar de donde se puede bajar el proyecto.

download-mirror: URI de un "espejo" del lugar donde se puede bajar el proyecto.

repository: URI de un `doap:Repository` que describe un repositorio de código fuente del proyecto.

release: Un `doap:Version` que describe el *release* actual del proyecto.

maintainter: Una `foaf:Person` que describe el desarrollador principal o líder del proyecto.

developer: Una `foaf:Person` que describe un desarrollador del proyecto.

documenter: Una `foaf:Person` que describe una persona que a aportado documentación al proyecto.

translator: Una `foaf:Person` que describe una persona que a aportado traducciones al proyecto.

helper: Una `foaf:Person` que describe una persona que ha contribuido al proyecto de una manera no cubierta por las propiedades anteriormente mencionadas.

4.8. Clase `Version`

La clase *Version* es usada para describir, únicamente, la versión actual del proyecto. La clase *Version* representa una instancia de una versión (*Release*) de software.

branch: Una cadena que describe la "rama" de esta versión, como puede ser *stable* (estable), *unstable* (inestable), gnome24, gnome26, etc.

name: El nombre de la versión, e.g. Panther.

created: La fecha de la versión en formato: AAAA-MM-DD.

revision: Número de la versión, e.g. 1.0.

4.9. Clase Repository

La clase *Repository* tiene varias subclases, una por cada tipo de repositorio (CVS, Arch, etc.) definido en el vocabulario. Con ella se puede especificar dónde podemos acceder al repositorio de software del proyecto.

Subclases de la clase *Repository*:

SVNRepository: Repositorio Subversion.

BKRepository: Repositorio BitKeeper.

CVSRepository: Repositorio CVS.

ArchRepository: Repositorio Arch.

Estos repositorios pueden tener diferentes propiedades:

anon-root: "Path" de la raíz del repositorio público.

module: Nombre del módulo del código fuente dentro del repositorio.

browse: URL de la interfaz web del repositorio.

location: URL base del repositorio.

Estas propiedades no aplican por igual a todas las subclases de *Repository*, las propiedades son apropiadas o no dependiendo de su clase específica, como se ve en el cuadro 4.1.

Propiedad	SVNRepository	BKRepository	CVSRepository	ArchRepository
anon-root			*	
module			*	
browse	*	*	*	
location	*	*		*

Cuadro 4.1: Propiedades de las subclases de la clase `doap:Repository`

4.10. Conclusión

A través del esquema de RDF DOAP se pretende dar una descripción que cubra todas las necesidades de comunicación acerca de un proyecto de software, particularmente si se trata de software libre. Haciendo uso de esta descripción y extendiéndola con nuevas propiedades adecuadas a nuestras necesidades podemos acercarnos más a la meta de esta tesis.

Capítulo 5

Obtención de categorías desde `freshmeat.net`

5.1. Introducción

Nuestro propósito básico es clasificar los paquetes dentro del sistema operativo Debian GNU/Linux por medio de categorías para facilitar su localización y poder manipular conjuntos de paquetes relacionados.

Esto parece una tarea colosal, considerando que Debian tiene miles de paquetes. Sin embargo ya existen recursos a nuestra disposición que nos permiten abreviar un poco esta tarea.

Freshmeat.net es probablemente el directorio de software libre con más proyectos registrados. Este directorio exporta la información en su base de datos por medio de archivos de XML en la dirección `http://freshmeat.net/backend`[1]. Aquí podemos encontrar varios archivos que nos pueden ser de utilidad.

5.2. Trove

Freshmeat.net tiene una clasificación que puede ser aplicada a los proyectos registrados en este directorio. Esta clasificación consiste de categorías organizadas jerárquicamente en 5 árboles, cada uno de los cuales se refiere a un aspecto del proyecto de software. Los aspectos que cada uno de estos árboles define son:

[1] Aunque varios de estos archivos tienen terminación "rdf", en realidad son archivos de XML que no representan RDF

- Estado de desarrollo (*Development Status*).
- Ambiente (*Envionment*).
- Licencia (*License*).
- Ambiente de red (*Network Environment*).
- Sistema Operativo (*Operating Systems*).
- Lenguaje de Programación (*Programming Language*).
- Tema (*Topic*).
- Traducciones (*Translations*).

El archivo que contiene todas las categorías que se pueden aplicar a los proyectos, así como la estructura de los árboles que forman estas categorías y los proyectos que caen dentro de cada una de estas categorías, se pueden encontrar en el archivo `http://download.freshmeat.net/backend/fm-trove.rdf`. La estructura de este archivo se encuentra especificada por un DTD que se puede encontrar en la dirección `http://download.freshmeat.net/backend/fm-trove-0.1.dtd` y la parte sustantiva del cual se puede ver en la Figura 5.1 (el resto son definiciones de entidades de uso común). El archivo `fm-trove.rdf` tiene una gran cantidad de información que nos puede servir como base para clasificar los paquetes de Debian dentro de categorías (dado que los paquetes de Debian no son otra cosa que el resultado de tomar proyectos de software libre, muchos de los cuales se encuentran registrados en freshmeat.net, y ponerlos en un formato específico).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT trove-listing (descriminator*)>
<!ELEMENT descriminator (id, name, parent_id, root_id, projects?)>
<!ELEMENT projects (project_id*)>
<!ELEMENT project_id (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT parent_id (#PCDATA)>
<!ELEMENT root_id (#PCDATA)>
```

Figura 5.1: fm-trove DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE
 trove-listing
 SYSTEM
 "http://freshmeat.net/backend/fm-trove-0.1.dtd">
<trove-listing>
  <descriminator>
    <id>6</id>
    <name>Development Status</name>
    <parent_id>0</parent_id>
    <root_id>0</root_id>
  </descriminator>
  <descriminator>
    <id>7</id>
    <name>
      Development Status :: 1 - Planning (disabled category)
    </name>
    <parent_id>6</parent_id>
    <root_id>6</root_id>
    <projects>
      <project_id>302</project_id>
      <project_id>4754</project_id>
      <project_id>7408</project_id>
      <project_id>8708</project_id>
    </projects>
  </descriminator>
</trove-listing>
```

Figura 5.2: Parte de fm-trove.rdf

Parte del archivo `fm-trove.rdf` que define este DTD se puede ver en la Figura 5.2.

Así pues, me dí a la tarea de especificar un esquema de RDF que fuera equivalente semánticamente (según la semántica que podemos inferir del archivo XML) al DTD definido por freshmeat.net. El resultado se puede apreciar en la Figura 5.3

Observando la Figura 5.3, podemos notar, en primer lugar, que no tenemos nada equivalente al elemento `trove-listing`, no me pareció necesario tener un contenedor para lo que básicamente son una serie de elementos que forman un árbol. Los elementos `descriminator` (sic.) serán representados en instancias de RDF como elementos de la clase `Discriminator`, esta clase tiene únicamente dos propiedades asociadas, la propiedad `name` que tendrá como valor el contenido del elemento `name` correspondiente en el archivo `fm-trove.rdf` y la propiedad `parent` que indicará el padre de un `Discriminator` en el árbol que forman las categorías. El elemento `root_id`, que sirve para identificar a la raíz del árbol en el archivo `fm-trove.rdf`, en realidad es innecesario. Asimismo, es innecesario incluir equivalentes a los elementos `projects` y sus subclementos `project_id` en los archivos RDF, ya que esta información se encuentra especificada en otros archivos. En la Figura 5.5 podemos ver un fragmento de lo que resulta al convertir el archivo `fm-trove.rdf` a formato RDF (según el esquema de RDF especificado). Esta instancia fue obtenida a partir del archivo `fm-trove.rdf` mediante la transformación XSLT de la Figura 5.4.

Aquí podemos notar que las categorías son recursos, identificados por URIs, por lo cual podemos decir más de ellos en este documento de RDF o en otros.

5.3. Proyectos

freshmeat.net exporta otro archivo que es de utilidad, pues es el archivo que lista toda la información que tienen sobre proyectos. Este archivo es `fm-projects.rdf.bz2` y se puede encontrar en `http://freshmeat.net/backend/fm-projects.rdf.bz2`. El DTD que define su estructura está en `http://freshmeat.net/backend/fm-projects-0.4.dtd` y la parte sustancial del mismo se puede ver en la Figura 5.6 (de nuevo, el resto son definiciones de entidades estándar).

Un fragmento del archivo `fm-projects.rdf` se puede apreciar en la Figura 5.7.

Ya tenemos una ontología para proyectos de software, la ontología DOAP,

```
<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:owl="http://www.w3.org/2002/07/owl#"
  xmlns:vs="http://www.w3.org/2003/06/sw-vocab-status/ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:doap="http://usefulinc.com/ns/doap#">

  <owl:Ontology rdf:about="http://example.org/ns/trove#">
    <owl:imports rdf:resource="http://xmlns.com/foaf/0.1/index.rdf" />
    <dc:description>
      Vocabulario RDF para la clasificación Trove
    </dc:description>
    <dc:creator>Jorge Santos</dc:creator>
    <dc:format>application/rdf+xml</dc:format>
    <foaf:maker>
      <foaf:Person>
        <foaf:name>Jorge Santos</foaf:name>
        <foaf:mbox rdf:resource="mailto:jsf@ciencias.unam.mx" />
      </foaf:Person>
    </foaf:maker>
  </owl:Ontology>
  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/ns/trove#Discriminator">
    <rdfs:isDefinedBy
     rdf:resource="http://dep3.fciencias.unam.mx/ns/trove#" />
    <rdfs:label xml:lang="es">Un discriminador</rdfs:label>
  </rdfs:Class>
  <!-- Properties -->
  <rdf:Property rdf:about="http://example.org/ns/trove#name">
    <rdfs:isDefinedBy
     rdf:resource="http://dep3.fciencias.unam.mx/ns/trove#" />
    <rdfs:label xml:lang="es">El nombre del discriminador.</rdfs:label>
    <rdfs:domain
     rdf:resource="http://example.org/ns/trove#Discriminator" />
  </rdf:Property>
  <rdf:Property rdf:about="http://example.orgns/trove#parent">
    <rdfs:isDefinedBy rdf:resource="http://usefulinc.com/ns/doap#" />
    <rdfs:label xml:lang="es">El padre del discriminador.</rdfs:label>
    <rdfs:domain
     rdf:resource="http://example.org/ns/trove#Discriminator" />
    <rdfs:range
     rdf:resource="http://example.org/ns/trove#Discriminator" />
  </rdf:Property>
</rdf:RDF>
```

Figura 5.3: Esquema trove

```
<xsl:stylesheet version = '1.0'
                xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
                xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
                xmlns:trove='http://example.org/ns/trove#'>

  <xsl:template match="trove-listing">
    <rdf:RDF>
      <xsl:for-each select="descriminator">
        <trove:Discriminator
         rdf:about="http://software.freshmeat.net/browse/{id}">
          <trove:name>
            <xsl:value-of select="name" />
          </trove:name>
          <trove:parent
           rdf:resource="http://software.freshmeat.net/browse/{parent_id}"/>
        </trove:Discriminator>
      </xsl:for-each>
    </rdf:RDF>
  </xsl:template>

</xsl:stylesheet>
```

Figura 5.4: Transformación XSLT para fm-trove.rdf

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:trove="http://example.org/ns/trove#">
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/6">
    <trove:name>Development Status</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/0" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/7">
    <trove:name>1 - Planning (disabled category)</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/8">
    <trove:name>2 - Pre-Alpha</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/9">
    <trove:name>3 - Alpha</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/10">
    <trove:name>4 - Beta</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/11">
    <trove:name>5 - Production/Stable</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
  <trove:Discriminator
   rdf:about="http://software.freshmeat.net/browse/12">
    <trove:name>6 - Mature</trove:name>
    <trove:parent
     rdf:resource="http://software.freshmeat.net/browse/6" />
  </trove:Discriminator>
</rdf:RDF>
```

Figura 5.5: Fragmento de una instancia del esquema definido para Trove

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT project-listing (project*)>
<!ELEMENT project (project_id, date_added, date_updated,
projectname_short, projectname_full, desc_short,
desc_full, vitality_score, vitality_percent, vitality_rank,
popularity_score, popularity_percent, popularity_rank, rating,
rating_count, rating_rank, subscriptions, branch_name,
url_project_page, url_homepage, url_tgz, url_bz2, url_zip,
url_changelog, url_rpm, url_deb, url_osx, url_bsdport,
url_purchase, url_cvs, url_bugtracker, url_list, url_mirror,
url_demo, license, latest_release, screenshot_thumb, authors,
descriminators, dependencies)>

<!ATTLIST dependency type CDATA #REQUIRED>
```

Figura 5.6: Fragmento de fm-projects-0.4.dtd

por lo que no es necesario crear una nueva. Lo que hice a continuación es transformar la información relevante a DOAP que contiene este archivo de XML mediante una trasformación de XSLT, la cual se puede observar en la Figura 5.8

Como podemos ver, de toda la información que se encuentra en el archivo `fm-projects.rdf`, es realmente poco lo que podemos rescatar, sin embargo, para este trabajo lo que realmente nos interesa son las clasificaciones, por lo que es afortunado que éstas se puedan obtener más o menos fácilmente, también cabe notar que se ha vuelto a omitir el elemento más exterior, en este caso `project-listing` por no considerarse de utilidad. Parte del resultado de transformar el archivo `fm-projects.rdf` a RDF (de acuerdo al esquema DOAP) se puede apreciar en la Figura5.9.

5.4. Conclusión

Aunque transformar toda esta información puede parecer un ejercicio fútil, en realidad nos trae varias ventajas. En primer lugar, el árbol que forman las categorías de freshmeat.net puede ser aumentado independientemente del proyecto freshmeat.net, si se le quisiera añadir otra categoría al árbol en el formato XML, habría que modificar el archivo XML en sí, en formato RDF, cualquiera puede añadir nuevas categorías sin necesidad de ponerse en contacto con los administradores de freshmeat.net.

Por otro lado, al transformar el archivo `fm-projects.rdf` a RDF po-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE project-listing
  SYSTEM "http://freshmeat.net/backend/fm-projects-0.4.dtd">
<project-listing>
  <project>
    <project_id>2296</project_id>
    <date_added>1998-08-21 11:02:40</date_added>
    <date_updated>2005-08-07 08:24:38</date_updated>
    <projectname_short>emacs</projectname_short>
    <projectname_full>Emacs</projectname_full>
    <desc_short>
      The extensible, customizable, self-documenting,
      real-time display editor.
    </desc_short>
    <desc_full>
      Emacs is the extensible, customizable, self-documenting,
      real-time display editor. Emacs has special code editing modes,
      a scripting language (elisp), and comes with many packages for
      doing mail, news and more, all in your editor.
    </desc_full>
    <vitality_rank>142</vitality_rank>
    <popularity_rank>206</popularity_rank>
    <rating>8.67</rating>
    <rating_count>72</rating_count>
    <rating_rank>88</rating_rank>
    <subscriptions>217</subscriptions>
    <branch_name>Default</branch_name>
    <url_project_page>http://freshmeat.net/pro/emacs/</url_project_page>
    <url_homepage>http://freshmeat.net/emacs/url_homepage/</url_homepage>
    <url_tgz>http://freshmeat.net/redir/emacs/2296/url_tgz/</url_tgz>
    <url_bz2></url_bz2> <url_changelog></url_changelog> <url_rpm></url_rpm>
    <url_deb>http://freshmeat.net/redir/emacs/2296/url_deb/</url_deb>
    <url_bsdport></url_bsdport>
    <url_cvs>http://freshmeat.net/redir/emacs/2296/url_cvs/</url_cvs>
    <url_bugtracker></url_bugtracker>
    <url_list></url_list>
    <license>GNU General Public License (GPL)</license>
    <latest_release>
      <latest_release_version>21.3</latest_release_version>
      <latest_release_id>122343</latest_release_id>
      <latest_release_date>2003-05-09 10:22:19</latest_release_date>
    </latest_release>
    <descriminators>
      <trove_id>226</trove_id>
      <trove_id>2</trove_id>
      <trove_id>15</trove_id>
    </descriminators>
    <dependencies>
    </dependencies>
  </project>
</project-listing>
```

Figura 5.7: Un fragmento del archivo fm-projects.rdf

```
<xsl:stylesheet version = '1.0'
                xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
                xmlns:rdf='http://www.w3.org/1999/02/rdf-synt-ns#'
                xmlns:trove='http://example.org/ns/trove#'
                xmlns:doap="http://usefulinc.com/ns/doap#">

  <xsl:template match="project-listing">
    <rdf:RDF>
      <xsl:for-each select="project">
        <doap:Project>
          <doap:name>
            <xsl:value-of select="projectname_full" />
          </doap:name>
          <doap:shortdesc>
            <xsl:value-of select="desc_short" />
          </doap:shortdesc>
          <doap:description>
            <xsl:value-of select="desc_full" />
          </doap:description>
          <doap:homepage>
            <xsl:value-of select="url_homepage" />
          </doap:homepage>
          <xsl:for-each select="descriminators/trove_id">
            <category
             rdf:resource="http://software
                           .freshmeat.net/browse/{.}/"/>
          </xsl:for-each>
        </doap:Project>
      </xsl:for-each>
    </rdf:RDF>
  </xsl:template>

</xsl:stylesheet>
```

Figura 5.8: La transformación fm-projects.xsl (fragmento)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:trove="http://example.org/ns/trove#"
         xmlns:doap="http://usefulinc.com/ns/doap#">
  <doap:Project>
    <doap:name>Emacs</doap:name>
    <doap:shortdesc>The extensible, customizable, self-documenting,
    real-time display editor.</doap:shortdesc>
    <doap:description>Emacs is the extensible, customizable,
    self-documenting real-time display editor. Emacs has special
    code editing modes, a scripting language (elisp), and comes
    with many packages for doing mail, news and more, all in your
    editor.</doap:description>
    <doap:homepage
     rdf:resource="http://www.gnu.org/software/emacs/" />
    <doap:category
     rdf:resource="http://software.freshmeat.net/browse/226/" />
    <doap:category
     rdf:resource="http://software.freshmeat.net/browse/2/" />
    <doap:category
     rdf:resource="http://software.freshmeat.net/browse/15/" />
  </doap:Project>
</rdf:RDF>
```

Figura 5.9: Parte del archivo fm-projects-doap.rdf

demos hacer afirmaciones adicionales sobre el proyecto, por ejemplo, que está empacado por tal o cuál paquete de Debian.

Capítulo 6

Implementación

Para realizar la implementación se usaron las siguientes tecnologías: el lenguaje de programación Ruby, librdf con la interfaz para Ruby y "Ruby on Rails"[27] una plataforma para desarrollo de aplicaciones web.

6.0.1. Ruby

Ruby es un lenguaje de programación sencillo y poderoso orientado a objetos, creado por Yukihiro Matsumoto (también conocido como "matz").

Tiene las características usuales de los lenguajes orientados a objetos, clases, herencia, métodos, iteradores, etc. Sin embargo, a diferencia de otros lenguajes orientados a objetos populares, en Ruby todo es un objeto, de tal manera que se pueden hacer cosas como:

```
> 5.class.name
=> ''Fixnum''
> 5.class.superclass.name
=> ''Integer''
```

Además, tiene mecanismos menos usuales, como métodos unitarios ("singleton methods"), los cuales se pueden agregar a objetos directamente, sin necesidad de crear una nueva clase, de manera similar a como se hace el los lenguajes basados en prototipos, como Self o JavaScript [7]. También tiene "mixins" por módulos, que nos permiten tener una buena alternativa a la herencia múltiple.

Ruby tiene una sintaxis sencilla y consistente; a continuación, un ejemplo de definición de método en Ruby:

```
def foo
```

```
  if @baz > 5
    puts ''bar''
  end
end
```

Es interpretado, tiene recolección de basura y su sistema de tipos es dinámico, de tal manera que nos permite desarrollar programas rápidamente.

Tiene facilidades para el procesamiento de texto, por ejemplo, expresiones regulares compatibles con Perl.

Algunas otras características de Ruby son tomadas de los lenguajes funcionales como las cerraduras (closures) y las continuaciones (continuations).

Ruby me permitió desarrollar el sistema de una manera extremadamente ágil.

6.0.2. librdf

Librdf es una biblioteca modular para manipulación de RDF desarrollada en C y pensada especialmente para ser usada fácilmente desde otros lenguajes. Tiene analizadores sintácticos y serializadores para leer y escribir RDF como RDF/XML, N-Triples y Turtle (un formato parecido a N-Triples). Tiene una variedad de métodos para almacenar las gráficas, y es muy portable.

Sobre los métodos de almacenaje de librdf

librdf cuenta con cinco métodos diferentes para almacenar el modelo de RDF:

hashes: Este método es el más probado y se ha usado con modelos con entre dos y tres millones de afirmaciones. El modelo puede ser almacenado en memoria o persistentemente usando Berkeley DB. Tiene un buen desempeño pues hace uso de índices.

memory: Este método guarda el modelo en memoria pero sin hacer uso de índices.

file: Este método crea un modelo en memoria a partir de un archivo, el cual puede posteriormente reescribirse al mismo.

mysql: Este método hace uso del manejador de bases de datos MySQL para guardar las tripletas en una base de datos relacional.

tstore: Con este método se puede acceder a repositorios de RDF "AKT Triplestore"[1], el cual es un repositorio especializado para RDF.

uri: Finalmente, este tipo de repositorio permite obtener la información RDF/XML de un URI, usarla en memoria, por el momento no permite escribir a la dirección URI.

Para este proyecto fue utilizado el método de "hashes" respaldado por Berkeley DB, esto nos permite alcanzar un buen desempeño y tener persistencia del modelo.

6.0.3. Ruby on Rails

Esta plataforma de desarrollo web hace uso de las características particulares de Ruby para proveer un ambiente dinámico y disfrutable.

Está compuesto por varios componentes, los cuales se pueden utilizar independientemente:

Active Record: Proporciona un mapeo a objetos de bases de datos relacionales.

Action Pack: Direcciona las peticiones a través de controladores, asignando un método a cada acción y permite que las vistas sean generadas por plantillas de Ruby.

Action Mailer: Proporciona una interfaz de servicios de correo electrónico fácil de probar sobre un servidor de correo.

Además de estos componentes, existen otras partes de la plataforma que los unen para facilitar el desarrollo.

Para este proyecto únicamente se utilizó "Action Pack", así como las herramientas adicionales que proporciona Ruby on Rails y el lenguaje de plantillas ERb, basado en Ruby.

Se procuró seguir un modelo de tres capas para separar la vista del diseño y el controlador.

En la parte del modelo, se elaboró una interfaz que permitió al controlador interactuar limpiamente con el mismo. Este modelo, a su vez, implementa la lógica necesaria para tratar con el modelo RDF. De esta manera, el resto de la aplicación queda aislada de los detalles de la implementación

[1] Para más información consultar `http://www.aktors.org/technologies/3store/` y [21]

del modelo, lo cual nos permitiría utilizar otro modelo de datos en caso de que esto sea más conveniente.

Para este proyecto se utilizaron, además, las facilidades que provee Ruby on Rails para desarrollar una aplicación de las que se conocen últimamente por el término de Web 2.0, estas son aplicaciones que, mediante el uso de JavaScript y la comunicación con el servidor sin pedir una nueva página (cuando sea posible y conveniente), proporcionan una experiencia más dinámica que las aplicaciones tradicionales de web. Ruby on Rails provee facilidades para que el desarrollo de estas aplicaciones sea más sencillo, haciendo uso de las bibliotecas prototype[2] y script.aculo.us[3].

6.0.4. Observaciones

En esta sección se analizarán las ventajas y problemas que trae consigo el desarrollo de aplicaciones utilizando de RDF.

En principio, es conveniente definir a qué tipo de aplicaciones de RDF me refiero. Vamos a considerar aplicaciones en las cuales se manejen grandes cantidades de datos, por lo que una parte importante del desarrollo se refiere a las consideraciones necesarias para capturar, guardar, recuperar, analizar e intercambiar la información.

Actualmente, en este tipo de aplicaciones, lo más común es pensar en las bases de datos relacionales. El modelo relacional de bases de datos está bien entendido, es ampliamente conocido y tiene bases teóricas sólidas. Además, existe una gran variedad de implementaciones de este modelo, o al menos de algo parecido a este modelo. Esto nos da una base sólida para desarrollar aplicaciones de todo tipo.

Gracias al gran parecido entre el modelo relacional y el de RDF[4], dichas aplicaciones pueden hacer uso de RDF sin hacer grandes modificaciones. Se puede realizar un mapeo del modelo relacional al modelo RDF de manera manual, adicionalmente, hay trabajos que buscan formalizar este proceso [3]. Así, es posible exportar e importar datos representados por medio de un modelo de RDF a una base de datos relacional.

Sin embargo, para este trabajo decidí usar el método menos convencional de usar RDF como único modelo de datos. Esto, como dijimos antes, tiene sus ventajas y desventajas.

Una ventaja evidente, es la extensibilidad que nos proporciona RDF. En este caso en particular, fue necesario extender el esquema RDF de DOAP,

[2]Ver `http://prototype.conio.net/`

[3]Ver `http://script.aculo.us/`

[4]Ver `http://www.w3.org/DesignIssues/RDB-RDF.html`

dado que en el esquema original no existían propiedades para indicar un paquete de un proyecto de software. Asimismo, se puede suponer que con una intención de generalidad, no se especificaron dominios para algunas propiedades, lo cual dificultaba obtener las propiedades de un proyecto, para así poder generar formas de manera automática. Estos objetivos se lograron mediante la creación de un nuevo esquema de RDF que especificaba las nuevas propiedades deseadas, como se puede ver en la Figura 6.1.

Esta extensibilidad permite compartir la parte común de varios proyectos de manera descordinana y permite la interoperabilidad entre estos sin ningún esfuerzo adicional.

Un aspecto importante a considerar en el desarrollo de sistemas basados en RDF es el hecho de que aún falta mucho trabajo por ser desarrollado en cuanto al desempeño de los mismos. Mientras que en una base de datos relacional podemos estar bastante seguros del tipo de desempeño que podemos esperar, no sucede lo mismo con los repositorios de tripletas de RDF, este es un campo en el que todavía queda mucho por hacer [29].

Una ventaja futura de RDF debería consistir en la posibilidad de realizar inferencia sobre modelos de datos. Esto nos permitiría hacer consultas más sofisticadas de las que son posibles de hacer con SQL, el lenguaje habitual para hacer consultas a bases de datos relacionales[5].

A los lenguajes de consulta de RDF todavía les queda camino por andar, en la implementación de RDF utilizada se pueden utilizar dos lenguajes de consulta: RDQL [26] y SPARQL [14]. RDQL es muy sencillo, no está muy alejado de las consultas por medio de patrones (explicadas más abajo). SPARQL es más sofisticado[6], pero aún se encuentra en la etapa de desarrollo, aunque ya existen implementaciones de borradores de la recomendación oficial.

Una dificultad actual para el trabajo con RDF, es la ausencia de herramientas para el mapeo a objetos, análogas a las que encontramos para las bases de datos relacionales. Esto nos obliga a trabajar directamente con lenguajes de consulta o bien la obtención de tripletas por medio de patrones sencillos (del tipo [sujeto, objeto, predicado], donde estamos interesados en las tuplas que cumplan con el patrón y el sujeto, objeto y predicado pueden ser nulos).

[5] Ver `http://www.ninebynine.org/SWAD-E/Scenario-HomeNetwork/HomeNetwork/HomeNetwork-330.htm`, `http://www.ilrt.bris.ac.uk/discovery/rdf-dev/purls/papers/QL98-queryservice-19981118/` y `http://xml.coverpages.org/RIL-20010510.html`

[6] Gracias a la opción de especificar partes opcionales del patrón de la gráfica, entre otras características

```
  <rdf:Description rdf:about="http://usefulinc.com/ns/doap#Project">
    <rdfs:label xml:lang="en">Minimal requirements</rdfs:label>
    <rdfs:comment xml:lang="en">
      Miminal requirements for a project description.
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
<owl:onProperty
        rdf:resource="http://usefulinc.com/ns/doap#homepage"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
         1
       </owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </rdf:Description>

  <rdf:Property rdf:about="http://usefulinc.com/ns/doap#Package">
    <rdfs:isDefinedBy rdf.resource="http://usefulinc.com/ns/doap#" />
    <rdfs:label xml:lang="en">Package</rdfs:label>
    <rdfs:comment xml:lang="en">
      A package for this project
    </rdfs:comment>
    <rdfs:domain rdf:resource="http://usefulinc.com/ns/doap#Project" />
  </rdf:Property>

  <rdf:Property rdf:about="http://usefulinc.com/ns/doap#name">
    <rdfs:isDefinedBy rdf:resource="http://usefulinc.com/ns/doap#" />
    <rdfs:domain rdf:resource="http://usefulinc.com/ns/doap#Project" />
  </rdf:Property>

  <rdf:Property rdf:about="http://usefulinc.com/ns/doap#DebianPackage">
    <rdfs:isDefinedBy rdf:resource="http://usefulinc.com/ns/doap#" />
    <rdfs:label xml:lang="en">Debian Package</rdfs:label>
    <rdfs:comment xml:lang="en">A package of a project</rdfs:comment>
    <rdfs:domain rdf:resource="http://usefulinc.com/ns/doap#Project" />
    <rdfs:subPropertyOf
     rdf:resource="http://http://usefulinc.com/ns/doap#Package" />
  </rdf:Property>
```

Figura 6.1: Fragmento del esquema de RDF adicional (fragmento)

Otra dificultad encontrada durante el desarrollo del sistema, es la generación automática de vistas para los datos. En muchos casos, existe información en el modelo de datos que puede no ser interesante para el usuario, por ejemplo identificadores usados a nivel interno. Asimismo, no existe una jerarquía en muchas de las gráficas de RDF, lo cual presenta problemas a la hora de organizar la información. En ocasiones, podemos querer presentar una etiqueta para un recurso, pero puede haber varias adecuadas o inclusive podemos preferir presentar URIref del mismo. En resumen, no es posible hacer un sistema que presente modelos RDF arbitrarios de manera útil para el usuario. Sin embargo, ya existe al menos un proyecto de investigación, *Haystack*, que incluye un lenguaje de descripción de interfaces en términos semánticos, basado en RDF y que es muy prometedor[22].

Un problema encontrado en el desarrollo, aunque no grave, fue la imposibilidad de usar los identificadores naturales de RDF, las URIrefs, como identificadores de elementos en HTML, esto hubiese simplificado enormemente el tránsito de información entre el navegador y el servidor, pero dadas las restricciones en la sintaxis de los identificadores de HTML, en particular, el no poder usar diagonales en los mismos [13], fue necesario mapear cada uno de estos identificadores a un hash, por lo que fue necesario mantener estado adicional, guardando la relación entre estos hashes y los URIrefs. Gracias a las virtudes de RDF esto pudo ser hecho directamente en el modelo, y gracias a que éste no tiene un esquema predeterminado fue más sencillo de lo que hubiese sido en un modelo relacional.

Una gran ventaja de usar RDF reside en el intercambio de información. Gracias a RDF podemos exportar la información en un formato estándar y, por medio de los esquemas de RDF, podemos ponernos de acuerdo en la información que vamos a enviar. En el sistema se pueden importar archivos de RDF, así como recibirlos para incrementar nuestro modelo.

Capítulo 7

Conclusiones

El desarrollo de aplicaciones basándonos exclusivamente en RDF puede ser una tarea instructiva, mas, en mi opinión, no es una opción viable actualmente, en general, para sistemas con necesidad de manejar grandes cantidades de información, debido a las limitaciones en cuanto a las herramientas para consultar la información y desarrollar aplicaciones como se mencionó en la sección 6.0.4.

En el caso de aplicaciones específicas, donde la intención sea claramente hacer y guardar afirmaciones muy específicas sobre recursos de un dominio muy bien definido y en donde las gráficas que representan la información sean relativamente sencillas, como por ejemplo aplicaciones del tipo del.icio.us (un sistema para almacenar ligas a páginas y asignar etiquetas a estas) o de flickr.com (que asigna etiquetas a fotografías) tal vez sea factible desarrollar sistemas que se basen exclusivamente en modelos RDF, sin embargo, se deben tener en cuenta las consideraciones de escalabilidad de los mismos, aunque en modelos de datos sencillos es de esperarse que las consultas sean, a su vez, sencillas, lo cual no debería de representar un problema para los repositorios actuales.

En general, por el nivel en el que se encuentran los lenguajes de consulta para RDF y las herramientas que existen para trabajar con este, mi impresión es que por el momento el mejor uso que se le puede dar a RDF es para representar modelos de datos simples y, sobre todo, en los que las consultas sean sencillas, o bien, exclusivamente para el intercambio de información. Esto debido a que, por lo menos hasta que haya una capa de inferencia bien definida para el Web Semántico, será difícil hacer consultas complicadas a repositorios de afirmaciones, del tipo de las que se hacen rutinariamente en bases de datos relacionales (y aún más poderosas). Esto debido a que,

aunque actualmente existen varios lenguajes de extracción de datos para RDF, en varios estados de madurez, de muy distintas filosofías y de diferente poder, todavía no hay un claro candidato a ser un lenguaje estándar [20] además de SPARQL, y este todavía se encuentra en la etapa de borrador.

Tampoco queda claro que el desarrollo de aplicaciones basadas únicamente en el modelo RDF sea la mejor opción, aún cuando la infraestructura de la Web Semántica esté más madura. Es muy pronto para decirlo, pero probablemente el mejor camino sea un punto medio entre el uso del modelo de RDF y el modelo relacional. Por el momento, ciertamente, las herramientas para trabajar con bases de datos relacionales son mucho más maduras que las existentes para el modelo de RDF.

Existen proyectos para el desarrollo de aplicaciones web basadas en RDF (como por ejemplo SWAP, de Tim Bernes-Lee[1]), sin embargo no parecen haber ganado mucha tracción en la comunidad web, probablemente porque el conocimiento y uso de RDF todavía no está muy difundido, sobre todo comparado con el modelo relacional.

Sin embargo, no debemos olvidar que hoy en día RDF es muy útil para el intercambio de información, gracias a sus fundamentos teóricos sólidos, representaciones estándar y extensibilidad, es una herramienta ideal para este fin y en efecto se usa ampliamente (ver por ejemplo `http://web.resource.org/rss/1.0/` y `http://ruby.codezoo.com/about/doap.csp`). Y me gustaría ver que mas proyectos, como freshmeat.net o gnomefiles.net hiciesen uso de este formato.

Entre los puntos más importantes en el desarrollo futuro de RDF y el Web Semántico podemos considerar:

- Desarrollo de mecanismos de inferencia estándar que permitan explotar el potencial de información de los modelos RDF.
- Mecanismos de mapeo del modelo RDF a objetos en lenguajes de programación orientados a objetos, de manera análoga a los sistemas existentes para mapear el modelo relacional a objetos.
- Asimismo, como se mencionó en la sección 6.0.4, sería interesante trabajar sobre los lenguajes de descripción de interfaces basado en RDF.
- Desarrollo de plataformas para aplicaciones web que hagan uso de RDF.

[1] ver `http://www.w3.org/2000/10/swap/`

- Desarrollo de lenguajes de consulta para modelos de RDF. Es importante, sobre todo, la definición de un lenguaje estándar lo suficientemente poderoso.
- Investigar el desempeño de los repositorios de tripletas de RDF.
- Creación de más herramientas para trabajar con RDF, así como la maduración de las existentes.

Bibliografía

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2005.

[2] Tim Bernes-Lee. Cool uris don't change. 1998. URL `http://www.w3.org/Provider/Style/URI.html`.

[3] Christian Bizer and Andy Seaborne. D2rq treating non-rdf databases as virtual rdf graphs. 2004.

[4] Tim Bray et al. Extensible markup language (xml) 1.0 (third edition). 2004. URL `http://w3c.org/TR/2004/REC-xml-20040204/`.

[5] Dan Brickley and Libby Miller. Foaf vocabulary specification. 2004. URL `http://xmlns.com/foaf/0.1/`.

[6] The Unicode Consortium. The unicode standard, version 3. 2000. URL `http://www.unicode.org/unicode/standard/versions/`.

[7] Christophe Dony, Jacques Malenfant, and Pierre Cointe. Prototype-based languages: from a new taxonomy to constructive proposals and their validation. 2002.

[8] Edd Dumbill. Describe open source projects with xml, part 1. 2004. URL `http://www-106.ibm.com/developerworks/xml/library/x-osproj.html`.

[9] Edd Dumbill. Describe open source projects with xml, part 2. 2004. URL `http://www-106.ibm.com/developerworks/xml/library/x-osproj2`.

[10] Edd Dumbill. Describe open source projects with xml, part 3. 2004. URL `http://www-106.ibm.com/developerworks/xml/library/x-osproj3`.

[11] Edd Dumbill. Describe open source projects with xml, part 4. 2004. URL http://www-106.ibm.com/developerworks/xml/library/x-osproj4.

[12] Dave Beckett (editor) and Brian McBride (series editor). Rdf/xml syntax specification (revised). 2004. URL http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[13] Dave Raggett (editor), Arnaud Le Hors (editor), and Ian Jacobs (editor). Html 4.01 specification. 1999. URL http://www.w3.org/TR/html4/.

[14] Eric Prud'hommeaux (editor) and Andy Seaborne (editor). Sparql query language for rdf. 2005. URL http://www.w3.org/TR/rdf-sparql-query/.

[15] Graham Klyne (editor), Jeremy J. Carroll (editor), and Brian McBride (series editor). Resource description framework (rdf): Concepts and abstract syntax. 2004. URL http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[16] Henry S. Thompson (editor), David Beech (editor), Murray Maloney (editor), and Noah Mendelsohn (editor). Xml schema part 1: Structures second edition. 2004. URL http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[17] Jan Grant (editor), Dave Beckett (editor), and Brian McBride (series editor). Rdf test cases. 2004. URL http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/.

[18] Paul V. Biron (editor) and Ashok Malhotra (editor). Xml schema part 2: Datatypes second edition. 2004. URL http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[19] Bdale Garbee et al. A brief history of debian. 2004. URL http://www.debian.org/doc/manuals/project-history/.

[20] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. 2004.

[21] Stephen Harris and Nicholas Gibbins. 3store: Efficient bulk rdf storage. 2003.

[22] David Huynh, David Karger, and Dennis Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. 2002.

[23] D.L. Mcguinness, R. Fikes, J. Hendler, and L. A. Stein. Daml+oil: an ontology language for the semantic web. 2002.

[24] Peter F. Patel-Schneider. What is owl (and why should i give a hoot)? 2004.

[25] Shelley Powers. *Practical RDF.* 2003.

[26] Andy Seaborne. Rdql - a query language for rdf. 2005. URL `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`.

[27] Dave Thomas and David Heinemeier Hansson. *Agile Web Development with Rails.* 2005.

[28] Wolfgang Wahlster, Henry Lieberman, James Hendler, and Dieter Fensel (editor). *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* 2002.

[29] Kevin Wilkinson, Craig Sayers, Harumi Kuno, and Dave Reynolds. Efficient rdf storage and retrieval in jena2. 2004.