Añadiendo Información Semántica a Paquetes de Software

Jorge Jesús Santos Fierro

14 junio de 2005

El Problema

- Miles de paquetes en las distribuciones de Linux.
- Es difícil encontrar lo que se busca.
- En el mejor de los casos están clasificados por categorías amplias (Desarrollo de software, Juegos, Gnome, etc.)
- Cuando una categoría tiene miles de paquetes, deja de ser útil.

Subcategorías

- Podemos organizar los paquetes de manera jerárquica pero...
- ¿Qué hacemos cuando un paquete puede pertenecer a dos o más categorías?
- Podemos entonces clasificarlos por medio de etiquetas, de esta manera un paquete puede pertenecer a más de una “categoría”.

Debian Package Tags

- Usa clasificación “facetada”.
- Les asigna etiquetas a los paquetes, estas etiquetas a su vez pertenecen a algún punto de vista desde el cuál podemos ver el paquete.

Ventajas de Debtags

- No limita la clasificación a un formato jerárquico.
- Esto da mayor flexibilidad de clasificación.
- Las etiquetas disponibles y sus asociaciones con paquetes se pueden actualizar independientemente de los paquetes.
- Se puede actualizar la información remotamente.

Desventajas de Debtags

- Usa un formato ad-hoc para guardar sus datos.
- Requiere que las etiquetas y sus asociaciones se definan de manera central.
- Requiere de la captura de mucha información para ser útil.

Propuesta

- Se propone la creación de un sistema parecido a Debtags pero basado en RDF.
- Se procura hacerlo lo más general posible para que se pueda usar en más de una distribución.
- Al ser basado en RDF y hacerlo general la información se puede usar para otros fines (e.g. genearción automática de distribuciones a la medida).

Propuesta (cont.)

- Se usan las tecnologías existentes para mejorar la interoperabilidad y facilitar el desarrollo.
- Se usa la información existente para tener algo útil sin esperar a que alguien se siente horas y horas capturando información.

Tecnología

- Se hace uso de RDF, RDF Schema y OWL para establecer la información que tendremos disponible, así como para expresar dicha información.
- En particular, se usa el esquema de RDF DOAP, el cual establece información que es deseable tener sobre proyectos de software libre.

Información

- Se usa información de freshmeat.net, el directorio de software libre más grande del mundo.
- Desafortunadamente, la información que exportan no está en formato RDF.
- Afortunadamente, está en XML, y es relativamente sencillo generar archivos de RDF (que correspondan al esquema DOAP) a partir de estos archivos XML.

El esquema DOAP

- Es un esquema que define la información que es deseable tener sobre proyectos de software libre:
  - Nombre del proyecto.
  - Página principal.
  - Bug database.
  - Reposito de software.
  - Colaboradores.
  - Etc.

Lo que le falta a DOAP

- Al esquema DOAP le falta, para nuestros fines. La información correspondiente al paquete de software que contiene el proyecto de software libre.
- Para esto creé, gracias a la flexibilidad que otorga RDFS un esquema complementario que define el formato de esta información.

Información sobre el paquete

```
<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:label xml:lang="es">Paquete</rdfs:label>
        <rdfs:comment xml:lang="en">A package for this project</rdfs:comment>
        <rdfs:comment xml:lang="es">Un paquete que encapsula este
proyecto.</rdfs:comment>
        <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:label xml:lang="es">Paquete Debian</rdfs:label>
        <rdfs:comment xml:lang="en"></rdfs:comment>
        <rdfs:comment xml:lang="es">El paquete de Debian que encapsula este
proyecto.</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>

<rdf:Property
rdf:about="http://usefulinc.com/ns/doap#DebianPackageDistribution">
        <rdfs:isDefinedBy rdf:resource="http://usefulinc.com/ns/doap#" />
        <rdfs:label xml:lang="en">The Debian Package's Distribution</rdfs:label>
        <rdfs:label xml:lang="es">La distribuciónn del Paquete
Debian</rdfs:label>
        <rdfs:comment xml:lang="en"></rdfs:comment>
        <rdfs:comment xml:lang="es">La distribución del paquete de
        Debian que encapsula este proyecto.</rdfs:comment>
```

Juntándolo Todo

```
  <doap:Project>
    <doap:name>123mp3</doap:name>
    <doap:shortdesc>Converts CD tracks to MP3
    files.</doap:shortdesc>
    <doap:description>123mp3 is a frontend to BladeEnc and
    cdparanoia that produces MP3s from selected tracks on a CD in
    one automated process. Support for CDDB is also
    available.</doap:description>
    <doap:homepage
rdf:resource="http://yallara.cs.rmit.edu.au/~bnuske/linux/linux.html" />
    <doap:category rdf:resource="http://software.freshmeat.net/browse/15/" />
    <doap:category rdf:resource="http://software.freshmeat.net/browse/113/" />
    <doap:category rdf:resource="http://software.freshmeat.net/browse/118/" />
    <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/164/" />
    <doap:category rdf:resource="http://software.freshmeat.net/browse/10/" />
    <doap:DebianPackage
rdf:resource="http://packages.debian.org/unstable/sound/emacs21.html" />
  </doap:Project>
```

Clasificación

- Nótese cómo en la diapositiva anterior tenemos varios statements del tipo:

  project description http://foo.bar/1.

- Esto es lo que nos permite buscar paquetes de software por medio de las categorías de freshmeat.net
- Al ser URLs podemos conservar una relación entre las diferentes categorías.

Implementación

- La información se puede navegar y modificar a través de una interfaz web.
- Se usa un repositorio de RDF para guardar la información y se usó el “framework” Ruby on Rails para crear las interfaces que permiten consultar y modificar dicha base de datos.
- La información inicial se cargó a partir de archivos de RDF, generados de freshmeat.net.

Detalles de la implementación

- MVC
- Se programó un modelo que extrae la información de un repositorio de RDF.
- Se generaron las vistas de los recursos más o menos automáticamente (ver Haystack).
-

Generación automática de esquemas

- No parece trivial hacer esto en general, ¿cómo establecemos las llaves primarias?
- ¿Cómo afecta la normalidad de la base de datos la manera en que creamos las tablas?

Implementación de una fuente de datos RDF en Rails

- Necesitamos un repositorio de RDF adecuado. Opciones:
  - 3store
  - rdfDB
  - Otras.
- Rails está orientado a aplicaciones sobre bases de datos. Está por verse qué tanto se puede reutilizar del mismo para trabajar con RDF (al menos en cuanto a la persistencia).

Código de Rails

```
def scaffold(model_id, options = {})
  validate_options([ :class_name, :suffix ], options.keys)

  singular_name = model_id.id2name
  class_name    = options[:class_name] || Inflector.camelize(singular_name)
  plural_name   = Inflector.pluralize(singular_name)
  suffix        = options[:suffix] ? "_#{singular_name}" : ""

  unless options[:suffix]
    module_eval <<-"end_eval", __FILE__, __LINE__
      def index
        list
      end
    end_eval
  end

  module_eval <<-"end_eval", __FILE__, __LINE__
    def list#{suffix}
      @#{singular_name}_pages, @#{plural_name} = paginate :#{singular_name}, :per_page =>
10

      render#{suffix}_scaffold "list#{suffix}"
    end

    def show#{suffix}
      @#{singular_name} = #{class_name}.find(@params["id"])
      render#{suffix}_scaffold
    end
```