

Dirigiendo Google Calendar

DÍA D, HORA H

Con el protocolo GData (el API de datos de Google) tendremos un acceso completo a los datos de las aplicaciones diseñadas por Google, incluido Google Calendar, de una manera unificada y sencilla.

POR ALBERTO PLANAS

El API de Google para la integración de resultados de búsquedas que hemos tenido la oportunidad de conocer hace uso del protocolo SOAP para el acceso a servicios web[1]. SOAP es un protocolo complejo que requiere de la instalación y uso de librerías de terceros para lograr simplificar la creación de *stubs*, o funciones proxy o intermedias, a partir del lenguaje de descripción de servicios web (WSDL). Con estas funciones, creadas de manera automática, lograremos realizar llamadas de acceso a objetos localizados en servidores remotos. También se encargan de codificar los mensajes, realizar la conexión por HTTP (o cualquier otro protocolo de transporte) con el servidor y controlar los errores de transmisión. Además los mensajes SOAP, en formato XML, son algo redundantes y recargados. Si programamos en lenguaje Python podemos hacer uso del interesante PyGoogle[2] de Mark Pilgrim, autor también del excelente libro *Dive Into Python*[3]. Esta librería simplifica el acceso al API SOAP de Google, proporcionando interfaces para la realización de búsquedas, obtención de la versión cacheada de un resultado e, incluso, la consulta a las sugerencias de escritura de parámetros de búsqueda. Esta librería necesita de otra para la generación de mensajes SOAP (SOAPpy). Como vemos, no es directo acceder a la funcionalidad de Google proporcionada desde el API de basado en SOAP. Necesitamos poner dos capas de funciones, objetos y métodos entre nuestra aplicación y los servicios ofrecidos por Google.

Para simplificar este escenario, Google ha desarrollado el API de acceso a datos de Google. GData hace uso del protocolo HTTP para la transmisión de datos de sindicación en formato XML. Con este protocolo podemos enviar y recibir documentos en formato Atom 1.0 y RSS 2.0. Además extiende el protocolo de publicación Atom (APP) para la realización de consultas parametrizadas. A diferencia del protocolo basado en SOAP, podremos consul-

tar, crear, actualizar y eliminar diversos elementos de datos con los comandos simples de HTTP: GET, POST, PUT, DELETE. Esta manera de trabajar con servicios web no es nueva, ya que sigue los principios básicos de la arquitectura REST (ver Tabla GData y REST). Además este sistema está siendo aplicado con éxito en la creación de otros servicios web de empresas tales como Amazon, eBay, Yahoo! y más recientemente en Flickr y Youtube. Una ventaja frente a la aproximación SOAP, además de su simplicidad e inmediatez, es su baja latencia. Al disminuir la complejidad del sistema y eliminar datos redundantes logramos disminuir el tiempo entre la petición del servicio y la respuesta generada. Google utiliza este mismo API para acceder a múltiples servicios[4], entre los que se cuentan Blogger, Google Calendar y Google Base-Data.

En este artículo nos centraremos en Google Calendar[5]. Aunque es posible acceder a toda la funcionalidad desde cualquier lenguaje de programación de manera directa, los desarrolladores de Google proporcionan librerías en Java, .NET, y dentro del proyecto Zend Framework encontramos una versión de la librería en PHP. No hay ninguna versión para Python, así que vamos a plantear a lo largo de este artículo las bases para un futuro proyecto que denominaremos PyGData[6].

Enigma

Si tenemos una cuenta en Google Mail, ya podemos acceder al calendario desde el pequeño menú de la esquina superior izquierda del interfaz Gmail. También podemos entrar directamente visitando la web del proyecto[7]. Dar de alta un evento, asignar una visibilidad (evento público o evento privado), añadir un comentario o asignar una periodicidad y una alarma de avisos al evento son cosas sencillas de hacer desde el mismo interfaz web de la herramienta. Otras cosas que podemos hacer es la de realizar búsqu-

das de texto libre por todos los eventos del calendario y recolocarlos en otro día o cambiar la duración del evento mediante el uso del ratón, como buena aplicación AJAX que es.

Pero lo interesante no es sólo tener este interfaz de acceso, sino poder comprobar que podemos acceder a toda esta funcionalidad desde la línea de comandos. Para ello necesitamos previamente encontrar una URL que identifique de manera única a nuestro calendario. Si pulsamos sobre la flechita asociada al calendario en la lista de calendarios situada en la columna de la izquierda, veremos un desplegable que nos permite realizar acciones de configuración, creación de eventos y cambio de colores de los eventos. Debemos seleccionar la entrada que pone "Configuración del calendario". En el formulario que aparece ahora pulsamos sobre el botón naranja etiquetado 'XML' de la sección 'Dirección privada' al final del mismo. Ahora aparecerá una URL que debemos copiar. Esta URL tiene el siguiente formato:

```
http://www.google.com/calendar/?
feeds/<I>email<I>/private-?
<I>magicCookie<I>/basic
```

En el campo *email* aparece nuestra cuenta Gmail y en *magicCookie* una cadena alfanumérica secreta que podemos regenerar (en el caso de que quede descubierta por descuido) desde el enlace 'Restablecer URL privada' situado cerca del botón anterior. Con esta URL podremos acceder, en modo de solo lectura, a la lista de eventos del calendario con un simple *wget URL*. Si lo hacemos obtendremos algo parecido a lo mostrado en el Listado 1. Es un documento XML en formato Atom[8] que contiene una entrada (tag *entry*) por cada evento del repositorio. Además se indica el identificador del calendario (primer campo *id* no asociado a ninguna entrada), el dueño del mismo (¿adivinan qué tag tiene esa información?) y, para cada entrada de calendario, el

título y descripción del evento. Por último, si cambiamos en la URL de acceso anterior la palabra *basic* por *full* y volvemos a realizar una petición con *wget* obtendremos una versión extendida del mismo documento de listado de eventos.. En ella aparecerán nuevos campos que hasta ahora han quedado ocultos, como por ejemplo la referencia (otra URL) a los comentarios asociados a cada uno de los eventos.

Esta consulta se puede parametrizar. Así, si queremos obtener una lista de eventos para un rango de tiempo dado podemos añadir dos parámetros a la URL: *start-min* y *start-max*. Por ejemplo, para saber los eventos que me esperan durante la semana del 06/11/2006, tengo que preparar una consulta del siguiente tipo:

```
http://www.google.com/
calendar/feeds/<I>email<I>/
private-<I>magicCookie<I>/
basic?start-min=2006-11-
```

```
06T00:00:00&start-max=
2006-11-10T23:59:59
```

Las fechas y horas siguen el formato ISO. El resultado de la consulta mantendrá la misma estructura que la anterior. En ambas consultas el resultado viene ordenado por la fecha de actualización, de modo que si queremos ordenar por la fecha de suceso del evento (algo natural en la realización de un informe) deberemos añadir otro parámetro más al final de la URL: *orderby=starttime*. Siendo Google una empresa que ha creado un buscador (vale, 'el buscador') no puede faltar la funcionalidad de búsqueda de eventos por palabras. Los parámetros de la consulta los indicamos con el parámetro *q=palabra1+palabra2*. De esta manera se realizarán búsquedas con esas dos palabras por todo el texto del evento y descargará un listado con aquéllos que contengan ambas coincidencias. Si usamos las dobles comillas (") indicaremos una búsqueda exacta, y si ponemos por delante de la palabra un guión (-) excluirémos los eventos que con-

tenham la palabra prefijada. Podemos establecer un limitador de coincidencias descargadas con el parámetro *max-results*, que por defecto vale 25. Para poder ir paginando las consultas deberemos hacer uso de *start-index*.

Como hemos podido ver, el procedimiento para acceder a Google Calendar es sencillo. Con el protocolo GData es suficiente con enviar el comando GET de HTTP a una URL que representa un repositorio o fuente de datos. Así obtendremos un documento Atom que representa a este repositorio de datos o a un recurso determinado (p.ej, un evento), y eso es todo (o casi) , ya que es en esta sencillez donde radica el principal atractivo de la arquitectura REST. Pero es verdad que no podemos hacer muchas más cosas con nuestro calendario si antes no nos autentificamos. La URL que hemos localizado al principio del artículo no es suficiente para poder crear nuevos eventos, modificar o eliminar los ya existentes. Deberemos realizar dos pasos para autentificarnos. El primer paso es transformar la URL de acceso: deberemos eliminar el resu-

Listado 1: Extracto XML

```
01 <?xml version='1.0'
encoding='utf-8'?>
02 <feed xmlns=
'http://www.w3.org/2005/Atom'
03 xmlns:openSearch=
'http://a9.com/-/spec/opensearch
rss/1.0/'
04 xmlns:gd=
'http://schemas.google.com/g/200
5'
05 xmlns:gCal=
'http://schemas.google.com/gCal/
2005'>
06 <id>
07 http://www.google.com/
calendar/feeds/aplanas@gmail.com
/private-<I>magicCookie<I>/basic
</id>
08 <updated>2006-11-07T12:05:
21.000Z</updated>
09 <title type='text'>Alberto
Planas</title>
10 <subtitle type='text'> Alberto
Planas</subtitle>
11 <link rel='http://schemas.
google.com/g/2005#feed'
12 type='application/atom+xml'
13 href='http://www.google.com/
calendar/feeds/aplanas@gmail.com
/private-<I>magicCookie<I>/basic
'>
14 </link>
15 <link rel='self'
type='application/atom+xml'
16 href='http://www.google.com/
calendar/feeds/aplanas@gmail.com
/private-<I>magicCookie<I>/basic
?max-results=25'>
17 </link>
18 <author>
19 <name>Alberto Planas</name>
20 <email>aplanas@gmail.com</email>
21 </author>
22 <generator version='1.0'
uri='http://www.google.com/calen
dar'>
23 Google Calendar</generator>
24 <openSearch:startIndex>1
</openSearch:startIndex>
25 <openSearch:itemsPerPage>25
</openSearch:itemsPerPage>
26 <gCal:timezone value='Europe
/Madrid'></gCal:timezone>
27 <entry>
28 <id>
29 http://www.google.com/
calendar/feeds/aplanas@gmail.com
/private-<I>magicCookie<I>/basic
/<I>entryID<I></id>
30 <published>2006-11-07T06:
13:31.000Z</published>
31 <updated>2006-11-07T06:
13:37.000Z</updated>
32 <category scheme='http://sch
emas.google.com/g/2005#kind'
33 term='http://schemas.google.
com/g/2005#event'></category>
34 <title type='text'>Reunión con
empresa Exp.#299</title>
35 <summary type='html'>Cuándo:
del mar 7 de nov de 2006 11:00
36 al 14:30& CET <br><br>Dónde:
Antequera</summary>
37 <content type='text'>Cuándo:
del mar 7 de nov de 2006 11:00
38 al 14:30& CET <br><br>Dónde:
Antequera <br><br>Descripción del
39 evento: Reunión con la empresa
con Exp. #299 para el proyecto
40 N.P.</content>
41 <link rel='alternate'
type='text/html'
42 href='http://www.google.com/
calendar/event?eid=<I>eventID<I>
'>
43 title='alternate'></link>
44 <link rel='self'
type='application/atom+xml'
45 href='http://www.google.com/
calendar/feeds/aplanas@gmail.com
/private-<I>magicCookie<I>/basic
/<I>entryID<I>'>
46 </link>
47 <author>
48 <name>Alberto Planas</name>
49 <email>aplanas@gmail.com
</email>
50 </author>
51 <gCal:sendEventNotifications
value='false'>
52 </gCal:sendEventNotifications>
53 </entry>
54 </feed>
```

men *magicCookie* de la URL hasta dejarlo como sigue:

```
<U>http://www.google.com/?
calendar/feeds/<I>email<I>/?
private/full<U>
```

También vale si en la URL anterior usamos la versión *basic* en vez de la *full*. El segundo paso es algo más complejo: tenemos que autenticar la sesión frente a Google. Hay dos procedimientos para hacer esto dependiendo de si la aplicación que se va a autenticar es de escritorio o un desarrollo Web. En la primera opción, la aplicación debe consultar directamente del usuario la cuenta Gmail y la contraseña de la cuenta. Este es un método que puede ser seguro para aplicaciones de escritorio, pero nefasto si la aplicación es vía web (¿quién piensa poner su contraseña de correo en un formulario web de otra aplicación diferente a Gmail?). La segunda opción de autenticación está especialmente diseñada para aplicaciones web. En ella no introducimos la contraseña en ningún formulario de la aplicación web, sino que esta aplicación nos redirige a un formulario de Google donde nosotros procedemos a identificarnos. Si la identificación es correcta seremos redirigidos a una URL de la aplicación web solicitante. Vamos a usar la primera técnica, denominada *ClientLogin API*[9]. Para ello tendremos que enviar el comando HTTP POST con una serie de parámetros de formulario (que corresponden con el *Content-Type* de tipo *application/x-www-form-urlencoded* a la siguiente URL:

```
https://www.google.com/?
accounts/<I>ClientLogin<I>
```

Los parámetros que debemos pasar por POST son:

- Email*: cuenta Gmail del usuario.
- Passwd*: contraseña de la cuenta.
- source*: identificador de la aplicación.
- service*: texto 'cl' para identificar el servicio de calendario.

El campo *source* debe seguir el siguiente patrón: *nombreDeLaCompañía-nombreDeLaAplicación-versión*. Si hay algún error en el procedimiento de autenticación el sistema devolverá el código HTTP 403 para indicar que el acceso queda denegado. El usar códigos de error para devolver el resultado de una operación es algo que veremos con frecuencia en el protocolo GData. Se hace un uso intensivo de códigos que representan una redirección, una prohibición, o cuando todo sale correctamente (incluido el proceso de autenti-

ficación), un código HTTP 200 OK. En este caso, además del código 200, devolverá varias líneas de texto. Cada línea es del tipo 'clave' = 'valor'. Necesitamos obtener el valor asociado a la clave *Auth* puesto que es una cadena de texto que deberemos entregar como testigo de una correcta identificación en aquellas operaciones que cambien el estado del calendario, y que requieran de una autorización. En el Listado 2 podemos encontrar un procedimiento para extraer esta variable dentro del método *client_login()* de la clase *GData*.

¡Fuego!

Una vez autenticados Google Calendar nos dejará crear eventos de calendario mediante un POST a la URL transformada, o si sólo tenemos un calendario también podemos hacer uso de la siguiente URL:

```
http://www.google.com/?
calendar/feeds/default/?
private/full
```

Deberemos completar este POST añadiendo una cabecera HTTP de autorización, formada de la siguiente manera:

```
Authorization: GoogleLogin?
auth=<I>Auth<I>
```

Es ahí donde debemos colocar el token que se nos entregó durante el proceso de autenticación anterior. Tenemos que completar la cabecera estableciendo un *Content-Type* de tipo *application/atom+xml*, puesto que lo que vamos a transmitir es un documento XML Atom. Tras la realización del POST, Google Calendar deberá retornar un código HTTP 302 de redirección. Esto es así porque aprovecha para cambiar la URL que establecimos en el POST, añadiendo un parámetro adicional. Se trata del identificador de sesión *gsessionid*. Así pues deberemos relanzar otra vez el POST, con las mismas cabeceras HTTP, el mismo documento XML Atom, pero ahora a la nueva URL completada durante el proceso de redirección. Si finalmente el proceso de creación de eventos termina adecuadamente, recibiremos un código HTTP 201 para indicar su creación y una versión del documento que enviamos al que se le han añadido algunos tags nuevos.

Hay varios tipos de documentos XML Atom que podemos enviar por medio de GData [10]. El principal tipo que debemos usar para ponernos en contacto con Google Calendar es el tipo evento. Un ejemplo de un evento lo podemos ver en el Listado 3. Cada docu-

mento Atom XML tiene una raíz de tipo *entry*, en él se declaran los distintos espacios de nombre (xmlns), tanto de Atom como de GData, de esta manera podemos utilizar nuevos tags extendidos que no pertenecen a la especificación Atom. Estos nuevos tags exclusivos de GData son los del tipo *<gd>*. Para indicar que estamos ante una entrada de tipo evento hacemos uso del tag *category* indicando en el atributo *term* la URL *http://schemas.google.com/g/2005#event*. El resto de los tags son más o menos autodescriptivos: con *title* establecemos el texto publicado en el evento que aparece en la vista de calendario, en el *content* hacemos una descripción más detallada del evento. El tag *author* contiene el nombre y email del creador del evento (que puede ser diferente al dueño del calendario). Si el tag *<gd:transparency>* tiene el valor *http://schemas.google.com/g/2005#event*.

opaque estaremos indicando que el evento va a consumir tiempo en el calendario, va a ser un tiempo marcado como 'ocupado'. Con *<gd:eventStatus>* podemos indicar si el evento ha sido confirmado o no, o si ha sido cancelado. Los valores de visibilidad (evento privado / evento público) se establecen por medio del tag *<gd:visibility>*. Podemos indicar el lugar donde se va a producir el evento, dónde se espera poder encontrar aparcamiento (curioso, ¿no?) y una localización alternativa para la realización del evento. Para ello usamos el tag *<gd:where>* poniendo lo que queremos describir en el parámetro *value* y un texto libre en el campo *valueString*. Para indicar varios tipos de localizaciones las pondremos una detrás de la otra. Y, naturalmente, podemos establecer una fecha y duración del evento usando *<gd:when>*. Hay más tags que describen otros componentes que constituyen un evento, como la repetición del evento, un enlace a comentarios asociados o el conjunto de personas relacionadas con el mismo.

Este mismo procedimiento de autenticación puede ser usado para listar los calendarios disponibles por el usuario (podemos crear y eliminar calendarios desde la interfaz web). Tenemos que enviar el comando HTTP GET a la URL siguiente (sustituyendo el campo *email* por la cuenta Gmail del usuario):

```
https://www.google.com/?
calendar/feeds/<I>email<I>
```

Naturalmente hay que poner la misma cabecera de autenticación para lograr pasar el parámetro *Auth*. Un ejemplo de todo el proceso lo tenemos en el método *list_calendars* del Listado 2.

Nueva Orden

Los eventos conseguidos tras una consulta contienen más campos de los que pusimos en nuestro documento XML Atom. El campo *id*

contiene una URL que referencia de manera única al evento. Además, hay varios campos de tipo *link*, cada uno con un parámetro *rel* diferente. Si el valor de *rel* es *self* veremos que la URL asociada al parámetro *href* del tag de

enlace coincide exactamente con el identificador del evento. Podemos usar esta dirección para obtener una copia del objeto por medio del, ya explicado, comando HTTP GET. El enlace que tiene un valor para *rel* = *'alternate'*

Listado 2:PyGData.py

```

001 #! /usr/bin/env python
002 # -*- coding: UTF-8 -*-
003
004 import urllib
005 import urllib2
006
007 class LoginError(Exception):
008     pass
009
010 class
011     GDataRedirectHandler(urllib2.HTTP
012     RedirectHandler):
013     """
014     Recuperamos los datos enviados
015     antes de la redirección y los
016     volvemos
017     a enviar incluyendo esta vez el
018     parámetro gsessionid. El código,
019     por
020     tanto, es idéntico al de la
021     clase HTTPRedirectHandler, excepto
022     que
023     esta vez sí retomo los datos del
024     POST.
025     """
026     def redirect_request(self, req,
027     fp, code, msg, headers, newurl):
028         print req.get_data()
029         m = req.get_method()
030         if (code in (301, 302, 303,
031         307) and m in ("GET", "HEAD")
032         or code in (301, 302, 303) and
033         m == "POST"):
034             return urllib2.Request(newurl,
035             headers=req.headers,
036             origin_req_host=req.get_origin_re
037             q_host(),
038             unverifiable=True,
039             data=req.get_data())
040         else:
041             raise
042             HTTPError(req.get_full_url(),
043             code, msg, headers, fp)
044
045
046
047
048
049
050
051
052
053
054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
100
101
102
103
104
105
106
107
108
109
110
111
112
113

```

Listado 3: Evento Atom XML

```

01 <entry
02   xmlns='http://www.w3.org/2005/
03   Atom'
04   xmlns:gd='http://schemas.google.
05   com/g/2005#'
06   <category
07     scheme='http://schemas.google.
08     com/g/2005#kind'
09     term='http://schemas.google.co
10     m/g/2005#event'></category>
11   <title type='text'>Comida
12   LiMa</title>
13   <content type='text'>Lasaña
14   con los socios de
15   LiMa</content>
16   <author>
17     <name>aplanas</name>
18     <email>aplanas@gmail.com</emai
19     l>
20     </author>
21     <gd:transparency
22       value='http://schemas.google.c
23       om/g/2005#event.opaque' />
24     <gd:eventStatus
25       value='http://schemas.google.c
26       om/g/2005#event.tentative' />
27     <gd:where
28       value='http://schemas.google.c
29       om/g/2005#event'
30     valueString='La Caprichosa,
31     Campanillas' />
32     <gd:where
33       value='http://schemas.google.c
34       om/g/2005#event.parking'
35     valueString='Campanillas'
36     />
37     <gd:when
38       startTime='2006-11-10T14:30:00
39       .000Z'
40       endTime='2006-11-10T15:30:00.0
41       00Z'
42     valueString='Cuando
43     siempre' />
44   </entry>

```

contiene una URL a un documento HTML normal y corriente, que muestra el contenido de los distintos campos de este evento. Podemos comprobarlo copiando y pegando esa dirección en nuestro navegador. Cuando *rel = 'edit'*, la URL asociada tiene una utilidad muy interesante. Esta URL es igual a la del enlace de tipo *self* pero que tiene añadido un número largo al final de la misma. Este número es un identificador de versión del evento, y se utiliza para lograr un acceso concurrente al mismo desde varias aplicaciones sin que exista riesgo de colisiones entre las mismas. Con esta URL extendida podremos realizar las operaciones de borrado y actualización del evento. Por tanto, necesitaremos consultar este valor dentro del documento XML de un evento dado antes de proceder a su eliminación o modificación.

Eliminar un evento es sencillo. Es suficiente con enviar el comando HTTP DELETE a la URL almacenada en el enlace de tipo *edit* con la cabecera de autorización de Google, y tratar su correspondiente redirección por medio del código HTTP 302. Ya está. Los problemas aparecen cuando estamos conectados por medio de algún proxy que no acepte, por razones de seguridad, comandos HTTP diferentes de GET y POST. En estos casos Google proporciona la siguiente alternativa. Emitimos un comando POST con un valor para la cabecera 'Content-Length' de 0 y, finalmente, creamos un nuevo campo en la cabecera HTTP de tipo 'X-HTTP-Method-Override' con el valor de la operación que deseamos realizar (en este caso 'DELETE'). Esta técnica se puede utilizar también para la actualización de un evento. La manera natural pasa por emitir un comando

HTTP PUT sobre la dirección URL del enlace de edición, pero si estamos delante de uno de estos proxys podremos emitir un comando POST e indicar la operación PUT dentro de la cabecera 'X-HTTP-Method-Override'. Hay que tener cuidado en un aspecto. Para la modificación del evento hay que enviar el XML con todos los campos originales del mismo, no sólo los que van a ser modificados. Un ejemplo del procedimiento de comunicación HTTP por medio de la cabecera 'X-HTTP-Method-Override' lo podemos encontrar en los métodos *update_entry()* y *delete_entry()* del Listado 2.

Victoria

La alternativa a SOAP que propone GData y su arquitectura basada en REST suponen una clara simplificación en el acceso a servicios web. Con un simple *wget* podemos empezar a realizar scripts bash, python o perl para el acceso a dichos servicios, sin necesidad de depender de ninguna librería externa. Para los accesos autenticados necesitamos crear nuevas cabeceras HTTP, pero hasta con aplicaciones en línea de comando como *curl*, y mediante el uso del parámetro *-d* para el envío de parámetros de formulario, y de *-H* para la inserción de nuevas entradas de cabecera, podremos tener acceso a toda la funcionalidad que proporciona el protocolo GData. Naturalmente todo el proceso se simplifica si tenemos la oportunidad de hacer uso de algún lenguaje más sencillo que bash. Los trozos de código mostrados en el Listado 2 pueden ser el germen del proyecto

PyGData para el acceso al protocolo GData desde Python. De hecho, la complejidad de un proyecto como éste no está en la implementación del protocolo, sino en el API que se ha de proporcionar para la construcción de los documentos XML Atom. Y es precisamente en lo que estoy trabajando ahora mismo ¿me ayudas? ■

RECURSOS

- [1] Google SOAP Search API: <http://code.google.com/apis/soapsearch/>
- [2] PyGoogle: <http://pygoogle.sourceforge.net>
- [3] Dive Into Python: <http://www.diveintopython.org>
- [4] APIs de Google: <http://code.google.com/apis.html>
- [5] API de Calendar: <http://code.google.com/apis/gdata/calendar.html>
- [6] PyGData: <http://code.google.com/p/pygdata>
- [7] Google Calendar: <http://www.google.com/calendar/>
- [8] Formato Atom: <http://tools.ietf.org/html/rfc4287>
- [9] ClientLogin API: <http://code.google.com/apis/accounts/AuthForInstalledApps.html>
- [10] Tipos de documentos: <http://code.google.com/apis/gdata/common-elements.html>
- [11] Roy T. Fielding: <http://www.ics.uci.edu/~fielding/>
- [12] GData: <http://code.google.com/apis/gdata/overview.html>