UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Firetag – Una extensión para Mozilla Firefox que mejore el manejo de tags en distintas redes sociales.
Autores: Juan Ignacio Vidal.
Director: Dra. Alicia V. Díaz.
Codirector: - .
Asesor profesional: - .
Carrera: Licenciatura en Informática (Plan 90).

Resumen

En el presente trabajo se propone una solución a los problemas relacionados con la consistencia de los tags de un usuario, mediante el desarrollo de una extensión para Mozilla Firefox.
Firetag, la extensión propuesta, permite a un usuario interactuar desde el navegador de Internet con las cuentas que éste posee en las distintas redes sociales en las que participa. De esta manera el usuario puede compartir información y modificar la información ya compartida de un modo más consistente.
Mediante el acceso a los datos compartidos por el usuario, Firetag brindará sugerencias de los tags que puede utilizar el usuario basándose en todo el conjunto de tags que éste ha utilizado en las distintas redes sociales, logrando así, servir de gran ayuda para el mantenimiento de la consistencia dentro del conjunto de tags que el usuario utiliza en Internet.
Lograr una mayor consistencia en el vocabulario de los usuarios, tendrá como efecto a largo plazo, mejorar el criterio de los tags utilizados en conjunto por todos los usuarios y ayudar así a una mejor categorización de la información compartida en las distintas redes sociales.

Palabras Claves

Tags, Folksonomías, Redes Sociales, Web Semántica, Extensión de Mozilla Firefox, Bibsonomy, Delicious, Flickr.

Conclusiones

La actividad conjunta del usuario para con sus tags en distintas folksonomías y desde una aplicación que permite dicha unificación, resulta de suma utilidad y genera un volumen de datos mayor y bien definido para poder realizar recomendaciones basadas en uso previo. Mantener los datos de un usuario unificados quizás permita mejorar las recomendaciones al usuario en distintas situaciones como recomendaciones de sitios web a visitar en un buscador, recomendaciones de productos en sitios de e-commerce, etc.

Trabajos Realizados

Durante el desarrollo del presente trabajo se realizó una investigación sobre las distintas APIs de sitios como Bibsonomy, Delicious y Flickr. Además se investigó sobre distintas aplicaciones que plantean soluciones a problemáticas similares a las de este escrito.
Finalmente, se desarrolló una extensión de Mozilla Firefox que cumple con las ideas planteadas en el trabajo. Para ello fue necesario llevar a cabo una investigación previa sobre el funcionamiento y la implementación de los complementos para el navegador mencionado.

Trabajos Futuros

- Definición de una arquitectura de comunicación de salida de la extensión Firetag de la misma manera que se definió una arquitectura para la comunicación de entrada.
- Ampliación hacia otras redes sociales como LinkedIn, Youtube, etc.
- Incorporación de más servicios para la interacción del usuario con sus perfiles en las distintas redes sociales.
Universalización de la extensión a otros navegadores populares como Google Chrome, Internet Explorer, Safari, etc.

Fecha de la presentación: Febrero de 2011

Tesina de Grado
Licenciatura en Informática Plan 90

Firetag – Una extensión para Mozilla Firefox que mejore el manejo de tags en distintas redes sociales

Juan Ignacio Vidal

Directora: Dra. Alicia V. Díaz

Febrero de 2011

Abstract

Las aplicaciones web que mantienen un sistema de tagueo como *Delicious*[1], *Flickr*[2] o *Bibsonomy*[3] han ganando gran popularidad en los últimos años. Este tipo de aplicaciones web permite a los usuarios clasificar los objetos compartidos (ya sean *URLs*, fotos, publicaciones, etc.) de forma sencilla y no estructurada. De esto se obtiene una gran ventaja ya que los usuarios no necesitan ningún tipo de conocimiento experto, lo que bien podría explicar por qué el uso de estas redes sociales se ha incrementado, despertando el interés en una gran cantidad de usuarios.
Como se detalló anteriormente, la clasificación de objetos mediante tags es sencilla y no estructurada, lo que no obstante genera ciertas problemáticas a la hora de realizar y analizar una clasificación. Algunos de estos inconvenientes se enumeran a continuación:

- Errores de escritura u ortográficos: un error en la escritura de un tag, o un error ortográfico puede generar información inexacta.
- Diferencias de idioma: usuarios con distinta lengua nativa, realizarán tagueos en distintos idiomas.
- Utilización de sinónimos: distintos usuarios o incluso un único usuario podrían taguear varios objetos que refieren a un mismo concepto o idea con palabras diferentes de igual significado (sinónimos).

En el presente trabajo se propone una solución a estos problemas relacionados con los tags de un usuario, mediante el desarrollo de una extensión para Mozilla Firefox.
Firetag, la extensión propuesta, permite a un usuario interactuar desde el navegador de Internet con las cuentas que éste posee en las distintas redes sociales en las que participa. De esta manera el usuario puede compartir información y modificar la información ya compartida de un modo más consistente.
Mediante el acceso a los datos compartidos por el usuario, *Firetag* brindará sugerencias de los tags que puede utilizar el usuario basándose en todo el conjunto de tags que éste ha utilizado en las distintas redes sociales, logrando así servir de gran ayuda para el mantenimiento de la consistencia dentro del conjunto de tags que el usuario utiliza en Internet; más adelante esto será definido como “vocabulario”.
Lograr una mayor consistencia en el “vocabulario” de los usuarios tendrá, como efecto a largo plazo, mejorar el criterio de los tags utilizados en conjunto por todos los usuarios y ayudar así a una mejor categorización de la información compartida en las distintas redes sociales.

[1] http://delicious.com
[2] http://www.flickr.com
[3] http://www.bibsonomy.org

Agradecimientos

No son pocas las personas que merecen mi agradecimiento en este trabajo, espero poder expresar en este líneas parte del reconocimiento que éstas merecen.

En primer lugar, a mis padres por inculcarme desde siempre la cultura del estudio y por su apoyo incondicional durante todos los años de mi carrera y de mi vida. A mi hermano por mostrarme la otra cara de la realidad y por darme una mano hasta último momento con las correcciones y las redacciones.

A mi abuela, por su incansable interés, preocupación y apoyo por mis estudios, al igual que mis tíos, mis tías, mis primos y mis primas.

También a mis amigos, esas personas maravillosas que uno no sabe en dónde conoce (o sabe, pero no importa), porque siempre estuvieron ahí cuando los necesité y siempre van a estar. Tanto a mis amigos de toda la vida, que me alentaron en los momentos más difíciles de la carrera y se alegraron con mis progresos, como a mis amigos que conocí durante la facultad; porque, además, nunca se cansaron de ayudarme en incontables materias.

A Alicia, por su paciencia y guía constante durante este trabajo y durante los años de investigación, disciplina en la que participo con gran entusiasmo desde hace más de tres años gracias a otro amigo, Diego.

Al Saracom F.C. (y todos sus integrantes) por darme las mejores horas para despejarme del trajín de las largas horas de estudio.

A la educación pública y gratuita por darme la formación académica a cambio de una sola cosa, mi esfuerzo.

A todos ellos, muchas gracias.

Juan Ignacio Vidal.

Índice General

Índice de imágenes

Capítulo 1

Introducción

No resulta del todo posible describir el problema y en consecuencia la solución que se desarrollará en el presente trabajo sin definir el concepto de folksonomía. A continuación daremos una primera definición básica para abordar el tema, más luego se desarrollará con mayor detalle este concepto.

Una folksonomía es el resultado de asignar tags de manera libre y personal a información y/o recursos presentes en la web, cualquier dato que posea una *URI*. El tagueo es realizado en un entorno social (usualmente compartido y abierto a otros usuarios). La folksonomía se crea mediante las acciones de tagueos realizadas por la persona que, a su vez, consume la información publicada por otras personas.

La principal ventaja que se obtiene a partir de una folksonomía es la categorización de los recursos compartidos, de manera no estructurada. Esto se logra mediante la participación de todos los usuarios (o al menos la gran mayoría de ellos) en el proceso de asignación de tags que reflejen, desde su punto de vista, el significado del recurso en cuestión.

Claro que la libre asignación de tags también implica algunos problemas que se convierten en una desventaja. Al realizarse libremente la asociación entre tags y objetos pueden darse situaciones erróneas e indeseadas debido a las diferentes maneras en que los usuarios realizan sus tagueos.

Los usuarios que posean distintas lenguas nativas realizarán tagueos de manera muy diferente. Si bien esto podría no ocasionar problemas considerando que un objeto recibirá tags en distintos idiomas obteniendo así traducciones de su significado, puede presentarse la confusión entre términos de distintos idiomas pero con idéntica escritura. Palabras como “dice”, “fin”, “pie”, “can” son algunos ejemplos de términos que pueden interpretarse en inglés y en español y su significado varía radicalmente.

Los errores ortográficos o errores de escritura también generan problemas en la definición de una folksonomía. Por error o desconocimiento, un usuario puede asignar como tag una palabra inexistente o, peor aún, tags que tienen un significado no relacionado con el objeto al que refieren.

Por último, consideraremos el problema de los sinónimos. Si bien que un objeto tenga asignado distintos sinónimos no genera ninguna inconsistencia en la información, dos objetos que estén relacionados semánticamente pueden tener asignados tags totalmente distintos, tags cuyos términos son sinónimos entre sí.

Esta situación podría ocasionar algunas fallas en búsquedas. Por ejemplo: no sería posible, al buscar el término “barco”, encontrar una imagen de un velero que fue tagueada con el término “nave”.

Los problemas previamente enumerados no sólo se manifiestan en los tags emitidos por distintos usuarios, puede darse el caso en que un mismo usuario no respete cierta coherencia entre los tags utilizados en distintas publicaciones. Como un usuario realiza los distintos tagueos durante lapsos de tiempo eventualmente muy distantes, y en redes sociales completamente distintas, el conjunto de tags que representa el vocabulario del usuario en las redes sociales puede tener un grado de cohesión muy bajo ya sea por errores puntuales o por no mantener un criterio unificado a la hora de asignar uno o varios tags a distintas publicaciones.

Esta última situación es la que se propone mejorar mediante el presente trabajo. La idea de permitirle a un usuario interactuar con sus cuentas en las distintas redes sociales a través de una extensión del navegador Mozilla Firefox, que además de unificar el criterio de interacción con las redes sociales brinde ayuda y/o recomendaciones en el momento de asignar un nuevo tag a un objeto existente o a uno que está por ser creado, es el concepto central en el que se basa el desarrollo de este trabajo.

La aplicación que se plantea, denominada Firetag, debe resolver un problema crucial para encarar la solución en cuestión. Dicho problema es la comunicación desde la aplicación hacia las distintas redes sociales con que ésta interactúa, el cual se logra resolver mediante el acceso vía AJAX a las APIs (Application Programming Interface) que poseen las redes con las que se comunica Firetag.

Esta interacción se realiza en una serie de etapas y con una arquitectura de comunicación particular, las cuales serán descriptas en detalle en secciones posteriores.

Serán, éstas, las herramientas que permitirán resolver los problemas de:

- Identificar al usuario en las distintas redes sociales.
- Enviar la información necesaria para la ejecución de la tarea deseada.
- Interpretar los resultados de la ejecución de la tarea e informárselos al usuario.

Estas situaciones en conjunto, resueltas, se asemejan a la idea de lo que se denomina en el contexto de una página web como *mashup*. Este concepto se centra en aprovechar la información compartida por otros sitios para utilizarla y aplicarla en el entorno de nuestro propio sitio, con el objetivo de mejorar los datos que se comparten en nuestra página.

Así, aplicando esta idea dentro de una extensión del navegador de Internet, surge la noción de una aplicación que permita la interacción de un usuario con datos, información y principalmente tags, que se encuentran en distintas aplicaciones webs cuyas APIs nos permiten manipular los datos de manera remota.

Si bien la idea de *mashups* apunta a información pública y no privada, como en la presente propuesta, la idea que se plantea es sustancialmente la misma. Esto es,

conglomerar toda la información que un usuario mantiene en distintas redes para permitirle manipular estos datos de forma más eficiente y con una mayor coherencia entre los tags que, según el usuario, definen la información que comparte.
Podría afirmarse, entonces, que la aplicación Firetag no es más que un *mashup* de distintas redes sociales con la diferencia de que, en lugar de manipular la información pública de todo el contenido de las redes sociales, sólo se utiliza la información privada perteneciente a un usuario en particular.

1.1. Objetivos

El objetivo central de este trabajo puede definirse como la obtención de una aplicación pequeña, de fácil acceso y de simple utilización, que permita a sus usuarios manipular en forma más eficiente los tags que definen sus vocabularios en el contexto de las redes sociales.
Una extensión de Mozilla Firefox simplifica los primeros problemas, es decir, el tamaño reducido de la aplicación y la facilidad de acceso. Si bien podemos considerar el problema adicional de la falta de universalidad de nuestra aplicación (no todos los usuarios utilizan el mismo navegador) la idea de esta extensión puede ser replicada con mayor o menor complejidad, según sea el caso, en los distintos navegadores utilizados popularmente. Si bien ésta no es la idea del presente trabajo, podría considerarse en el contexto del trabajo a futuro.
La extensión propuesta deberá permitir el acceso de manera sencilla a las distintas redes sociales con las que ésta interactúa, brindándole al usuario la seguridad pertinente y la confianza para que éste manipule sus datos de manera más eficiente.
En adición, el usuario deberá recibir recomendaciones de los tags que desee utilizar basándose en la información obtenida a partir de los datos ya publicados por el usuario dentro de las redes sociales en cuestión. Estas recomendaciones serán, en consecuencia, las que permitan manipular en forma más eficiente el vocabulario del usuario con el objetivo de lograr una mayor cohesión y coherencia dentro de los términos utilizados.
En conjunto, todas estas ideas, podrán mejorar la semántica de la asignación de los tags en las redes sociales, siempre y cuando los usuarios utilicen de manera razonable las recomendaciones y la cantidad de usuarios interactuando mediante la extensión aumente de forma significativa.
El usuario deberá tener la posibilidad de interactuar normalmente mediante la aplicación, de la misma manera que lo hace con las distintas redes sociales. En este caso particular se trabajará con Flickr, Delicious y Bibsonomy; por ello el usuario deberá poder manipular la información de estas redes de forma análoga.
Será de suma importancia para el objetivo del presente trabajo que el usuario pueda agregar información; esto es, realizar nuevas publicaciones o modificar publicaciones anteriores mediante la ejecución de cambios en el conjunto de tags que éstas posean. Se

centra el interés en tales actividades ya que son éstas las que definirán la mejora en la coherencia y cohesión del vocabulario del usuario.
Actividades como la eliminación de una publicación, no presentan mayor aporte a la mejora del vocabulario, ya que no podrá realizarse recomendación alguna sobre los tags que se perderán al eliminar una publicación si el usuario ha tomado la decisión de quitarlos de su conjunto de publicaciones. Lo mismo sucede con la visualización de las publicaciones; ésta es una actividad sin efecto para lo que al vocabulario respecta y es por ello que no debería tomar mayor importancia en el desarrollo de la aplicación.

1.2. Resultados

Los resultados obtenidos durante el desarrollo del presente trabajo pueden ser divididos en dos grandes grupos. Por un lado se considerarán los resultados teóricos y por el otro los resultados prácticos.

1.2.1. Resultados teóricos

Los resultados teóricos pueden definirse en función del cambio de paradigma que se plantea en el actual trabajo. La nueva forma de interacción que se plantea se ve reflejada en otras extensiones existentes para Mozilla Firefox, las cuales permiten manipular datos en forma remota de una única aplicación web o bien compartir datos en distintas redes sociales pero simplificando sólo el acceso a las páginas de cada red social, y no mediante la real interacción de la aplicación con el sitio.
A diferencia de estos casos, Firetag permite utilizar datos de distintas redes sociales y aprovecharlos para ayudar al usuario a mantener un vocabulario más conciso, más coherente. Se logra unificar el criterio de un usuario en una persona y no en un conjunto de perfiles definidos en un conglomerado de páginas web. Toda esta unificación se centra en torno a los tags, son éstos los que se consideran como definidores del perfil del usuario, de su vocabulario. Al manipularlos en conjunto y recomendar tags para nuevas publicaciones en base al grupo de tags ya utilizados, se unifica el criterio de tagueo en las distintas redes en las que el usuario participa; logrando así aunar la semántica que define el usuario en los sitios con los que interactúa.

1.2.2. Resultados prácticos

El principal resultado práctico es la aplicación obtenida, Firetag. La misma se concentra en la manipulación de los tags de un usuario dentro de distintas redes sociales, en particular Flickr, Bibsonomy y Delicious. Se logró interactuar de manera remota con sendas aplicaciones, permitiéndoles a los usuarios realizar publicaciones en las redes Flickr y Delicious, así como agregar tags a publicaciones existentes en Flickr; en estos casos la aplicación brinda recomendaciones basándose en la frecuencia de uso de tags del conjunto total de las cuentas del usuario.

Además, Firetag permite eliminar tags de las publicaciones en Flickr y Delicious, pero en estos casos no se generan recomendaciones, ya que el usuario decidirá qué tag eliminar y no se le presentarán las confusiones descriptas previamente que sí se manifiestan al momento de agregar información.

La otra funcionalidad interesante que permite realizar la aplicación es la visualización de todos los tags utilizados por el usuario. Estos tags se verán agrupados y ponderados por su frecuencia de uso, sin importar la red social en la cual han sido definidos. Esta herramienta es la que permite obtener el conjunto de datos necesarios para realizar recomendaciones al usuario, cuando éste desee agregar un nuevo tag a su vocabulario.

Algunos resultados prácticos secundarios del desarrollo e investigación del presente trabajo se obtuvieron cuando se resolvió el problema de unificar la interacción de la aplicación con las distintas redes sociales. A priori, cada red social brinda una forma de comunicación distinta en la cual los datos obtenidos no necesariamente respetan un único formato. Esta situación se resolvió mediante la transformación de los datos recibidos en formato XML en información que sea interpretable por la aplicación, esto es, XUL.

El cambio de un formato a otro se realiza mediante la aplicación de archivos XSLT, donde cada uno de estos archivos describe la forma en la que se debe interpretar la información obtenida desde las distintas redes y cómo transformarla en datos interpretables por la extensión del navegador.

1.3. Contribución

El concepto de interacción con múltiples sitios desde una única herramienta es la principal contribución del presente trabajo. Es la idea de cambiar la manera en que un usuario se relaciona con sus cuentas en distintas redes sociales, este enfoque es el que permite manipular de manera más eficiente los tags que libremente define un usuario. Esa libertad de acción se mantiene porque es el principio fundamental de las folksonomías, pero se agrega un concepto unificador para el usuario permitiéndole definir los nuevos términos que utilizará mediante la ayuda basada en el conocimiento aportado por los términos ya definidos.

Esta globalización de las cuentas y/o perfiles del usuario, en lo que a sus tags se refiere, permite una mejor comprensión del vocabulario del usuario, de sus características a la hora de realizar un tagueo. Surge desde la individualidad de un usuario el concepto de ayuda para cohesionar su vocabulario, pero es el objetivo lograr una mayor coherencia en la semántica que definirá los objetos compartidos en las redes sociales por todos los usuarios. Objetivo que se verá realizado o no si un gran porcentaje de usuarios comienzan a interactuar mediante aplicaciones como la que se plantea, aplicaciones que brinden ayuda al usuario pero con el objetivo de mejorar las folksonomías en su conjunto, tratando de solucionar los problemas más comunes que estas categorizaciones suelen acarrear.

En adición a este criterio teórico planteado, se contribuye con el presente trabajo mediante una nueva extensión al navegador Mozilla Firefox. Firetag, la extensión desarrollada, se encuentra publicada en el repositorio de “add-ons” de Mozilla y actualmente tiene una tasa de descarga cercana a las dos descargas diarias y ya ha superado el centenar de usuarios que utilizan esta extensión.
La contribución a la comunidad de “add-ons” es secundaria, pero no así despreciable; es el principal fundamento del presente trabajo el de valorar la participación comunitaria por parte de los usuarios en la nueva metodología de interacción con la web. Es por ello que la evaluación de la aplicación a través de la comunidad se considera parte de los logros del presente trabajo.

1.4. Estructura del documento

El resto del documento se compone de los temas en el orden que se describe a continuación. En el capítulo 2 se describen las tecnologías, herramientas y conceptos involucrados en el desarrollo de Firetag con el fin de definir una base de conocimiento sobre los mismos. Además se desarrollan las bases de conocimiento previo en las que fue basada la idea del presente trabajo.
El capítulo 3 abarca los temas referidos al concepto general de administración de tags por parte de los usuarios, definiendo una categorización teórico-práctica y luego proyectando estos conceptos sobre el mecanismo manipulación de tags que posee Firetag.
El capítulo 4 consta de la descripción del funcionamiento de las APIs con las que interactúa Firetag y la delineación en detalle del mecanismo mediante el cual la aplicación desarrollada efectúa la interacción con las distintas APIs.
Finalmente, en el capítulo 5 se desarrollan las conclusiones, los aportes realizados y los trabajos a futuro dentro del área de la presente investigación.

Capítulo 2

Bases y tecnologías

En este capítulo se abordarán los temas necesarios para introducir los conceptos relacionados con el desarrollo de extensiones para Mozilla Firefox, así como también se describirán algunas extensiones relacionadas con Firetag y además se detallarán una serie de tecnologías y conceptos teórico-prácticos que fueron utilizados durante el desarrollo de la extensión propuesta en este trabajo.
En capítulos posteriores se hará mención de estas herramientas, por lo que resulta indispensable partir de un conocimiento básico para completar la descripción de Firetag y lograr un completo entendimiento de su funcionamiento.

2.1. Extensiones para Mozilla Firefox

Los complementos de Mozilla son un software de tamaño reducido creados por distintas personas o usuarios, que agregan nuevas características o funcionalidades tanto al navegador Firefox como a otros productos de Mozilla, como el cliente de mail Thunderbird. Los complementos pueden aumentar las funcionalidades de Firefox con nuevos motores de búsqueda, diccionarios de lenguas extranjeras, o cambiar la apariencia visual de Firefox. Con los complementos, un usuario puede modificar o agregar funcionalidades al navegador con el fin de satisfacer sus necesidades.
Existen distintos tipos de complementos agrupados en tres categorías:

- Extensiones:
  Las extensiones agregan nuevas características a Firefox, o modifican algunas ya existentes. Existen extensiones que permiten bloquear anuncios, descargar vídeos de sitios web, incrementar la integración del navegador con sitios como Facebook o Twitter, o incluso añadir características sólo disponibles en otros navegadores.

- Themes:
  Los temas cambian la apariencia visual de Firefox. Algunos temas simplemente cambian el color de fondo de las barras de herramientas mientras que otros pueden hacer de Firefox un programa completamente distinto.

- Plugins:

  Un plugin es un fragmento de software que permite utilizar contenido de Internet para el cual Firefox no está diseñado para procesar. Este contenido incluye normalmente formatos de videos con patentes, audio, juegos online, presentaciones y más. Los plugins son creados y distribuidos en general por las compañías que poseen las patentes de los formatos o que han desarrollado algún formato en particular. Uno de los plugins más populares es el que permite visualizar los archivos de formato ".pdf" en el navegador.

El complemento desarrollado en este trabajo corresponde a la categoría de extensión, es por eso que no se mencionarán más detalles de las otras categorías de complementos.

2.2. Extensiones relacionadas

A continuación se enunciarán una serie de extensiones para Mozilla Firefox que por distintos motivos poseen alguna relación con la extensión propuesta, y de una forma u otra han servido para el desarrollo de la aplicación en cuestión.

2.2.1. BibSonomy (https://addons.mozilla.org/firefox/addon/52326/)

Esta extensión permite integrar el navegador con la red social de publicaciones BibSonomy. La capacidad de compartir *bookmarks* de Firefox es incrementada mediante la sincronización de los *bookmarks* del usuario con BibSonomy.

Mediante una simple configuración, la extensión BibSonomy permite utilizar de manera análoga tanto los *bookmarks* que el usuario estableció en su navegador como los que compartió en su perfil de BibSonomy. Esto se logra mediante la posibilidad de los *bookmarks* hacia BibSonomy desde el navegador y viceversa; unificando así el criterio de interacción de los mismos.

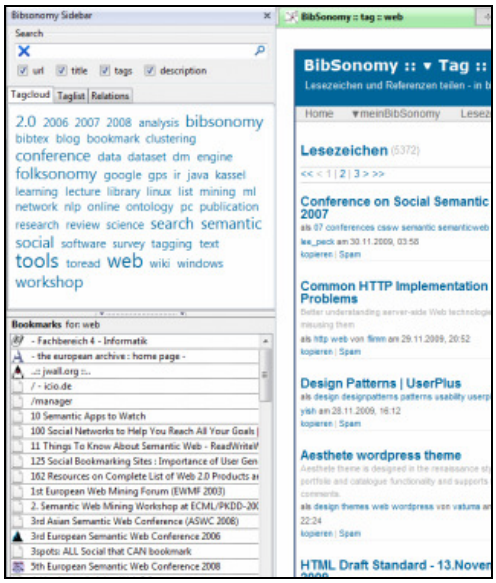

Imagen 2.1 – Una captura de pantalla de la extensión BibSonomy

2.2.2. Delicious Bookmarks (https://addons.mozilla.org/firefox/addon/3615/)

Esta extensión integra el navegador con Delicious. Al igual que la extensión mencionada anteriormente, lo hace mediante el incremento de la capacidad de manejo de *bookmarks* de Firefox agregando las siguientes funcionalidades:

- Búsqueda y navegación de los *bookmarks* de Delicious de un usuario.
- Acceso a los *bookmarks* propios desde cualquier computadora en cualquier momento.
- Organización de los *bookmarks* mediante el uso de tags.
- Posibilidad de compartir los *bookmarks* con contactos en Internet.
- Capacidad de importar los *bookmarks* de Firefox a la cuenta Delicious.

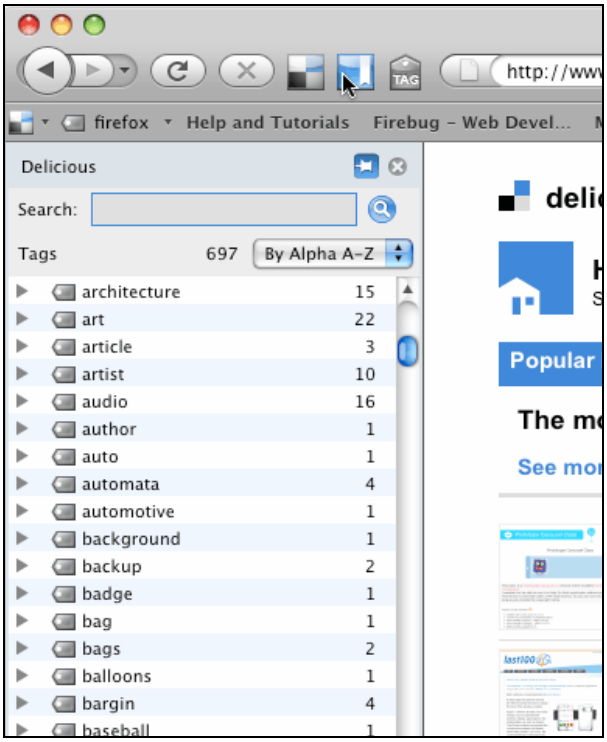

Imagen 2.2 – Una captura de pantalla de la extensión Delicious Bookmarks

2.2.3 Shareaholic (https://addons.mozilla.org/firefox/addon/5457/)

Esta extensión permite compartir links en una gran cantidad de redes como Facebook, Twitter, Google Mail, Reader, Bookmarks, Evernote, Bitly, StumbleUpon, LinkedIn, entre otros.

No realiza la publicación de un link en una red, sino que redirecciona al usuario hacia la página en la cual debe publicar el nuevo link. Si bien la herramienta no es compleja, y posee una utilidad relativamente acotada, la idea en la que se basa es similar a uno de los conceptos de la aplicación que se desarrolla en el presente trabajo; esto es, permitirle al usuario la interacción unificada con las distintas redes sociales en las cuales participa, haciendo esto desde un único lugar: el navegador.

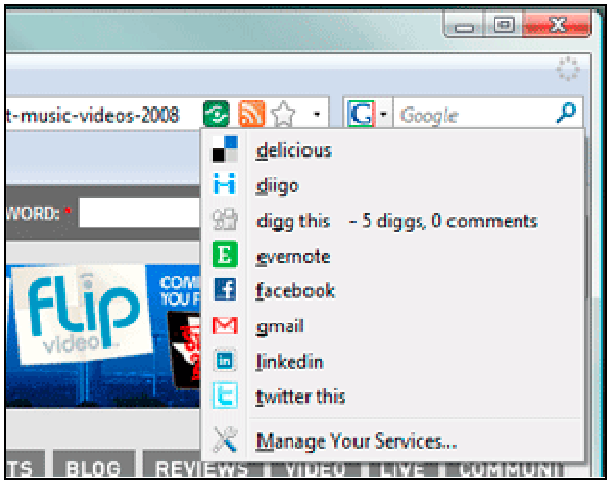

Imagen 2.3 – Una captura de pantalla de la extensión Shareaholic

2.2.4 Yoono (https://addons.mozilla.org/firefox/addon/1833/)

Yoono es una extensión que agrega una barra lateral a Firefox que ayuda a simplificar la interacción con la vida social *on-line* conectándose a Facebook, MySpace, Twitter, LinkedIn, Youtube, Flickr, Friendfeed, AIM, entre otros y todo desde el mismo lugar: el navegador. Brinda una forma muy sencilla para compartir datos en las redes sociales que participa el usuario, compartir links, imágenes y videos de la página que se está visitando en cualquiera de las redes sociales ya mencionadas en forma simultánea.

Imagen 2.4 – Captura de pantalla de la extensión Yoono

2.3. Folksonomías

El concepto de folksonomía es el principal motivador del presente trabajo, por ello su definición y explicación debe ser considerada de manera prioritaria. El objetivo principal que se plantea apunta a mejorar la coherencia de los tags de un usuario utilizando como base de conocimiento los tags previamente agregados en distintas redes sociales; esto conllevará a la mejora de las folksonomías en menor medida, y en mayor medida a la mejora del uso de tags por parte de un usuario.

En [4] se define a una folksonomía como el resultado de asignar tags de manera libre y personal a información y/o objetos (cualquier dato que posea una *URL*). El tagueo es realizado en un entorno social (usualmente compartido y abierto a otros usuarios). La folksonomía se crea mediante las acciones de tagueos realizadas por la persona que, a su vez, consume la información publicada por otras personas.

Los valores que toman estos tagueos externos dependen de los usuarios, quienes utilizan su propio vocabulario agregándole un significado explícito a los tags, el cual puede considerarse como inferido desde el propio entendimiento del usuario hacia el objeto o información que éste ha tagueado.

Son tres los principios de una folksonomía:

- Los tags.
- Los objetos que son tagueados.
- Los usuarios que realizan los tagueos.

Estos principios son fundamentales para desambiguar los términos de los tags y brindar una correcta comprensión del objeto tagueado. Con esta información, una folksonomía puede definirse formalmente como en [1] y [2], esto es F:=(U, T, R, A, <).

F, la folksonomía, se define como tres conjuntos U (los usuarios), T (los tags) y R (los recursos, información u objetos). Además la relación ternaria A que define las asignaciones de tags, estableciendo las tuplas $\{a:=(u, t, r) \mid a \in A; u \in U; t \in T; r \in R\}$.

Finalmente <, vincula a los tags propios de un usuario, definiendo la relación de sub/super entre ellos. Esta relación no la define el usuario, es definida automáticamente por el sistema.

En [3] se menciona uno de los aspectos más importantes de una folksonomía; ésta se encuentra compuesta por términos en un espacio de nombres chato, es decir, no existe una jerarquía y no se especifican directamente relaciones padre/hijo o de hermanos entre los términos. Existe, sin embargo, relaciones entre tags generadas automáticamente, las cuales agrupan tags asociados a los mismos objetos. Esto es completamente distinto a las taxonomías formales y esquemas de clasificación, en donde se plantean varios tipos de relaciones explícitas entre los términos. Relaciones como *es mayor que*, *es menor que*, *está relacionado con*, son comunes en estas clasificaciones. Las folksonomías, en cambio, son simplemente el conjunto de términos que fueron utilizados por un grupo de

usuarios para taguear contenido, no existe un conjunto predeterminado de términos o etiquetas para las clasificaciones.

En resumen, aunque el término “clasificación” es usado comúnmente en relación a estos sistemas de folksonomías, es la “categorización” la forma más acertada de asemejar su comportamiento. Una categorización es generalmente menos rigurosa y sus límites son menos definidos. Se basa más en una síntesis de similitud que en una organización sistemática de los componentes. Más importante aún, cada documento puede tener varios términos asociados a él, lo que no sucede en una clasificación, que suele enfocarse en proveer una única clasificación para un ítem, con relaciones jerárquicas definidas previamente.

2.4. Mashups

En [5] y [6] se definen conceptos relacionados a la denominada web 2.0 y particularmente los *mashups*. Dentro del desarrollo web, un *mashup* es una página o aplicación que utiliza y combina datos, representación o funcionalidades de dos o más recursos con el fin de crear nuevos servicios.

Este término implica facilidad, velocidad de integración APIs abiertas de uso frecuente y datos para producir resultados enriquecidos que, no necesariamente, tendrán el propósito original para el cual fueron producidos.

Las características principales de un *mashup* son la combinación, la visualización y el agregado. Son de gran utilidad para brindar mayor utilidad a datos existentes, más aún para usos personales y profesionales.

Para poder acceder en forma permanente a los datos de otros servicios, los *mashups* suelen ser aplicaciones cliente o aplicaciones alojadas online. Desde el año 2010, dos grandes proveedores de *mashups* agregaron soporte para alojar implementaciones basados en soluciones de *Cloud Computing*; que son cómputos realizados en un ambiente basado en Internet en los cuales los recursos compartidos, el software y la información son provistos a las computadoras y a otros dispositivos según demanda.

En la actualidad, son cada vez más las aplicaciones web que han publicado sus APIs para permitir a los desarrolladores de software integrar fácilmente los datos y funciones brindados, sin necesidad de desarrollarlos por su cuenta. Los *mashups* tienen un rol muy activo en la evolución del software social y la web 2.0. Las herramientas para combinar *mashups* son de gran facilidad y suelen ser usadas por usuarios finales. Éstas no requieren habilidades en programación, en cambio brindan un soporte de interfaces gráficas, servicios y componentes. En consecuencia, estas herramientas contribuyen a una nueva visión de la web, en la cual los usuarios finales pueden participar activamente.

Los *mashups* pueden clasificarse en tres grandes tipos, *mashups* de datos, *mashups* consumidores y *mashups* de negocios; aunque el más común suele ser los *mashups* consumidores, apuntados al público general.

- *Mahsups* de datos: combinan tipos similares de medios e información desde múltiples recursos hacia una única representación. La combinación de todos estos recursos crea un nuevo servicio web, que no era originalmente provisto por ninguno de los otros recursos.
- *Mashups* consumidores: en oposición al *mashup* de datos, combina diferentes tipos de datos. Generalmente elementos visuales y datos desde varios recursos (por ejemplo, Wikipediavision[4] combina Google Map[5] y Wikipedia API[6]).
- *Mashups* de negocios: generalmente definen aplicaciones que combinan sus propios recursos, aplicaciones y datos, con otros servicios web externos. Enfocan los datos en una única representación y permiten la acción colaborativa de distintos negocios y desarrolladores. Son aplicaciones web seguras y visualmente ricas, que exponen información procesable desde recursos diversos, tanto externos como internos.

2.5. Tecnologías

En esta sección se describirán los conceptos básicos de las tecnologías y herramientas principales utilizadas para el desarrollo de Firetag. El conocimiento básico de estas tecnologías es fundamental para comprender, en capítulos posteriores, la arquitectura y el funcionamiento substancial de la aplicación desarrollada.

2.5.1 *JavaScript*

En [7] se define a *JavaScript* como un lenguaje de programación interpretado, dialecto del estándar *ECMAScript*. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de *JavaScript* del lado del servidor (*Server-side JavaScript* o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente *widgets*) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y *JavaScript* no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del *Document Object Model* (DOM).

---

[4] http://www.lkozma.net/wpv/index.html
[5] http://www.lkozma.net/wpv/index.html
[6] http://www.mediawiki.org/wiki/API

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de *Mocha*, el cual fue renombrado posteriormente a *LiveScript*, para finalmente quedar como *JavaScript*. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.0B3 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de marketing de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web.

Microsoft dio como nombre a su dialecto de JavaScript, *JScript*, para evitar problemas relacionados con la marca. *JScript* fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad en las funciones de fecha para el Efecto 2000, una diferencia con los que se basaban en ese momento. Los dialectos pueden parecer tan similares que los términos "*JavaScript*" y "*JScript*" a menudo se utilizan indistintamente, pero la especificación de *JScript* es incompatible con la de ECMA en muchos aspectos.

Para evitar estas incompatibilidades, el *World Wide Web Consortium* diseñó el estándar *Document Object Model* (DOM, ó Modelo de Objetos del Documento en castellano), que incorporan Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera la versión 7, y *Mozilla Application Suite*, Mozilla desde su primera versión.

En 1997 los autores propusieron JavaScript para que fuera adoptado como estándar de la *European Computer Manufacturers Association* ECMA, que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de *ECMAScript*. Poco después también como un estándar ISO.

A continuación se mencionan las características comunes de todas las implementaciones de *ECMAScript*.

- JavaScript soporta toda la sintaxis de programación estructurada en C, con algunas diferencias menores en el trascurso de las diferentes versiones del lenguaje.
- Como en la mayoría de los lenguajes de *scripting*, los tipos son asociados con los valores, no con las variables. Por ejemplo, una variable x puede ser ligada a un número y más tarde asociada a un *string*.
- JavaScript está casi en su totalidad orientado a objetos. Los objetos son *arrays* asociativos, ampliados con prototipos.
- El lenguaje incluye una función *eval* que permite ejecutar sentencias proveyéndole *strings* en tiempo de ejecución.

- Las funciones son objetos por sí mismas. Como tales, tienen propiedades y métodos como *length* y *call()*.
- Las funciones *inner* o *nested* son funciones definidas dentro de otra función. Son creadas cada vez que la función externa es invocada. Además el alcance de la función externa forma parte de las funciones internas.
- *JavaScript* utiliza prototipos en lugar de clases para implementar la herencia.
- A diferencia de muchos otros lenguajes de programación orientados a objetos, no existe diferencia en la definición de un método o una función. La diferenciación ocurre durante el llamado a la función, una función puede ser invocada como un método.
- *JavaScript* se basa en un ambiente de ejecución en *run-time* (por ejemplo, un navegador web) para proveer objetos y métodos mediante los cuales los scripts pueden interactuar con "el mundo exterior".
- Una función puede recibir un número indefinido de parámetros. La función puede acceder a ellos a través de los parámetros formales y también mediante el objeto de argumentos local.
- *JavaScript* soporta el manejo de expresiones regulares, las cuales proveen una sintaxis concisa y potente para la manipulación de texto.

2.5.2. Servicios web

Como se describe en [8] y en [9], un servicio web es una API (*application programming interface*) o una API web la cual es accedida vía HTTP (*Hypertext Transfer Protocol*) y ejecutada en un sistema remoto que aloja al servicio requerido. Los servicios web suelen agruparse en dos grandes campos: *big web services* y *RESTful web services*.

La W3C define a un servicio web como un sistema de software diseñado para soportar interacciones interoperables entre máquinas sobre una red. Posee una interfaz descripta en un formato procesable por máquina (en particular *Web Services Description Language* WSDL). Otros sistemas interactúan con el servicio web acorde a lo detallado en su descripción mediante mensajes *SOAP*, generalmente transmitidos vía HTTP con una serialización en XML junto a otros standards vinculados a la web.

La W3C, además, estipula que se pueden definir dos grandes clases de servicios web: *REST-compliant web services*, en los cuales el principal propósito del servicio es manipular las representaciones XML de los recursos web mediante un conjunto de operación que no mantienen el estado, y *arbitrary web services*, en estos casos el servicio puede producir un conjunto arbitrario de operaciones.

2.5.2.1. Big web services

En [9] y [10] se describe que los *big web services* utilizan mensajes en XML que mantienen el estándar SOAP y han logrado una gran popularidad entre las empresas de desarrollo de software. En estos tipos de sistemas se presenta una descripción legible por

máquina de las operaciones que ofrece el servicio, descriptas en *Web Services Description Language* (WSDL). Este último no es un requisito de SOAP, pero es un prerrequisito para los clientes con generación automática de código en muchos *frameworks* SOAP de Java y .NET. En algunos casos, ciertas organizaciones como WS-I requieren tanto SOAP como WSDL en la definición de sus servicios web.

2.5.2.2. Remote procedure calls (RPC)

Los servicios web RPC presentan una interfaz para la invocación de un método o función distribuida que resulta familiar para los desarrolladores. Típicamente, la unidad básica de un servicio web RPC es la operación WSDL.

Las primeras herramientas de servicios web estaban enfocadas en RPC. Como resultado de esto, este tipo de desarrollo es ampliamente utilizado y soportado. Sin embargo, suele recibir una serie de críticas por no ser débilmente acoplado, ya que solía ser implementado asociando a los servicios directamente a invocaciones de funciones o métodos de un lenguaje específico.

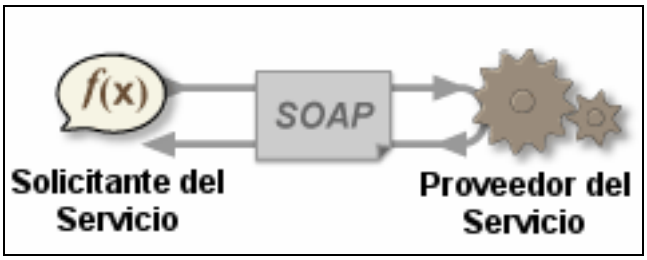

Imagen 2.5 – Elementos de la arquitectura XML-RPC

2.5.2.3. Service-oriented architecture

Los servicios web pueden ser utilizados para implementar una arquitectura acorde a los conceptos de *Service Oriented Architecture* (SOA), en donde la unidad básica de comunicación es el mensaje en lugar de la operación. Suele referirse a estos servicios como servicios orientados a mensajes.

Los servicios web SOA son soportados por la mayoría de los proveedores de software. A diferencia de los servicios web RPC, el acoplamiento débil es más viable debido a que está enfocado al “contrato” que provee WSDL en lugar de hacerlo mediante los detalles de implementación.

2.5.2.4. Representational state transfer (REST)

Como se define en [9] y [10], REST intenta describir las arquitecturas que usan HTTP o protocolos similares restringiendo la interfaz a un conjunto de operaciones standard como k, *POST*, *DELETE* en HTTP. Esta idea se centra en interactuar con recursos que mantengan su estado en lugar de hacerlo mediante mensajes u operaciones.

Una arquitectura basada en REST puede usar WSDL para describir los mensajes SOAP vía HTTP, puede implementar una abstracción sobre SOAP (por ejemplo *WS-Transfer*) o puede crearse sin utilizar en ningún momento SOAP.
WSDL 2.0 brinda soporte para asociar a todos los métodos de *request* HTTP (no sólo *GET* y *POST*). De esta manera permite una mejor implementación de los servicios web *RESTful*. Sin embargo, el soporte de esta especificación no ha sido muy desarrollado por los softwares comúnmente utilizados para el desarrollo de aplicación REST. En general suelen ofrecer herramientas que sólo soportan versiones anteriores de WSDL.

2.5.2.5. Críticas

Los detractores de los servicios webs que no son *RESTful* suelen argumentar que estos son muy complejos y se basan en un gran conjunto de proveedores o integradores en lugar de realizar implementaciones de código abierto (existen algunas implementaciones de código abierto como Apache Axis y Apache CXF).
Una de las preocupaciones principales de los desarrolladores de servicios web REST se centra en la facilidad de definición de nuevas interfaces de interacción remota que brindan las herramientas SOAP WS, generalmente basadas en introspección para obtener el WSDL, lo que ocasiona que un mínimo cambio en el servidor puede resultar en un WSDL diferente y, en consecuencia, en una interfaz distinta del servicio. Las clases del lado del cliente que pueden ser generadas a partir de las descripciones de los servicios WSDL y XSD pueden estar atadas a una versión particular de SOAP y esto puede conllevar a ciertos problemas si se modifica el SOAP del lado del cliente.
Pueden mencionarse también algunas preocupaciones acerca de la performance a causa de la utilización de XML como formato y de SOAP/HTTP para la envoltura y el transporte de los servicios web.

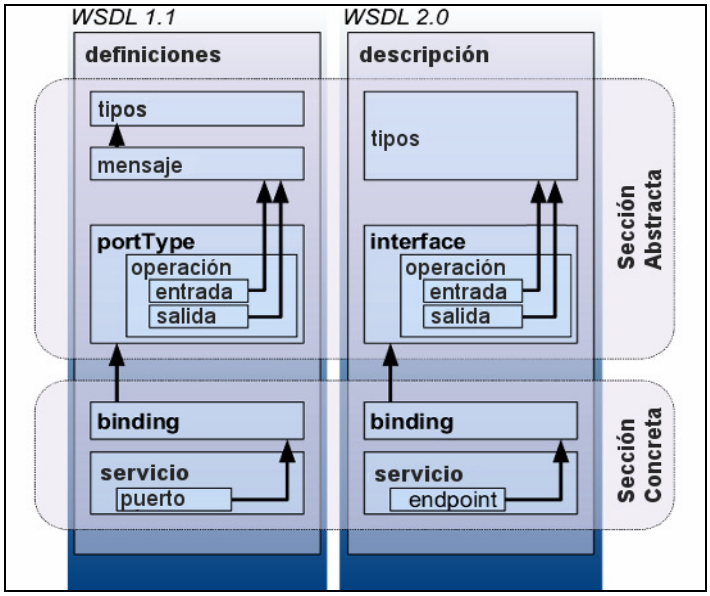

Imagen 2.6 – Conceptos definidos por los documentos WSDL1.1 y WSDL 2.0

2.5.3 API Web

Una API (*application programming interface*) web es la interfaz que un sistema, librería o aplicación provee con el fin de permitir a otros programas realizar requerimientos (*requests*) a servicios y/o permitir el intercambio de datos entre ellos.

Una API Web es una implementación de servicios web (dentro del movimiento denominado web 2.0) en donde se ha optado por reemplazar a los servicios basados en SOAP por comunicaciones basadas en REST. Los servicios REST no requieren de XML, SOAP, o definiciones de los servicios de la API mediante WSDL.

Las APIs Web permiten la combinación de servicios web múltiples en nuevas aplicaciones, normalmente conocidas como mashups.

Cuando se utiliza en el contexto de un desarrollo web, una API Web se conforma de un conjunto de mensajes de *request* HTTP, junto a la definición de la estructura de los mensajes de respuesta, normalmente expresados en un formato XML o JSON.

Cuando se ejecutan servicios web compuestos, cada sub-servicio puede ser considerado como autónomo. El usuario no tiene control sobre estos servicios, además los servicios webs por sí solos no son confiables dado que el proveedor de éstos puede eliminar, cambiar o modificar sus servicios sin brindarles notificaciones a los usuarios. La confiabilidad y la tolerancia de fallos no es soportada correctamente; pueden suceder algunas fallas durante la ejecución. El manejo de excepciones en el contexto de los servicios web sigue siendo un tema abierto y en proceso de investigación. De todas maneras, pueden ser manejadas mediante la respuesta de un objeto de error al cliente.

2.5.4. AJAX

Como se describe en [11] y [12], AJAX, acrónimo de *Asynchronous JavaScript And XML* (*JavaScript* asincrónico y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente. Es decir, se hace en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

AJAX es una tecnología asíncrona en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. *JavaScript* es el lenguaje interpretado (*scripting language*) en el que normalmente se efectúan las funciones de llamada de AJAX mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

AJAX es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como *JavaScript* y *Document Object Model* (DOM).

Ajax es una combinación de cuatro tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- *Document Object Model* (DOM) accedido con un lenguaje de *scripting* por parte del usuario, especialmente implementaciones *ECMAScript* como *JavaScript* y *JScript*, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto *XMLHttpRequest* para intercambiar datos de forma asíncrona con el servidor web. En algunos *frameworks* y en algunas situaciones concretas, se usa un objeto *iframe* en lugar del *XMLHttpRequest* para realizar dichos intercambios.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML pre-formateado, texto plano, JSON y hasta EBML.

Como el DHTML, LAMP o SPA, AJAX no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

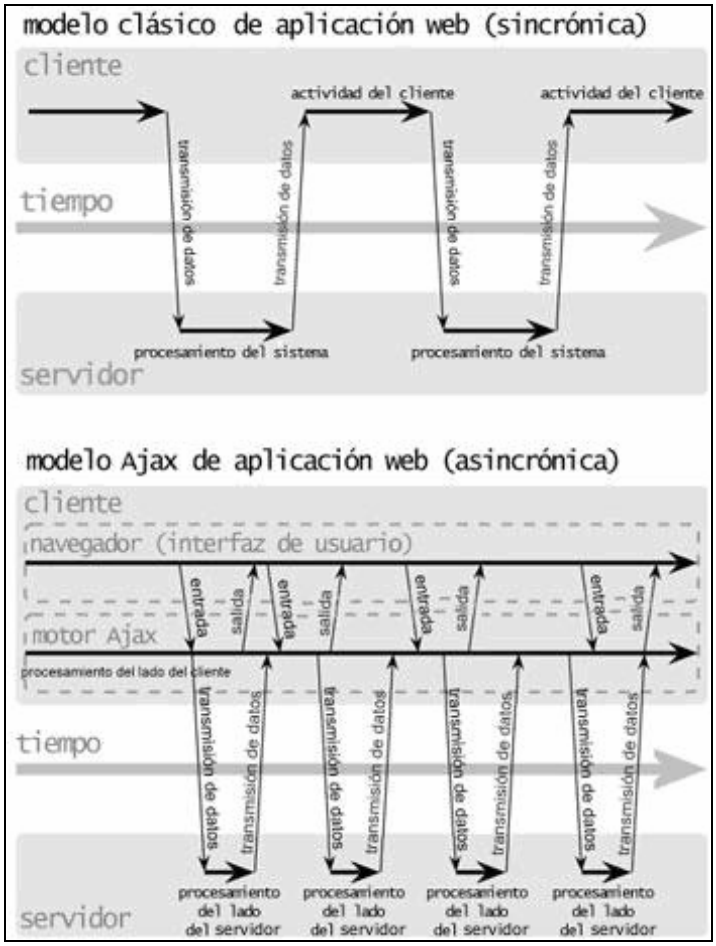

Imagen 2.7 – Diferencia del modelo de interacción de una aplicación web tradicional y una aplicación web AJAX

A pesar de que el término AJAX fue creado en 2005, la historia de las tecnologías que permiten AJAX se remonta a una década antes con la iniciativa de Microsoft en el desarrollo de s*cripting* remoto. Sin embargo, las técnicas para la carga asincrónica de contenidos en una página existente sin requerir recarga completa remontan al tiempo del elemento *iframe* (introducido en Internet Explorer 3 en 1996) y el tipo de elemento *layer* (introducido en Netscape 4 en 1997, abandonado durante las primeras etapas de desarrollo de Mozilla). Ambos tipos de elemento tenían el atributo "*src*" que podía tomar cualquier URL externa, y cargando una página que contenga *javascript* que manipule la página paterna, se lograban efectos parecidos a los de AJAX.

El *Microsoft's Remote Scripting* (o MSRS, introducido en 1998) resultó un sustituto más elegante para estas técnicas, con envío de datos a través de un *applet* Java el cual se podía comunicar con el cliente usando *JavaScript*. Esta técnica funcionó en ambos navegadores, Internet Explorer versión 4 y Netscape Navigator versión 4. Microsoft la utilizó en el Outlook Web Access provisto con la versión 2000 de Microsoft Exchange Server.

La comunidad de desarrolladores web, primero colaborando por medio del grupo de noticias *microsoft.public.scripting.remote* y después utilizando blogs, desarrollaron una gama de técnicas de *scripting* remoto para conseguir los mismos resultados en diferentes navegadores. Los primeros ejemplos incluyen la biblioteca JSRS en el año 2000, la introducción a la técnica imagen/*cookie* en el mismo año y la técnica *JavaScript* bajo demanda (*JavaScript on Demand*) en 2002. En ese año se realizó una modificación por parte de la comunidad de usuarios al *Microsoft's Remote Scripting* para reemplazar el *applet* Java por el objeto *XMLHttpRequest*.

Frameworks de *Scripting* Remoto como el ARSCIF aparecieron en 2003 poco antes de que Microsoft introdujera *Callbacks* en ASP .NET.

Desde que *XMLHttpRequest* está implementado en la mayoría de los navegadores, raramente se usan técnicas alternativas. Sin embargo, todavía se utilizan donde se requiere una mayor compatibilidad, una reducida implementación, o acceso cruzado entre sitios web.

2.5.5. XML

Como se ve en [13] y [14] XML, siglas en inglés de *eXtensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML proviene de un lenguaje creado por IBM en la década del setenta, llamado GML (*Generalized Markup Language*). Este lenguaje surgió por la necesidad de la empresa de almacenar grandes cantidades de información. En su momento resultó interesante para la ISO, por lo que en 1986 se comenzó a trabajar para normalizarlo, creando SGML (*Standard Generalized Markup Language*), capaz de adaptarse a un gran abanico de problemas. A partir de éste se han creado otros sistemas para almacenar información.

En el año 1989 Tim Berners Lee creó la web, y junto con ella el lenguaje HTML. Este lenguaje se definió en el marco de SGML y comenzó a ser la aplicación más conocida de este estándar. Los navegadores web, sin embargo, siempre han puesto pocas exigencias al código HTML que interpretan, generando así cierto caos en algunas páginas web que no cumplen con la especificación de la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación de HTML es que cada documento pertenece a un vocabulario fijo, establecido por un DTD. No se pueden combinar elementos de diferentes vocabularios. Asimismo es imposible para un intérprete (por ejemplo un navegador) analizar el documento sin tener conocimiento de su gramática (DTD).

Se buscó entonces definir un subconjunto de SGML que permita:

- Mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles.
- La creación de analizadores simples, sin ninguna lógica especial para cada lenguaje.
- Empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis.

Para lograr esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos. XML, en cambio, está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

A continuación se enumeran algunas ventajas de XML:

- Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla.
- Mejora la compatibilidad entre aplicaciones. Podemos comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos.
- Transformación de datos en información, se le añade un significado concreto y se lo asocia a un contexto, con lo cual se obtiene flexibilidad para estructurar los documentos.

2.5.5.1. Estructura de los documentos XML

La tecnología XML busca dar una solución al problema de expresar información, estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y esas partes se componen a su vez de otras partes. Entonces, se obtiene un árbol de porciones de información. Estas partes se llaman elementos y se las señala mediante etiquetas.
Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma *<nombre>*, donde *nombre* es el nombre del elemento que se está señalando.

2.5.5.2. Documentos XML bien formados

Los documentos denominados como "bien formados" son aquellos que cumplen con todas las definiciones básicas de formato y pueden analizarse correctamente por cualquier *parser* que cumpla con la norma.

- Los documentos deben seguir una estructura estrictamente jerárquica respecto a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, sólo pueden tener un elemento inicial.

- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos "entendibles" por las personas.

2.5.5.3. Validez

Que un documento esté bien formado se refiere, solamente, a su estructura sintáctica básica. Esto es, que esté compuesto de elementos, atributos y comentarios como XML especifica que se escriban. Ahora bien, cada aplicación de XML, es decir, cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento.

Esta relación entre elementos se especifica en un documento externo o definición (expresada como DTD [*Document Type Definition*] o como XSchema). Crear una definición equivale a crear un nuevo lenguaje de marcado, para una aplicación específica.

2.5.6. XSLT

En [15] y [16] se define a XSLT (*Extensible Stylesheet Language Transformations*) como un lenguaje declarativo basado en XML utilizado para realizar transformaciones de un documento XML a otro. El documento original no recibe modificación alguna, por el contrario, un nuevo documento es creado basándose en el contenido del documento existente. El documento nuevo puede ser serializado por el procesador en una sintaxis estándar XML o en otro formato como HTML o texto plano. XSLT suele ser utilizado como un convertidor de datos XML en documentos HTML o XHTML que serán visualizados en una página web; esta transformación puede suceder tanto en el servidor como en el cliente.

Otro uso que suele recibir XSLT es el de crear la salida para imprimir o visualizar un video, normalmente mediante la transformación del XML original en XSL Formatting Objects para crear una salida con un formato específico que luego podrá ser convertida a una serie de formatos como PDF, PostScript o PNG.

También puede usarse para traducir mensajes XML entre dos esquemas XML diferentes, o para modificar un documento dentro del alcance de un único esquema, por ejemplo eliminando las partes de un mensaje que no se necesitan.

El modelo de procesamiento de XSLT involucra:

- al menos un documento XML como dato de origen.
- al menos un módulo *stylesheet* XSLT.
- el procesador de *templates* XSLT.
- al menos un documento de resultado.

El procesador XSLT, normalmente, toma dos documentos de entrada, un documento XML y un módulo *stylesheet* XSLT, y produce un documento de salida. El *stylesheet* XSLT contiene una colección de reglas *template* que sirven de instrucciones para guiar al procesador en la producción del documento de salida.

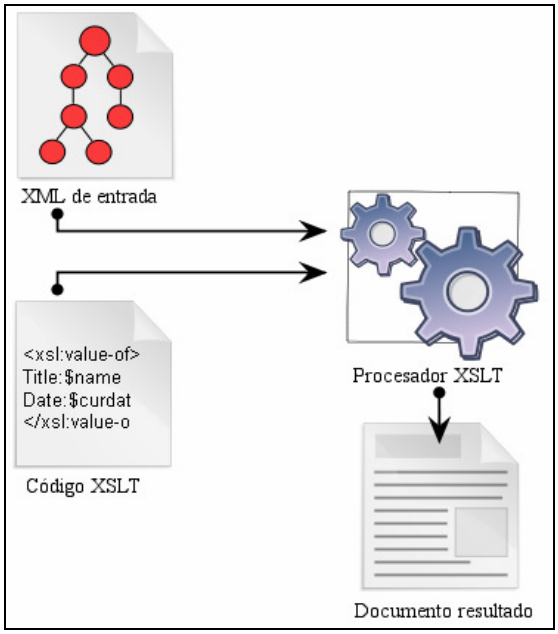

Imagen 2.8 – Diagrama de los elementos básicos y el flujo de proceso de una transformación XSLT

2.5.6.1. XPath

Como se menciona en [17] y [18] XML *Path Language* es un lenguaje de consultas diseñado y definido por la W3C para seleccionar nodos en un documento XML y para computar algunos valores (*strings*, números o booleanos) dentro del contenido del documento.

XPath está basado en una representación en forma de árbol del documento XML, y brinda la capacidad de navegar sobre este árbol seleccionando nodos mediante una variedad de criterios. Popularmente (aunque no sea así en la especificación oficial), una expresión XPath suele ser referida, simplemente, como un XPath.

Surgido originalmente para proveer una sintaxis y comportamiento común entre XPointer y XSLT, subconjuntos del lenguaje de consultas XPath son utilizados en otras especificaciones de la W3C como XML *Schema*, XForms e ITS (*Internationalization Tag Set*).

Existen actualmente dos versiones en uso de XPath:

- XPath 1.0 se convirtió en una recomendación a fines de 1999 y sigue siendo ampliamente utilizado e implementado, tanto por sí solo como embebido en leguajes como XSLT, XProc, XML *Schema* o XForms.
- XPath 2.0 es la versión actual del lenguaje y se convirtió en una recomendación a principios de 2007. Existe una gran cantidad de implementaciones, pero no son tan utilizadas como XPath 1.0. La nueva especificación es mucho más grande que su predecesora y posee modificaciones de conceptos básicos del lenguaje como el sistema de tipos.

XPath 2.0 es, en realidad, un subconjunto de Xquery 1.0. Ofrece una expresión *for* que es una reducción de las expresiones "*FLOWR*" en XQuery. Es posible describir el lenguaje listando las partes de XQuery que se eliminan. Los ejemplos principales son las consultas *prolog*, los constructores de elementos y atributos, el resto de la sintaxis del *FLOWR* y las expresiones de cambio de tipos.

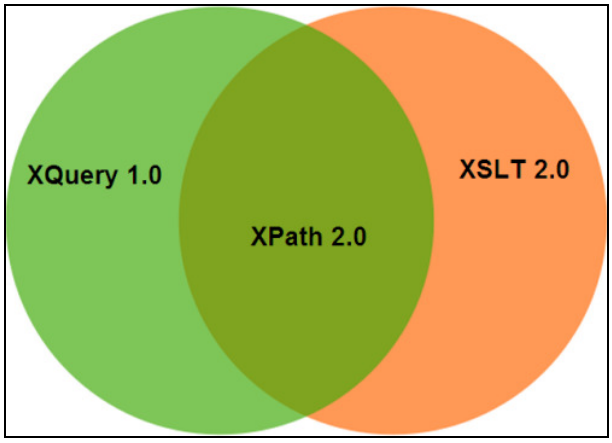

Imagen 2.9 – Diagrama de Venn representado la relación entre XQuery, XPath y XSLT

2.5.6.2. Procesamiento de las reglas template

El lenguaje XSLT es declarativo, las reglas template solo definen cómo manejar un nodo que coincide con un patrón XPath. Es por eso que el procesador sigue un algoritmo fijo en todas sus ejecuciones, pero varía los resultados en función de las reglas *template* utilizadas, que indican al procesador las operaciones que debe realizar.

Asumiendo que una *stylesheet* fue leída y preparada, el procesador construye un árbol fuente a partir del documento XML de entrada. Luego comienza a procesar los nodos a partir del nodo raíz del árbol, buscando en la *stylesheet* el *template* que mejor coincida con el nodo actual y evaluando el contenido del *template* sobre el nodo. Las instrucciones en cada *template* generalmente indican al procesador que cree nodos en el árbol resultado o que procese más nodos del árbol fuente de la misma forma que se hace con el nodo raíz. Finalmente, a partir del árbol resultado, se genera el documento de salida.

2.5.7. XUL

En [19] y [20] se define a XUL (XML *User Interface Language*) como un lenguaje de marcado XML de interfaz de usuario desarrollado por el proyecto Mozilla. XUL opera en aplicaciones *cross-plataforms* de Mozilla como Firefox y Flock. El motor de diseño de Mozilla Gecko provee una implementación de XUL utilizada en el navegador Firefox.

XUL se basa en las múltiples tecnologías y estándares de la web, incluyendo CSS, *JavaScript* y DOM. Esta relación hace que XUL sea relativamente sencillo de aprender para personas que posean conocimientos en el desarrollo, diseño y programación de páginas web.

XUL no posee una especificación formal y no interopera con implementaciones que no pertenezcan a Mozilla Gecko. Sin embargo, utiliza una implementación *open source* de Gecko bajo las licencias GPL, LGPL y MPL.

Mozilla provee un constructor experimental, XULRunner, para permitir a los desarrolladores realizar sus aplicaciones por encima del *framework* de aplicaciones Mozilla y de algún XUL en particular.

XUL posee definiciones portables de *widgets* comunes, permitiendo que estos se transporten fácilmente a cualquier plataforma en la cual correr aplicaciones Mozilla.

A pesar de que XUL, principalmente, se utiliza para construir aplicaciones Mozilla y sus extensiones, también puede ser utilizado en aplicaciones web transmitidas por HTTP. El *Mozilla Amazon Browser*, una antigua aplicación XUL de este tipo, proveía una interfaz para realizar búsquedas de libros en Amazon.com.

Sin embargo, la mayoría de las herramientas más poderosas de Mozilla (como los objetos XPCOM) no se encuentran disponibles para los documentos XUL sin privilegios, a menos que el *script* posea una firma digital y el usuario obtenga los permisos para los privilegios requeridos por la aplicación. Este tipo de documentos también involucra varias limitaciones del navegador, incluyendo la incapacidad de cargar XUL remotos, DTA y documentos RDF.

Como Gecko provee la única implementación completa de XUL, estas aplicaciones se mantienen inaccesibles para los usuarios de navegadores no basados en Mozilla.

Otras aplicaciones que utilizan XUL son:

- El *framework* de interfaces de usuarios de código abierto Ample SDK, que provee una implementación multi-navegador de XUL en *JavaScript*.
- La IDE ActivateState Komodo utiliza XUL al igual que el proyecto *Open Komodo*.
- El reproductor de música Songbird y el reproductor de videos Miro.
- Cyclone3 CMS utiliza XUL, en su formato de extensión para Mozilla Firefox.
- El sistema de manejo de contenido Elixon WCMS/XUL utiliza exclusivamente XUL remoto, incluso resolviendo alguno de los problemas de los documentos remotos XUL previamente mencionados.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window id="vbox example" title="Ejemplo...."
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <vbox>
    <button id="yes" label="Sí"/>
    <button id="no" label="No"/>
    <button id="maybe" label="Quizás"/>
  </vbox>
</window>
```

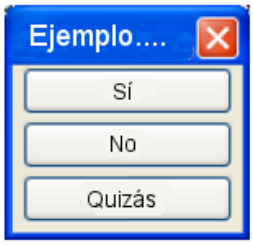

Imagen 2.10 – Ejemplo de código y aplicación resultante de un documento XUL

Capítulo 3

Administración de los tags en Firetag

En este capítulo se desarrollarán los conceptos involucrados en la forma de administración de los tags que un usuario posee en las distintas redes sociales. Se enunciará la idea general de cómo se realiza usualmente dicha manipulación y, luego, se modificará la idea abstracta planteando los criterios de interacción que debería seguir una aplicación con características como las de Firetag.

Como parte inicial de este capítulo, el autor definirá una categorización de los métodos mediante los cuales un usuario puede administrar sus tags. Dicha categorización se compone de dos grandes grupos: la administración directa y la administración indirecta. Este último grupo, a su vez, se divide en dos subgrupos: la administración indirecta centralizada y la administración indirecta distribuida.

Como eje central del capítulo, se explicarán en detalle los mecanismos de interacción que posee Firetag. Fundamentalmente se detallará la forma en que se implementa el mecanismo de interacción indirecta distribuida y las ventajas obtenidas al seguir un paradigma de interacción distinto al tradicional.

3.1 Categorización de los métodos de administración de tags

En esta sección se definirá la categorización que el autor considera necesaria para agrupar a los conjuntos de métodos de administración de tags utilizados por distintas aplicaciones. Esta categorización se define en el presente trabajo debido a la necesidad de contrastar las distintas formas de control de tags que puede utilizar el usuario, estableciendo las características y funcionalidades necesarias de cada una de ellas logrando, finalmente, definir el conjunto de servicios que deberán poseer las aplicaciones con características similares a Firetag. Éstos últimos serán definidos en el siguiente capítulo ya que resulta necesario realizar una descripción previa de las APIs web y los servicios de las mismas que han sido investigados y utilizados.

3.1.1. Administración directa

La administración directa es la más simple y tradicional de las formas mediante las cuales un usuario puede manipular su conjunto de tags dentro de una red social.

Como se ha descripto anteriormente, redes sociales como Delicious o Flickr permiten a los usuarios agregar uno o más tags a los objetos que ellos publican. Este proceso de tagueo puede realizarse tanto al momento de realizar la publicación como cuando el

objeto ya se encuentra publicado. Incluso, en algunos casos, un usuario puede agregar tags a objetos que no han sido publicados por él; pero para ello deben cumplirse una serie de requisitos de permisos y visibilidad establecidos por el usuario al que le pertenece el objeto publicado.

Esta forma de controlar los tags, es lo que se denomina administración directa. La manipulación se realiza de manera directa desde la aplicación web o red social que mantiene los tags, los objetos tagueados y los usuarios que realizan los tagueos. Es el mismo sitio web el que se encarga de brindar facilidades al usuario para que éste pueda agregar, eliminar o modificar su conjunto de tags asociado a distintos objetos.

A pesar de ser una forma de administrar tags muy utilizada y aceptada, tiene la gran desventaja de obligar al usuario a interactuar directamente con un sitio web distinto para manipular cada uno de sus conjuntos de tags; los cuales no poseen ningún vínculo más que el de la persona física que los maneja. Es el usuario el único responsable de mantener la coherencia entre sus tags, en cada una de las redes sociales en las que se involucra.

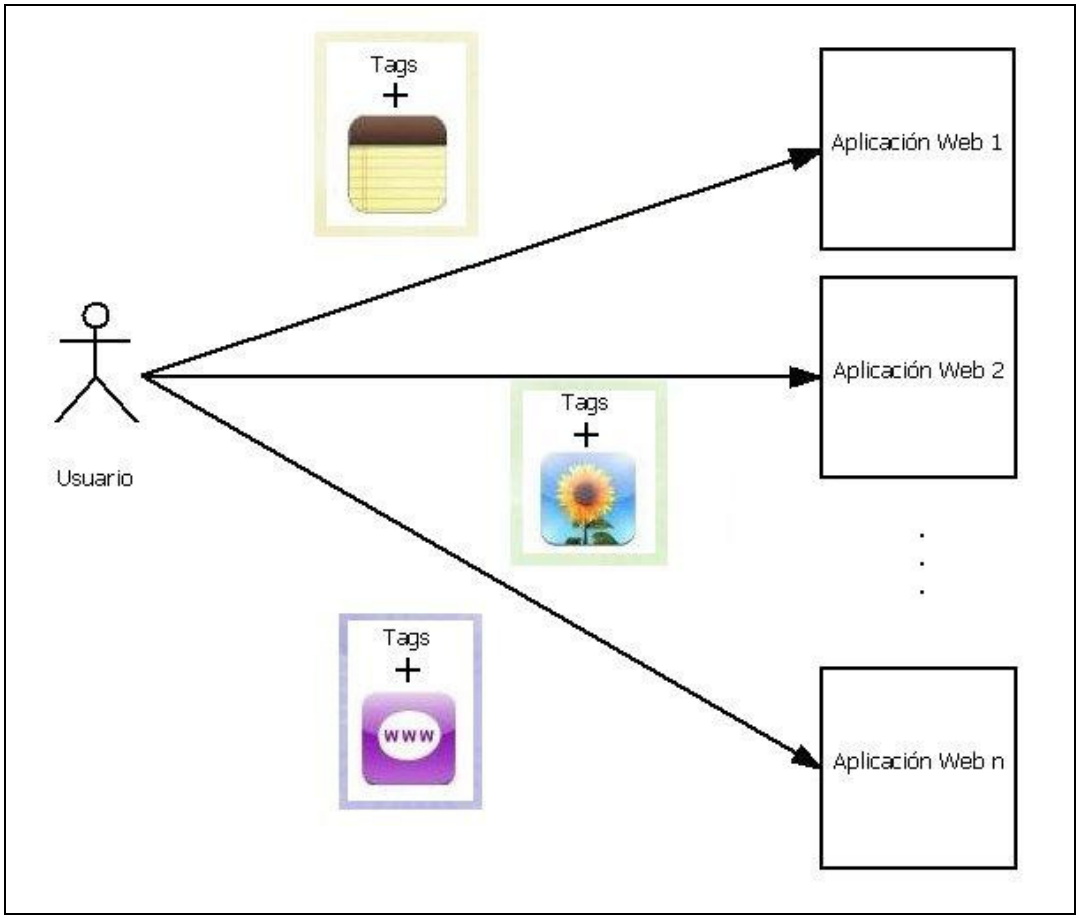

Imagen 3.1 – Administración directa por parte de un usuario hacia sus tags en distintas aplicaciones web

Como se muestra en la imagen 3.1, el usuario interactúa con las distintas aplicaciones web de manera directa. Esto es, al realizar una nueva publicación se envía un paquete con el objeto a ser publicado y el conjunto de tags que el usuario desea asociarle. No

existe ningún vínculo entre los paquetes dirigidos a distintas aplicaciones. Por ello, es el usuario el responsable de mantener coherentemente el conjunto de tags que utiliza; incluso si las publicaciones se han realizado en lapsos de tiempo muy grandes.
La principal desventaja de este estilo de administración de tags es, precisamente, la falta de relación entre los tags publicados en las distintas aplicaciones web. Al ser el usuario el encargado de relacionar coherentemente los tags utilizados en los distintos paquetes, la administración de éstos se vuelve compleja con el transcurso del tiempo. Si bien el usuario podrá mantener un grado de coherencia significativo en las publicaciones que realice en intervalos de tiempo relativamente cortos, cuando estos intervalos sean más grandes, la capacidad del usuario de mantener la coherencia en su vocabulario se verá seriamente afectada.

3.1.2. Administración indirecta

La administración indirecta de tags se distingue por una característica primordial: el usuario no interactúa directamente con la aplicación web que mantiene sus tags. Por el contrario, lo hace mediante una aplicación intermediaria que puede estar implementada como una aplicación de escritorio, una aplicación web, una aplicación para dispositivos móviles, o una extensión para el navegador web; y brindar funcionalidades extras que no están disponibles en la aplicación web en cuestión o simplemente permitir acceder a la cuenta del usuario de una manera distinta.
La aplicación intermediaria puede permitirle al usuario interactuar con varias aplicaciones de redes sociales o simplemente con una. En el presente trabajo, estos tipos de administración indirecta se definen como administración indirecta distribuida y administración indirecta centralizada, respectivamente.

3.1.2.1. Administración indirecta centralizada

Como se mencionó previamente, en la administración indirecta centralizada la aplicación intermediaria con la cual interactúa el usuario se relaciona sólo con una red de folksonomía.
No siempre se presentan características de mejora hacia el mecanismo de manipulación de tags de un usuario, pero al manejarse los tags de manera remota y no directamente sobre la plataforma que los almacena, puede brindársele al usuario mayores facilidades para modificar su conjunto de tags.
Normalmente, el usuario ingresa los datos pertinentes para realizar una publicación (el objeto a ser publicado, los tags y eventualmente los datos de identificación para acceder a la red social) y la aplicación intermedia accede a la red social mediante alguna API que ésta posea. Con el servicio específico de la API se realiza la publicación de la información que el usuario ha ingresado y se concluye la transacción informándole al usuario el resultado de la misma.

Esta interacción básica de una aplicación que brinde administración indirecta centralizada, puede incorporar algunas características extras. Por ejemplo: cuando el usuario ingresa los datos e indica que sean enviados, la aplicación podría comprobar, con un servicio de corrección ortográfica, errores en los tags ingresados y sugerirle al usuario ciertas modificaciones; o podría brindar un servicio de traducción, obteniendo la traducción a otro idioma de los tags mediante otro servicio remoto.

Las posibilidades son infinitas dado que cada día existen nuevos y mejores servicios web de distintas aplicaciones y distintas características, que permitirán agregar mejoras y características adicionales de diversa utilidad a una aplicación intermedia para el manejo de tags.

Las mejoras hacia el manejo de tags son claras, y si bien con algunas de las técnicas mencionadas pueden solucionarse errores de tipeo o de ortografía, sigue pendiente el manejo del vocabulario completo en las distintas redes en las que participa el usuario.

Como muestra la imagen 3.2, el usuario ingresa a la aplicación intermedia los datos necesarios (tags y objetos a publicar, aunque podría ser necesario datos de autenticación, como se mencionó previamente) y esta aplicación es la que se encarga de armar los paquetes de información y enviarlos a la aplicación web que publica y mantiene los datos del usuario, sus tags y sus objetos compartidos.

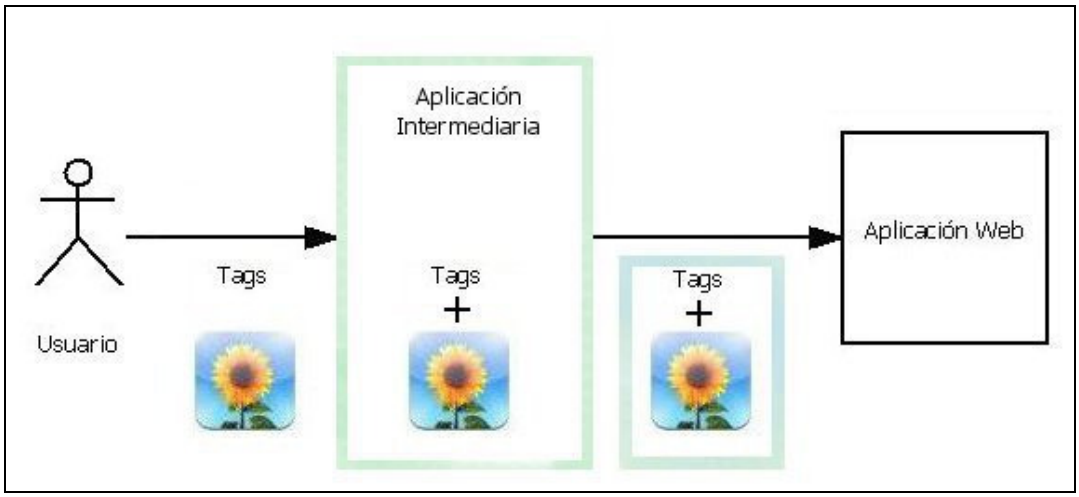

Imagen 3.2 – Administración indirecta centralizada de los tags de un usuario en una aplicación web particular

3.1.2.2. Administración indirecta distribuida

El caso de la administración indirecta distribuida de tags resulta la más interesante del presente trabajo. Este tipo particular de administración indirecta de tags podría considerarse como un subconjunto de la administración indirecta centralizada, donde sus elementos en lugar de interactuar entre el usuario y una red de folksonomías lo hacen con un conjunto de estas aplicaciones.

Un usuario, normalmente, interactúa con varios sitios que permiten realizar tagueos sobre los objetos que se comparten. Estos sitios, en general, permitirán compartir distintos tipos

de objetos; el usuario deberá, en consecuencia, trabajar con un sitio distinto cada vez que quiera realizar una publicación de un tipo de objeto en particular.

Cuando el usuario interactúa con una aplicación que brinda una administración de tags indirecta y distribuida, es la aplicación la que se encarga de conectarse y publicar los objetos en las distintas redes sociales. El usuario sólo se preocupa por seleccionar los datos que desea enviar (básicamente objetos y tags) e ingresar sus datos de autenticación, conforme la aplicación lo requiera. Por lo tanto, el usuario realiza la administración de los tags de manera análoga para todos los conjuntos de tags que la aplicación le permita manipular.

La mayor ventaja de este tipo de aplicaciones es que permite conocer el vocabulario del usuario en un porcentaje mayor. No sólo se conocen los tags del usuario en una folksonomía, sino que en varias de ellas.

Esta ventaja otorga a la aplicación la capacidad de realizar recomendaciones al usuario basándose en datos más extensos, de mayor poder de cobertura que los datos de una única red. Si la aplicación aprovecha estos datos, se puede brindar al usuario la ayuda necesaria para que éste mantenga un grado de coherencia mayor en su vocabulario; evitando errores, confusiones o selecciones poco específicas de los tags que se asociarán a un objeto.

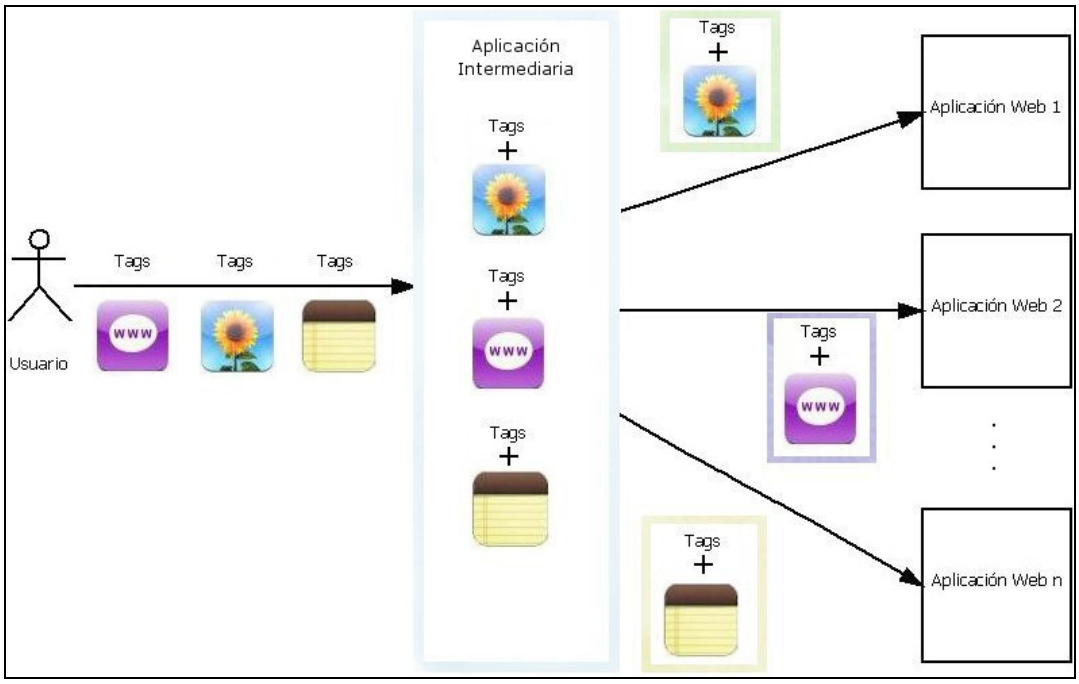

Imagen 3.3 – Administración indirecta distribuida de los tags de un usuario en distintas aplicaciones web

La imagen 3.3 refleja la interacción entre un usuario y las aplicaciones web en las que participa.

El usuario sólo participa activamente con la aplicación intermediaria, enviándole los objetos de distinto tipo junto a los conjuntos de tags que desea asociarle a cada uno de ellos.

La aplicación intermediaria se encarga de armar los paquetes de información, que luego serán enviados a la aplicación que corresponda. Normalmente, será necesario que el usuario, además, ingrese en algún punto de la transacción sus datos de identificación; estos requerimientos varían y dependen pura y exclusivamente de cada una de las aplicaciones web.

Finalmente, la aplicación intermediaria recibirá por parte de las aplicaciones con las que está interactuando, un mensaje de respuesta indicando el resultado de la operación. Deberá, entonces, informar al usuario del éxito de la transacción o del error y sus posibles causas.

3.2. Mecanismo de interacción de Firetag

Firetag es una extensión de Mozilla Firefox que permite manipular de manera conjunta los tags que un usuario utiliza en las cuentas que éste posee en Delicious, Flickr y Bibsonomy. Si bien permite realizar varias operaciones, como la agregación o eliminación de publicaciones y tags, el concepto más interesante que se plantea desde su desarrollo es el de permitirle a los usuarios administrar los tags utilizados en distintas aplicaciones web desde una única aplicación. Esta característica brinda la posibilidad de realizar recomendaciones de tags basándose en todo el conjunto de tags que el usuario ha utilizado en sus perfiles; generando recomendaciones que serán más adecuadas al usuario y no al contexto de la red social con la que se está interactuando.

El mecanismo de interacción de Firetag debe ser agrupado dentro de la categoría de administración indirecta distribuida de tags.

Al permitirle al usuario interactuar desde la misma aplicación con distintas aplicaciones web como Flickr, Delicious y Bibsonomy, y al mantener la comunicación en manos de la extensión misma y no del usuario, Firetag reúne las características principales enunciadas anteriormente. Estas características son las que ubican a la aplicación desarrollada dentro de la mencionada categoría.

Firetag logra brindar estos servicios al usuario mediante la comunicación con las distintas aplicaciones web, utilizando las API que éstas poseen para intercambiar datos o información a través de JavaScript asincrónico y obteniendo los resultados en formato XML. En otras palabras, AJAX.

Obtenido el resultado XML, se presenta el problema de interpretarlo. Esto se debe a que cada una de las distintas APIs utiliza un formato propio y no unificado de respuesta. Para solucionar este problema de manera escalable (ya que se agrega complejidad si se considera el eventual cambio en una API y en su formato de respuesta) se optó por utilizar el mismo algoritmo de interpretación de datos para todos los XML, variando solamente en la utilización de un archivo XSLT distinto para cada uno de los formatos.

Luego de la interpretación de los resultados, los datos quedan transformados en XUL, información que es interpretada automáticamente por el navegador Firefox; sólo resta agregar los datos al nodo XUL correspondiente de la aplicación y el usuario recibirá su resultado.

3.3. Arquitectura de comunicación

El intercambio más complejo de información que realiza Firetag ocurre en el momento de obtener los tags que el usuario mantiene en las distintas folksonomías de las que forma parte.

Como se describió anteriormente, se obtiene información que no posee el mismo formato, y dicha información podría, circunstancialmente, recibirse en un formato distinto debido a cambios en los servicios web de las aplicaciones con las que se interactúa.

Se plantea a continuación la arquitectura de comunicación utilizada para resolver este problema de manera escalable y confiable.

Para resolver la comunicación de salida, es decir, las situaciones en las que el usuario realiza una publicación o una modificación a una publicación existente o a su conjunto de tags, la interacción es relativamente más sencilla en lo que a interpretación de resultados se refiere. Sin embargo, estas actividades requieren de la identificación del usuario, lo que conlleva a manejar distintos tipos de datos y métodos de autenticación de los datos del mismo.

Para solucionar este problema, se optó por interactuar de manera distinta acorde a lo exigido por cada aplicación web; siempre manteniendo el mismo método de comunicación (vía AJAX); pero cambiando en la forma según los requerimientos de cada API. De manera similar a lo planteado en [21] se define la siguiente arquitectura de comunicación.

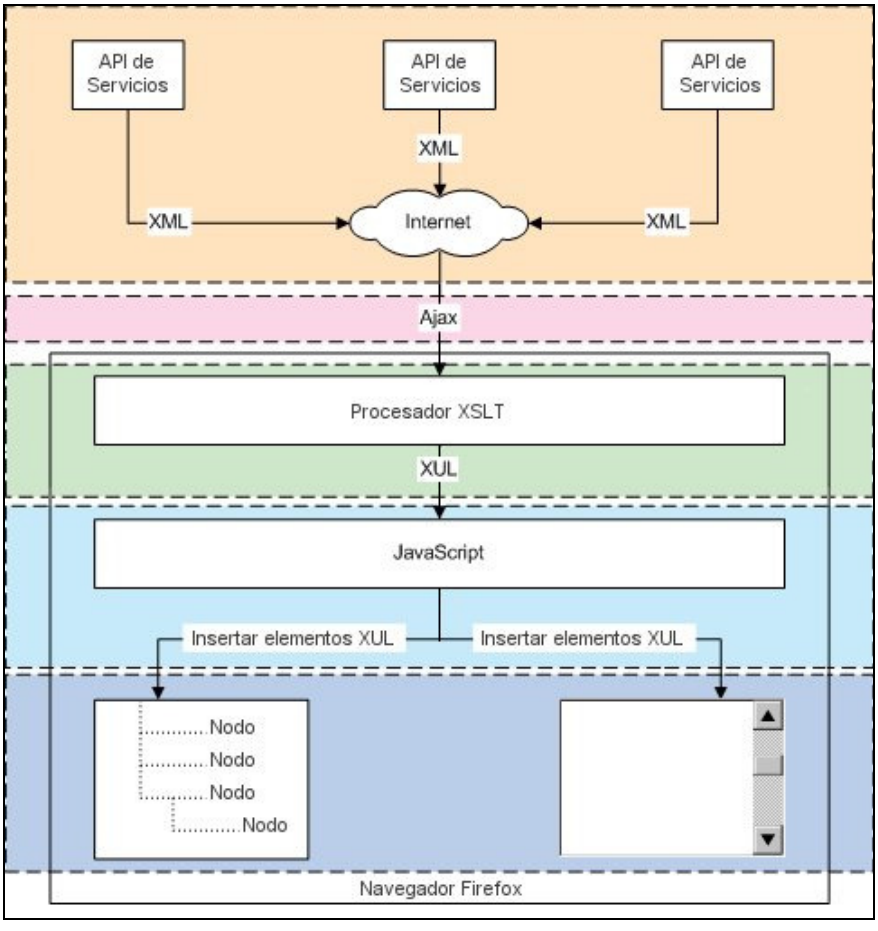

Imagen 3.4 – Esquema de la arquitectura de comunicación de entrada de Firetag

3.4. Comunicación de entrada

Se mencionó anteriormente la complejidad presentada por el mecanismo de comunicación de entrada en Firetag. En esta sección se describirá en detalle los distintos pasos que abarca dicho proceso.

3.4.1. API de servicios y XML vía Internet

Esta sección de la arquitectura se encuentra implementada en las aplicaciones web que mantienen la información del usuario. Cada una de ellas brinda una API que se utiliza desde Firetag para recibir la información deseada en formato XML.

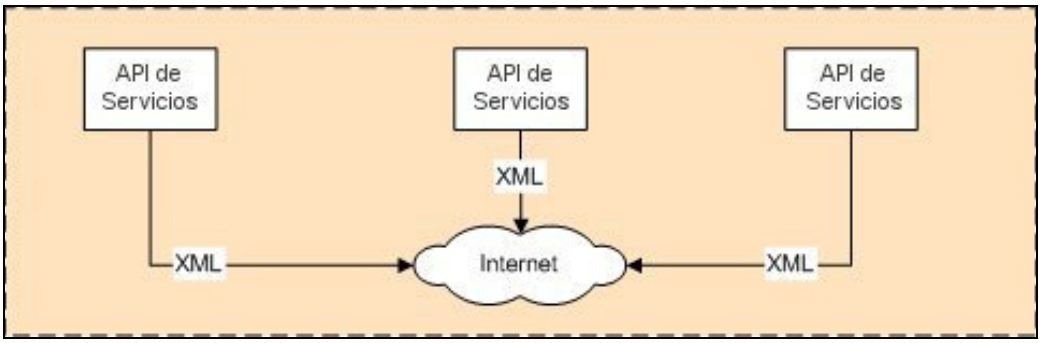

Imagen 3.5 – Porción de la arquitectura de comunicación de Firetag que representa las APIs de servicios web

Como muestra la imagen 3.5, las APIs de servicios envían en formato XML la información solicitada a través de Internet. Esto será recibido mediante AJAX, situación que se describe en la sección siguiente.

3.4.2. Comunicación AJAX

La comunicación AJAX resulta de suma importancia dentro de la arquitectura de comunicación de Firetag. Es el mecanismo que permite comunicar a la aplicación con los servicios web brindados en las distintas APIs.

Mediante la utilización del objeto XMLHttpRequest, se envía una petición remota de un servicio de una de las APIs; además de la URL del servicio, deben enviarse los parámetros necesarios para la autenticación del usuario, los cuales acarrean cierta complejidad en algunos casos. En el capítulo 4 se detallará el funcionamiento de las APIs y sus mecanismos de autenticación.

Una vez resuelta la URL con los parámetros correspondientes según el caso, se obtiene una respuesta, la cual se procesa invocando a una función manejadora que se indica a través del atributo *onreadystatechange* de XMLHttpRequest. Esta función es la que se encarga, en caso de que los datos de respuesta hayan llegado en forma correcta, de recibir la respuesta y enviarla para ser procesada por el archivo XSLT correspondiente.

A continuación se muestra una porción de código que permite ejemplificar el comportamiento descripto anteriormente.

```
/**Initializes and sends the request needed to get the tags at
Flickr.*/
function loadFlickrTags(){
      var userId=getElement("flickrUserId").value;
      initialize(flickrBaseURL+flickrGetMethod+
                  "&api_key="+flickrApiKey+"&user_id="+userId,
                  "chrome://firetag/content/xslt/flickr.xslt");
      sendXsltRequest();
}

/**Actually sends the XMLHttpRequest to get the XSLT file.*/
function sendXsltRequest(){
      requestXslt.overrideMimeType('text/xml');
      requestXslt.open("GET", xsltPath, true);
      requestXslt.onreadystatechange=processAjaxRequestXslt;
      requestXslt.send(null);
}

/**Processes the ajax request used for getting the XSLT file.*/
function processAjaxRequestXslt(){
      if (requestXslt.readyState==4){
            transform=requestXslt.responseXML;
            sendRequest();
   }
}

/**Actually sends the XMLHttpRequest.*/
function sendRequest(){
      request.overrideMimeType('text/xml');
      request.open("GET", url, true);
      request.onreadystatechange=loadTransform;
      request.send(null);
}
```

El código anterior presenta cuatro funciones que son las encargadas de realizar la solicitud a una de las APIs (en este caso, la API de Flickr), cargar el archivo que utilizará el procesador XSLT, y recibir los datos enviados por la API.

La función *loadFlickrTags* toma los datos de entrada ingresados por el usuario en el formulario correspondiente de Firetag, inicializa los valores de las distintas variables que serán utilizadas con posterioridad e invoca a la función *sendXsltRequest*.

Esta última función se encarga de obtener el archivo XSLT que será utilizado, cargando los valores necesarios en un objeto *XMLHttpRequest* (referido en la función como *requestXslt*) y luego realiza el envío de la petición mediante la función *send* de dicho objeto.

Cuando *requestXslt* realiza la petición y modifica su estado a *ready* se ejecuta el manejador de dicho estado, la función *processAjaxRequestXslt*. El manejador se ocupa de verificar que el estado sea válido y, de ser así, almacena el resultado obtenido (el contenido del archivo XSLT correspondiente) e invoca a la función *sendRequest*.

Es esta última función la que realizará, realmente, la petición a la API de Flickr. Primero inicializará otro objeto *XMLHttpRequest*, en este caso alojado en la variable *request*, y luego enviará la solicitud mediante la función *send*.

La interpretación del resultado y su transformación mediante los datos obtenidos del archivo XSLT corresponden a la sección siguiente. En ella se describirá su funcionamiento.

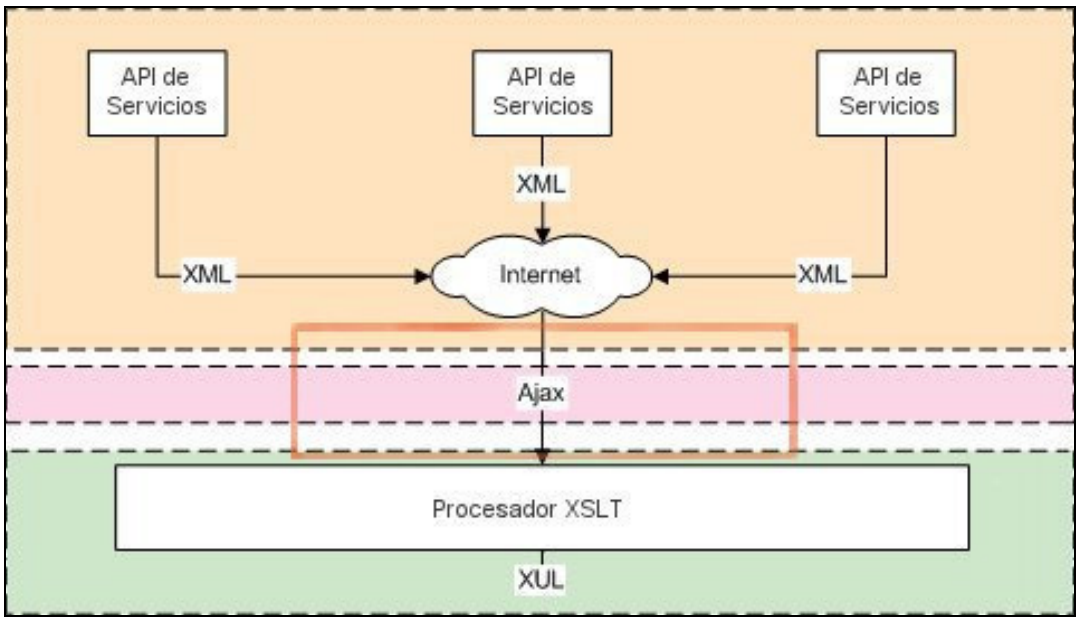

Imagen 3.6 – Fragmento de la arquitectura que muestra la comunicación vía AJAX. El recuadro rojo indica la parte que se refleja en el código citado anteriormente

3.4.3. Conversión a XUL

Uno de los problemas más complejos a resolver a la hora de recibir información de distintas redes sociales, es el de poder estandarizar la interpretación de los resultados.

Cada una de las redes enviará los datos correspondientes a la petición realizada en un formato propio, el cual no necesariamente coincidirá con los formatos de los demás sitios.
Entonces, se plantea el siguiente dilema: ¿Cómo interpretar de manera unívoca los datos recibidos por las distintas redes sociales?
La pregunta anterior tiene un anexo, ¿resulta nuestro método de interpretación de resultados escalable? Es decir, si el día de mañana necesitáramos interpretar datos de alguna otra red en algún otro formato, ¿podríamos hacerlo con facilidad y sin necesidad de modificar nuestro intérprete de datos?
La solución a tales problemas que se plantea en este trabajo consiste en la conversión de los datos recibidos en formato XML a formato XUL.
¿Qué significa esto? Simplemente, convertir cada uno de los datos recibidos en una porción de datos que sea interpretable por la extensión del navegador.
La forma más sencilla de realizar esto es mediante la utilización de archivos XSLT, los cuales brindan la posibilidad de convertir datos XML en un formato particular a otro formato XML. Siendo XUL un lenguaje XML, la conversión puede realizar sin mayores inconvenientes.
Esta técnica mencionada suele utilizarse para convertir, en aplicaciones web, datos XML en código HTML; el cual es interpretable por cualquier navegador web y puede ser visualizado directamente.

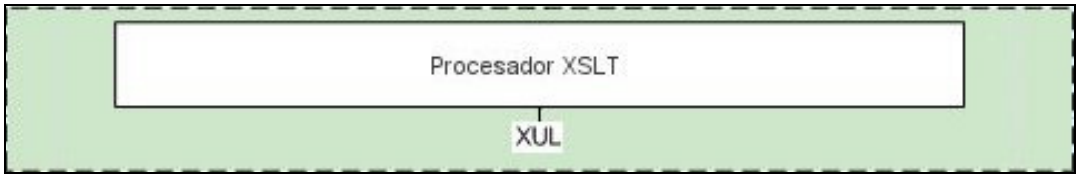

Imagen 3.7 – Extracto de la arquitectura que muestra la conversión de los datos recibidos a XUL

Como muestra la imagen 3.7, el procesador XSLT se encargará de convertir los datos recibidos en XUL. Los datos llegan mediante una comunicación AJAX con las distintas redes sociales, como se describió anteriormente.
A continuación se muestra una porción de código que resume el comportamiento descripto previamente. El código ha sido simplificado respecto al original, para evitar detalles técnicos que dificultan su interpretación, pero no modifican el concepto general que se intenta desarrollar.

```
/**Triggered once the XSLT document is loaded*/
function loadTransform(){
	try {
		processor.importStylesheet(transform);
	}
	catch(e) {
		setMessage("Error: Couldn't load the data.");
	}
	processXML("lista");
	invokeNextLoading();
}

/**Reflectively calls the next method to invoke for getting the
tags.*/
function invokeNextLoading(){
	if(nextMethod!=null){
		callNextMethod();
	}
}
/**Generic function to process the retrieved XML.*/
function processXML(id) {
	var xulItems;
	if(request.responseXML!=null){
		try {
	xulItems=processor.transformToDocument(request.responseXML);
		}
		catch(e) {
		setMessage("Error: Couldn't process received data.");
		}
	}
	var node=getElement(id);
	renderXUL(xulItems.firstChild.childNodes, node);
}
```

En el extracto de código mostrado se definen tres funciones, las cuales forman parte del mecanismo principal de transformación de datos XML a XUL.

La función loadTransform, que es invocada luego de que se recibe la respuesta a la solicitud de datos realizada en la función sendRequest (ver extracto de código en página 47), se encarga de asignar el archivo XSLT correspondiente al objeto *XSLTProcessor* llamado *processor*. El archivo XSLT que se necesita se obtiene mediante la función *processAjaxRequestXslt*, también descripta en la página 47.

Una vez establecido el archivo XSLT, se invoca a la función *processXML*. Esta función, luego de controlar que los datos XML que deben ser transformados son correctos, le indica al objeto *XSLTProcessor* que realice la transformación del documento, invocando a la función *transformToDocument* y enviándole como parámetro el contenido del documento XML que se desea transformar.

Una vez transformado los datos, se almacenan en la variable *xulItems*, la cual será el parámetro de la invocación a la función *renderXUL*. Esta última se encargará de mostrar los datos obtenidos dentro de la extensión en el navegador. Su funcionamiento en detalle se describe en la sección posterior.

La transformación de los datos a un formato interpretable por el navegador se realiza de manera transparente en el código, es decir, el mismo no depende de los distintos formatos con los que se puedan recibir los datos desde los servicios web.

Como ya se mencionó, esto se logra a través de la transformación XSLT, que depende de los archivos con formato XSLT los cuales le dan al XSLT *processor* la información necesaria para convertir los datos al esquema deseado.

A continuación se muestra un ejemplo de archivo XSLT. Este se utiliza para convertir los datos de los tags de un usuario recibidos desde Delicious.

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="xml" version="1.0" encoding="ISO-8859-1"
indent="yes"/>
      <xsl:template match="/">
            <pre>
                  <xsl:for-each select="tags/tag">
                        <listitem>
                              <listcell>
                                    <xsl:value-of select="@tag"/>
                              </listcell>
                              <listcell>
                                    <xsl:value-of select="@count"/>
                              </listcell>
                        </listitem>
                  </xsl:for-each>
            </pre>
      </xsl:template>
</xsl:stylesheet>
```

El archivo XSLT descripto indica, principalmente, dos tipos de información. En primer lugar se define el recorrido que tendrá que hacer el XSLT *processor* en el conjunto de datos de origen y en segundo lugar de qué manera se convertirán los datos.
En el tag *xsl:for-each* se indica que se deben recorrer todas las ramas que cumplan con el patrón de anidamiento "*tags/tag*" y en cada uno de estos elementos que se encuentren se deberá crear un nodo *listitem* que contenga dos nodos *listcell* donde cada uno de estos alojará la información correspondiente a los atributos *tag* y *count* encontrados en el respectivo nodo *tag* del conjunto de datos original.
Los demás archivos XSLT utilizados son análogos, sólo cambia la forma de recorrer el árbol de datos en función de su estructura.
Si se necesitara interactuar con un servicio web distinto que defina una estructura diferente, o si alguno de los servicios utilizados modificase el formato de respuesta; bastaría con agregar un archivo XSLT que contemple el formato del servicio nuevo, o con modificar el archivo correspondiente para que éste sea capaz de interpretar el nuevo formato de datos.

3.4.4. Inserción de elementos

Ya se ha definido, hasta el momento, el mecanismo para obtener los datos desde los servicios web remotos y procesarlos de manera abstracta para obtener un formato que sea interpretable por Mozilla Firefox. Para completar la transacción de obtención de datos desde una aplicación web remota, sólo resta insertar los datos obtenidos en nuestra aplicación.
Al obtener un documento DOM en formato XUL luego de convertir los datos mediante el XSLT *processor*; se podría suponer que, idealmente, sólo basta con insertar el documento en el punto de entrada correspondiente dentro de la interfaz de nuestra aplicación. Desgraciadamente, al realizar esto, el documento DOM del navegador se actualiza correctamente, pero no así la interfaz que se visualiza.
Para solucionar el problema descripto, se optó por crear uno a uno los elementos XUL que se han obtenido en la transformación mediante XSLT, utilizando la función *JavaScritpt createElement*. Esta función toma como parámetro el nombre de un elemento XUL y lo crea dinámicamente.
Dado que ya se conocen los nombres de los elementos XUL y su estructura jerárquica, establecidos en el resultado de la transformación XSLT, se utilizó una función *JavaScript* recursiva para crear cada uno de los elementos requeridos.
La función que se utiliza, en alto nivel, consiste en recorrer el árbol XUL creado por el XSLT *processor*, y utilizar los nombres y atributos de los nodos que se van encontrando para crear el nuevo nodo XUL con los atributos requeridos.

A continuación se muestra un extracto del código que realiza la función previamente descripta. Este se encuentra modificado con respecto al código original para evitar detalles técnicos que dificulten su comprensión; sin embargo la idea de ejecución del algoritmo se mantiene inalterada.

```
/**Setups the nodes to be rendered and calls renderNode function*/
function renderXUL(topNodes, node) {
	if(topNodes!=null && node!=null){
		for (var i=0; i<topNodes.length; i++){
				mergedTags.add(topNodes[i]);
		}
		if(nextMethod==null){
			mergedTags.sort();
			for (var i=0; i<mergedTags.length; i++){
					renderNode(mergedTags.get(i), node);
			}
		}
	}
}

/**Renders each node recursively.*/
function renderNode(topNode, node) {
	var thisNode;
	if(topNode!=null && node!=null){
		//Base case, create a new text item with the same value
		if (isTextNode(topNode)) {
			thisNode=createTextItem(topNode.nodeValue);
		}
		//Otherwise, if we are dealing with a parent node,
		//we create an identical item and call recursively.
		else{
			thisNode=createItem(topNode.nodeName);
			var nodes=topNode.childNodes;
			for(var i=0; i<nodes.length; i++) {
				renderNode(nodes[i], thisNode);
			}
		}
		//Append the new child to the argument received node.
		node.appendChild(thisNode);
	}
}
```

En el código anterior se definen dos funciones que se encargan de desarrollar el comportamiento ya mencionado. La función *renderXul* recibe un conjunto de nodos, los cuales son el resultado de la conversión mediante XSLT, y un nodo extra que indica el lugar en donde se deberá agregar a la interfaz de la aplicación. De ser correctos estos parámetros, se agrega cada uno de los nodos incluidos en *topNodes* a un objeto TagsMerger referenciado por la variable *mergedTags*; este objeto se encarga de mantener los nodos que representan tags ordenados para su fácil acceso.

Una vez que no hay más nodos para agregar, se recorre el conjunto de nodos obtenido y se invoca a la función *renderNode* enviándole como parámetros cada uno de estos nodos y el nodo destino en la interfaz de usuario.

La función *renderNode* se define recursivamente, y es la que se encarga de crear los nuevos nodos y agregarlos en la vista. Si los parámetros son correctos, se evalúa que el nodo a copiar sea un nodo de texto; este es el caso base, en donde se deberá crear un nodo idéntico y finalmente agregarlo al nodo destino.

El caso recursivo se presenta cuando el nodo que se está recorriendo no es un nodo de texto. En esta situación se debe crear un nuevo nodo idéntico al nodo padre que se está procesando e invocar recursivamente a la función *renderNode* con cada uno de los hijos del nodo procesado y el nuevo nodo creado como nodo destino. Finalmente, al igual que en el caso base, se agrega el nodo creado al nodo destino; completando así el proceso de inserción de los datos provenientes de la transformación XSLT a la interfaz de usuario de la aplicación.

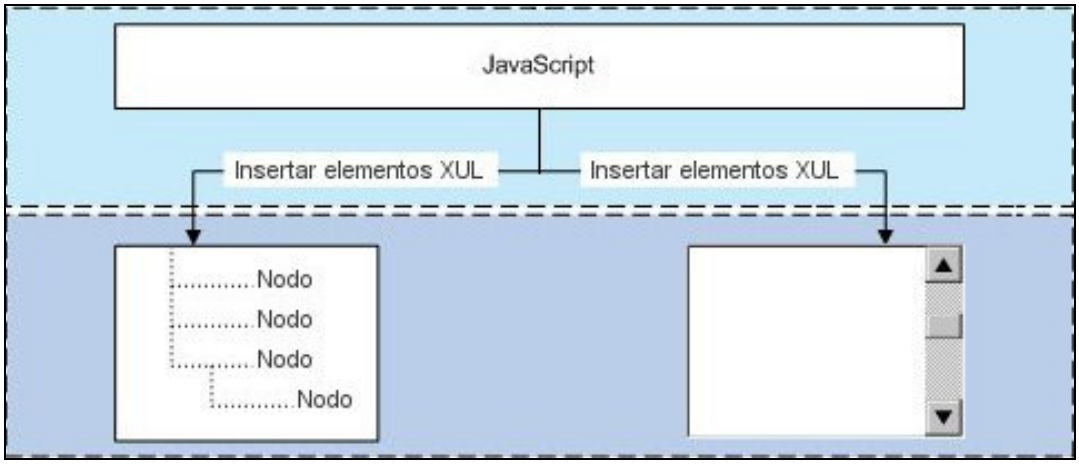

Imagen 3.8 – Porción de la arquitectura que refleja el proceso de inserción de los nodos obtenidos de la transformación XSLT a la interfaz mediante *JavaScript*

La imagen 3.8 refleja la etapa que se implementa con el código anterior, es decir, el momento en que los datos ya convertidos a XUL mediante un XSLT *processor*, se insertan en la aplicación para que puedan ser visualizados correctamente por el usuario.

3.5. Comunicación de salida

La comunicación de salida en Firetag resulta significativamente más sencilla que la comunicación de entrada.

Como se describió en secciones anteriores, la comunicación de entrada requiere de una arquitectura de cierta complejidad que contemple la solicitud de información, su recepción, su interpretación y su visualización.

Distinto es el caso de la comunicación de salida. Para realizar envíos de información desde Firetag hacia las distintas redes sociales sólo se requiere de la autenticación del usuario (quizás el paso más complejo de la transacción debido a la variación en los mecanismos de cada API), la invocación a la solicitud de envío de datos junto al conjunto de datos necesario y la recepción de la respuesta; que a diferencia de la comunicación de entrada, sólo indica si el envío de datos ha sido correcto o no.

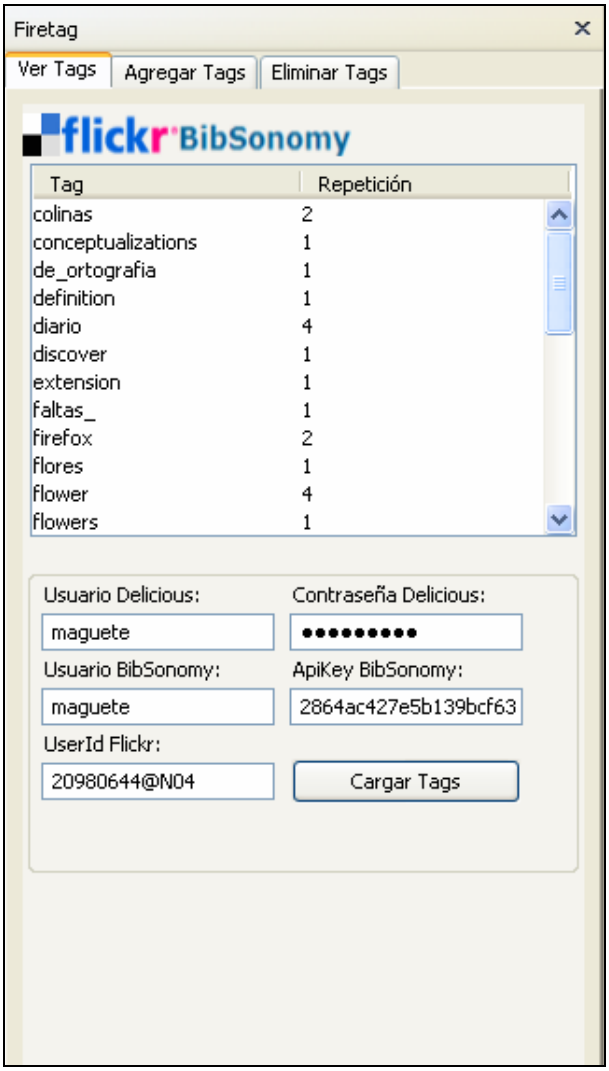

Imagen 3.9 – Captura de pantalla de Firetag que muestra los tags utilizados por el usuario en Delicious, Flickr y Bibsonomy

Lo más relevante dentro de la comunicación de salida de Firetag es el proceso de recomendación de tags que se realiza en el momento en que el usuario define el conjunto de tags que utilizará en una publicación.

Para ello se utiliza la información obtenida mediante la comunicación de entrada, es decir, la obtención de todos los tags que el usuario ha utilizado en las redes sociales.

La recomendación se realiza mediante un campo que brinda la posibilidad de autocompletar el término que se está escribiendo. Este campo obtiene la información de los tags y muestra las opciones que coinciden con lo que el usuario está escribiendo junto a la cantidad de veces que el usuario ha utilizado cada término.

En la imagen 3.11 se puede apreciar el conjunto de tags mediante el que Firetag se basará para realizar las recomendaciones al usuario en el momento en que éste realice una publicación.

Imagen 3.10 – Captura de pantalla de Firetag que muestra las recomendaciones que se le realizan al usuario

En la imagen 3.12 puede verse cómo se despliega un menú en el cual figuran los distintos tags que Firetag le recomienda al usuario cuando éste se encuentra completando los datos requeridos para realizar una publicación en Delicious.

Los tags que aparecen en el menú desplegable se definen en base al conjunto de tags que fueron cargados a la aplicación, como se muestra en la imagen 3.11. Además se realiza una selección de ese conjunto de tags en base a los caracteres que ha ingresado el usuario mientras escribe los tags que desea publicar. Para cada tag que escriba, aparecerá el menú desplegable con distinta información según el contenido ingresado.

Capítulo 4

Firetag y su interacción con las APIs web

En este capítulo se describirán los procesos básicos de interacción con las distintas APIs web utilizadas en la implementación de Firetag. Además de realizar una descripción de las características de cada una de las APIs, se desarrollarán los servicios que fueron necesarios utilizar para satisfacer los distintos requerimientos de la extensión desarrollada.

4.1 Interacción con Bibsonomy

Bibsonomy provee una API, definida en [22], con servicios web mediante el uso de REST (*Representational State Transfer*), un estilo arquitectónico de software para sistemas de hipermedia distribuidos en donde los servicios se acceden mediante los distintos métodos HTTP (*DELETE*, *GET*, *POST*, *PUT*) y los nombres de los servicios se definen representando a los objetos que modifican o con los que interactúan. Por ejemplo, si necesitáramos manipular las publicaciones (*posts*) de un usuario (*users*) en bibsonomy, accederíamos a una URL del estilo: www.bibsonomy.org/users/*username*/posts/*id*.

Dado que Bibsonomy, en particular, permite la publicación de referencias literarias o publicaciones científicas, se presenta el problema de detectar las publicaciones duplicadas; ya que existen muchas posibilidades en la manera en que los usuarios completarán los campos de sus publicaciones (nombre de la revista, autor, etc.). Por un lado resulta aceptable permitirle a los usuarios realizar publicaciones que difieran sólo en algunos detalles, pero por el otro lado una persona podría querer encontrar publicaciones de otros usuarios que refieran al mismo *paper* o al mismo libro aunque éstas no sean idénticas.

Para cumplir con estos objetivos, como se muestra en [23], la API de Bibsonomy implementa dos tipos de *hashes* que permiten comparar las publicaciones. La primera permite comparar las publicaciones de un único usuario (se la denomina *intra user hash*) y la segunda se utiliza para comparar las publicaciones de distintos usuarios (llamada *inter user hash*). La comparación es posible gracias a la normalización y concatenación de los campos BibTex y la posterior encriptación de estos datos mediante el algoritmo MD5.

La *intra user hash* es bastante estricta ya que considera los campos de autor, título de la publicación, editor, año, tipo de entrada, revista en la que fue publicada, título del libro en

el que fue publicada, volumen y número. Esto permite tener artículos con el mismo título, los mismos autores y en el mismo año pero con diferentes volúmenes (por ejemplo, un reporte técnico y el correspondiente artículo publicado en una revista).
En contraste a esto, la *inter user hash* es menos específica y sólo incluye el título, el año y el autor o editor (dependiendo de los datos que el usuario haya ingresado).
Con esta información y dado que los datos utilizados para realizar las *hashes* se ormalizan mediante la eliminación de espacios, signos, etcétera, se logra una comparación más precisa entre publicaciones similares o una unificación de las mismas en la búsqueda, según el usuario lo requiera.

4.1.1. Autenticación

El mecanismo de autenticación de Bibsonomy resulta bastante sencillo. Se implementa mediante autenticación http, esto es, la concatenación del nombre de usuario y la contraseña separados por el carácter ":" más el símbolo "@" al final.
Para evitar violaciones a la seguridad de la cuenta del usuario que está accediendo a la API, en lugar de enviarse la contraseña, se envía una clave de API, la cual es brindada por el sitio y cada usuario puede acceder a ella o solicitar una nueva dentro de su perfil.
No es un método por demás seguro, pero sí resulta sencilla su implementación y su acceso. Para lograrlo sólo basta con agregar al inicio de la URL correspondiente del servicio que se desea acceder, los datos descriptos anteriormente; para recibir como resultado la información pertinente al método accedido o la notificación de un error en su ejecución.

4.1.2. Servicios principales

La API de Bibsonomy brinda una gran cantidad de servicios útiles para la manipulación remota de la cuenta de un usuario y para el acceso a información de búsquedas dentro del sitio. A continuación se describe el conjunto de los métodos que se consideran de mayor importancia.

Agregar una publicación: le permite al usuario crear una nueva publicación. Para ello, como se muestra en [24], se accede a este servicio remoto mediante el envío de un método *POST* HTTP a la url **www.bibsonomy.org/api/users/[usuario]/posts**; en donde [usuario] debe reemplazarse por el nombre del usuario al cual se le agregará la publicación nueva.
A continuación se muestra un ejemplo de cómo debería formarse los datos para enviar en el método específico para agregar una publicación:

```
POST www.bibsonomy.org/api/users/maguete/posts

<?xml version="1.0"?>
<bibsonomy>
  <post description="El buscador por excelencia.">
    <user name="maguete"/>
    <tag name="tag1"/>
    <tag name="tag2"/>
    <tag name="tag3"/>
    <group name="public"/>
    <bookmark url="http://www.google.com.ar" title="Google"/>
  </post>
</bibsonomy>
```

Modificar una publicación: permite realizar una modificación sobre una publicación existente de un usuario. A diferencia del servicio anterior, como se muestra en [25], se utiliza el método *PUT* de http, pero la url de acceso es la misma.
Se deberá enviar el *PUT* a la url **www.bibsonomy.org/api/users/[usuario]/posts/[hash]**; en donde deberá reemplazarse [usuario] por el nombre del usuario correspondiente y [hash] por el valor de la hash que represente a la publicación que se desea modificar.

```
PUT www.bibsonomy.org/api/users/maguete/posts/aabbccddeeff01234

<?xml version="1.0"?>
<bibsonomy>
  <post description="El buscador por excelencia.">
    <user name="maguete"/>
    <tag name="tag1"/>
    <tag name="tag2"/>
    <tag name="tag3"/>
    <tag name="tag4"/>
    <bookmark url="http://www.google.com.ar"
      intrahash="ed646a3334ca891fd3467db131372140"
      interhash="ed646a3334ca891fd3467db131372140"/>
  </post>
</bibsonomy>
```

En este ejemplo puede verse cómo se agrega un tag a la publicación que había sido realizada en el ejemplo anterior. El servicio enviará, luego, una respuesta indicando si la modificación se realizó correctamente o hubo algún inconveniente en dicho proceso.

Eliminar una publicación: definido en [26], este servicio permite eliminar la publicación de un usuario. Se realiza mediante un método *DELETE* de http, llamando a la url: **www.bibsonomy.org/api/users/[usuario]/posts/[hash]**.
Como no se requiere más información que la del nombre usuario y la del hash de la publicación (indicadas ambas en la url de llamada), no es necesario armar un paquete de datos como en los servicios anteriores; en lugar de ello sólo basta con realizar una invocación de la siguiente manera:

```
DELETE www.bibsonomy.org/api/users/maguete/posts/aabbcdef01244432
```

Listar publicaciones: este servicio permite obtener un listado de todas las publicaciones que se encuentran en el sitio. Como se muestra en [27], mediante parámetros ubicados dentro de la url de llamada se puede especificar información de búsqueda como los tags que poseen, los grupos o usuarios a los que pertenecen, el tipo de publicación, etcétera.

```
GET
www.bibsonomy.org/api/posts?resource=aaaabbbbbccccc&start=0&end=1

<?xml version="1.0"?>
<bibsonomy>
  <posts start="0" end="1" next="
www.bibsonomy.org/api/posts?resource=aaaabbbbbccccc&start=2&end=3
   ">
    <post description="el buscador por excelencia"
          postingdate="2006-05-21T17:47+01:00">
      <user name="maguete"/>
      <tag name="tag1"/>
      <tag name="tag2"/>
      <tag name="tag3"/>
      <bookmark url="http://www.google.com.ar"
                intrahash="aaaabbbbbccccc"
                href="http://www.bibsonomy.org/api/
                     posts?resource=aaaabbbbbccccc"/>
    </post>
  </posts>
</bibsonomy>
```

La invocación a este servicio debe hacerse mediante el método *GET* de http, hacia la url **www.bibsonomy.org/api/posts**, agregando los parámetros necesarios para especificar la búsqueda.
En el ejemplo citado puede verse el resultado de solicitar el listado de publicaciones restringiendo la búsqueda por la hash de un recurso.

Listar tags: Utilizando este servicio, como se ve en [28], podemos obtener el conjunto de tags acorde a los parámetros que indicamos para la realización de la búsqueda. Se pueden filtrar tags mediante expresiones regulares o bien indicando a qué usuario o grupo pertenecen.
La llamada se debe hacer mediante el método *GET* de http hacia la siguiente url:
www.bibsonomy.org/api/tags.
Parámetros como ?filter=[regex] o ?user=[username] deberán agregarse para acotar los resultados de la búsqueda.

```
GET www.bibsonomy.org/api/tags?end=1&user=maguete

<?xml version="1.0"?>
<bibsonomy>
  <tags start="0" end="1" next=
"http://www.bibsonomy.org/api/users?start=2&end=3&user=maguete">
    <tag name="tag1" globalcount="13" usercount="5"/>
    <tag name="tag2" globalcount="23" usercount="15"/>
  </tags>
</bibsonomy>
```

En el ejemplo anterior puede verse el resultado de solicitar los tags del usuario “maguete” limitándolos a los que se encuentran entre la posición 2 (*start*) y la posición 3 (*end*).

4.1.3. Servicios utilizados por Firetag

La interacción que realiza Firetag con Bibsonomy es unidireccional. Esta red se utiliza sólo para obtener información de los tags empleados por el usuario y no se le permite al mismo realizar publicaciones o modificaciones a las publicaciones existentes dentro de Bibsonomy.
Si bien la interacción completa podría haberse logrado sin mayor complejidad, se prefirió dejar a un lado está funcionalidad debido a que se encontraba ya reflejada en la interacción con las otras aplicaciones web (Delicious y Flickr). La recolección de los tags utilizados por el usuario se mantuvo con el fin de obtener una mayor cantidad de

información para realizar las recomendaciones sobre los nuevos tags que utilizará el usuario.
Debido a esta reducción de la actividad con Bibsonomy, sólo se utiliza el servicio web que permite obtener los tags dentro del sistema. Invocando a la url www.bibsonomy.org/api/tags&user=[username] se obtiene como resultado todos los tags que ha utilizado el usuario; los mismos se tendrán en consideración al momento de recomendar un nuevo tag en una nueva publicación o en una publicación existente.

4.2 Interacción con Delicious

El sitio Delicious brinda acceso a una API de gran versatilidad y facilidad de uso definida en [29]. Esta se encuentra en desarrollo, es por ello que los servicios van cambiando y actualizándose, presentando mejoras en performance y seguridad constantemente.
En esta sección se describirá el funcionamiento básico de la interacción con esta API, junto a los principales servicios que la misma ofrece. Finalmente se detallarán los servicios utilizados dentro de la implementación de Firetag y la forma en la que éstos fueron empleados.

4.2.1. Autenticación

El mecanismo de autenticación de la API de Delicious, al igual que en el caso de Bibsonomy, se basa en la autenticación http. Como ya se mencionó, para acceder a un servicio que deba ser autenticado por este método, deberá incorporarse en la url el nombre del usuario junto a la contraseña separado por el carácter ":" y concatenarlo a la url junto a un símbolo "@".
La gran desventaja de la implementación de autenticación de Delicious es que no se utiliza ninguna encriptación de datos, por lo que la invocación a las urls deberá ser con la contraseña del usuario concatenada en forma literal. Aunque la comunicación se realiza mediante el protocolo https, la falta de encriptación en la contraseña deja un hueco en la seguridad de acceso.
Recientemente, Yahoo[7] incorporó la red Delicious dentro de sus servicios, por ello es que la API de este sitio (y en especial su mecanismo de autenticación) ha sufrido grandes cambios.

Con el fin de posibilitar a los usuarios acceder al sitio con su Yahoo ID, la empresa se vio obligada a agregar a la API el mecanismo de autenticación OAuth; mecanismo utilizado por Yahoo en sus redes para identificar de manera segura a los usuarios.
Esta actualización trajo grandes ventajas, siendo la principal la posibilidad de contar con un mecanismo de autenticación robusto que brinde a los usuarios seguridad a la hora de usar aplicaciones que interactúen con la API de Delicious. Por el otro lado, quedó desfasado el uso de la red de manera remota, ya que para los usuarios que poseían

[7] http://www.yahoo.com

cuentas sin Yahoo ID, la autenticación sigue siendo vía http y para los usuarios nuevos, vía OAuth. Esta situación acarrea un grado de complejidad en las aplicaciones que trabajaban con la API original, como es el caso de Firetag, en cuya implementación se optó por mantener el acceso sólo a los usuarios originales de Delicious.
La autenticación OAuth es utilizada también por la red Flickr (que también pertenece a Yahoo), en la siguiente sección se describirá la API de este sitio en detalle junto a dicho mecanismo de autenticación.

4.2.2. Servicios principales

Dentro de los servicios publicados por Delicious en su API, se encuentran los que se describen a continuación. Estos son los más importantes o al menos los más interesantes dentro del desarrollo del presente trabajo.
Además de los servicios que se detallarán en esta sección, Delicious brinda la posibilidad de manejar de manera remota lo que denominan "*bundles*" que no es más que un conjunto de tags agrupados por un nombre. Estas características y sus servicios asociados exceden a los contenidos del presente trabajo, es por ello que no serán tenidos en cuenta.

Obtener tags: este servicio, definido en [30], permite obtener la lista de los tags que han sido utilizados por el usuario autenticado. Además de los tags, la lista contiene la cantidad de veces que el usuario ha utilizado cada tag.
Para invocar a este servicio, es necesario realizar un envío http a la url **https://api.del.icio.us/v1/tags/get**, agregando los datos de autenticación del usuario, como se describió previamente.
Un ejemplo de la respuesta obtenida se muestra a continuación:

```
<tags>
  <tag count="1" tag="noticias" />
  <tag count="1" tag="diario" />
  <tag count="3" tag="deportes" />
  <tag count="5" tag="argentina" />
  <tag count="1" tag="fútbol" />
  <tag count="1" tag="tenis" />
</tags>
```

Eliminar un tag: como se indica en [31], mediante la utilización de este servicio se puede realizar la eliminación de todas las apariciones de un tag dentro del perfil de las publicaciones de un usuario. Para realizar este proceso debe enviarse un requerimiento http, siempre agregando los datos de autenticación del usuario, a la url

https://api.del.icio.us/v1/tags/delete?tag=[tag]. En el parámetro tag deberá enviarse el valor del tag que se desea eliminar.
La respuesta a este requerimiento será, simplemente, un documento que indique si la operación fue finalizada con éxito o si hubo algún error.

Renombrar un tag: con este servicio, como se muestra en [32], es posible modificar un tag. Esto significa, cambiar la palabra de un tag dentro del perfil de las publicaciones de un usuario por otra palabra distinta. Como es de esperarse, la modificación se realiza a nivel del tag y no de una asociación de un tag con una publicación, por lo tanto se modifican todas las asignaciones de este tag dentro del perfil de un usuario.
El requerimiento http para realizar esta operación debe hacerse a la url **https://api.del.icio.us/v1/tags/rename?old=[tag]&new=[tag]**, donde deben agregarse los datos de autenticación y enviar como valores de los parámetros old y new el término que se desea reemplazar y el nuevo término, respectivamente.
Al igual que en el servicio de eliminación de un tag, la respuesta es simplemente la confirmación o la indicación del error de la operación.

Obtener sugerencias de tags: un servicio más que interesante de la API de Delicious. Mediante su utilización se pueden obtener sugerencias de los tags más populares utilizados por los usuarios para una url en particular, según lo definido en [33].
Su utilización es sencilla y sólo requiere el envío de una solicitud http a la url **https://api.del.icio.us/v1/posts/suggest?url=[url]**. Además de los datos de autenticación del usuario, se debe enviar en el parámetro url el valor de la url para la cual se desea obtener tags recomendados.
Un ejemplo de la respuesta obtenida luego de invocar a este método es el siguiente:

```
https://api.del.icio.us/v1/posts/suggest?url=www.pagina12.com.ar
<suggest>
        <recommended>noticias</recommended>
        <recommended>diario</recommended>
        <popular>argentina</popular>
        <popular>diarios</popular>
        <popular>noticias</popular>
        <popular>diario</popular>
        <popular>news</popular>
        <network>for:alicia.diaz</network>
        <network>for:magic.towers</network>
</suggest>
```

Obtener publicaciones: este servicio, como se describe en [34], brinda la posibilidad de acceder a las publicaciones de un usuario, el que se ha autenticado, junto a una serie de restricciones definidas en parámetros de la url del servicio. Las publicaciones que se obtendrán corresponderán a un día en particular; si no se define el día de búsqueda, se utilizará el último en que se ha realizado una publicación
A continuación se muestra un ejemplo del resultado obtenido:

```
https://api.del.icio.us/v1/posts/get?tag=noticias+diario
<posts user="maguete" dt="2009-07-29t07:00:00" tag="noticias diario">
      <post href="http://www.la nacion.com.ar/"
           hash="29a89b7b547e2753602ff6c73a3d6e02"
           description="Diario La Nacion, Argentina."
           tag="diario noticias"
           time="2009-07-29T14:09:03Z"
           extended=""/>
      <post href="http://www.pagina12.com.ar/"
           hash="915b5874f5af3de58cd66b231ae2b91c"
           description="Diario Pagina12, Argentina."
           tag="diario noticias"
           time="2009-07-29T14:07:14Z"
           extended=""/>
      <post href="http://www.clarin.com.ar/"
           hash="a028a3eac7873b5fe9831ab259d02742"
           description="Diario Clarin, Argentina."
           tag="diario noticias"
           time="2009-07-29T13:59:24Z"
           extended=""/>
</posts>
```

Para su utilización debe invocarse a la url **https://api.del.icio.us/v1/posts/get?{parámetros}**. El conjunto de parámetros es opcional, y en el mismo se pueden utilizar algunos de los siguientes argumentos:

- Tag: restringe el resultado a las publicaciones que posean todos los tags enumerados. Debe agregarse a la url **&tag=[TAG]+[TAG]+...+[TAG]** reemplazando los valores de los tags que se deseen utilizar.
- Fecha: filtra los resultados a las publicaciones que se hayan realizado en la fecha indicada. Para su uso debe agregarse el parámetro **&dt=[YYYY-MM-DDThh:mm:ssZ]** en donde debe reemplazarse la fecha en el formato año-mes-día-hora-minutos-segundos.

- Url: permite obtener las publicaciones que se hayan realizado para una url en particular. Debe incorporarse el parámetro **&url=[URL]** y asignar en éste el valor de la url por la cual se desea realizar la búsqueda. Debe codificarse la url para que ésta no incluya espacios o caracteres especiales.

Eliminar una publicación: este servicio permite eliminar una publicación del usuario que se ha autenticado. Su utilización, como se define en [35], se hace mediante la invocación a un requerimiento http de la url **https://api.del.icio.us/v1/posts/delete?url=[url]**, en donde debe reemplazarse el valor del único parámetro que requiere por la dirección de la url que posee la publicación que se pretende eliminar.
La respuesta a la invocación a este servicio es, simplemente, un documento xml que indica si la operación fue correcta o no; indicando el error ocurrido si el caso fuera el último.

Agregar una publicación: para agregar una nueva publicación al conjunto de publicaciones del usuario autenticado, la API de Delicious brinda un servicio que permite realizar esta función. Para ello debe invocarse, según [36], mediante una solicitud http a la url **https://api.del.icio.us/v1/posts/add**. Esta url admite una serie de parámetros que se describen a continuación:

- Url: es obligatorio este parámetro, indica la dirección del *bookmark* que se publicará. Debe utilizarse con el formato **&url=[URL]**, reemplazando el valor del argumento por el de la url que se pretende publicar.
- Descripción: es el otro parámetro obligatorio, define la descripción que llevará la nueva publicación. Para definirlo se debe usar el parámetro **&descritpion=[…]** en donde se adjuntará el valor que se desee asociar a la publicación.
- Tags: es opcional utilizar este parámetro. Indica el conjunto de tags que serán vinculados a la publicación. Debe usarse agregando a la url el parámetro **&tags=[t1 t2 …]** reemplazando los valores de los tags que se utilizarán, separados por espacios.
- Reemplazar: también opcional. Indica si la publicación se reemplazará en caso de encontrarse una existente con el mismo *bookmark*. Debe agregarse el parámetro **&replace=no** si se desea dejar la publicación original. En caso de no definir este parámetro, se reemplazará la publicación existente.
- Compartido: otro de los parámetros opcionales. Indica si la publicación será privada o compartida. Si no se define, se considerará compartida. Para modificar esto debe utilizarse el parámetro **&shared=no**, que indicará la privacidad de la nueva publicación.

La respuesta recibida, luego de invocar a este servicio, es un documento que indica la corrección o no de la acción realizada.

4.2.3. Servicios utilizados por Firetag

Firetag interactúa con la API de Delicious mediante la utilización de tres servicios web. Éstos son necesarios para permitirle al usuario agregar publicaciones nuevas y eliminar tags, así como visualizar los tags que ya ha publicado en el sitio.

A continuación se describirán cada uno de estos servicios, centrándose en la forma en que se utilizan para la implementación de Firetag. La visualización de tags será obviada en esta sección porque ya fue descripta en el capítulo anterior y repetir la descripción de su funcionamiento resultaría redundante.

Agregar una publicación: para implementar este requerimiento, Firetag hace uso del servicio de la API de Delicious que permite agregar una publicación nueva al conjunto de publicaciones de un usuario.

Imagen 4.1 – Formulario de Firetag para agregar una publicación en Delicious

La imagen 4.1 refleja el formulario que permite realizar una nueva publicación en Delicious. Al completar los datos requeridos (usuario y contraseña, URL, tags y descripción) y ejecutar el botón “Agregar”, la aplicación cargará los datos ingresado y realizará, de ser estos datos correctos, una invocación remota al servicio de la API Delicious que permite crear una nueva publicación.

A continuación se muestra parte del código que realiza esta función. Éste ha sido modificado para simplificar su comprensión; sin embargo, su contenido sigue siendo el mismo.

De las tres funciones que se citan, *addBookmark* es la que se invoca desde la interfaz de usuario. Esta función se encarga de tomar los datos desde el formulario e inicializar los valores correspondientes dentro del objeto *XMLHttpRequest* que se utilizará para la invocación remota del servicio. Una vez cargados estos valores, se invoca a *sendRequestPost*, función que se encarga de realizar la solicitud vía AJAX (mediante el

objeto XMLHttpRequest inicializado previamente) para ejecutar el servicio de la API de Delicious.

Una vez ejecutado el servicio, se invocará al manejador del cambio de estado del objeto que implementa el request. En esta función, llamada *processAjaxRequestPost*, se procesará el resultado obtenido en la respuesta del requerimiento, mostrándole al usuario en la interfaz si fue correcto o no.

```
/**Initializes the variables and starts the post.*/
function addBookmark(){
      var user=getElement("deliciousPostUsername").value;
      var psswd=getElement("deliciousPostPsswd").value;
      var url=getElement("url").value;
      var tags=getElement("tags").value;
      var description=getElement("description").value;
      initializePost("https://"+user+":"+psswd+"@api.del.icio.us/
                        v1/posts/add?url="+url+
                        "&description="+description+"&tags="+tags);
      sendRequestPost();
}

/**Sends, via XMLHttpRequest, the request to add a post.*/
function sendRequestPost(){
      postRequest.overrideMimeType('text/xml');
      postRequest.open("GET", postUrl, true);
      postRequest.onreadystatechange=processAjaxRequestPost;
      postRequest.send(null);
}

/**Processes the response obtained after trying to add a post*/
function processAjaxRequestPost() {
      getElement("result").value="";
      if (postRequest.readyState==4) {
            if(postRequest.responseText.match("done")==null){
                getElement("result").value="Something went wrong!";
            }
            else{
                  getElement("result").value="OK!";
            }
   }
}
```

Eliminar un tag: El servicio que brinda la API de Delicious para la eliminación de un tag, se utiliza desde Firetag para que el usuario pueda realizar esta acción sin necesidad de acceder al sitio.

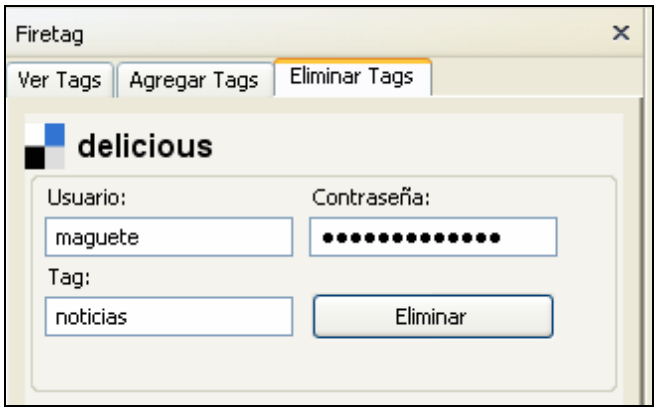

Imagen 4.2 – Formulario de Firetag para eliminar un tag de Delicious

En la imagen 4.2 puede verse el formulario que se utiliza para realizar la eliminación de un tag en Delicious. Es un formulario sencillo que además de los datos de autenticación (usuario y contraseña) posee un campo para que sea ingresado el valor del tag a eliminar. Una vez que se ingresaron estos datos, se deberá ejecutar el botón “Eliminar”, el cual invocará a la función *removeTag* que se muestra a continuación.

La función *removeTag* se encarga de tomar los valores del formulario, inicializar los datos para realizar la solicitud del servicio y llamar a la función *sendRequesPost*. Siempre y cuando el usuario confirme la eliminación del tag. En caso de no hacerlo, la función concluye instantáneamente.

Es *sendRequestPost* la función que se encarga de inicializar las propiedades del objeto *XMLHttpRequest* que llevará a cabo el requerimiento del servicio. En este objeto, referido como *postRequest*, se cargará el valor de la url que permite acceder a la API y eliminar un tag, como se detalló en secciones previas.

Una vez inicializado el objeto en *postRequest* y enviada la solicitud, se ejecutará el manejador cuando se notifique un cambio de estado en dicho objeto.

El manejador se definió como la función *processAjaxRequestPost*, que es la que se encargará de evaluar la respuesta de la ejecución del servicio, para luego informar mediante la interfaz de usuario si el resultado de la operación ha sido correcto o no.

En el código que sigue se detallan las funciones descriptas. Como ya se ha mencionado en otros casos, éste ha recibido algunas modificaciones menores para simplificar su entendimiento; sin embargo, tales modificaciones no afectan a la idea de ejecución general.

```
/**Initializes the post, get the values from the view and calls
the method to send the request.*/
function removeTag(){
	getElement("result").value="";
	var tag=getElement("tag").value;
	if(confirm("Are you sure you want to remove the tag
			\""+tag+"\" from all your posts?"))
	{
		var user=getElement("username").value;
		var psswd=getElement("psswd").value;
		initializePost("https://"+user+":"+psswd+
		"@api.del.icio.us/v1/tags/delete?tag="+tag);
		sendRequestPost();
	}
}

/**Sends, via XMLHttpRequest, the request to remove a tag from
delicious.*/
function sendRequestPost(){
	postRequest.overrideMimeType('text/xml');
	postRequest.open("GET", postUrl, true);
	postRequest.onreadystatechange=processAjaxRequestPost;
	postRequest.send(null);
}

/**Processes the response obtained after */
function processAjaxRequestPost() {
	getElement("result").value="";
	if (postRequest.readyState==4) {
		if(postRequest.responseText.match("done")==null){
		    getElement("result").value="Something went wrong!";
		}
		else{
			getElement("result").value="OK!";
		}
   }
}
```

4.3 Interacción con Flickr

Flickr posee una API, especificada en [37], que consiste en un gran conjunto de servicios, cerca de 180, que permiten acceder a gran parte de la información publicada en el sitio.
Estos servicios permiten manipular todo tipo de información, como la actividad de los usuarios, los procesos de autenticación, los contactos de un usuario, galerías de fotos, grupos, fotos, tags, información geográfica de las fotos, comentarios, estadísticas, etcétera.
A continuación se describirán las características más interesantes de esta API, detallando su mecanismo de autenticación, los servicios más importantes y su utilización en Firetag.

4.3.1. Autenticación

El mecanismo de autenticación de la API de Flickr funciona bajo los conceptos de la autenticación OAuth.
OAuth es un protocolo abierto que permite la autenticación segura de una API de modo estándar para aplicaciones de escritorio, web y móviles.
Para desarrolladores de aplicaciones consumidoras de información OAuth es un método que permite publicar e interactuar con datos protegidos. Para desarrolladores de aplicaciones proveedoras de servicio OAuth proporciona a los usuarios un acceso a sus datos al mismo tiempo que protege las credenciales de su cuenta. En otras palabras, OAuth permite a un usuario de un sitio en particular compartir su información en este sitio (proveedor de servicio) con algún otro sitio (denominado consumidor) sin compartir toda su identidad.

A continuación, se describe el funcionamiento del mecanismo de autenticación de Flickr para aplicaciones de escritorio según lo especificado en [38]. Existen algunas diferencias con las aplicaciones web o móviles, pero no serán descriptas en el presente trabajo ya que no corresponden a las características de la aplicación desarrollada.
Lo primero que debe mencionarse en la descripción de la autenticación OAuth es el concepto de firma. Todas las llamadas API que usen un *token* de autenticación deben estar firmadas. La firma debe generarse mediante un proceso particular que involucra los siguientes pasos:

- Ordenar la lista de argumentos alfabéticamente.
- Agregar, al principio, el valor de la clave secreta.
- Calcular el *hash* MD5 de la cadena obtenida.
- Esta cadena debe agregarse en la lista de parámetros de la url como valor del argumento api_sig.

Ejemplo: si nuestra url tiene los parámetros c=1, b=2 y a=3 junto a una clave con valor "miClave", deberemos generar el valor *hash* MD5 de la cadena miClavea3b2c1 y enviarlo como valor del parámetro api_sig.

El primer paso de autenticación para la API Flickr es realizar una llamada al método flickr.auth.getFrob con los parámetros api_key y api_sig (firma). La respuesta a la llamada a este método contiene un valor *frob* que luego debe usarse para crear una dirección url con el fin de dirigir al usuario a la página que le otorgará los permisos deseados.

Con el valor *frob* obtenido deberá dirigirse, en algún navegador de internet, a la dirección formada de la siguiente manera:

```
http://flickr.com/services/auth/?api_key=[api_key]&
                                perms=[perms]&
                                frob=[frob]&
                                api_sig=[api_sig]
```

Los parámetros utilizados son:

- api_key: es la clave pública que posee el desarrollador de la aplicación que utiliza la API de Flickr. No debe confundirse con la clave secreta. Si bien ambas son otorgadas por Flickr, su uso es distinto.
- perms: son los permisos que se desean obtener (*read*, *write* o *delete*, son inclusivos en ese orden).
- frob: es el valor obtenido luego a la llamada del servicio flickr.auth.getFrob.
- api_sig: es la firma creada como se describió anteriormente.

En la imagen 4.3 se puede ver la página a la que dirige la url descripta anteriormente. En ella el usuario deberá permitir (o no) que la aplicación desarrollada acceda a sus datos publicados en Flickr.

Imagen 4.3 – Captura de pantalla de la ventana que se muestra al realizar la autorización en Flickr

Una vez autenticado el usuario mediante, debe invocarse al método flickr.auth.getToken, en la llamada deben incorporarse los parámetros api_key (con la correspondiente clave pública otorgada por Flickr), frob (con el valor obtenido en la respuesta a la llamada a flikcr.auth.getFrob) y la firma (siempre generada de la misma forma, pero con distintos parámetros). La respuesta a esta invocación devolverá, de ser correcto el llamado, un valor *token* el cual deberá usarse luego para realizar las invocaciones a los servicios que se requieran. Es este valor el que asegura la autenticación del usuario cuando se utilizan métodos remotos en aplicaciones que manipulan información publicada en Flickr.
Cada *token* es único por usuario y por clave pública de aplicación. El *token* debe poseer los permisos necesarios para acceder al servicio solicitado. Puede configurarse la aplicación para que los *token* no caduquen, aunque no es recomendable. Por ello es necesario controlar la validez de un *token* antes de usarlo, o bien solicitar uno nuevo cada vez que se realice una invocación a un método de la API.

Para cerrar esta sección, se describirá un ejemplo de los pasos que debe seguir una aplicación para ejecutar un servicio de la API Flickr.

- El usuario debe hacer clic en un botón de autenticación.
- La aplicación realiza una llamada oculta a flickr.auth.getFrob:
  http://flickr.com/services/rest/?method=flickr.auth.getFrob&
  api_key=987654321&
  api_sig=5f3870be274f6c49b3e31a0c6728957f
- La llamada devuelve el valor *frob* de “12345abcde”.
- La aplicación abre una ventana del explorador con la URL
  http://flickr.com/services/auth/?api_key=987654321&
  perms=write&
  frob=12345abcde&
  api_sig=6f3870be274f6c49b3e31a0c6728957f
- El usuario inicia sesión en la venta, si es que no se encuentra logueado.
- Se le pregunta al usuario si desea permitir el acceso de la aplicación a su cuenta.
- El usuario responde afirmativamente y vuelve a la aplicación.
- Hace clic en el botón que indique la continuación de la función que se desea realizar y se efectúa una llamada oculta a flickr.auth.getToken:
  http://flickr.com/services/rest/?method=flickr.auth.getToken&
  api_key=987654321&
  frob=12345abcde&
  api_sig=7f3870be274f6c49b3e31a0c6728957f.
- Se obtiene el *token* de autenticación “1234567”
- La aplicación realiza la solicitud del servicio correspondiente enviando los parámetros necesarios que incluirán: la clave pública, el *token* y la firma.

4.3.2. Servicios principales

Debido a la gran cantidad de servicios que brinda la API de Flickr, sólo se realizará la descripción de algunos pocos, los que se consideran más interesantes para los conceptos aplicados en el presente trabajo.

Agregar tags: este método, como se define en [39], permite agregar tags a una foto publicada. Para invocarlo es necesario obtener permiso de escritura mediante la autenticación OAuth y su invocación debe realizarse mediante un *POST* http.
Se debe invocar a la url **http://flickr.com/services/rest/?method=flickr.photos.addTags**, a la cual deben agregarse los siguientes parámetros:

- api_key: es obligatorio, es la clave pública de la aplicación.
- photo_id: es obligatorio, es el identificador de la foto a la que se le desea agregar tags.
- tags: es obligatorio, son los tags que se desean agregar a la foto.

La respuesta es vacía si se ejecutó sin errores, en caso contrario se describe el error sucedido.

Asignar conjunto de tags: reemplaza los tags (si no posee los asigna) de una foto por el conjunto de tags especificado. Requiere permiso de escritura mediante la autenticación OAuth y su invocación debe realizarse mediante un *POST* http, según se muestra en [40].
Para ejecutarlo la url **http://flickr.com/services/rest/?method=flickr.photos.setTags** debe ser invocada agregándole los siguientes parámetros:

- api_key: es obligatorio, es la clave pública de la aplicación.
- photo_id: es obligatorio, es el identificador de la foto a la que se le desea agregar tags.
- tags: es obligatorio, son los tags que se desean asignar a la foto.

La respuesta, de ejecutarse correctamente el servicio, es vacía. En caso contrario se obtiene una respuesta indicando el error producido.

Eliminar tag: como se muestra en [41], este servicio se utiliza para eliminar un tag de una foto publicada en Flickr. Al igual que los casos anteriores, se debe poseer permiso de escritura obtenido mediante autenticación OAuth y su requerimiento debe hacerse mediante un *POST* http. La url que debe ser invocada es **http://flickr.com/services/rest/?method=flickr.photos.removeTag**, además deben agregarse los parámetro que se enumeran a continuación:

- api_key: es obligatorio, es la clave pública de la aplicación.
- tag_id: es obligatorio, el id del tag que se desea eliminar.

Al igual que en los casos anteriores, la respuesta es vacía si la ejecución fue correcta o se indica el error si no pudo completarse la ejecución.

Obtener tags de un usuario: este servicio brinda, según lo establecido en [42], la posibilidad de obtener la lista de los tags de un usuario o del usuario que se encuentra logueado. Este método no requiere autenticación y debe accederse mediante la url **http://flickr.com/services/rest/?method=flickr.tags.getListUser** a la cual se le agregarán los parámetros:

- api_key: es obligatorio, es la clave pública de la aplicación.
- user_id: es opcional, es el id del usuario del que se desean consultar sus tags, si no se especifica se toma al usuario logueado.

La respuesta tiene el siguiente formato:

```
<who id="12037949754@N01">
    <tags>
        <tag>tesis</tag>
        <tag>tag1</tag>
        <tag>tag2</tag>
        <tag>tags</tag>
        <tag>test</tag>
    </tags>
</who>
```

Es importante destacar que no se retornan todos los tags, sino que se realiza una reducción de estos eliminando los tags similares.

Obtener tags en bruto de un usuario: su especificación se encuentra en [43]. Es análogo al servicio anterior, pero retorna todos los tags del usuario logueado sin eliminar los que se consideran similares. La url que debe utilizarse es **http://flickr.com/services/rest/?method=flickr.tags.getListUserRaw** con los mismos permisos de autenticación que el servicio descripto anteriormente, pero con los parámetros:

- api_key: es obligatorio, es la clave pública de la aplicación.
- tag: es opcional, es el tag del cual se obtendrán sus tags similares utilizados por el usuario logueado.

La respuesta tiene otro formato con el siguiente patrón:

```
<who id="12037949754@N01">
    <tags>
        <tag clean="tag1">
            <raw>tag1</raw>
            <raw>Tag1</raw>
            <raw>t:ag1</raw>
        </tag>
    </tags>
</who>
```

Obtener tags populares de un usuario: permite obtener, como se establece en [44], una lista de los tags populares del usuario dado o del usuario logueado. No requiere autenticación y debe accederse mediante la url **http://flickr.com/services/rest/?method=flickr.tags.getListUserPopular**. A la url citada, deben agregarse los siguientes parámetros:

- api_key: es obligatorio, es la clave pública de la aplicación.
- user_id: es opcional, es el id del usuario del que se desean consultar sus tags, si no se especifica se toma al usuario logueado.
- count: opcional, indica la cantidad de tags que se desean obtener, si no se indica se toma el valor 10 por defecto.

La respuesta tiene un formato como el que sigue:

```
<who id="12037949754@N01">
    <tags>
        <tag count="10">bar</tag>
        <tag count="11">foo</tag>
        <tag count="147">gull</tag>
        <tag count="3">tags</tag>
        <tag count="3">test</tag>
    </tags>
</who>
```

Obtener tags relacionados: según [45] permite acceder a una lista de los tags relacionados al tag dado, basado en análisis de uso y agrupamiento. No requiere autenticación, y debe utilizarse invocando a la url **http://flickr.com/services/rest/?method=flickr.tags.getRelated**, a la que se deben agregar los siguientes parámetros:

- api_key: es obligatorio, representa a la clave pública de la aplicación que utiliza el servicio.
- tag: también obligatorio, es el tag para el cual se buscarán tags relacionados.

La respuesta a este servicio será de la siguiente forma:

```
<tags source="london">
    <tag>england</tag>
    <tag>thames</tag>
    <tag>tube</tag>
    <tag>bigben</tag>
    <tag>uk</tag>
</tags>
```

Obtener tags de una foto: siguiendo lo especificado en [46], permite recibir la lista de tags asociados a una foto, no requiere autenticación y su acceso remoto debe hacerse mediante una solicitud a la url **http://flickr.com/services/rest/?method=flickr.tags.getListPhoto**. A esta url deberán agregarse los siguientes argumentos:

- api_key: obligatorio, es la clave pública de la aplicación que utiliza el servicio.
- photo_id: obligatorio, es el id de la foto para la cual se solicitan los tags.

La respuesta sigue el siguiente formato:

```
<photo id="2619">
    <tags>
        <tag id="156" author="12037949754@N01"
            authorname="Bees" raw="tag 1">tag1</tag>
        <tag id="157" author="12037949754@N01"
            authorname="Bees" raw="tag 2">tag2</tag>
    </tags>
</photo>
```

Cargar una foto: mediante este servicio, como se define en [47], puede publicarse una nueva foto en Flickr. Para ello debe realizarse un *POST* http a la url **http://api.flickr.com/services/upload/** agregando los siguientes parámetros:

- photo: es obligatorio, se envía aquí el archivo que desea cargarse.
- title: opcional, es el título de la foto.
- description: opcional, representa la descripción de la foto, puede contener HTML.
- tags: opcional, una lista de tags que se desean asociar a la foto.
- is_public, is_friend, is_family: opcionales, especifica quiénes pueden ver la foto, se envía 0 si no se da permiso y 1 para darlo.
- safety_level: opcional, indica el nivel de seguridad, 1 seguro, 2 moderado y 3 restringido.
- content_type: opcional, indica qué tipo de imagen es, 1 para fotos, 2 para capturas de pantallas y 3 para otros tipos.
- hidden: opcional, se envía el valor 1 si se desea mostrar la foto en los resultados globales de búsquedas, 2 en caso contrario.

Todos estos valores, a excepción del argumento photo, deben incluirse en la firma para la autenticación de la solicitud. Los permisos necesarios para realizar este proceso son de escritura (*write*).

Eliminar una foto: permite eliminar una foto de Flickr. Para su utilización, según lo establecido en [48], deben tenerse permisos de eliminación (*delete*) y debe realizarse una solicitud *POST* http a la url **http://flickr.com/services/rest/?method=flickr.photos.delete** a la que se le deberán agregar los siguientes parámetros:

- api_key: obligatorio, es la clave pública de la aplicación que utiliza el servicio.
- photo_id: obligatorio, es el id de la foto para la cual se solicitan los tags.

La respuesta será en blanco si el proceso es correcto, en caso contrario se detallará el error.

Comentar una foto: agrega un comentario a una foto con el usuario que se encuentra logueado. Requiere permisos de escritura (*write*) y debe enviarse una solicitud *POST* http a la url **http://flickr.com/services/rest/?method=flickr.photos.comments.addComment** a la cual se incorporarán, acorde a lo definido en [49], los siguientes argumentos:

- api_key: obligatorio, es la clave pública de la aplicación que utiliza el servicio.
- photo_id: obligatorio, es el id de la foto para la cual se solicitan los tags.
- comment_text: obligatorio, es el texto del comentario que se agregará.

4.3.3. Servicios utilizados por Firetag

La interacción de Firetag con Flickr se realiza mediante la utilización de cuatro de los servicios mencionados en la sección anterior. Estos son: Obtener tags populares de un usuario, Agregar tags, Cargar una foto y Eliminar tag.
Como se vio en el capítulo 3, la carga de los tags utilizados por Firetag se hace mediante servicios de las distintas APIs. En el caso de Flickr, mediante el servicio Obtener tags populares de un usuario. Como ya fue descripto este comportamiento en el presente trabajo, sólo se describirá el uso de los otros servicios enumerados.

Agregar tags: este servicio de la API de Flickr se utiliza dentro de Firetag para permitirle al usuario agregar uno o más tags en una foto ya existente.
En la imagen 4.4 se muestra el formulario que permite realizar la función de agregar tags a una foto publicada en flickr.

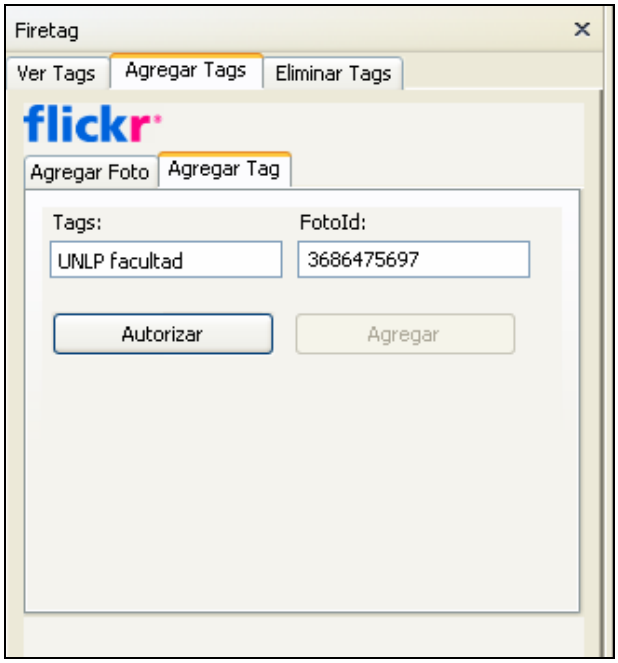

Imagen 4.4 – Captura de pantalla de Firetag que muestra el formulario que permite agregar tags a una foto ya publicada

Para completar esta función, el usuario deberá ingresar el conjunto de tags deseados y el id de la foto a la que desea agregárselos. Una vez hecho esto deberá ejecutar el botón "Autorizar" que comenzará el proceso de autenticación descripto en la sección 4.3.1. Una vez concluido este proceso deberá accionarse el botón "Agregar" que disparará la función que ejecutará el servicio deseado. Finalmente, una vez ejecutada la operación, se le informará al usuario de su correcta finalización o de los errores que hayan surgido.

A continuación se muestra el código que refleja el comportamiento descripto. Este fue modificado para simplificar su entendimiento, sin embargo su esencia es la misma.

Una vez realizada la autenticación, al ejecutar el botón "Agregar" se invocará a la función *doSubmit*. Ésta se encarga de inicializar las variables con los valores necesarios para la invocación (la url del servicio) e invocar a la función *sendSubmitRequest*, que se ocupa de preparar el objeto *XMLHttpRequest* referenciado en *flickrRequest* con los valores correspondientes a la solicitud que se desea realizar. Finalmente inicia esta solicitud.

Una vez concluida la solicitud, cuando cambie el estado del objeto a *ready* se ejecutará el manejador de este evento, definido en *sendSubmitRequest* como *processSubmitAjaxRequest*. Es esta última función la que realiza la evaluación de la respuesta y le indica al usuario en la interfaz gráfica si fue correcta o si se produjo algún error.

```
/**Starts de submitting process*/
function doSubmit(){
      initializeFlickr("http://api.flickr.com/services/rest/?
                  method=flickr.photos.addTags" +
                  "&api_key="+apiKey+
                  "&photo_id="+getElement("photoId").value+
                  "&tags="+getElement("tags").value+
                  "&auth_token="+getElement("authToken").value+
                  "&api_sig="+getElement("apiSig").value);
      sendSubmitRequest();
}

/**Sends, via XMLHttpRequest, the request to do add the tag.*/
function sendSubmitRequest(){
      flickrRequest.overrideMimeType('text/xml');
      flickrRequest.open("GET", flickrUrl, true);
      flickrRequest.onreadystatechange=processSubmitAjaxRequest;
      flickrRequest.send(null);
}

/**Processes the response*/
function processSubmitAjaxRequest() {
      if (flickrRequest.readyState==4) {
            var doc=flickrRequest.responseXML;
            if(getStatus(doc)!="ok"){
                  getElement("result").value=getErrorMessage(doc);
            }
            else{
                  getElement("result").value="OK!";
            }
   }
}
```

Cargar una foto: el servicio de carga de fotos de Flickr se utiliza en Firetag para permitirle al usuario publicar una nueva foto. El mismo se hace desde el formulario que se muestra en la imagen 4.5, luego de cargar los datos requeridos y realizar la autenticación, se deberá ejecutar el botón “Agregar” que disparará la operación de carga remota mediante el servicio de la API Flickr.

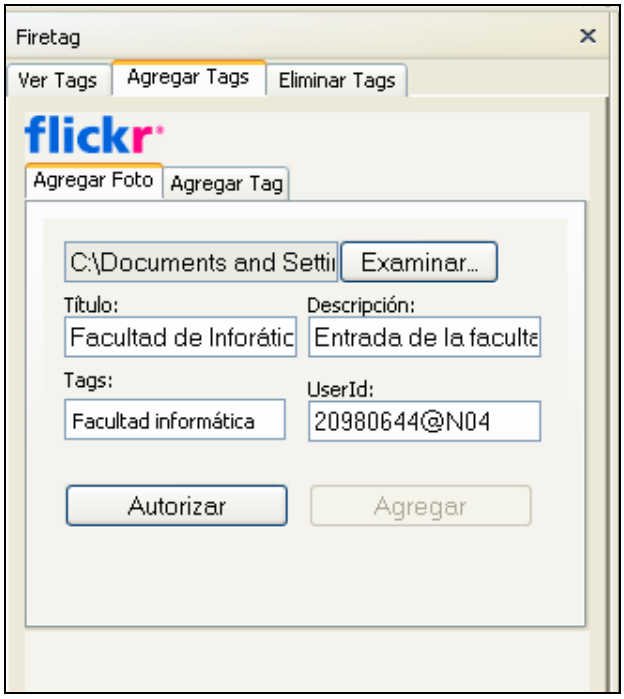

Imagen 4.5 – Captura de pantalla de Firetag que muestra el formulario que permite realizar la carga de fotos a Flickr

La ejecución que se realiza luego de presionar el botón “Agregar” difiere de la utilizada en el servicio anterior. Al necesitar enviar la imagen mediante un *POST* http, se utilizó un formulario HTML con un campo de manejo de archivos, el cual ya manipula los datos necesarios a realizar un envío mediante un post.

El formulario que se utiliza posee como atributo *action* (atributo que indica la url a la cual se invocará cuando se realice un *submit*) la url http://api.flickr.com/services/upload/, la cual brinda acceso al servicio utilizado para cargar nuevas imágenes en Flickr.

Esta situación hace que la ejecución se simplifique a dos funciones principales obviando, al igual que en los casos anteriores, las tareas secundarias que dificultarían la compresión del algoritmo. Al ejecutarse el botón “Agregar” se invoca la función d*oSubmit* que se encarga de realizar la autenticación y generar el evento de *submit* del formulario HTML que se muestra en la imagen 4.5. Una vez realizado el envío de los datos, incluyendo la firma y el *authToken* necesarios para la autenticación y los datos del archivo seleccionado por el usuario, se invoca a la función *showResult* que se encarga de mostrar el resultado de la operación detallando los errores que hayan ocurrido.

```
/**Submits, after a few seconds, to avoid certain concurrency
issues.*/
function doSubmit(){
	getElement("result").value="";
	getAuthToken();
	document.postPhoto.submit();
	showResult();
}

/**Shows the result of attempting to load a photo.*/
function showResult(){
	var doc=parent.resultFrame.document;
	var textNode=document.createTextNode("");
	if(getStatus(doc)!="ok"){
		textNode.nodeValue=getErrorMessage(doc);
		disableEnableButtons();
	}
	else{
		textNode.nodeValue="OK!";
	}
	getElement("result").appendChild(textNode);
}
```

Eliminar un tag: Este servicio se utiliza dentro de Firetag para permitirle al usuario eliminar un tag de una foto. Para ello debe completarse el formulario que se muestra en la imagen 4.6, realizar la autenticación de la solicitud y finalmente ejecutar el botón “Eliminar”; el cual disparará la ejecución de las funciones necesarias para la invocación al servicio en cuestión.

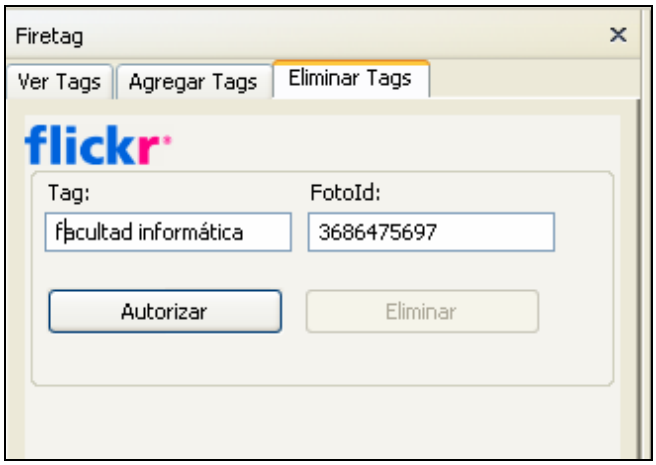

Imagen 4.6 – Captura de pantalla que muestra el formulario utilizado para eliminar un tag de una imagen de Flickr

```
/**Starts the authentication, and calls removeTag.*/
function doSubmit(){
      getElement("result").value="";
      if(confirm("Are you sure you want to remove the tag
                  \""+getElement("tag").value+"\" from the
                   selected Photo?")){
            flickrUrl="http://api.flickr.com/services/rest/
                        ?method=flickr.photos.removeTag"
            getAuthToken();
            removeTag();
      }
}

/**Starts the request to remove a tag from the photo.*/
function removeTag(){
      initializeFlickr("http://api.flickr.com/services/rest/?
                  method=flickr.photos.removeTag"+
                  "&api_key="+apiKey+
                  "&tag_id="+tagId+
                  "&auth_token="+getElement("authToken").value+
                  "&api_sig="+getElement("apiSig").value);
      sendSubmitRequest();
}

/**Actually sends the request to submit the tag removal.*/
function sendSubmitRequest(){
      flickrRequest.overrideMimeType('text/xml');
      flickrRequest.open("GET", flickrUrl, true);
      flickrRequest.onreadystatechange=processSubmitAjaxRequest;
      flickrRequest.send(null);
}

/**Processes the response after request used to remove a tag.*/
function processSubmitAjaxRequest() {
      getElement("result").value="";
      if (flickrRequest.readyState==4) {
            var doc=flickrRequest.responseXML;
            if(getStatus(doc)!="ok"){
                  getElement("result").value=getErrorMessage(doc);
            }
            else{
                  getElement("result").value="OK!";
            }
   }
}
```

La porción de código citada muestra la ejecución luego de presionar el botón “Eliminar” mostrado en la imagen 4.6. Este botón ejecutará función *doSubmit* que desencadenará la ejecución del algoritmo de manera similar a los otros algoritmos citados. Sólo varían los parámetros y las url utilizadas; fuera de eso, la ejecución es casi idéntica.

4.4. Consideraciones generales sobre las APIs web

Como ha sido desarrollado en el presente capítulo, la interacción con distintas APIs web genera algunos inconvenientes cuya resolución no siempre resulta trivial. La implementación arbitraria de las aplicaciones que comparten APIs públicas implica un esfuerzo importante por parte de los desarrolladores de software que manipulen estos recursos al momento de implementar las funcionalidades necesarias para interactuar con los servicios requeridos.

A continuación se mencionan algunos de los inconvenientes que pueden suceder en esta situación:

- Diferencias en los métodos de autenticación/identificación de los usuarios: cada API plantea, eventualmente, distintos métodos para permitir el acceso a los servicios. Esto implica, además del esfuerzo de investigación y estudio sobre los mismos, la implementación de distintos algoritmos y funciones para lograr el acceso. Si en cambio se tratara con un criterio unificado de autenticación, la implementación consistiría en un único conjunto de funciones que sólo necesitaría una modificación en los parámetros de invocación para realizar la identificación en las distintas APIs.
- Diferencias en los servicios existentes: dentro del conjunto de servicios de las APIs puede ocurrir la inexistencia de algunos servicios en un subconjunto de las APIs utilizadas. Este problema implica una modificación en los requerimientos de la aplicación que se desarrolle, ya que algunos requerimientos no estarán disponibles para la interacción con ciertas redes sociales.
- Diferencias semánticas de los mismos servicios: si todas las APIs implementan un servicio en particular, podría darse la situación de que estos no realicen exactamente la misma operación. A modo de ejemplo, durante la implementación de Firetag se descubrió que la eliminación de un tag mediante la API de Delicious implica la eliminación de éste de todas las publicaciones del usuario; mientras que en Flickr se elimina sólo en la publicación indicada.
- Diferencias en los formatos de respuesta: los formatos de respuesta de las distintas APIs puede variar drásticamente según sus implementaciones. Esto genera el inconveniente de tener que realizar una manera de interpretación distinta para cada uno. En el desarrollo de Firetag logró solucionarse este inconveniente mediante la transformación de los datos con archivos *XSLT*, como se describe en el capítulo 3. Pero nada asegura que esta situación puede aplicarse a cualquier API o a cualquier aplicación, la solución depende de que el formato de respuesta sea en *XML*.

Como conclusión general de estos conceptos se definen las características y funciones deseables dentro de una API que sea utilizada por aplicaciones como Firetag; es decir, que manipulen los datos de las publicaciones de los usuarios y sus tags de manera conjunta.

En primer lugar se requiere de un método de autenticación unificado, logrando así evitar los problemas de implementación de distintas funciones para distintos métodos de autenticación. En particular, el autor considera que el mecanismo *OAuth* (descripto en la sección 4.3.1 del presente capítulo) es el más consistente y que mayor seguridad y confiabilidad brinda al usuario. Si bien su utilización desde el punto de vista implementativo es bastante compleja, la información que publicarán los usuarios pertenece a su ámbito privado y resulta primordial brindarle confiabilidad en la privacidad de sus datos.

Por otro lado, el tipo de respuesta debe ser lo bastante consistente como para poder interpretar los mismos de manera análoga. Si bien pueden ocurrir pequeños cambios, al menos debe respetarse la igualdad entre los formatos de las respuestas. En particular se ha trabajado con *XML*, pero otros formatos como *JSON* podrían permitir una implementación similar en función de la arquitectura de comunicación de entrada.

Finalmente, los servicios mínimos e indispensables para una aplicación de las características de Firetag deben permitir realizar las siguientes funcionalidades:

- Autenticación. Un servicio que permita realizar la autenticación de los usuarios. En particular, estandarizar el mecanismo de autenticación resulta de suma importancia para solucionar los problemas previamente descriptos.
- Publicaciones. Será necesario que se provea un servicio que permita agregar y eliminar publicaciones. De manera opcional, la modificación de una publicación también resulta importante; aunque podría ser implementada como una eliminación junto a una agregación posterior.
- Tags. El manejo de los tags es sumamente importante. Su acceso, así como la eliminación, agregación y modificación resultan fundamentales para el desarrollo de una aplicación que los manipule mediante servicios web remotos.
- Varios. Manejo de datos como el perfil del usuario, vínculos con otros usuarios (amistades), publicaciones o tags utilizados últimamente, etcétera. Pueden ser de gran utilidad para el desarrollo de aplicaciones de características similares a la propuesta en el presente trabajo; pero las mismas pueden prescindir de estos servicios ya que no hacen a la funcionalidad primordial del trabajo propuesto.

Capítulo 5

Conclusiones y trabajo a futuro

Las conclusiones y el trabajo a futuro que se obtienen luego del desarrollo del presente trabajo, pueden enmarcarse dentro de dos grandes grupos. Por un lado la rama teórica y conceptual que se encara al plantear un paradigma distinto de interacción entre el usuario y sus cuentas en distintas redes sociales. Por otro lado, la rama práctica que incluye el desarrollo de una aplicación que permita aplicar los conceptos planteados a nivel teórico.
En este capítulo se desarrollarán las ideas vinculadas a ambas áreas a modo de finalización del presente trabajo de investigación y desarrollo.

5.1. Conclusiones

Como conclusión principal del trabajo desarrollado, se deduce que la actividad conjunta del usuario para con sus tags en distintas folksonomías y desde una aplicación que permite dicha unificación, resulta de suma utilidad y genera un volumen de datos mayor y bien definido para poder realizar recomendaciones basadas en uso previo.
El problema que se presenta en la situación descripta es la falta de aprovechamiento de herramientas de ayuda y recomendación que suele manifestarse en usuarios no expertos; es decir, en los usuarios tradicionales de redes sociales y folksonomías. Esta situación podría generar la falta de motivación hacia la utilización de herramientas como la que se presenta en este trabajo o herramientas similares. Si bien la evolución de Internet, en su conjunto, ha sufrido grandes cambios en cuanto a su mecanismo de interacción, no puede aseverarse con total precisión que la evolución de las formas de interacción tome un rumbo dirigido hacia el estilo de aplicaciones como Firetag; pero aquí se plantea una de las posibilidades que pueden presentarse.
Las ideas teóricas que se definen, como la división de las formas de administración de tags en tres categorías (directa, indirecta centralizada e indirecta distribuida) resultan por demás interesantes dentro de la rama teórica del presente trabajo.
Si bien las categorías fueron definidas arbitrariamente por el autor, se las considera con la capacidad de abarcar y separar a las distintas aplicaciones que permiten la manipulación de tags, en forma correcta y bien definida. Como contrapartida, la categorización planteada no se considera absoluta. Pueden existir aplicaciones o mecanismos de administración de tags que presenten características muy distintas a las que se plantean y ello conlleve a no poder incluirlas en ninguna de las categorías mencionadas. Sin

embargo, y en el marco del presente trabajo, la definición de estos conceptos ha sido de gran utilidad para poder definir y concretar las ideas que se desarrollan en la totalidad de este escrito.

En resumen, el presente trabajo plantea una unificación del manejo de las cuentas de un usuario en el contexto de la administración de sus tags. Esta situación es concreta y bien definida, pero la idea puede ser generalizada y aplicada en otros contextos. Mantener los datos de un usuario unificados quizás permita mejorar las recomendaciones al usuario en distintas situaciones como recomendaciones de amistad en una red social, recomendaciones de sitios web a visitar en un buscador, avisos publicitarios personalizados en sitios con patrocinadores, recomendaciones de productos en sitios de *e-commerce*, etcétera.

5.2. Aportes

El principal aporte realizado es la implementación de una extensión para Mozilla Firefox que permite al usuario administrar sus tags de manera remota y utilizar el conjunto de tags de sus distintas cuentas para obtener recomendaciones de los tags que utilizará a futuro.

La aplicación desarrollada, Firetag, se encuentra publicada en el repositorio de extensiones de Mozilla (**http://addons.mozilla.org/firefox/addon/firetag-161877**) en donde ha recibido más de doscientas descargas desde su publicación. Este dato puede tomarse como un indicio del interés de los usuarios por aplicaciones que permitan el manejo remoto de sus cuentas y que a su vez agreguen alguna funcionalidad adicional.
Dentro de la implementación se definió una arquitectura de comunicación para los datos de entrada. Esta arquitectura es, también, un aporte del presente trabajo. Mediante ella se especifican las capas necesarias a desarrollar para obtener información en formato XML y procesarlo para su conversión e inserción en la interfaz de usuario; haciendo hincapié en la diferencia de los formatos recibidos y cómo lograr la abstracción de ellos en la implementación mediante la utilización de procesadores XSLT.
Como aporte desde la concepción teórica de este desarrollo, se detalló una jerarquización en los métodos de administración de los tags del usuario. Si bien, en principio, esta categorización fue definida para diferenciar la aplicación desarrollada de otras aplicaciones similares u otras formas de manipulación de tags, la misma puede ser utilizada en el contexto de otros trabajos o aplicaciones que se centren en la manipulación de datos remotos de un único usuario. Si en lugar de tags, la aplicación que se plantee administra publicaciones, o imágenes, o relaciones de amistad (siempre en distintas redes sociales), la categorización definida puede aplicarse de manera análoga a como fue aplicada a los tags.

Dentro de lo que podrían considerarse cuestiones teóricas, se definió un conjunto de requerimientos necesarios para las APIs web que sean empleadas en el desarrollo de aplicaciones de características similares a la presentada en este trabajo. En este punto se consideraron temas como la autenticación del usuario, los formatos de respuesta de los servicios y los servicios mínimos que serán necesarios para concluir con una aplicación que permita la manipulación remota de los perfiles de un usuario y en particular sus tags, dentro de distintas redes sociales.

5.3. Trabajo a futuro

Como trabajo a futuro se plantean los siguientes ítems:

- Definición de una arquitectura de comunicación de salida: en el presente trabajo se desarrolló una arquitectura para la comunicación de entrada. En cuanto a la comunicación de salida desde Firetag, se ha implementado en su totalidad, pero la implementación carece de una definición abstracta de la forma de interacción.
  Resulta interesante lograr definir e implementar una arquitectura para la comunicación de salida, dado que los distintos servicios que se invoca mantienen un patrón de comunicación que se basa en la autenticación, solicitud del servicio, recepción de la respuesta (satisfactoria o errónea) y su interpretación.
- Ampliación hacia otras redes sociales: en este trabajo sólo se manipulan los tags de Bibsonomy, Delicious y Flickr. La incorporación al mismo de otras redes sociales que permitan realizar tagueos como Youtube[8] o LinkedIn[9] será interesante de desarrollar ya que se tendrá una mayor fuente de datos y se le permitirá al usuario manejar aún más perfiles desde la misma aplicación.
- Agregar más servicios: dentro de las redes con las que se permite interactuar, existen muchos más servicios que pueden ser utilizados desde Firetag. Muchos de estos, descriptos en su mayoría en el capítulo 4, pueden ser de gran utilidad para los usuarios de la aplicación y así mejorar su funcionalidad y aceptación.
- Solución a problemas de concurrencia: algunos problemas de concurrencia se manifiestan en la aplicación durante la ejecución de sucesivas solicitudes vía AJAX, si bien éstas fueron solucionadas de manera transitoria, queda pendiente definir una solución definitiva para este problema.
- Internacionalización: si bien la aplicación está parcialmente internacionalizada para los idiomas español e inglés, los mensajes de error o las advertencias no han sido traducidos y éstos se suceden sólo en inglés. Modificar esta situación junto a la posibilidad de realizar traducciones a otros idiomas como francés o portugués resulta una actividad interesante para lograr aumentar la cantidad de usuarios de Firetag.

[8] http:// www.youtube.com
[9] http://www.linkedin.com

- Universalización de la extensión: actualmente la extensión se encuentra desarrollada solamente para Mozilla Firefox. Esto representa una limitación ya que no tienen posibilidad de acceso los usuarios de navegadores populares como Internet Explorer, Google Chrome o Safari. El desarrollo de versiones de Firetag para sendos navegadores incluirá a una gran cantidad de usuarios, factor necesario para realizar evaluaciones a gran escala de la aplicación.

Bibliografía

[1] Hotho, A., Jäschke, R., Schmitz, C., Stumme, G. BibSonomy: A Social Bookmark and Publication Sharing System. In A. de Moor, S. Polovina, and H. Delugach, editors, Proceedings of the Conceptual Structures Tool Interoperability Workshop at the 14th International Conference on Conceptual Structures, Aalborg, Denmark, 2006.

[2] Schmitz, C., Hotho, A., Jäschke, R., Stumme, G. Mining association rules in folksonomies. Springer, Berlin, Heidelberg, 2006.

[3] Mathes, A. Folksonomies - cooperative classication and communication through shared metadata. Computer Mediated Communication, LIS590CMC (Doctoral Seminar), University of Illinois Urbana-Champaign, 2004.

[4] Folksonomy. http://www.vanderwal.net/folksonomy.html.

[5] Jhingran, A. Enterprise Information Mashups: Integrating Information, Simply. Keynote Address, VLDB 2006.

[6] O'Reilly, T. What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software, O'Reilly Media Inc., 2005.

[7] JavaScript. http://en.wikipedia.org/wiki/JavaScript.

[8] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 2002.

[9] Web service. http://en.wikipedia.org/wiki/Web_service.

[10] Pautasso, C., Zimmermann, O., Leymann, F. RESTful Web services vs. "big" Web services: making the right architectural decision. Proceeding of the 17th international conference on World Wide Web. ACM, 2008.

[11] Garrett, J. Ajax: A new approach to web applications. 2005.

[12] Ajax (Programming). http://en.wikipedia.org/wiki/Ajax_(programming).

[13] Extensible Markup Language (XML). http://www.w3.org/XML/.

[14] XML. http://en.wikipedia.org/wiki/XML.

[15] XSL Transformations (XSLT). http://www.w3.org/TR/xslt.

[16] XSLT. http://en.wikipedia.org/wiki/XSLT.

[17] XML Path Language (XPath). http://www.w3.org/TR/xpath/.

[18] XPath. http://en.wikipedia.org/wiki/XPath.

[19] XUL. http://developer.mozilla.org/en/xul.

[20] XUL. http://en.wikipedia.org/wiki/XUL.

[21] Create dynamic Firefox user interfaces - Architecture overview.
https://www6.software.ibm.com/developerworks/education/x-ajaxxul/section2.html.

[22] BibSonomy – API. http://www.bibsonomy.org/help/doc/api.html.

[23] BibSonomy – Inside BibSonomy. http://www.bibsonomy.org/help/doc/inside.html.

[24] BibSonomy – Add Post.
http://www.bibsonomy.org/help/doc/methods/CreatePost.html.

[25] BibSonomy – Change Post.
http://www.bibsonomy.org/help/doc/methods/ChangePost.html.

[26] BibSonomy – Delete Post.
http://www.bibsonomy.org/help/doc/methods/DeletePost.html.

[27] BibSonomy – List of all Post.
http://www.bibsonomy.org/help/doc/methods/ListOfAllPosts.html.

[28] BibSonomy – List of all Tags.
http://www.bibsonomy.org/help/doc/methods/ListOfAllTags.html.

[29] Delicious – API. http://www.delicious.com/help/api.

[30] Delicious – Get Tags. http://www.delicious.com/help/api#tags_get.

[31] Delicious – Delete Tag. http://www.delicious.com/help/api#tags_delete.

[32] Delicious – Rename Tag. http://www.delicious.com/help/api#tags_rename.

[33] Delicious – Suggest Popular Tags. http://www.delicious.com/help/api#posts_suggest.

[34] Delicious – Get Posts. http://www.delicious.com/help/api#posts_get.

[35] Delicious – Delete Post. http://www.delicious.com/help/api#posts_delete.

[36] Delicious – Add Post. http://www.delicious.com/help/api#posts_add.

[37] Flickr – API. http://www.flickr.com/services/api/.

[38] Flickr – Authentication API. http://www.flickr.com/services/api/auth.spec.html.

[39] Flickr – Add Tags. http://www.flickr.com/services/api/flickr.photos.addTags.html.

[40] Flickr – Set Tags. http://www.flickr.com/services/api/flickr.photos.setTags.html.

[41] Flickr – Remove Tag. http://www.flickr.com/services/api/flickr.photos.removeTag.html.

[42] Flickr – Get User's Tags List.
http://www.flickr.com/services/api/flickr.tags.getListUser.html.

[43] Flickr – Get User's Raw Tags List.
http://www.flickr.com/services/api/flickr.tags.getListUserRaw.html.

[44] Flickr – Get User's Popular Tags List.
http://www.flickr.com/services/api/flickr.tags.getListUserPopular.html.

[45] Flickr – Get Related Tags.
http://www.flickr.com/services/api/flickr.tags.getRelated.html.

[46] Flickr – Get Photo Tag List.
http://www.flickr.com/services/api/flickr.tags.getListPhoto.html.

[47] Flickr – Upload Photo. http://www.flickr.com/services/api/upload.api.html.

[48] Flickr – Delete Photo. http://www.flickr.com/services/api/flickr.photos.delete.html

[49] Flickr – Add Photo Comment.
http://www.flickr.com/services/api/flickr.photos.comments.addComment.html.