UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Enriqueciendo contenidos web mediante la inclusión no intrusiva de aspectos de redes sociales

Autores: Lucas, Eduardo Omar y Zecchin, Danilo Gabriel
Director: Prof. ª Alicia Díaz
Codirector: Prof. Alejandro Fernández
Carrera: Licenciatura en Sistemas – Licenciatura en Informática

Resumen

Existen aspectos comunes relacionados con el manejo de los contenidos publicados en diversas redes sociales. En este trabajo es de interés la forma de interactuar que proveen dichos servicios, particularmente la posibilidad de agregar aspectos de socialización. Además, es de igual importancia el mecanismo de propagación y notificación de dicha información, permitiendo que nuestra interacción social llegue a otros usuarios en la red social.

En esta tesina hemos analizado y estudiado el problema de incorporar aspectos de interacción social a contenidos diversos, con el objetivo principal que dicha inclusión sea no intrusiva y permita a los usuarios interactuar con diferentes proveedores sociales de manera transparente. En esencia, buscamos potenciar la inclusión de aspectos sociales en sitios de contenidos. Nuestra hipótesis de trabajo es que podemos aumentar el impacto de los sitios de redes sociales si simplificamos la inserción de funcionalidad de redes sociales en sitios existentes de contenidos. Un punto fundamental de esa simplificación es que logremos proveer herramientas que automaticen el enriquecimiento de sitios, las cuales nos permitan combinar funcionalidad de distintos proveedores sociales, escondiendo la complejidad de cada uno de estos.

Palabras Claves

Redes Sociales, Definición de Red Social, Aspectos de Redes Sociales, Sitio Enriquecido Socialmente, Socialización por capas, API de Aspectos Sociales, Plataformas de Servicios Sociales, Servicios Web Semánticos, REST, Integración No Intrusiva, Separación de Componentes, Arquitectura Cliente-Servidor, OAuth, XML, XSL

Trabajos Realizados

- *Definición del problema de enriquececimiento e identificación de los requerimientos clave.*
- *Revisión del estado del arte en enriquecimiento de sitios web con contenidos sociales.*
- *Estudio de los principios, tecnologías y casos más representativos de redes sociales.*
- *Evaluación de tecnologías para desarrollar servicios web.*
- *Un diseño general de solución que puede ser implementado en diversos lenguajes y extendido para dar soporte a nuevos proveedores sociales*
- *Una prueba de concepto para la evaluación de la efectividad del diseño e implementación de referencia.*
- *Una API genérica de aspectos sociales.*

Conclusiones

- *La integración es una necesidad creciente que tienen los sitios de redes sociales para llegar a usuarios que consumen contenidos fuera de ellos.*
- *La API genérica definida provee un enfoque abstracto para abordar el problema de cuáles son los aspectos interesantes para la interacción social.*
- *El mecanismo provisto permite la interacción social a partir de servicios de datos existentes de un modo transparente.*

Trabajos Futuros

- *Relación con Web Semántica.*
- *Granularidad del componente Configurador, para refinar aún más la especificación de socialización.*
- *Soporte a la navegabilidad.*

Fecha de la presentación: Diciembre 2010

Índice

1. Motivación 3

1.1. Contexto 3

1.2. Definiciones 4

1.2.1. ¿Qué es un sitio de red social? 4

1.2.2. Características de Sitios de Redes Sociales 5

1.2.3. Sitios Enriquecidos 6

1.2.4. Antecedentes y perspectivas sobre el uso de SNS 10

1.3. Conclusión 12

1.4. Organización de este informe 12

2. Análisis del problema 14

2.1. Enriqueciendo un sitio con contenido social 14

2.2. Tendencia hacia los servicios semánticos 19

2.3. Partes del problema 21

2.4. Requerimientos para emprender una solución 22

3. Estado del arte 24

3.1. Tecnologías existentes 24

3.1.1. Registración, autenticación y autorización 24

3.1.1.1. Autenticación en Facebook 24

3.1.1.2. Estándar OAuth 26

3.1.2. Manejo de información social 35

3.1.3. Open Social 45

3.2. Proveedores de Servicios Sociales 46

3.2.1. Facebook 46

3.2.2. Twitter 53

4. Socialización por capas 57

4.1. Especificación de la API estándar del Socializador 57

4.2. ¿Cómo se implementan los aspectos definidos en las Redes Sociales estudiadas? 60

4.3. Enfoque general mediante separación de capas 62

4.4. Arquitectura general propuesta 63

4.4.1. Generador de contenidos 63

4.4.2. Socializador 65

4.4.3. Visualizador 69

4.5. Estrategia de solución 70

5. Arquitectura de socialización por capas 73

5.1. Generador de Contenidos 73

5.2. Socializador 79

5.3. Visualizador 88

5.4. Configurador 94

5.5. Vista general de la solución propuesta 99

6. Enriqueciendo un sitio mediante la socialización por capas 101

6.1. Usuario administrador 101

6.2. Usuario final 103

7. Conclusiones, contribuciones y trabajos futuros 108

7.1. Conclusiones 108

7.2. Contribuciones 110

7.3. Trabajos futuros 112

8. Anexo 116

8.1. Manual del desarrollador 116

Bibliografía 125

Motivación

Contexto

Los seres humanos somos criaturas sociales. Cómo tales, la raíz de todo el éxito de los sitios de redes sociales está basada en la necesidad de conectarse, comunicarse y expandir dichas conexiones. A lo largo de la historia, tales conexiones estaban limitadas en gran parte por consideraciones económicas o geográficas. Internet permitió a las personas expandir sus conexiones. En primer lugar, esta necesidad se manifestó con el advenimiento de Usenet y BBS (Bulletin Board System, sistema de tablón de anuncios), donde los miembros formaban comunidades entorno a intereses específicos y que, actualmente, se han expandido hacia el reino de las redes sociales, donde cada persona puede descubrir más sobre otras personas que son como ella.

La proliferación de los blogs ha demostrado que las personas aprecian compartir su opinión. Además, la proliferación del movimiento "código libre" (Open Source) muestra que compartir conocimientos y recursos es otra actividad que muchas personas disfrutan. Las redes sociales atraen el lado altruista de las personas, permitiéndoles compartir conexiones, conocimientos e introducir amigos a otros amigos.

Otro aspecto importante de las redes sociales es que proporcionan un canal de entretenimiento compartido, que se manifiesta a través de actividades como juegos, chat, compartir fotos o videos, que pueden involucrar nuestras conexiones. Existen estudios que revelan que el 73% de personas entre 18 y 24 años en EEUU y el 61% del Reino Unido ven a las redes sociales como una forma de entretenimiento.

Las redes sociales "fuera de línea" han sido motivo de estudio para antropólogos y otros por muchos años [Clemons, Barnett, & Appadurai 2007]. Las redes sociales pueden ser grupos de personas que han interactuado en el pasado por un propósito en común o interés, y que tienen relaciones actuales con miembros del grupo. La membresía en las redes puede ser relativamente permanente (por ejemplo, relaciones familiares) o flexible y de corto tiempo (por ejemplo, los miembros pueden llegar e irse de acuerdo a sus intereses y necesidades de cambio de su membresía). Los miembros pueden construir relaciones de confianza a partir de las experiencias y valores compartidos, logrando percibir que la información compartida es creíble y digna de confianza [Clemons et al., 2007].

En la década pasada, los avances en tecnología hicieron posible el uso de herramientas electrónicas de comunicación para crear aplicaciones de colaboración, comunicación y redes sociales en línea. Estas aplicaciones, a veces llamadas herramientas de red social, son lugares web que permiten a los usuarios crear un perfil propio y conectarse con otras personas (que usan la misma aplicación) para construir y mantener una red personal [Skiba, 2007]. Este tipo de aplicación es parte de la evolución de la web 2.0 hacia una mayor colaboración a través de

la web y ejemplos incluyen a Facebook, MySpace, LinkedIn, por nombrar algunas de entre todas las existentes. Mientras que la terminología usada para describir estos sitios varia, recientemente el término Sitios de Redes Sociales (SNS, Social Networking Sites) se ha transformado en la forma común de referirse a ellos.

Definiciones

Desde el punto de vista sociológico, Stanley Wasserman y Katherine Faust establecen que la perspectiva de una red social abarca teorías, modelos, y aplicaciones que son expresadas en términos de conceptos o procesos relacionales. Esto hace referencia a que las relaciones definidas por enlaces a través de unidades son un componente fundamental de las teorías de redes sociales. Además del uso de conceptos relacionales, dichos autores enfatizan que los siguientes aspectos son de igual importancia:

- Los actores y sus acciones son vistos como unidades interdependientes en lugar de independientes o autónomas.
- Los enlaces relacionales entre actores son canales para la transferencia o "flujo" de recursos (materiales o no).
- Los modelos de red enfocados en individuos ven el ambiente estructural de la red como proveedor de oportunidades para las acciones individuales, o restricciones sobre estas últimas.
- Los modelos de red conceptualizan la estructura (social, económica, etc.) como patrones de relaciones entre los actores.

Sin embargo, hoy en día notamos que no existe una clasificación o caracterización clara de lo que es un servicio de red social. Existe una gran variedad de alternativas posibles que son catalogadas como proveedores de servicios sociales en línea. Podemos mencionar como ejemplo Facebook, Twitter, YouTube, Orkut, Ning, LinkedIn entre otros, y notar que cada aplicación define un patrón de interacción entre los usuarios que puede ajustarse en mayor o menor medida a lo definido anteriormente. Con esto apuntamos a que, desde el punto de vista del dominio, tales aplicaciones no necesariamente partieron del análisis teórico vinculado a la interacción social desde una perspectiva sociológica, planteada en la definición inicial.

Si pensamos en las redes sociales diseñadas, desarrolladas e implementadas sobre la web, podemos definirlas como servicios que básicamente proporcionan la capacidad a los individuos de definir un perfil (público, semi-público, privado), construir y alterar una lista de usuarios con los cuales se comparten conexiones, explorar y recorrer tanto sus listas de conexiones como aquellas establecidas por otros en el sistema. La naturaleza, nomenclatura y semántica de estos vínculos pueden variar de un sitio social a otro.

¿Qué es un sitio de red social?

Boyd & Ellison (2007) definieron apropiadamente SNS como "servicios web que permiten a los individuos (1) construir un perfil público o semi-público dentro de un sistema, (2) articular una

lista usuarios con los cuales se comparte una conexión, y (3) ver y explorar sus conexiones y aquellas que son establecidas por otros en el sistema". Es importante aclarar que en la literatura en idioma inglés se resalta la diferencia entre los términos "network" y "networking", notando la preferencia de los autores por el segundo si bien estos son intercambiables. Dicha preferencia es justificada por esos autores argumentando que el término "networking" enfatiza la iniciación de una relación, a menudo entre desconocidos y donde no todos los usuarios la experimentan. Recordemos que muchos usuarios usan los SNS para comunicarse con personas que ya conocen (por ejemplo, personas que forman parte de su entorno social).

Otros autores expresan su punto de vista indicando que el término "networking" es más apropiado puesto que el término "network" formando la sigla "Social Network Site" otorga un significado más amplio a la misma. Como soporte a esta visión, es posible enunciar las definiciones del término "networking" brindadas por dictionary.reference.com y websters-online-dictionary.org, respectivamente:

- Un sistema que provee soporte a los individuos o grupos de individuos para compartir servicios e información en torno a un interés común.
- El intercambio de información o servicios a través de individuos, grupos, o instituciones; específicamente: el cultivo de relaciones productivas.

Ninguna de estas definiciones sugiere que el término "networking" debería incluir la creación de nuevas relaciones (además de mantener las relaciones existentes). Por entonces, mientras adoptamos las tres características claves enunciadas por Boyd y Ellison sobre SNS, encontramos que "Social Networking Sites" es el significado más apropiado para la sigla, teniendo en cuenta los fundamentos en el idioma. Nosotros adoptaremos en el resto del documento la sigla SNS, significando "Sitio de Red Social", teniendo en cuenta que compartimos la interpretación enunciada para el término "networking" aunque la traducción al español de la sigla no exprese dicho significado claramente.

Características de Sitios de Redes Sociales

Los SNSs están organizados alrededor de las personas. Las primeras comunidades en línea y sus sitios web estaban organizados en torno a intereses y tópicos. Una característica única de los SNSs es que los usuarios pueden especificar sus redes sociales y hacerlas visibles a otros [Boyd & Ellison, 2007]. Esto es logrado por los usuarios desarrollando sus perfiles e identificando relaciones (denominadas como "amistades" en la mayoría de los SNS). El punto inicial para un usuario nuevo es desarrollar su perfil que típicamente contiene su imagen, información demográfica como edad, ubicación, estudios, e intereses personales. Luego los usuarios identifican a otros usuarios en el sistema con los cuales tienen o desean establecer una relación. Muchos SNS requieren confirmación bidireccional, cuando un usuario solicita una conexión. El sistema envía esta solicitud a la potencial conexión, y si está acepta la solicitud, los perfiles se enlazan. De esta forma, la red social de un usuario se vuelve visible a

sus conexiones o amistades, y dichas conexiones pueden mostrar las superposiciones que existen entre las redes de cada usuario, y así establecer conexiones con amigos de un amigo. Típicamente, un rico mecanismo de almacenamiento y consulta permite que cada campo en los perfiles sea localizable, otorgando la posibilidad a un usuario de encontrar conexiones con las cuales comparte intereses y entornos. La visibilidad de los perfiles puede variar dependiendo en el diseño del sistema y las opciones de privacidad del usuario. Los SNS también proveen un único punto de acceso a múltiples herramientas de comunicación y dan soporte para que las personas tengan la habilidad de construir una identidad digital. Esta habilidad incluye características como mensaje instantáneo con los miembros de la red (comunicación sincrónica), mensaje asincrónico semi-publico (por ejemplo, en Facebook, un amigo puede publicar comentarios en el perfil de otro en un lugar llamado "Muro"), mensaje asincrónico privado (email) y características asociadas al blogging (por ejemplo, las notas en Facebook).

En todos los ejemplos que hemos mencionado, vemos ciertos aspectos comunes relacionados con el manejo de los contenidos publicados en dichas webs. Para nuestro caso, nos interesa la forma de interactuar que proveen dichos servicios, particularmente la posibilidad de agregarles capacidades de socialización tales como, comentar y compartir. Además, es de sumo interés el mecanismo de propagación y notificación de dicha información, de modo que sea visible para nuestros contactos, ya sea a través del perfil del usuario, vía mail, sms, etc.

De este modo, las redes sociales proporcionan un excelente mecanismo para difundir e incrementar conocimientos. Si los miembros de una red social específica tienen un profundo conocimiento sobre un tema particular en una cierta área, es posible aprovecharlo enriqueciendo el conocimiento propio a través de la interacción social. Nosotros creemos que este es uno de los aspectos más útiles de los sitios de redes sociales, aunque no todos lo promueven y usan tal capacidad.

Sitios enriquecidos

Es muy común que cada SNS provea widgets propietarios que pueden ser incluidos en sitios de terceros. En informática, un widget es un elemento de interfaz gráfica de usuario que muestra una disposición de información modificable por el usuario, tal como una ventana o un cuadro de texto. La característica distintiva de un widget es proveer un solo punto de interacción para la manipulación directa de un tipo de dato dado. Entonces, en el contexto de los SNS, cada widget provee un mecanismo o un aspecto diferente de interacción con la red social destino. Mediante ellos es posible establecer una vía de comunicación entre una red social arbitraria y un sitio externo, el cual queremos promover en ella. Así, es posible construir sitios que proponen un contenido específico según el negocio, y que, a su vez, cuentan con mecanismos de interacción social web.

Teniendo en cuenta lo anterior, podemos referirnos a estos sitios que incluyen widgets de interacción social, como sitios de terceros enriquecidos socialmente. Este es un fenómeno cada vez más común, y tiene fundamento tanto en el crecimiento y popularidad de las redes

sociales, como así también, y más importante, en la necesidad de comunicar, difundir y promocionar diversos contenidos, expandiendo así el público al cual llegan. También es importante comprender que mediante estas tecnologías se extiende la capacidad de interacción de los usuarios ante un contenido arbitrario. Antes solo era posible consumir esa información. Ahora, es posible compartirla, expresar opiniones relacionadas, interactuar con los demás usuarios que consumen esa información, tanto dentro de un entorno social virtual, como en el sitio al cual pertenece y donde aparece publicada.

Existen varios ejemplos de sitios que aprovechan esta funcionalidad. La mayoría de los sitios informativos, basados en artículos o noticias, proveen mecanismos para poder compartir este contenido en varios SNS. Con esta interacción, cada usuario tiene la posibilidad de difundir un contenido (en este caso una noticia o artículo) de su interés en su SNS, y que todos sus amigos (conexiones) tengan acceso al mismo. En particular, podemos citar sitios como: www.clarin.com, www.infobae.com, www.lanacion.com.ar o www.eldia.com.ar, la figura 1.1, refleja dicha situación. Cada uno de estos provee un mecanismo para compartir cada artículo publicado en la página en redes sociales como Facebook, MySpace o Twitter.

Figura 1.1 - Widget de "Compartir" y "Gustar" de Facebook y "Twittear" de Twitter en el sitio del diario El Día.

Tomemos como otro ejemplo a la figura 1.2, donde es posible escuchar la emisora de una radio en línea. Este sitio incluye un widget de Facebook, llamado "Comentarios en directo" que permite a los usuarios de Facebook conectarse, compartir y publicar actualizaciones en tiempo real. Del mismo modo, vemos la figura 1.3, que presenta un widget de "Insignia de página",

que además de permitir la posibilidad de gustar de dicha pagina, nos permite ver las últimas publicaciones en su perfil. Lo que pretendemos ilustrar al citar estos ejemplos es que las redes sociales no solo ponen a disposición widgets que proporcionan mecanismos de interacción más bien estáticos, como compartir o expresar un comentario sobre un articulo, sino que también incluyen formas más dinámicas para interactuar, en tiempo real, sobre contenidos más volátiles, como por ejemplo, los expresados en un medio radial.

Figura 1.2 - Widget de "Comentarios en Directo" de Facebook en el sitio en vivo de Radio10.

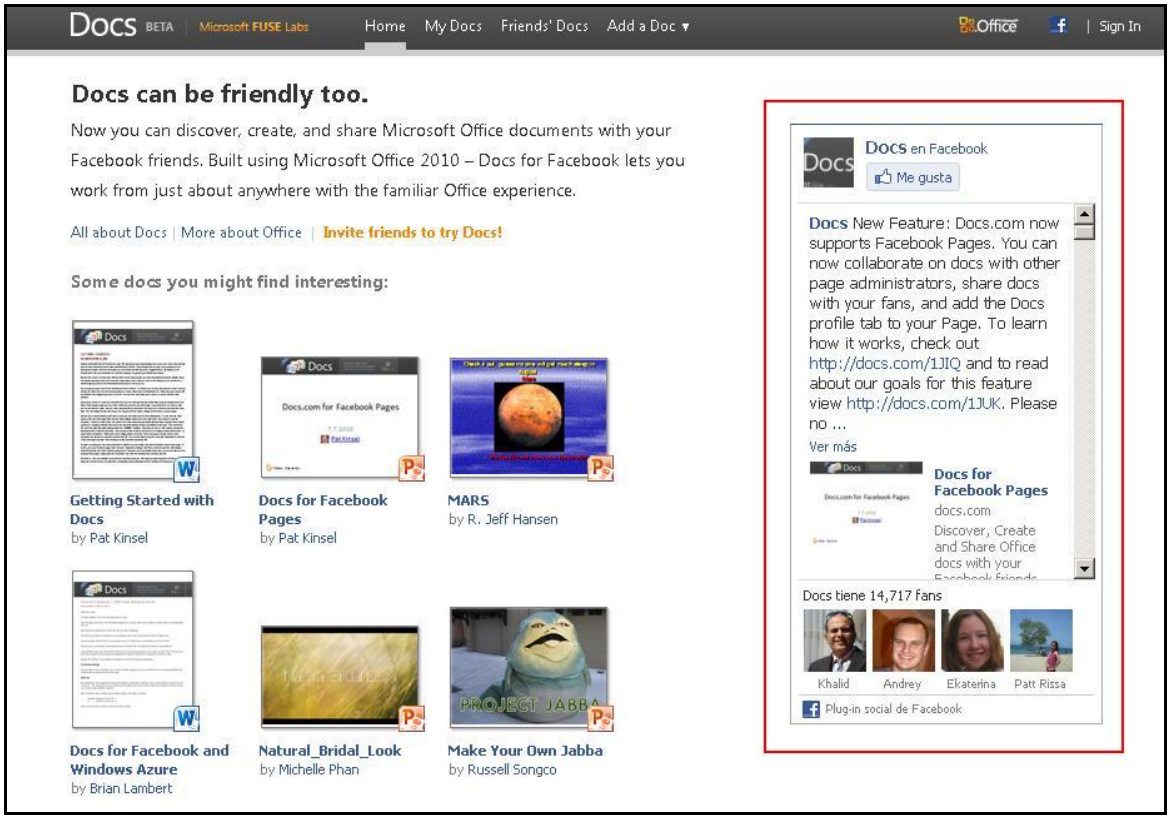


Figura 1.3 - Widget de "Insignia de pagina" en el sitio Docs.

Antecedentes y perspectivas sobre el uso de SNS

Existen numerosas líneas de estudio sobre el uso de las SNS. Con el fin de contextualizar el presente trabajo se discuten a continuación algunos de los más interesantes.

Un estudio examinó las relaciones entre el uso de SNS y el capital social. El capital social hace referencia a los recursos adquiridos a través de relaciones interpersonales y es visto como generalmente tiene un impacto positivo en las personas en las redes sociales (vía confianza incrementada, soporte mutuo, buena voluntad, etc.). Un estudio sobre 286 estudiantes universitarios estadounidenses reveló que su uso de Facebook era un predictor positivo significativo de capital social, sugiriendo que usar Facebook ayudaba a los estudiantes a mantener y reforzar las relaciones, tanto aquellas débiles como las más cercanas [Ellison, Steinfeld, & Lampe, 2007; Ellison et al., 2006]. Este estudio da soporte a la proposición de que el uso de SNS fortalece las redes sociales y es beneficioso, especialmente en un entorno universitario.

Los usuarios de SNS crean una imagen para los demás usuarios a través de la información personal que ellos eligen compartir a través de sus perfiles y a quienes están conectados como Amigos (conexiones). Varios estudios han examinado las impresiones creadas desde diferentes ángulos, incluyendo número de amigos, contenido del perfil, y apariencia. Por ejemplo, un resultado experimental con 153 estudiantes universitarios estadounidenses encontró que el

número de amigos en Facebook puede representar un parámetro significante sobre el cual, otros pueden hacer juicios sociales. Aquellos que tenían muy pocos amigos (aproximadamente 100) o demasiados (más de 500) eran percibidos más negativamente en términos de atractivo social [Tong, Van Der Heide, Langwell, & Walther, 2008]. Los participantes en este estudio tenían un promedio de 395 amigos en sus cuentas de Facebook, indicando que las personas juzgan otros atractivos sociales sobre el número de amigos que tienen y posiblemente usan su propio número de amigos como un punto de referencia.

La relación entre el número de amigos y la estructura del perfil (por ejemplo, que campos están completos) también ha sido investigada para determinar si un perfil más completo (más campos con información certera) predice amistad [Lampe, Ellison, & Steinfeld, 2007]. Más de 30.000 perfiles fueron examinados en una universidad de EEUU. En promedio, los usuarios completaron el 50% de los campos disponibles en Facebook. Las asociaciones positivas fueron encontradas entre el número de campos completos y el número de amigos. No es posible inferir a partir de esto consecuencias de la casualidad, puesto que existen varias explicaciones convincentes incluyendo: un perfil más completo genera más amigos, una mayor cantidad de usuarios activos podrían completar más campos y buscar amigos, y las personas con más amigos podrían sentirse más presionadas a agregar más información en sus perfiles.

Los antecedentes de compartir información y el desarrollo de nuevas relaciones fueron estudiados en 117 usuarios de MySpace y Facebook. Aspectos acerca de la privacidad, confianza en los SNS, y confianza en demás miembros de la SNS fueron propuestos como antecedentes a acciones de compartir información y desarrollo de relaciones. Los usuarios de Facebook tenían mayor confianza en el SNS y ellos revelaban más información personal tal como el nombre real y direcciones de correo electrónico. Los miembros de MySpace estaban más inclinados a revelar su estado de relación. Además, los usuarios de MySpace coincidían en que ellos podían encontrar nuevas personas más fácilmente a través de su SNS y extender nuevas relaciones en línea contactando a la persona vía otros medios de comunicación (por ejemplo, teléfono, mensajería instantánea, email). Los resultados son consistentes con la investigación previa, sugiriendo que los usuarios de Facebook usan el SNS para extender las relaciones existentes "fuera de línea" más frecuentemente que iniciar nuevas relaciones "en línea". Sin embargo, en MySpace, donde la confianza es menor, había una cantidad justa de actividad relacionada con el establecimiento de nuevas relaciones "en línea".

El tiempo destinado al uso de SNS es un aspecto creciente en muchas compañías. Con el creciente uso de SNS, las organizaciones detectan tanto amenazas como oportunidades. Las corporaciones están creando y adoptando SNS para estimular el uso interno y publicitando en los SNS para alcanzar mercados objetivos. Sin embargo, las compañías también reconocen amenazas en términos de tiempo de trabajo desperdiciado y aspectos de seguridad, a tal grado que algunas compañías han bloqueado el acceso a los SNSs durante el tiempo de trabajo [Boyd & Ellison, 2007].

Una encuesta en UK estimó que el uso de Facebook y MySpace estaba costando a las compañías aproximadamente 6.5 billones de libras anualmente en pérdidas de productividad, basándose en que un trabajador de oficina pasaba por lo menos 30 minutos por día en alguna red social [Hathi, 2008]. La asunción detrás de esta afirmación es que el tiempo demandado en el uso de algún SNS no tiene beneficio a la organización. Argumentos opuestos sugieren que el tiempo dedicado al uso de SNS puede ayudar con el reclutamiento, puede mejorar la moral y lealtad de los empleados, y mejorar la transparencia de la compañía. Mientras los departamentos IT típicamente quisieran suprimir el uso de SNS (por razones de seguridad), el uso de SNS en organizaciones es a menudo conducido por recursos humanos. Esto apunta a la construcción de equipos, reclutamiento, retención y cultura corporacional [Hathi, 2008].

Conclusión

En el marco de lo expuesto en este capítulo, buscamos potenciar la inclusión de SNS en sitios de contenidos. Nuestra hipótesis de trabajo es que se puede aumentar el impacto de las SNS si se simplifica la inserción de funcionalidad de redes sociales en sitios existentes de contenidos. Un punto fundamental de esa simplificación es proveer herramientas que automaticen el enriquecimiento de sitios, aun combinando funcionalidad de distintas SNS, escondiendo la complejidad de cada una de ellas.

Organización de este informe

Capitulo 1 - Motivación.

Inicialmente se introduce al lector a las redes sociales desde el punto de vista histórico y social, repasando la evolución de estas para finalmente abordarlas tal cual las conocemos hoy. Luego, se proveen definiciones y características de los sitios de redes sociales, para finalmente brindar ejemplos de sitios enriquecidos socialmente, mencionando los antecedentes y consecuencias de esto.

Capitulo 2 - Análisis del problema.

Se presentan las soluciones que proveen los sitios de redes sociales para enriquecer socialmente sitios de terceros. A partir de este análisis, se enumeran las problemáticas encontradas y se detallan los requerimientos para emprender una solución genérica.

Capitulo 3 - Estado del arte.

Este capítulo brinda un panorama de las tecnologías existentes, enfocándose en los mecanismos que se utilizan para llevar adelante el manejo de la información social. Se detallan proveedores de servicios sociales tales como Facebook y Twitter, mencionando las características, el modo de interactuar, el alcance y las limitaciones que nos brindan sus plataformas.

Capitulo 4 - Socialización por capas.

Se propone un Socializador como solución a la problemática tratada, brindando una API estándar que implementa los aspectos sociales definidos. Luego, se menciona el enfoque de separación por capas desarrollado, brindando una visión general de la arquitectura.

Capitulo 5 - Arquitectura de socialización por capas.

En este capítulo se analiza la arquitectura de la solución propuesta, detallando cada una de las componentes involucradas, sus características y la forma de interactuar de cada una con las restantes. Finalmente, se propone vista general de la solución.

Capitulo 6 - Enriqueciendo un sitio mediante la socialización por capas.

Se detalla el proceso de interacción tanto del usuario administrador como del usuario final, con la solución propuesta. Para esto, se resaltan factores relativos a la usabilidad e interacción.

Capitulo 7 - Conclusiones, contribuciones y trabajos futuros.

Luego de presentada la solución propuesta, se detallan tanto las conclusiones obtenidas a partir del presente desarrollo, así como las contribuciones hechas por el mismo. Finalmente se detalla un conjunto de posibles trabajos futuros con los cuales continuar enriqueciendo la herramienta.

Capitulo 8 - Anexo.

El anexo básicamente propone un manual para el desarrollador. En este se describe tanto lo relativo al entorno de desarrollo, como las herramientas utilizadas, así como los puntos de extensión de la aplicación desarrollada, como por ejemplo agregar un proveedor social.

Análisis del problema

Como se menciono al concluir el capítulo de motivación, nuestra hipótesis de trabajo es que se puede aumentar el impacto de las SNS si se simplifica la inserción de funcionalidad de redes sociales en sitios existentes de contenidos. Un punto fundamental de esa simplificación es proveer herramientas que automaticen el enriquecimiento de sitios, aun combinando funcionalidad de distintas SNS, escondiendo la complejidad de cada una de ellas.

Encontramos aspectos comunes relacionados con el manejo de los contenidos publicados en diversas webs. En nuestro caso, es de interés la forma de interactuar que proveen dichos servicios, particularmente la posibilidad de agregar aspectos de socialización tales como comentar y compartir. Además, es de igual importancia el mecanismo de propagación y notificación de dicha información, permitiendo que nuestra interacción social llegue a otros contactos en la red.

Encontramos que los sitios de redes sociales ponen a nuestra disposición la posibilidad de agregar aspectos de socialización a sitios de terceros. Esto nos brinda la oportunidad de distribuir el contenido, publicándolo en la red social a la cual pertenezcamos. Por lo tanto, permite interactuar con nuestras conexiones o relaciones, logrando potenciar el sitio y promoverlo en un entorno social virtual.

Enriqueciendo un sitio con contenido social

Es nuestra intención destinar algunos párrafos a explicar en términos generales que es y en qué consiste enriquecer un sitio de contenidos. Básicamente, un sitio enriquecido está compuesto por dos elementos o partes principales: contenido estático y contenido social. El primero está expresado básicamente por código HTML que despliega el contenido dispuesto por el sitio de acuerdo con su dominio de información. El segundo está representado por widgets de proveedores sociales. Brevemente, un widget es un componente dinámico construido en código HTML, que posibilita la interacción con el proveedor social de forma asíncrona.

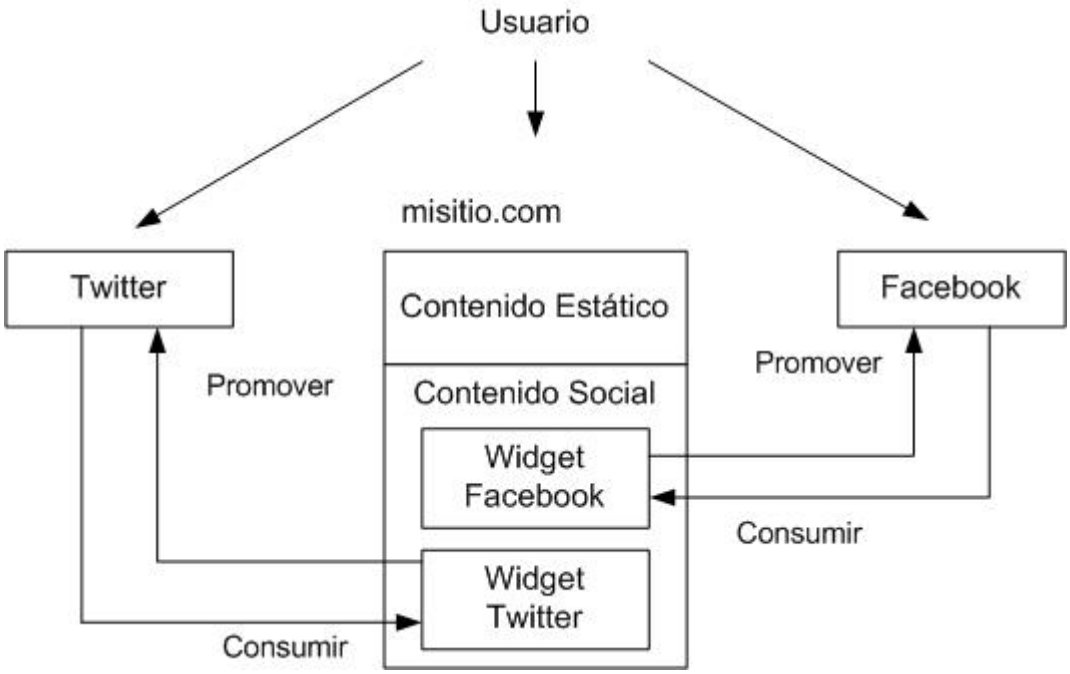


Figura 2.1 - Sitio enriquecido

La figura 2.1 muestra los elementos que intervienen en el enriquecimiento de una página. El usuario puede interactuar independientemente con el sitio enriquecido "misitio.com", y con las redes sociales Facebook y Twitter. El sitio enriquecido publica contenido estático (por ejemplo noticias periodísticas). Adicionalmente, cada elemento de contenido estático (por ejemplo, una noticia) es enriquecido por información consumida (conexiones etiquetadas "Consumir" en el diagrama) desde los sitios de redes sociales (por ejemplo, los comentarios correspondientes a esa noticia). Por otro lado, la interacción conseguida por medio de la inclusión de los widgets permite distribuir cada elemento de contenido estático (conexiones etiquetadas "Promover" en el diagrama) en los sitios de redes sociales (por ejemplo, compartiendo una noticia o artículo, o publicando un comentario o calificación).

Las redes sociales más populares proporcionan un conjunto de widgets que permiten llevar a cabo diferentes acciones sociales, como las mencionadas en el párrafo anterior. Cada uno dispone de guías completas en línea que permiten especificar cierta personalización de los widgets (relacionada con el contenido desplegado), y desde donde es posible copiar el código HTML correspondiente que será incluido en el sitio a enriquecer. La mayoría de ellos por ejemplo no permite modificar parámetros visuales, respetando la gráfica dispuesta en la página del proveedor. En el capítulo siguiente analizaremos los mecanismos de integración en detalle, incluyendo una reseña de widgets disponibles, pero es posible citar ahora como ejemplo los widgets que incluye el sitio www.eldia.com.ar, tal como se ilustra a continuación.

Figura 2.2 - Sitio enriquecido con widgets sociales

La figura 2.2 muestra varios widgets, de diferentes proveedores. Los que están ubicados en la parte superior permiten compartir (o promover) en el sitio de red social (en este caso Twitter o Facebook) el elemento de contenido estático (por ejemplo, un artículo periodístico). Al hacer clic sobre alguno de ellos, se despliega un formulario del proveedor que permite publicar el link que referencia dicho elemento de contenido estático. Las figuras 2.3 y 2.4 a continuación muestran esos formularios para los ejemplos de Twitter y Facebook en la figura 2.2, respectivamente.

Figura 2.3 - Compartir en Twitter

Figura 2.4 - Compartir en Facebook

Una vez que el contenido es compartido en el proveedor social, el mismo aparecerá como una notificación vinculada al usuario que ejecutó dicha acción. Además, el botón desplegado junto al contenido estático mostrará una cuenta (ver figura 2.2), indicando las veces que el mismo ha

sido compartido. Por ejemplo, el artículo de la figura 2.2 publicado en Twitter tendrá el aspecto que muestra la figura 2.5.

Figura 2.5 - Contenido compartido en Twitter

Por otro lado, el widget ubicado en la parte inferior de la figura 2.2 permite calificar el contenido estático al cual está asociado. Dicha calificación aparecerá desplegada como una notificación en la red social dentro del hilo de noticias correspondientes al usuario. Esto se ilustra en la figura 2.6.

Figura 2.6 - Calificación en Facebook

Además, dicho widget listará aquellos usuarios que hayan calificado ese elemento de contenido (interacción etiquetada como "consumir" en la figura 1.1). Entonces, luego de interactuar con los widgets y generar contenido social, el aspecto del sitio enriquecido se verá como lo muestra la figura 2.7.

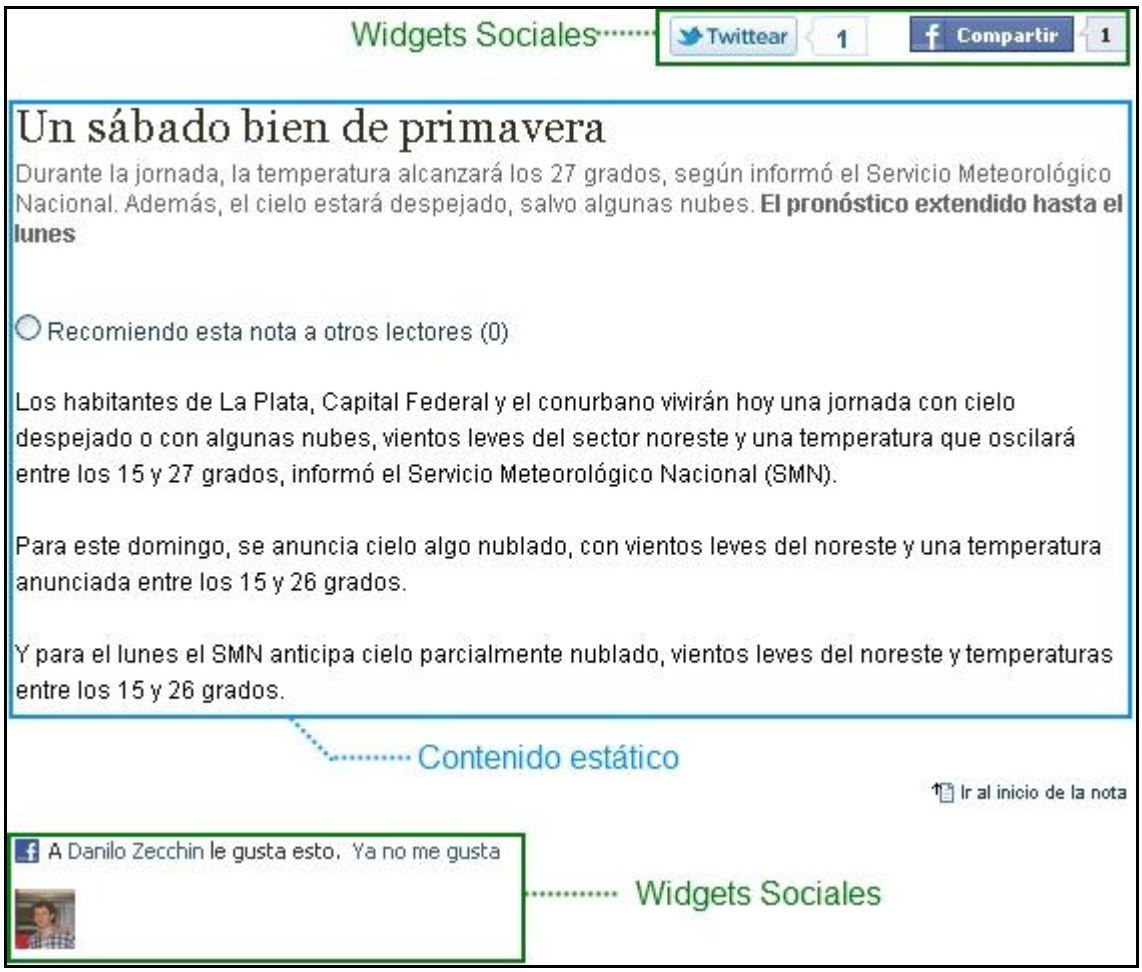


Figura 2.7 - Sitio enriquecido con widgets que muestran interacción social

Como se ve en la figura 2.7, los botones correspondientes a los widgets que permiten promover contenido estático en la red social muestran la cantidad de veces que se llevó a cabo dicha acción. De la misma forma, el widget que permite calificar el contenido lista los usuarios que han expresado dicha opinión. Este último caso en particular describe el caso específico en el cual el sitio de contenidos consume información social del proveedor.

Entonces, para enriquecer un sitio por medio de dichos widgets podemos utilizar el mecanismo de personalización provisto por cada red social para configurar cierta funcionalidad (siempre y cuando sea posible). Luego, obtenemos el código del mismo para ubicarlo en el lugar conveniente en el sitio de contenidos. Si queremos utilizar varios widgets, debemos repetir este proceso para cada uno. Lo mismo sucederá si queremos disponer widgets de distintos proveedores sociales, como se muestra en la figura 2.1 y 2.7.

Tendencia hacia los servicios semánticos

Otro tópico que queremos tratar antes de explicitar los problemas que debemos afrontar y los requisitos necesarios para alcanzar una solución, es la tendencia actual a la construcción de

servicios semánticos. Es posible afirmar que resulta cada vez más común que las aplicaciones web consideren, a la hora de su diseño e implementación, la incorporación de servicios semánticos que pueden ser consumidos de diferentes formas. Esta tendencia involucra tecnologías diversas, tales como REST o MashUps, entre otras, donde en general el medio o lenguaje de comunicación es XML. A continuación se brinda una breve descripción de estas tecnologías.

REST

Brevemente, *REST* es un estilo de arquitectura cliente-servidor, que se basa en la transferencia de representaciones de recursos. Un servicio RESTful permitirá manipular el estado de un objeto por medio de los métodos HTTP GET, PUT, POST o DELETE, usando algún lenguaje unificado para el intercambio, como por ejemplo XML. Así, un servicio RESTful puede ser considerado como una API web.

Muchos sitios de redes sociales ponen a disposición de los desarrolladores APIs que permiten consumir contenidos a través del mecanismo de interacción por llamados REST. Como ejemplo de esta tendencia podemos mencionar Facebook, Twitter, MySpace, FriendFeed, LinkedIn entre otros. A través de estas APIs, un programador web es capaz de incorporar interacción social a un sitio web determinado, y, a su vez, promocionar o difundir socialmente el contenido de dicho sitio. Entonces, con este tipo de tecnología se logra que una red social tenga mecanismos para expandir las conexiones y la comunicación, abarcando sitios externos.

MashUps

En desarrollo web, un *mashup* es una página web o aplicación que usa o combina datos o funcionalidad desde dos o más fuentes externas para crear un nuevo servicio. Permite lograr una integración rápida y fácil, usando APIs abiertas y fuentes de datos para producir resultados enriquecidos que no eran necesariamente la razón original para producir las fuentes de datos. Para ser capaces de acceder permanentemente a los datos de otros servicios, los mashups son generalmente aplicaciones clientes o alojadas en línea. En los últimos años, cada vez más aplicaciones web proveen APIs que posibilitan a los desarrolladores de software integrar fácilmente datos y funciones en lugar de construirlas ellos mismos. Esto último se refleja en lo mencionado anteriormente con respecto a la tecnología REST y su adopción por parte de los sitios de redes sociales. Es por esto que es posible considerar el fenómeno mashup como participante activo en la evolución del software social y la Web 2.0.

Web Semántica

REST, Mashups, y Microformats son solo un indicador de un fenómeno más grande que está transformando la manera en la que los contenidos web se representan y distribuyen. En la actualidad de habla de la Web Semántica para referirse a un conjunto de estrategias y tecnologías que apuntan a dotar a la información en la web de descripciones formales del conocimiento que representan. En la web semántica los contenidos, dado que son

acompañados por descripciones de su semántica, pueden ser consumidos, integrados, consultados y extendidos de maneras mucho más ricas. En el centro de la web semántica se encuentran lenguajes de representación de conocimiento como RDF y OWL. La relación entre las redes sociales y la web semántica es un tema emergente y muy interesante y será foco de discusión más detallada en la sección de trabajo futuro de esta tesis.

En todas las tecnologías mencionadas, está presente el formato XML como denominador común. Resulta vital la importancia de dicho estándar para llevar a cabo el intercambio de información con significado. En tal sentido, se debe destacar la posibilidad que nos provee el hecho de tener una estructura y la posibilidad de transmitir y transportar significado que puede ser consumido por otros servicios o clientes.

Partes del problema

A partir de la problemática global inicial desarrollada anteriormente, el enriquecimiento social de sitios de contenidos, se detallan a continuación cada uno de los problemas hallados:

- Los sitios de contenidos pueden necesitar integrarse a distintas redes, ya sea para llegar a más usuarios, como así también para incorporar distintos aspectos de socialización. Esto, por ejemplo, implicaría repetir el widget relacionado al aspecto social requerido, para cada proveedor social al cual deseamos integrarnos.
- Los widgets que ponen a disposición los sitios de redes sociales que se pueden incorporar en sitios de terceros resultan muy poco editables. Esto impacta directamente en la vista final del sitio, teniendo que adaptarse a dichos widgets o convivir con esta diferencia entre estilos.
- Existe una gran variedad de sitios que proveen algún mecanismo de enriquecimiento mediante aspectos sociales, como ejemplo de esto, tenemos a Facebook Connect o Google Friend Connect. Estas técnicas requieren conocimientos de programación web, dado que proporcionan un mecanismo de interacción con la API de los sitios de redes sociales a través de lenguajes de scripting del lado del cliente.
- Cada SNS define su propia API de acuerdo a los aspectos que fueron tenidos en cuenta al momento de su concepción, tanto de dicha API, como de la red social en sí misma.
- El modo en que cada SNS exporta y especifica el modo en que debe ser utilizada su API para consumir contenidos sociales establece un protocolo de comunicación server-to-server y define diferentes mecanismos para la integración con un sitio arbitrario.
- Con respecto a cómo cada proveedor social exporta su API, hay disponibles librerías en diferentes lenguajes de programación, que permiten la interacción server-to-server con el proveedor. Esto tiene como consecuencia lo siguiente:

    - Impone restricciones con respecto al lenguaje de programación a utilizar, existiendo la posibilidad de que un sitio de red social no provea librerías en determinados lenguajes.
    - Agrega dependencias de versionado. En la mayoría de los casos, la creación y el mantenimiento de dichas librerías no es responsabilidad de la red social, sino que dicha actividad es desarrollada por algún tercero.
- Para casos en los que se quiere enriquecer sitios utilizando una arquitectura server-to-server, los mecanismos de autenticación pueden variar de acuerdo a la red social que utilicemos.

De acuerdo con estos desafíos mencionados que se deben enfrentar, es claro que la lógica del sitio enriquecido no será homogénea, teniendo que convivir con los distintos mecanismos, políticas y formatos de socialización impuestas por cada proveedor objetivo. Tal como se menciona, esta situación impacta a lo largo de toda la aplicación, tanto al momento de identificarse en la red social, así como al momento de interactuar con el servidor de nuestro proveedor.

Requerimientos para emprender una solución

A partir de los problemas detectados, nos enfocaremos en detallar ciertos puntos que se deberán tener en cuenta para desarrollar una herramienta que trate de resolver las cuestiones antes planteadas:

1. Separación del contenido de la presentación: es una cualidad reconocible actualmente en el desarrollo web. Esto nos da la posibilidad de proveer servicios de publicación de información (por ejemplo, sin ninguna representación gráfica particular como ser HTML), y que pueden ser consumidos por otros servicios con diferentes objetivos, incluyendo algún mecanismo para su visualización en un navegador.
2. Una estrategia que permita a quienes desarrollan sitios de contenidos, enriquecerlos con distintas redes sociales: si el objetivo es poder abarcar la mayor cantidad de contactos y relaciones mejorando la integración, como así también ofrecer más aspectos de socialización, es necesario que se provean la mayor cantidad de redes sociales disponibles de manera transparente.
3. Permitir que sea una o muchas redes sociales al mismo tiempo: así como se deberá dar soporte a distintos proveedores de aspectos de socialización, se deberá poder personalizar la configuración de la herramienta, de modo de poder iniciar sesión simultáneamente en distintas redes sociales.
4. Reducir el esfuerzo del desarrollo del sitio: con el objetivo de minimizar el esfuerzo, la herramienta deberá resultar no intrusiva, evitando así la alteración o modificación del contenido original. Por otro lado, el código de la herramienta resultante deberá ser transparente y lo más homogéneo posible, siguiendo como premisa el empleo de estándares. Logrando esto, los aspectos relacionados con la interacción server-to-

server con un sitio de red social en particular deberán quedar encapsulados, exponiendo solo el punto de conexión entre el aspecto social y la red elegida.

5. Existencia de independencia de la tecnología de los sitios de red social: la interacción con la red social debe ser independiente de la tecnología con que se desarrolle el sitio, siendo necesario que la red social exporte o publique una interfaz entre la red y el sitio. El mecanismo que predomina entre los proveedores de servicios y aspectos sociales, son los llamados REST para el manejo de información (lectura o escritura de datos del usuario).
6. Extensibilidad para incorporar nuevas redes: La herramienta no solo deberá proveer un conjunto inicial de proveedores, sino que la arquitectura de la misma deberá dar soporte de modo que pueda extenderse de modo de incorporar nuevos proveedores de aspectos sociales, y que dicha adición sea transparente.
7. Que incluya herramientas de alto nivel que minimicen la necesidad de programar: Será necesario proveer un mecanismo mediante el cual se pueda crear una configuración de modo sencillo e interactivo que especifique de qué modo se socializará el sitio.
8. Que sea capaz de introducir la capa de socialización sin necesidad de modificar los contenidos originales: A partir de la configuración creada, la herramienta aplicará la socialización al sitio a partir de los parámetros introducidos.
9. Capa de visualización: Dado que no debe ser restrictivo para visitar un sitio tener una cuenta en alguna red social, el sitio que se enriquece socialmente deberá contar con dos vistas. Por un lado, debe ser posible acceder a una vista no socializada y que resulte transparente para aquellos usuarios que no tengan cuenta con ninguna red social. Así, esta vista mostrará el contenido original del sitio, sin ningún mecanismo de interacción social. Por otro lado, la vista acusará cambios cuando un usuario acceda a algún proveedor social, desplegando los mecanismos de interacción necesarios para poder llevar a cabo alguna acción social.

Estado del Arte

De acuerdo a lo desarrollado en el capitulo anterior, es nuestra intención dedicar los siguientes párrafos para analizar, estudiar y reseñar las tecnologías existentes que serán tenidas en cuenta para abordar los problemas detectados en el marco de desarrollo de aplicaciones web sociales, como así también aquellas existentes que no llegan a cubrir las necesidades planteadas. Luego, explicaremos como cada una de estas tecnologías se pone o no de manifiesto en los sitios de redes sociales más populares que hemos considerado para el desarrollo de esta tesina.

Tecnologías existentes

En el camino por encontrar soluciones a los problemas que detectamos, identificamos requerimientos que debemos resolver en las distintas etapas de la interacción con los proveedores de aspectos sociales. En tal sentido, agrupamos estas tareas en dos grandes grupos:

- Registración, autenticación y autorización.
- Manejo de Información Social

A continuación se menciona por cada una de estas tareas, las tecnologías y estándares analizados, así como el mecanismo que utilizan los proveedores sociales para hacer uso de esta.

Registración, autenticación y autorización

Autenticación en Facebook

El mecanismo original de autenticación en Facebook se denominó Facebook Connect. En agosto de 2006, se introdujo la primera versión de la API de Facebook, lo que permite a los usuarios compartir su información con los sitios web y aplicaciones de terceros que deseen. Cientos de empresas se han aprovechado de esta API, lo que permite a los usuarios conectarse de forma dinámica a su información de identidad de Facebook, como el perfil, amigos, fotos y más información, desde los sitios web de terceros, así como aplicaciones de escritorio y móviles [autenticación en Facebook].

En mayo de 2007, Facebook puso en marcha la plataforma de Facebook, que permitió a los desarrolladores de terceros crear aplicaciones sociales ricas dentro de Facebook. Más de 350.000 desarrolladores y empresarios de 225 países han firmado, y comenzaron el desarrollo de aplicaciones, y han visto una adopción significativa por parte de los usuarios de Facebook en todo el mundo.

En Mayo de 2008, se anuncia Facebook Connect. Facebook Connect es la siguiente iteración de la plataforma de Facebook que permite a los usuarios "conectar" su identidad de Facebook, con amigos y su privacidad con cualquier otro sitio. Esto ahora permitirá a sitios web de terceros para implementar y ofrecer más características, incluso fuera de la plataforma de Facebook.

A continuación se detallan algunas de las características que alcanza Facebook Connect:

- Autenticación confiable: los usuarios podrán conectar su cuenta de Facebook con cualquier sitio web utilizando un método de autenticación confiable. Ya sea en el inicio de sesión o en cualquier otro momento en el que el desarrollador guste, podrá añadir un contexto social a su sitio, de modo que el usuario final será capaz de autenticar y conectar su cuenta en un entorno de confianza. El usuario tendrá el control total de los permisos concedidos.
- Identidad Real: los usuarios de Facebook podrán representarse a sí mismos con sus nombres reales y verdaderas identidades. Con Facebook Connect, los usuarios pueden llevar su información de identidad real con ellos dondequiera que vayan en la Web, tales como: información del perfil, imagen de perfil, nombre, amigos, fotos, eventos, grupos, etc.
- Acceso a los Amigos: los usuarios emplean a Facebook para mantenerse conectados con sus amigos y familiares. Con Facebook Connect, los usuarios pueden llevar a sus amigos con ellos dondequiera que vayan en la Web. Los desarrolladores podrán añadir contexto social rico a sus sitios web. Los desarrolladores serán incluso capaces de mostrar de forma dinámica, cuáles de sus amigos de Facebook ya tienen cuentas en sus sitios.
- Privacidad Dinámica: como un usuario se mueve alrededor de la web, su configuración de privacidad lo va a seguir, asegurando que la información y las reglas de privacidad del usuario estén siempre al día. Por ejemplo, si un usuario cambia su foto de perfil, o quita una conexión con un amigo, este se actualizará automáticamente en el sitio externo.

Estos son algunos de los pasos que Facebook está tomando para hacer realidad la visión de la portabilidad de los datos de los usuarios en todo el mundo. La próxima evolución de la portabilidad de los datos es tratar de dar a los usuarios, la capacidad de transportar su identidad y amigos con ellos alrededor de la Web, al tiempo que poder confiar en que su información está siempre al día y siempre protegida por su configuración de privacidad.

¿Cómo funciona?

Facebook Connect simplifica en gran medida el desarrollo de aplicaciones para desarrolladores de Facebook. En su forma más simple, el desarrollo de una aplicación de Facebook Connect consiste en añadir etiquetas XFBML a una página HTML. Facebook Connect utiliza un canal de comunicación de dominio cruzado para abrir un 'iframe' en la página HTML para cada etiqueta

XFBML. Cuando el usuario hace clic en una etiqueta, Facebook Connect se encarga de la interacción y permite que el usuario se conecte, publique actualizaciones, etc.

Una aplicación de Facebook Connect puede utilizar cualquiera o todas las características siguientes:

- Etiquetas XFBML.
- JavaScript, con llamadas a la biblioteca cliente de JavaScript.
- Páginas PHP, con llamadas a la API PHP de Facebook.
- Código en cualquier lenguaje, con las llamadas RESTful a la API de Facebook.

Los casos más comunes son simples páginas web con etiquetas XFBML y las llamadas a la biblioteca de cliente JavaScript.

Uso de Etiquetas XFBML

XFBML es una manera muy fácil para empezar la construcción de una aplicación de Facebook Connect, ya que Facebook se encarga del manejo de inicio de sesión, de la sesión activa y la gestión interna de la misma. Para utilizar etiquetas XFBML en una página web, se necesitará:

- Un espacio de nombres XHTML de forma que el navegador ignore las etiquetas XFBML.
- Una llamada JavaScript que active el FeatureLoader de Facebook.
- Etiquetas XFBML, que ofrecen la experiencia de Facebook.
- Un archivo llamado 'xd_receiver.htm', para apoyar a los navegadores más antiguos.

Utilizando el SDK de JavaScript

La JavaScript SDK de código abierto proporciona una potente interfaz para aplicaciones de Facebook Connect. Se puede utilizar la JavaScript SDK de código abierto para atar una aplicación de Facebook Connect al mecanismo de de inicio de sesión del sitio. Del mismo modo, se utiliza esta SDK para recuperar el ID y estado del usuario, detectar si se ha conectado con su sitio, y luego actualizar su estado en el sitio Web de acuerdo al mecanismo utilizado.

Estándar OAuth

¿Qué ofrece OAuth?

El objetivo de este método de autenticación es poder acceder a recursos protegidos sin la necesidad de enviar nuestro usuario y contraseña para solicitar los mismos. ¿Cómo es posible? A través de nuestra aplicación será necesario generar una firma que nos identifique gracias a un conjunto de tokens facilitados por el servicio. A continuación se detallan los pasos requeridos para la utilización de este protocolo, utilizando como ejemplo la plataformas de Twitter, pero que también se aplican, claro está, al resto de las plataformas que opten por esta tecnología (como MySpace) [OAuth].

Workflow de OAuth

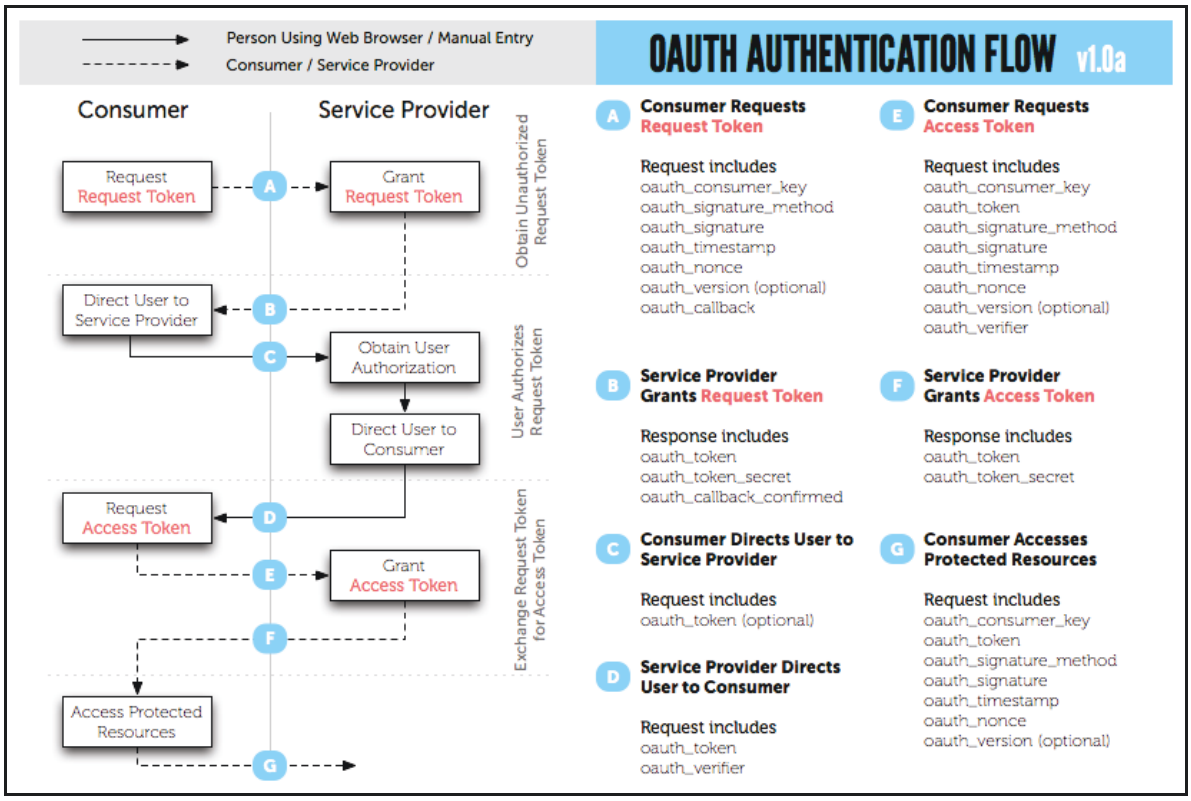


Figura 3.1 - Diagrama de flujo de OAuth

En la figura 3.1, podemos ver cuáles son los pasos para poder solicitar los recursos protegidos de un servicio. El objetivo fundamental es conseguir un *Access token* y un *Access Token secret*. Con ellos seremos capaces de generar la firma necesaria para obtener los recursos protegidos de nuestros usuarios sin necesidad de enviar sus credenciales en cada petición. En cualquier caso, debemos contar previamente con una aplicación en la red social con la que queramos interactuar. Los pasos que debemos seguir son los siguientes:

1. Creación de una aplicación y obtención de una Consumer Key y Consumer Secret.
2. Paso A, Solicitar un token temporal.
3. Pasos B, C y D, autenticación y autorización del usuario.
4. Pasos E y F, Intercambio del token temporal por un Access Token.

Creación de una aplicación y obtención de una Consumer Key y Consumer secret

Lo primero que necesitamos para comenzar es una **Consumer Key y Consumer Secret.** Ambos se consiguen cuando creamos "simbólicamente" una aplicación en cualquiera de las plataformas de la cual queremos hacer uso de sus servicios. En el caso de Twitter, para crear la aplicación, podemos acceder a la siguiente dirección http://twitter.com/apps. Obviamente, es necesario tener una cuenta para poder darlas de alta. Una vez creada, en ambos casos podemos ver en los detalles la Consumer Key y Consumer Secret generadas, tal como nos muestra la figura 3.2.

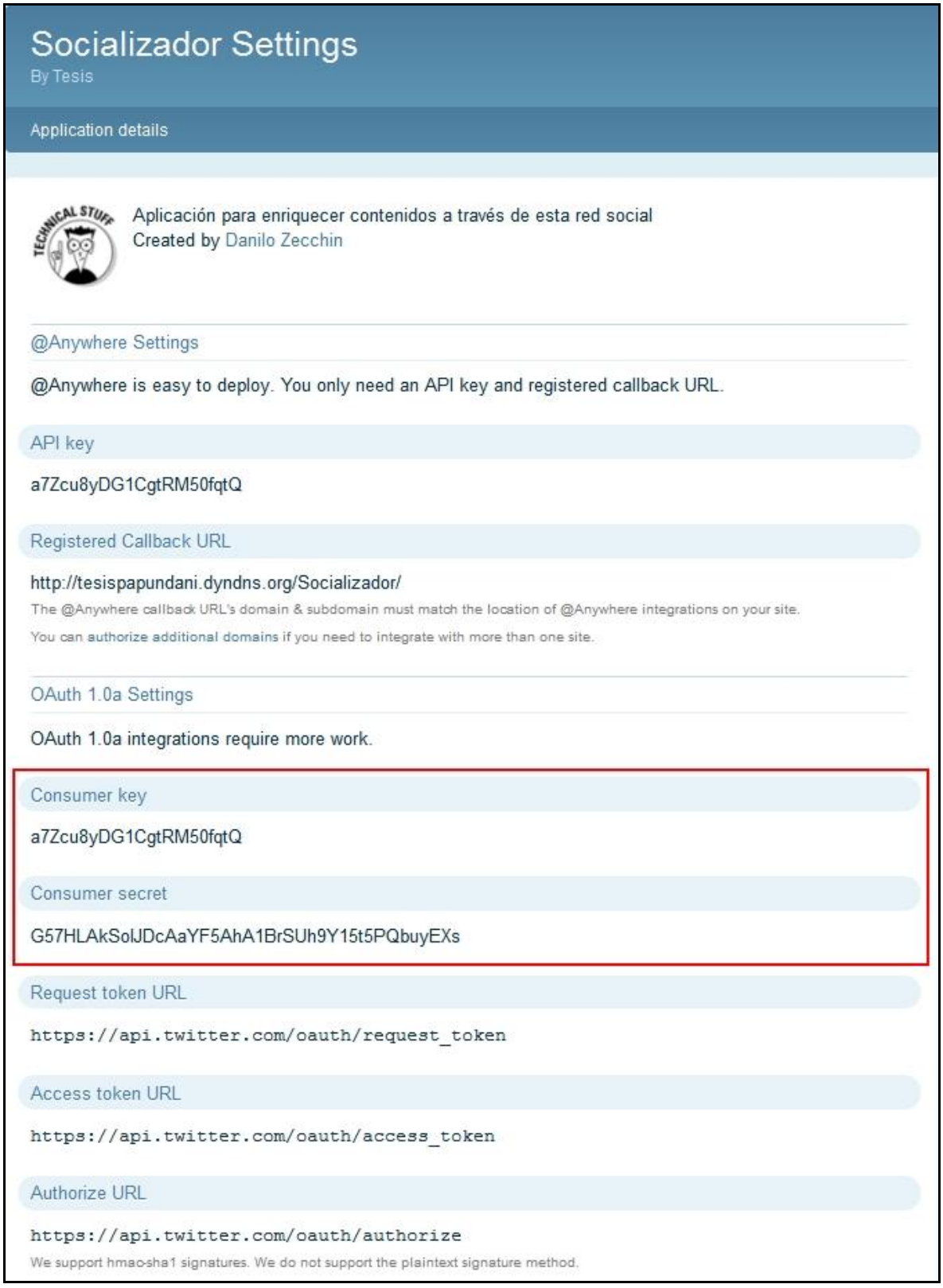


Figura 3.2 - Información detallada de una aplicación creada en Twitter.

Solicitar un token temporal

Tal como lo muestra el paso 'A' de la figura 3.1, lo primero que necesitamos antes de pedir la autorización del usuario para recuperar sus datos privados desde nuestra aplicación, es solicitar un token temporal para poder iniciar un proceso de autorización desde la plataforma en cuestión. Para solicitar un token al servicio serán necesarios los siguientes parámetros:

- **oauth_consumer_key**: Se obtuvo cuando dimos de alta la aplicación.
- **oauth_signature_method**: Se trata del método que utilizaremos para generar la firma. Usaremos **HMAC-SHA1** ya que, por ejemplo, MySpace y Twitter soportan este método hasta la fecha, pero también es posible usar **PLAINTEXT y RSA-SHA1.**
- **oauth_signature:** La firma generada por nuestra aplicación.
- **oauth_nonce**: Se trata de un string aleatorio que acompaña a un timestamp. Representa un valor usado **sólo una vez** y simplemente se nos pide la generación de un string diferente para cada petición en las especificaciones oficiales de OAuth 1.0.
- **oauth_timestamp**: El timestamp está comprendido entre el 1 de Enero de 1970 y la fecha actual y se utiliza de manera conjunta por el servidor con oauth_nonce para verificar que la petición nunca se ha realizado antes y así prevenir ataques.
- **oauth_version:** En la actualidad se está utilizando la versión 1.0 de OAuth. Recientemente se incorporó la nueva versión 2.0. Este parámetro es opcional.
- **oauth_callback:** Es importante mencionar que es posible utilizar OAuth tanto con aplicaciones de escritorio como aplicaciones web. Es por ello que este parámetro es importante para que el servicio se redirija a esta dirección después de que se autorice el acceso. En el caso de Twitter se puede omitir si la aplicación es de escritorio.

Creación de la firma

La firma es quizás la parte más compleja de la aplicación ya que está compuesta de una serie de pasos y si cometemos el error en alguno de ellos, el servidor rechazará la petición sin dar más explicación que un **signature_invalid**.

Todas las solicitudes OAuth 1.0 usan el mismo algoritmo básico para crear la firma base y la firma final. La firma base está compuesta por el método HTTP utilizado, seguido por el carácter ampersand ("&") y la URL accedida en formato codificado de URL, completa con la ruta (sin los parámetros de consulta), seguida nuevamente por un carácter ampersand. Luego, se toman todos los parámetros de consulta y los parámetros del cuerpo de la petición HTTP POST (en caso que el cuerpo del POST sea del tipo URL-Encoded, en otro caso el cuerpo del POST es ignorado), incluyendo los parámetros de OAuth necesarios para la negociación con la solicitud, y se acomodan de acuerdo al orden lexicográfico por nombre de parámetro seguido de su valor (en caso de parámetros duplicados), asegurando que tanto el nombre del parámetro como su valor están en formato URL-Encoded. En lugar de utilizar el carácter igual ("=") para marcar la relación clave/valor de los parámetros, se utiliza la cadena en formato URL-Encoded

correspondiente a dicho carácter "%3D". Cada parámetro es luego adjuntado utilizando la secuencia de escape correspondiente al ampersand en formato URL-Encoded: "%26".

Este algoritmo está expresado en el pseudo-código a continuación:

```
httpMethod + "&" +
url_encode( base_uri ) + "&" +
sorted_query_params.each { | k, v |
url_encode ( k ) +
"%3D" +
url_encode ( v )
}.join("%26")
```

Ejemplo de Firma base

Teniendo en cuenta los siguientes valores para los parámetros OAuth, la firma base tendrá la forma que se muestra más abajo.

- URL sobre la que se realiza el request - http://api.twitter.com/oauth/request_token.
- Método HTTP - POST
- oauth_callback - http://localhost:3005/the_dance/process_callback?service_provider_id=11
- oauth_consumer_key - a7Zcu8yDG1CgtRM50fqtQ
- oauth_nonce - QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk
- oauth_signature_method - HMAC-SHA1
- oauth_timestamp - 1272323042
- oauth_version - 1.0

```
POST&https%3A%2F%2Fapi.twitter.com%2Foauth%2Frequest_token&oauth_
callback%3Dhttp%253A%252F%252Flocalhost%253A3005%252Fthe_dance%25
2Fprocess_callback%253Fservice_provider_id%253D11%26oauth_consume
r_key%3Da7Zcu8yDG1CgtRM50fqtQ%26oauth_nonce%3DQP70eNmVz8jvdPevU3o
JD2AfF7R7odC2XJcn4XlZJqk%26oauth_signature_method%3DHMACSHA1%26oa
uth_timestamp%3D1272323042%26oauth_version%3D1.0
```

Firma encriptada y final

Una vez que se genera la firma base, concatenando y normalizando los parámetros según lo establecido por OAuth, debemos realizar un último paso para obtener la firma que finalmente pasaremos al servicio. En la misma, debemos concatenar primeramente el token **Consumer**

Secret y **Token Secret** si es que lo hubiera con un ampersand (&) entre ellos. Si utilizáramos PLAINTEXT deberíamos utilizar su código correspondiente en Unicode: %26. Si no existiera el segundo integrante, Token Secret, debemos indicar igualmente el ampersand dentro de la cadena.

Por último es necesario hacer uso del algoritmo de hash HMAC-SHA1, generado a partir del Consumer Secret y Token Secret concatenado anteriormente, para cifrar la firma base creada en el apartado anterior.

Llamada al servicio para la obtención del token temporal

Una vez lista la cabecera de nuestra petición, ya podemos realizar la llamada para recuperar el token temporal y un token secret que nos serán útiles para el último paso. Dependiendo del servicio con el que queramos contactar, realizaremos la llamada a una dirección u otra facilitada por la plataforma:

- Twitter: http://twitter.com/oauth/request_token

Puesto que esta solicitud no tiene parámetros *oauth_token* o *oauth_token_secret*, no incluimos un campo *oauth_token* en la firma base y no usamos *oauth_token_secret* para construir la clave de firmado compuesta. Nuestra clave para firmar es (notar el carácter ampersand al final) el valor correspondiente al parámetro oauth_consumer_secret de nuestra aplicación:

```
G57HLAkSoIJDcAaYF5AhA1BrSUh9Y15t5PQbuyEXs&
```

Entonces usamos esta clave compuesta de firmado para crear la firma que será el valor correspondiente del parámetro *oauth_signature* a partir de la firma base generada anteriormente por medio del algoritmo de cifrado seleccionado. Como resultado obtenemos dicho valor:

```
8wUi7m5HFQy76nowoCThusfgB+Q=
```

Ahora que tenemos la firma final, contamos con todos los elementos necesarios para realizar la petición HTTP a la URL https://api.twitter.com/oauth/request_token. Es posible generar un encabezado HTTP llamado "Authorization" con todos los parámetros relevantes de OAuth para la solicitud:

```
OAuth
oauth_nonce="QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk",
oauth_callback="http%3A%2F%2Flocalhost%3A3005%2Fthe_dance%2
Fprocess_callback%3Fservice_provider_id%3D11",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="1272323042",
oauth_consumer_key="a7Zcu8yDG1CgtRM50fqtQ",
oauth_signature="8wUi7m5HFQy76nowoCThusfgB%2BQ%3D",
oauth_version="1.0"
```

En particular, cuando Twitter recibe esta solicitud, responderá con oauth_token, oauth_token_secret (ambos en conjunto denominados como "request token" o token de solicitud), y un campo denominado oauth_callback_confirmed que indicará con el valor "true" si el parámetro Oauth_Callback es correcto. Entonces, dicha respuesta tendrá la forma

```
oauth_token=8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc&oauth
_token_secret=x6qpRnlEmW9JbQn4PQVVeVG8ZLPEx6A0TOebgwcuA&oau
th_callback_confirmed=true
```

Almacenamos los valores de oauth_token y oauth_token_secret por corto plazo mientras enviamos al usuario a que se autentique. Necesitaremos estos valores hasta que sean intercambiados para obtener el token de acceso.

Autenticación y autorización del usuario

Esta es la parte más simple del flujo estándar de OAuth. Una vez que ya hemos obtenido el token temporal (oauth_token), paso 'B' de la figura 3.1, ya podemos solicitar autorización al usuario para poder acceder a sus recursos protegidos, tal como lo muestra la figura 3.3. Dependiendo del servicio que estemos utilizando, debemos dirigirnos a su página de autorización con el token temporal como query string, paso 'C' de la figura 3.1. Típicamente, una aplicación web simplemente redirigirá al usuario a la página correspondiente de autorización del servicio utilizado. Continuando con los ejemplos en Twitter, el punto de entrada para la autorización es la URL https://api.twitter.com/oauth/authorize. Deberá tener un único parámetro adjuntado en la URL denominado oauth_token con el valor obtenido en el paso previo de solicitud de token temporal. De acuerdo con el ejemplo, el valor para oauth_token era "8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc". La URL de autorización resulta entonces

```
http://api.twitter.com/oauth/authorize?oauth_token=8ldIZyxQeVrFZXFO
ZH5tAwj6vzJYuLQpl0WUEYtWc
```

Si el usuario no se ha autenticado en Twitter recientemente, serán solicitadas sus credenciales de usuario. En caso contrario, se le presentará una petición de autorización para nuestra aplicación (como la que se muestra más abajo). Una vez que el usuario garantiza el acceso, el control es retornado a nuestra aplicación por medio de una redirección en función del valor del parámetro *oauth_callback* que habíamos especificado, paso 'D' de la figura 3.1. Esto recibe el nombre de flujo "callback" o flujo de retorno de llamada. En el caso de una aplicación de escritorio, el usuario recibirá un código PIN que deberá ingresar en nuestra aplicación.

Si estamos usando el flujo "callback", recibiremos en la etapa de retorno nuestro oauth_token (el mismo que habíamos enviado) y un campo llamado *oauth_verifier* que utilizaremos en el paso siguiente. Continuando con el ejemplo, recibiríamos:

```
oauth_token=8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc&oauth_verifie
r=pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY
```

Figura 3.3 - Solicitud de acceso a la información social del usuario.

Intercambio del token temporal por un Access Token

El último paso consiste en obtener un token de acceso. Para lograrlo necesitamos el token de solicitud o token temporal (formado por oauth_token y oauth_token_secret) y el valor de oauth_verifier obtenido en el paso previo. En el caso de Twitter, y continuando con nuestro

ejemplo, obtenemos el token de acceso desde la siguiente URL: https://api.twitter.com/oauth/access_token, pasos 'E' y 'F' de la figura 3.1. A continuación listamos todos los parámetros que formaran parte de nuestra solicitud de intercambio de tokens:

- oauth_consumer_key - a7Zcu8yDG1CgtRM50fqtQ
- oauth_nonce - 9zWH6qe0qG7Lc1telCn7FhUbLyVdjEaL3MO5uHxn8
- oauth_signature_method - HMAC-SHA1
- oauth_token - 8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc
- oauth_timestamp - 1272323047
- oauth_verifier - pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY
- oauth_version - 1.0

Como solo se utiliza en la parte de la construcción de la firma (oauth_signature), no aparece en esa lista oauth_token_secret, cuyo valor es el obtenido durante el primer paso:

```
x6qpRnlEmW9JbQn4PQVVeVG8ZLPEx6A0TOebgwcuA
```

Para llevar a cabo la solicitud, realizamos una llamada SSL utilizando el método POST. En primer lugar, preparamos nuestra firma base:

```
POST&https%3A%2F%2Fapi.twitter.com%2Foauth%2Faccess_token&oauth_con
sumer_key%3DGDdmIQH6jhtmLUypg82g%26oauth_nonce%3D9zWH6qe0qG7Lc1telC
n7FhUbLyVdjEaL3MO5uHxn8%26oauth_signature_method%3DHMACSHA1%26oauth
_timestamp%3D1272323047%26oauth_token%3D8ldIZyxQeVrFZXFOZH5tAwj6vzJ
YuLQpl0WUEYtWc%26oauth_verifier%3DpDNg57prOHapMbhv25RNf75lVRd6JDsni
1AJJIDYoTY%26oauth_version%3D1.0
```

Ahora, creamos una clave de firma compuesta utilizando tanto nuestro oauth_consumer_secret y oauth_token_secret (obtenido en el primer paso) concatenando estos valores con el carácter ampersand ("&"):

```
G57HLAkSoIJDcAaYF5AhA1BrSUh9Y15t5PQbuyEXs&x6qpRnlEmW9JbQn4PQVVeVG8Z
LPEx6A0TOebgwcuA
```

Entonces, firmamos nuestra solicitud con el campo oauth_signature resultante:

```
PUw/dHA4fnlJYM6RhXk5IU/0fCc=
```

Ahora estamos en condiciones de enviar la solicitud POST a Twitter. Creamos nuestra cabecera 'HTTP Athorization' usando nuevamente los parámetros relevantes de OAuth, incluyendo el valor de oauth_token (representado por nuestro token de solicitud) que estamos intercambiando por los tokens de acceso. Tendrá la siguiente estructura similar:

```
OAuth oauth_nonce="9zWH6qe0qG7Lc1telCn7FhUbLyVdjEaL3MO5uHxn8",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1272323047",
oauth_consumer_key="a7Zcu8yDG1CgtRM50fqtQ",
oauth_token="8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc",
oauth_verifier="pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY",
oauth_signature="PUw%2FdHA4fnlJYM6RhXk5IU%2F0fCc%3D",
oauth_version="1.0"
```

En el caso particular de Twitter, cuando este intercambio finaliza, responderá con más parámetros, incluyendo el nombre de pantalla e ID del usuario que ha sido autorizado, y un oauth_token y oauth_token_secret (en conjunto, nuestro token de acceso). En el ejemplo, una posible respuesta seria:

```
oauth_token=819797-
Jxq8aYUDRmykzVKrgoLhXSq67TEa5ruc4GJC2rWimw&oauth_token_secret=J6zix
3FfA9LofH0awS24M3HcBYXO5nI1iYe8EfBA&user_id=819797&screen_name=dzec
chin
```

Finalmente, con este token de acceso seremos capaces de construir solicitudes autorizadas que permitan interactuar con la información en torno al usuario en el servidor correspondiente. Así por ejemplo en Twitter, podremos publicar un nuevo twitt a través de su API, ilustrado en el paso 'G' de la figura 3.1.

Manejo de información social

REST

La Transferencia de Estado Representacional (REST - Representational State Transfer) fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP y a los servicios web basados en el Lenguaje de Descripción de Servicios Web (Web Services Description Language - WSDL). Ya varios grandes proveedores de Web 2.0 están migrando a esta tecnología, incluyendo a Yahoo, Google y Facebook, quienes marcaron como obsoletos a

sus servicios SOAP y WSDL y pasaron a usar un modelo más fácil de usar, orientado a los recursos [REST].

Presentando REST

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

REST no tuvo mucha atención cuando Roy Fielding lo presentó por primera vez en el año 2000 en la Universidad de California, durante la charla académica "Estilos de Arquitectura y el Diseño de Arquitecturas de Software basadas en Redes", la cual analizaba un conjunto de principios arquitectónicos de software para usar a la Web como una plataforma de Procesamiento Distribuido. Ahora, años después de su presentación, comienzan a aparecer varios frameworks REST y se convertirá en una parte integral de Java 6 a través de JSR-311.

Los 4 principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- utiliza los métodos HTTP de manera explícita.
- no mantiene estado.
- expone URIs con forma de directorios.
- transfiere XML, JavaScript Object Notation (JSON), o ambos.

A continuación vamos a ver en detalle estos cuatro principios, y explicaremos porqué son importantes a la hora de diseñar un servicio web REST.

REST utiliza los métodos HTTP de manera explícita

Una de las características claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- se usa POST para crear un recurso en el servidor
- se usa GET para obtener un recurso

- se usa PUT para cambiar el estado de un recurso o actualizarlo
- se usa DELETE para eliminar un recurso

Una falla de diseño poco afortunada que tienen muchas APIs web es el uso de métodos HTTP para otros propósitos. Por ejemplo, la petición del URI en un pedido HTTP GET, en general identifica a un recurso específico. O el string de consulta en el URI incluye un conjunto de parámetros que definen el criterio de búsqueda que usará el servidor para encontrar un conjunto de recursos. Al menos, así como el RFC HTTP/1.1 describe al GET.

Pero hay muchos casos de APIs web poco elegantes que usan el método HTTP GET para ejecutar algo transaccional en el servidor; por ejemplo, agregar registros a una base de datos. En estos casos, no se utiliza adecuadamente el URI de la petición HTTP, o al menos no se usa "a la manera REST". Si el API web utiliza GET para invocar un procedimiento remoto, seguramente se verá algo como esto:

```
GET /agregarusuario?nombre=Zim HTTP/1.
```

Este no es un diseño muy atractivo porque el método aquí arriba expone una operación que cambia estado sobre un método HTTP GET. Dicho de otra manera, la petición HTTP GET de aquí arriba tiene efectos secundarios. Si se procesa con éxito, el resultado de la petición es agregar un usuario nuevo (en el ejemplo, Zim) a la base de datos. El problema es básicamente semántico. Los servidores web están diseñados para responder a las peticiones HTTP GET con la búsqueda de recursos que concuerden con la ruta (o el criterio de búsqueda) en el URI de la petición, y devolver estos resultados o una representación de los mismos en la respuesta, y no añadir un registro a la base de datos. Desde el punto de vista del protocolo, y desde el punto de vista de servidor web compatible con HTTP/1.1, este uso del GET es inconsistente.

Más allá de la semántica, el otro problema con el GET es que al ejecutar eliminaciones, modificaciones o creación de registros en la base de datos, o al cambiar el estado de los recursos de cualquier manera, provoca que las herramientas de caché web y los motores de búsqueda (crawlers) puedan realizar cambios no intencionales en el servidor. Una forma simple de evitar este problema es mover los nombres y valores de los parámetros en la petición del URI a etiquetas XML. Las etiquetas resultantes, una representación en XML de la entidad a crear, pueden ser enviados en el cuerpo de un HTTP POST cuyo URI de petición es el padre de la entidad.

Antes:

```
GET /agregarusuario?nombre=Zim HTTP/1.
```

Después:

```
POST /usuarios HTTP/1.1
Host: miservidor
Content-type: application/xml
<usuario>
   <nombre>Zim</nombre>
</usuario>
```

El método anterior es un ejemplo de una petición REST: hay un uso correcto de HTTP POST y la inclusión de los datos en el cuerpo de la petición. Al recibir esta petición, la misma puede ser procesada de manera de agregar el recurso contenido en el cuerpo como un subordinado del recurso identificado en el URI de la petición; en este caso el nuevo recurso debería agregarse como hijo de /usuarios. Esta relación de contención entre la nueva entidad y su padre, como se indica en la petición del POST, es análoga a la forma en la que está subordinado un archivo a su directorio. El cliente indica esta relación entre la entidad y su padre y define el nuevo URI de la entidad en la petición del POST.

Luego, una aplicación cliente puede obtener una representación del recurso usando la nueva URI, sabiendo que al menos lógicamente el recurso se ubica bajo /usuarios

```
GET /usuarios/Zim
HTTP/1.1
Host: miservidor
Accept: application/xml
```

Es explícito el uso del GET de esta manera, ya que el GET se usa solamente para recuperar datos. GET es una operación que no debe tener efectos secundarios, una propiedad también conocida como idempotencia.

Se debe hacer un refactor similar de un método web que realice actualizaciones a través del método HTTP GET. El siguiente método GET intenta cambiar la propiedad "nombre" de un recurso. Si bien se puede usar el string de consulta para esta operación, evidentemente no es el uso apropiado y tiende a ser problemático en operaciones más complejas. Ya que nuestro objetivo es hacer uso explícito de los métodos HTTP, un enfoque REST sería enviar un HTTP PUT para actualizar el recurso, en vez de usar HTTP GET.

Antes:

```
GET /actualizarusuario?nombre=Zim&nuevoNombre=Dib HTTP/1.1
```

Después:

```
PUT /usuarios/Zim
HTTP/1.1
Host: miservidor
Content'Type:
application/xml
<usuario>
    <nombre>Dib</nombre>
</usuario>
```

Al usar el método PUT para reemplazar al recurso original se logra una interfaz más limpia que es consistente con los principios de REST y con la definición de los métodos HTTP. La petición PUT que se muestra arriba es explícita en el sentido que apunta al recurso a ser actualiza identificándolo en el URI de la petición, y también transfiere una nueva representación del recurso del cliente hacia el servidor en el cuerpo de la petición PUT, en vez de transferir los atributos del recurso como un conjunto suelo de parámetros (nombre = valor) en el mismo URI de la petición.

El PUT arriba mostrado también tiene el efecto de renombrar al recurso Zim a Dib, y al hacerlo cambia el URI a */usuarios/Dib*. En un servicio web REST, las peticiones siguientes al recurso que apunten a la URI anterior van a generar un error estándar "404 Not Found".

Como un principio de diseño general, ayuda seguir las reglas de REST que aconsejan usar sustantivos en vez de verbos en las URIs. En los servicios web REST, los verbos están claramente definidos por el mismo protocolo: POST, GET, PUT y DELETE. Idealmente, para mantener una interfaz general y para que los clientes puedan ser explícitos en las operaciones que invocan, los servicios web no deberían definir más verbos o procedimientos remotos, como ser */agregarusuario* y */actualizarusuario*. Este principio de diseño también aplica para el cuerpo de la petición HTTP, el cual debe usarse para transferir el estado de un recurso, y no para llevar el nombre de un método remoto a ser invocado.

REST no mantiene estado

Los servicios web REST necesitan escalar para poder satisfacer una demanda en constante crecimiento. Se usan clusters de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways de manera de conformar una topología con alta disponibilidad, que permita transferir peticiones de un equipo a otro para disminuir el tiempo total de respuesta de una invocación al servicio web. El uso de servidores intermedios para mejorar la escalabilidad hace necesario que los clientes de servicios web REST envíen peticiones completas e independientes; es decir, se deben enviar peticiones que incluyan todos los datos necesarios para cumplir el pedido, de manera que los componentes en los servidores intermedios puedan redireccionar y gestionar la carga sin mantener el estado localmente entre las peticiones.

Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o cliente de servicio web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

Servicios con estado versus servicios sin estado

La figura 3.4, presenta la ilustración de un servicio con estado, del cual una aplicación realiza peticiones para la página siguiente en un conjunto de resultados multipágina, asumiendo que el servicio mantiene información sobre la última página que pidió el cliente. En un diseño con estado, el servicio incrementa y almacena en algún lugar una variable *paginaAnterior* para poder responder a las peticiones siguientes.

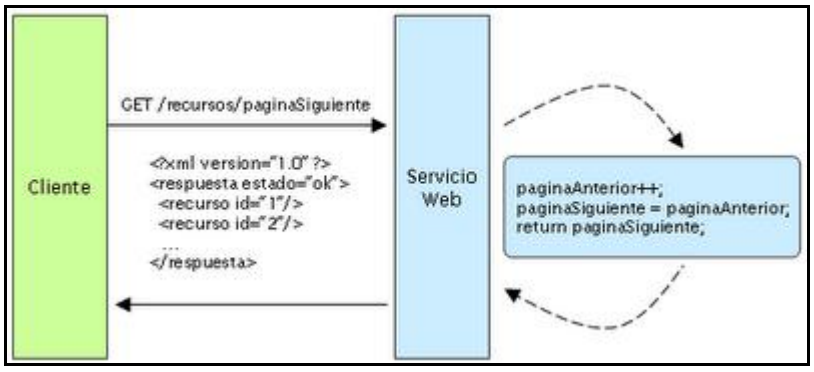


Figura 3.4 - Servicio con estado.

Los servicios con estado tienden a volverse complicados. En la plataforma Java Enterprise Edition (Java EE), un entorno de servicios con estado necesita bastante análisis y diseño desde el inicio para poder almacenar los datos eficientemente y poder sincronizar la sesión del cliente dentro de un clúster de servidores. En este tipo de ambientes, ocurre un problema que le resulta familiar a los desarrolladores de Servlets/JSP y EJB, quienes a menudo tienen que revolver buscando la causa de una *java.io.NotSerializableException* cuando ocurre la replicación de una sesión. Puede ocurrir tanto sea en el contenedor de Servlets al intentar replicar la *HttpSession* o por el contenedor de EJB al replicar un EJB con estado; en todos los casos, es un problema que puede costar mucho esfuerzo resolver, buscando el objeto que no implementa *Serializable* dentro de un grafo complejo de objetos que constituyen el estado del servidor. Además, la sincronización de sesiones es costosa en procesamiento, lo que impacta negativamente en el rendimiento general del servidor.

Por otro lado, los servicios sin estado (ver figura 3.5) son mucho más simples de diseñar, escribir y distribuir a través de múltiples servidores. Un servicio sin estado no sólo funciona mejor, sino que además mueve la responsabilidad de mantener el estado al cliente de la aplicación. En un servicio web REST, el servidor es responsable de generar las respuestas y

proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta. Por ejemplo, en el mismo ejemplo de una petición de datos en múltiples páginas, el cliente debería incluir el número de página a recuperar en vez de pedir "la siguiente", tal como se muestra en la siguiente figura:

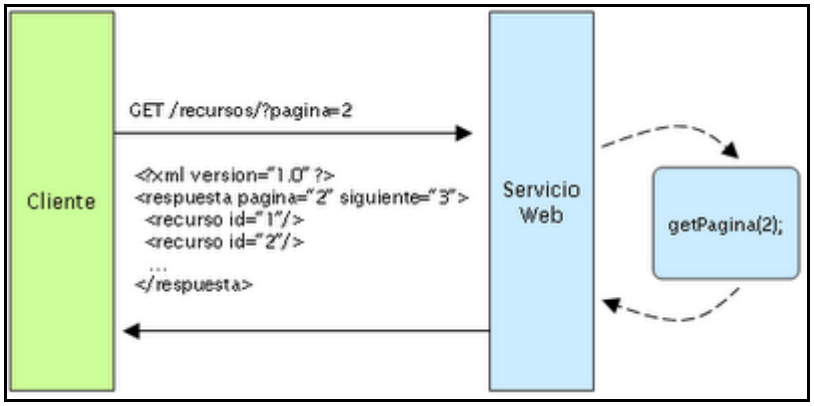


Figura 3.5 - Servicio sin estado.

Un servicio web sin estado genera una respuesta que se enlaza a la siguiente página del conjunto y le permite al cliente hacer todo lo que necesita para almacenar la página actual. Este aspecto del diseño de un servicio web REST puede descomponerse en dos conjuntos de responsabilidades, como una separación de alto nivel que clarifica cómo puede mantenerse un servicio sin estado.

Responsabilidad del servidor

- Genera respuestas que incluyen enlaces a otros recursos para permitirle a la aplicación navegar entre los recursos relacionados. Este tipo de respuestas tiene enlaces embebidos. De la misma manera, si la petición es hacia un padre o un recurso contenedor, entonces una respuesta REST típica debería también incluir enlaces hacia los hijos del padre o los recursos subordinados, de manera que se mantengan conectados.
- Genera respuestas que indican si son susceptibles de caché o no, para mejorar el rendimiento al reducir la cantidad de peticiones para recursos duplicados, y para lograr eliminar algunas peticiones completamente. El servidor utiliza los atributos Cache-Control y Last-Modified de la cabecera en la respuesta HTTP para indicarlo.

Responsabilidades del cliente de la aplicación

- Utiliza el atributo Cache-Control del encabezado de la respuesta para determinar si debe cachear el recurso (es decir, hacer una copia local del mismo) o no. El cliente también lee el atributo Last-Modified y envía la fecha en el atributo If-Modified-Since del encabezado para preguntarle al servidor si el recurso cambió desde entonces. Esto se conoce como *GET Condicional*, y ambos encabezados van de la mano con la respuesta del servidor 304 (No Modificado) y se omite al recurso que se había solicitado si no hubo cambios desde esa fecha. Una respuesta HTTP 304 significa que el

cliente puede seguir usando la copia local de manera segura, evitando así realizar las peticiones GET hasta tanto el recurso no cambie.

- envía peticiones completas que pueden ser serviciadas en forma independiente a otras peticiones. Esto implica que el cliente hace uso completo de los encabezados HTTP tal como está especificado por la interfaz del servicio web, y envía las representaciones del recurso en el cuerpo de la petición. El cliente envía peticiones que hacen muy pocas presunciones sobre las peticiones anteriores, la existencia de una sesión en el servidor, la capacidad del servidor para agregarle contexto a una petición, o sobre el estado de la aplicación que se mantiene entre las peticiones.

Esta colaboración entre el cliente y el servicio es esencial para crear un servicio web REST sin estado. Mejora el rendimiento, ya que ahorro ancho de banda y minimiza el estado de la aplicación en el servidor.

REST expone URIs con forma de directorios

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el servicio web REST, y si el servicio va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URI de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

Una forma de lograr este nivel de usabilidad es definir URIs con una estructura al estilo de los directorios. Este tipo de URIs es jerárquica, con una única ruta raíz, y va abriendo ramas a través de las subrutas para exponer las áreas principales del servicio. De acuerdo a esta definición, una URI no es solamente una cadena de caracteres delimitada por barras, sino más bien un árbol con subordinados y padres organizados como nodos. Por ejemplo, en un servicio de hilos de discusiones que tiene temas varios, se podría definir una estructura de URIs como esta:

```
http://www.miservicio.org/discusion/temas/{tema}
```

La raíz, */discusion*, tiene un nodo */temas* como hijo. Bajo este nodo hay un conjunto de nombres de temas (como ser tecnología, actualidad, y más), cada uno de los cuales apunta a un hilo de discusión. Dentro de esta estructura, resulta fácil recuperar hilos de discusión al escribir algo después de */temas/*.

En algunos casos, la ruta a un recurso encaja muy bien dentro de la idea de "estructura de directorios". Por ejemplo, tomemos algunos recursos organizados por fecha, que son muy prácticos de organizar usando una sintaxis jerárquica.

El siguiente ejemplo es intuitivo porque está basado en reglas:

```
http://www.miservicio.org/discusion/2008/12/23/{tema}
```

El primer fragmento de la ruta es un año de cuatro dígitos, el segundo fragmento es el mes de dos dígitos, y el tercer fragmento es el día de dos dígitos. Puede resultar un poco tonto explicarlo de esta manera, pero es justamente el nivel de simpleza que buscamos. Tanto humanos como máquinas pueden generar estas estructuras de URI porque están basadas en reglas. Como vemos, es fácil llenar las partes de esta URI, ya que existe un patrón para crearlas:

```
http://www.miservicio.org/discusion/{año}/{mes}/{dia}/{tema}
```

Podemos también enumerar algunas guías generales más al momento de crear URIs para un servicio web REST:

- ocultar la tecnología usada en el servidor que aparecería como extensión de archivos (.jsp, .php, .asp), de manera de poder portar la solución a otra tecnología sin cambiar las URI.
- mantener todo en minúsculas.
- sustituir los espacios con guiones o guiones bajos (uno u otro).
- evitar el uso de strings de consulta.
- en vez de usar un 404 Not Found si la petición es una URI parcial, devolver una página o un recurso predeterminado como respuesta.

Las URI deberían ser estáticas de manera que cuando cambie el recurso o cambie la implementación del servicio, el enlace se mantenga igual. Esto permite que el cliente pueda generar "favoritos" o bookmarks. También es importante que la relación entre los recursos que está explícita en las URI se mantenga independiente de las relaciones que existen en el medio de almacenamiento del recurso.

REST transfiere XML, JSON, o ambos

La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples "fotos" en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y etiquetas XML, donde los valores de

los elementos en el XML contienen los valores de las filas. O, si el sistema tiene un modelo de datos, la representación de un recurso es una fotografía de los atributos de una de las cosas en el modelo de datos del sistema. Estas son las cosas que serviciamos con servicios web REST.

La última restricción al momento de diseñar un servicio web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las peticiones/respuestas. Acá es donde realmente vale la pena mantener las cosas simples, legibles por humanos, y conectadas.

Los objetos del modelo de datos generalmente se relacionan de alguna manera, y las relaciones entre los objetos del modelo de datos (los recursos) deben reflejarse en la forma en la que se representan al momento de transferir los datos al cliente. En el servicio de hilos de discusión anterior, un ejemplo de una representación de un recurso conectado podría ser un tema de discusión raíz con todos sus atributos, y links embebidos a las respuestas al tema.

```
<discusion fecha="{fecha}" tema="{tema}">
      <comentario>{comentario}</comentario>
      <respuestas>
            <respuesta de="gaz@mail.com"
            href="/discusion/temas/{tema}/gaz"/>
            <respuesta de="gir@mail.com"
            href="/discusion/temas/{tema}/gir"/>
      </respuestas>
</discusion>
```

Por último, es bueno construir los servicios de manera que usen el atributo HTTP Accept del encabezado, en donde el valor de este campo es de tipo MIME. De esta manera, los clientes pueden pedir por un contenido en particular que mejor pueden analizar. Algunos de los tipos MIME más usados para los servicios web REST son:

MIME-Type	***Content-Type***
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado 'HTTP Accept' es un mecanismo conocido como *negociación de contenido*, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

Conclusión

No siempre REST es la mejor opción. Está surgiendo como una alternativa para diseñar servicios web con menos dependencia en middleware propietario (por ejemplo, un servidor de aplicaciones), que su contraparte SOAP y los servicios basados en WSDL. De algún modo, REST es la vuelta a la web antes de la aparición de los grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. Como examinamos en este artículo, XML sobre HTTP es una interfaz muy poderosa que permite que aplicaciones internas, como interfaces basadas en JavaScript Asincrónico + XML (AJAX) puedan conectarse, ubicar y consumir recursos. De hecho, es justamente esta gran combinación con AJAX que generó esta gran atención que tiene REST hoy en día.

Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera de brindar datos a distintas aplicaciones, formateados en distintas maneras. REST ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente (mashups) y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande.

Este artículo es tan sólo una breve introducción a los servicios web REST, y espera ser el puntapié inicial para que puedan continuar explorando esta forma de exponer servicios.

OpenSocial

El API de OpenSocial [OpenSocial] es un conjunto de interfaces para programación de aplicaciones (API) comunes que permite crear aplicaciones sociales en muchos sitios web. Hay dos formas de acceder al API de OpenSocial: a través de la aplicación cliente mediante el API de JavaScript y a través del servidor con las API de datos REST.

API JavaScript

El API de JavaScript se encuentra incluida en el espacio de nombres opensocial y proporciona acceso a tres áreas de funcionalidad principales:

- Personas: información sobre personas concretas y sus relaciones.
- Actividades: posibilidad de publicar y ver actualizaciones sobre lo que está haciendo la gente.
- Persistencia: un sencillo almacén de datos de valor-clave que admite aplicaciones que mantienen un seguimiento del estado sin necesidad de utilizar el servidor.

Éstas son algunas de las cosas que podrás hacer con el API de JavaScript:

- crear aplicaciones sin necesidad de mantener tu propio servidor,
- crear aplicaciones que incluyan un componente de servidor (para procesamiento sin conexión o acceso desde otros sitios web),
- crear aplicaciones completamente nuevas,
- mostrar aplicaciones web ya existentes dentro de sitios web sociales,

- añadir funciones de socialización a los gadgets existentes,
- crear una aplicación que se pueda ejecutar en el contexto de muchos sitios web de redes sociales distintos.

El API de JavaScript se ha diseñado para utilizar tecnologías web estándar:

- Está empaquetada como un conjunto de métodos en el espacio de nombre opensocial.*.
- Permite utilizar bibliotecas de terceros y cualquier técnica de programación JavaScript estándar.
- Incluye un sistema completo y asíncrono de devolución de llamadas para admitir una gran interactividad con AJAX.

API de datos REST

El API de datos REST proporciona funciones complementarias al API de JavaScript que te permitirán acceder a personas, actividades y datos almacenados en tu servidor.

El API de datos REST también está diseñado para utilizar tecnologías web estándar:

- Las interacciones con el servidor están basadas en el protocolo REST Atom Pub.
- La autenticación se gestiona con OAuth.

Proveedores de Servicios Sociales

Luego del análisis y reseña de las tecnologías existentes, procederemos a documentar las herramientas de integración que disponen los sitios de redes sociales más populares estudiados en este trabajo. Además, explicaremos cuales de las tecnologías mencionadas aparecen en cada red social.

Facebook

Actualmente, Facebook tiene como núcleo de su plataforma lo que denomina como grafo social, las personas y las conexiones que tienen a todo lo que les interesa. La plataforma Facebook es el conjunto de APIs y herramientas que permite la integración con el grafo social para manejar la registración, personalización, y el tráfico, ya sea a través de aplicaciones en Facebook o sitios y dispositivos externos. Entre los componentes que forman la plataforma, existen:

- Graph API: es el corazón de la plataforma, permitiendo leer y escribir datos en Facebook. Provee una vista consistente y simple del grafo social, representando uniformemente objetos (como personas, fotos, etc.) y las conexiones entre estos (como amistades, gustos, etc.).
- Autenticación: La autenticación de Facebook permite a nuestras aplicaciones interactuar con la Graph API sobre el comportamiento de los usuarios de Facebook, y

provee un único mecanismo de autenticación que se puede utilizar desde la web, dispositivos móviles, y aplicaciones de escritorio.

- Plugins sociales: los plugins sociales permiten proveer experiencias sociales atractivas a los usuarios con pocas líneas de código HTML. Estos plugins son servidos por Facebook, y por esta razón, el contenido es personalizado a quien lo visualiza dependiendo si está o no autenticado.
- Protocolo de grafo abierto: permite integrar nuestras páginas al grafo social. Estas páginas adquieren la funcionalidad de los demás objetos del grafo, incluyendo links de perfiles y actualizaciones de los usuarios conectados.

Referencias de las APIs provistas

Núcleo de la plataforma:

- Graph API: cada objeto en el grafo social tiene un ID único. Es posible recuperar los datos asociados con un objeto por medio de la URL https://graph.facebook.com/ID. Alternativamente, las personas y páginas con nombres de usuario pueden ser recuperadas usando dicho nombre de usuario como un ID. Todas las respuestas son objetos JSON. Todos los objetos en el grafo social de Facebook están conectados entre sí por medio de relaciones, como por ejemplo, amistad. Estas relaciones reciben el nombre de conexiones en la API. Es posible examinar las conexiones entre objetos usando URLs con la siguiente estructura https://graph.facebook.com/ID/TIPO_DE_CONEXION. Las conexiones soportadas para personas y paginas incluyen: amigos (friends), feed de noticias (home), feed de perfil (feed), gustos (likes), películas (movies), libros (books), notas (notes), etiquetas en fotos (photos), álbumes de fotos (albums), videos (videos), eventos (events) y grupos (groups).

  Además de recuperar datos desde el grafo social, es posible llevar a cabo las siguientes acciones a partir de esta API:

  - Selección: por defecto, la mayoría de las propiedades de los objetos son retornadas cuando ejecutamos una consulta. Es posible especificar los campos (o conexiones) que deseamos empleando el parámetro "fields" en la consulta.
  - Introspección: soportada por la Graph API, permite visualizar todas las conexiones que tiene un objeto. Para obtener esa información, se debe agregar el parámetro "metadata=1" a la URL del objeto, y la correspondiente respuesta JSON contendrá una propiedad "metadata" que lista todas las conexiones soportadas por el objeto dado.
  - Autorización: la Graph API usa OAuth 2.0 para autenticación, mecanismo descripto anteriormente en este capítulo.
  - Publicando en Facebook: es posible realizar publicaciones en el grafo Facebook por medio de solicitudes HTTP POST a las URLs de conexión apropiadas a continuación:

Método	**Descripción**	**Argumentos**
`PROFILE_ID/feed`	Escribe en el muro del perfil dado	`message`, `picture`, `link`, `name`, `caption`, `description`, `source`
`/POST_ID/comments`	Agrega un comentario en el post dado	`message`
`/POST_ID/likes`	Gustar el post dado	*ninguno*
`/PROFILE_ID/notes`	Escribir una nota en el perfil dado	`message`, `subject`
`/PROFILE_ID/links`	Publicar un link en el perfil dado	`link`, `message`
`/PROFILE_ID/events`	Crear un evento	`name`, `start_time`, `end_time`
`/EVENT_ID/attending`	Indicar asistencia al evento dado	*ninguno*
`/EVENT_ID/maybe`	Indicar la posibilidad de asistencia al evento dado	*ninguno*
`/EVENT_ID/declined`	Declinar el evento dado	*ninguno*
`/PROFILE_ID/albums`	Crear un álbum	`name`, `message`
`/ALBUM_ID/photos`	Subir una foto al álbum dado	`message`

    - Borrado de objetos: es posible eliminar objetos en el grafo mediante solicitudes HTTP del tipo DELETE sobre la URL del objeto determinado

- Plugins sociales: constituidos en forma de widget, proporcionan mecanismos de interacción social con la red que se pueden incorporar en sitios de terceros. Estos incluyen:

    - Botón de gustar: La figura 3.6 muestra la vista de este widget que permite expresar interés por un contenido, y dicha acción es notificada en la red social compartiendo el enlace del mismo.

Figura 3.6 - Vista del widget del botón gustar.

- Recomendaciones: en base a la actividad social del sitio, este plugin muestra sugerencias a los usuarios sobre ciertos contenidos que pueden ser de su interés. La figura 3.7 muestra la vista de este widget.

Figura 3.7 - Vista del widget de recomendaciones.

- Botón de Login: permite llevar a cabo la autenticación, como así también mostrar la imagen de perfil de aquellos usuarios amigos que han iniciado sesión.

- Comentarios: permite añadir un comentario relacionado a un contenido particular. La figura 3.8, muestra el aspecto de dicho widget.

Figura 3.8 - Vista del widget de comentarios.

- Feed de actividades: La figura 3.9 muestra la vista del widget correspondiente al feed de actividades. Este nos permite mostrar a los usuarios lo que están haciendo sus amigos en el sitio donde se coloca el widget a través de comentarios y gustos.

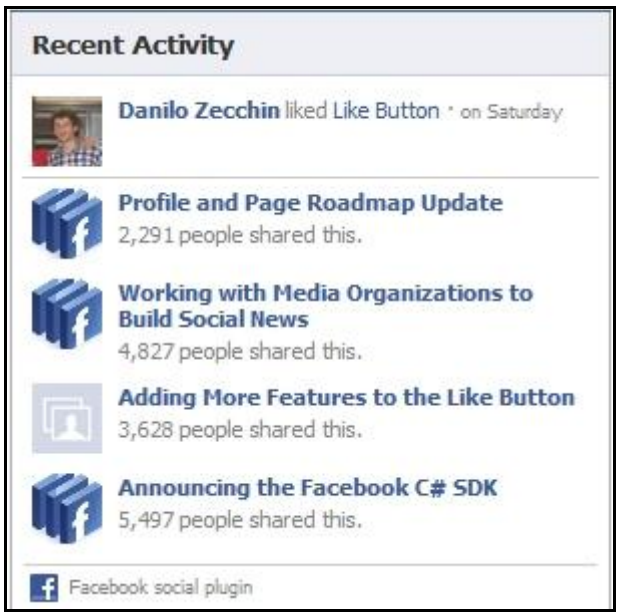


Figura 3.9 - Vista del widget feed de actividades.

- Stream en tiempo real: permite a los usuarios compartir en tiempo real comentarios y actividades como si estuvieran interactuando en un evento en vivo. La figura 3.10 muestra la vista de dicho widget.

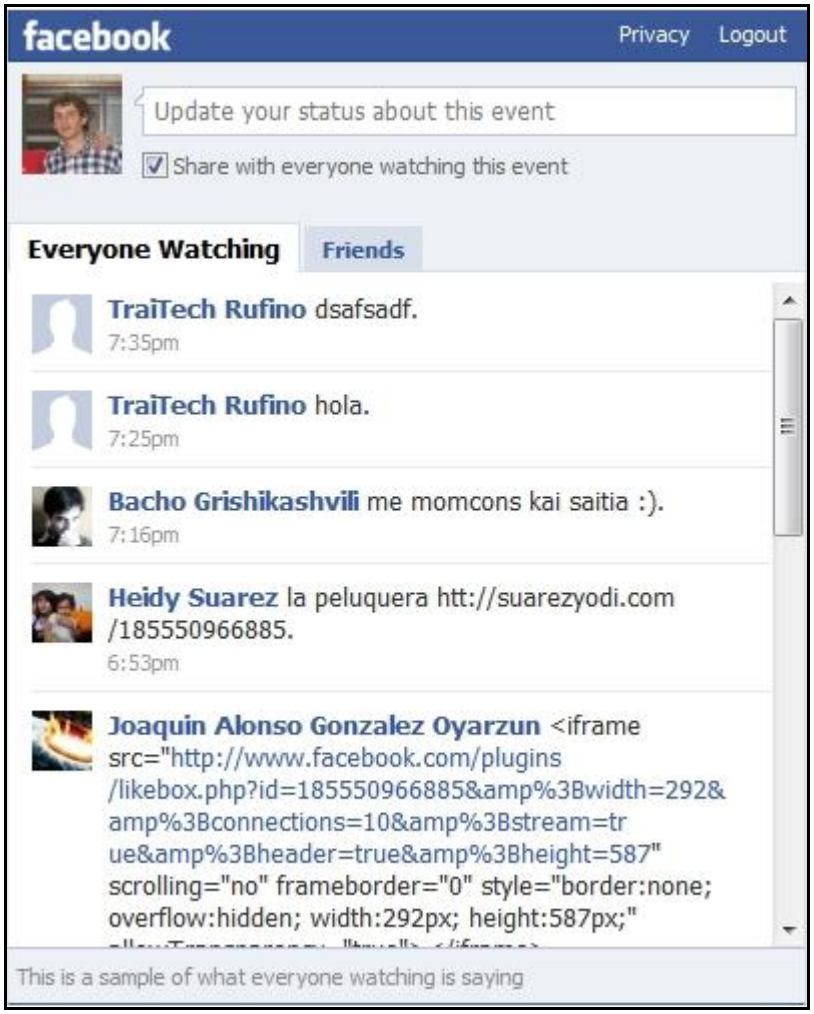


Figura 3.10 - Vista del widget stream en tiempo real.

APIs avanzadas

- Facebook Query Language (FQL): Facebook Query Language, o FQL, permite utilizar una interfaz de estilo SQL para consultar los datos expuestos por la Graph API. Provee algunas características avanzadas no disponibles en la Graph API, incluyendo procesamiento por lotes de varias consultas en una sola llamada. Se pueden ejecutar consultas FQL utilizando el siguiente vinculo [FQL]. también se puede especificar un formato de respuesta como XML o JSON con los parámetros `format` de la consulta. Las consultas son de la forma `SELECT [campos] FROM [tabla] WHERE [condiciones]`. A diferencia de SQL, la cláusula `FROM` del FQL sólo puede contener una sola tabla. Se puede utilizar la palabra clave `IN` en las cláusulas `SELECT` o `WHERE` de modo de hacer subconsultas, pero estas subconsultas no pueden referenciar variables fuera del ámbito de la consulta.

- Facebook Markup Language (FBML): Permite construir aplicaciones Facebook que se integran profundamente en la experiencia de uso de los usuarios de Facebook. Es

posible utilizar cada punto de integración provista por Facebook, incluyendo el perfil, acciones del perfil, y el entorno de interacción de la aplicación, denominado canvas. La utilización de este lenguaje de marcas implica trabajar con la API JavaScript de Facebook. Por ejemplo:

```
<script
src="http://static.ak.connect.facebook.com/connect.php/es_LA
" type="text/javascript"></script>
<script type="text/javascript">
      FB.init("e0f9d7a9afa7653a99053da01144be39");
</script>
<fb:comments xid="1234"></fb:comments>
```

El fragmento de código HTML anterior tiene como objetivo desplegar el widget de comentarios, visto antes en el desarrollo de esta red. Las etiquetas prefijadas con la cadena "fb" forman parte de este lenguaje diseñado por la red social.

- API REST previa: Es la versión previa a la Graph API, y la única existente al momento de comenzar el desarrollo de este trabajo. Permite interactuar con el sitio web de Facebook programáticamente a través de solicitudes HTTP. Esta API también soporta el mecanismo de autenticación OAuth 2.0, como así también el original Facebook Connect antes descripto. Para ejecutar solicitudes sobre la API REST usando OAuth, las URLs de las mismas respetan el siguiente formato: *https://api.facebook.com/method/**method_name**?access_token=**oauth_access_token** &arg0=value0&arg1=value1*
  Existe un amplio conjunto de métodos exportados por esta API. Por ejemplo, el método "comments.add" agrega un comentario asociado a un id único que los agrupa, publicando dicha actividad en el perfil del usuario. Así, la URL tendrá la siguiente estructura:
  https://api.facebook.com/method/comments.add?access_token=oauth_access_token &xid=20100707nota0001&text="comentario".
- JavaScript Client Library: El SDK de JavaScript permite acceder a todas las funciones de la Graph API a través de JavaScript, y proporciona un conjunto de funciones del lado del cliente para la autenticación y el intercambio. El SDK JavaScript es también necesario para utilizar versiones XFBML de los plugins sociales. Muchas funciones en el SDK de JavaScript exigen el identificador de una aplicación.
  La forma más eficiente para cargar el SDK en un sitio hacerlo de forma asíncrona, de modo que no bloquee la carga de otros elementos de la página.

Twitter

Vista General

Para utilizar la API de Twitter, lo primero que se debe hacer es registrar una aplicación cliente. A cada aplicación cliente que se registre, se le suministrara una 'consumer key' y una 'consumer secret'. Para aquellos que están familiarizados, este esquema de 'key' y 'secret' es similar a las claves públicas y privadas utilizadas en los protocolos como SSH. Dicha 'key' y 'secret', se utilizará junto con la biblioteca de OAuth (en el lenguaje de programación que se haya elegido), de modo de firmar cada solicitud que se realice a la API.

El 30 de junio de 2010, el mecanismo de autorización base (que se basa en pasar directamente un nombre de usuario y contraseña para todas las solicitudes a la API), ya no serán admitidos. Las aplicaciones web son alentadas a utilizar OAuth para autenticar usuarios y actuar en su nombre. Aplicaciones móviles y de escritorio son animadas a usar OAuth (también se ofrece la posibilidad de utilizar XAuth. Aunque para utilizar este protocolo, se debe enviar una solicitud justificando por que XAuth sería la mejor opción).

API

La API se ajusta a los principio de diseños REST, siendo completamente basado en HTTP. Los métodos para recuperar datos de la API de Twitter requieren una solicitud GET. Los métodos que envíen, cambien o destruyan los datos requieren una solicitud POST. Una solicitud DELETE también se acepta en los métodos que destruyan datos. Los métodos de la API que requieran una solicitud HTTP particular, devolverán un error si se hace el pedido con el método incorrecto. Los códigos de respuesta HTTP son bastante expresivos.

La API de Twitter se compone de tres partes: dos APIs REST y una API de Streaming. Las dos APIs REST son totalmente distintas debido a la evolución de la historia. Summize, Inc. fue originalmente una compañía independiente que proporcionaba capacidad de búsqueda para los datos de Twitter. Summize fue adquirida más tarde por Twitter y rebautizada como Twitter Search. Renombrar y rebautizar el sitio resulto fácil, aunque la plena integración de Twitter Search y su API de búsqueda con el código base de Twitter no resulto sencillo. Es la idea unificar las API, pero hasta tanto eso no sea posible, la API REST y la API de búsqueda se mantendrán como entidades separadas. La API de Streaming es distinta de las API REST dado que esta admite conexiones de larga duración en una arquitectura diferente.

Los métodos de la API REST de Twitter le permiten a los desarrolladores acceder a datos básicos de Twitter. Esto incluye los plazos de actualización, los datos de estado e información del usuario. Los métodos de búsqueda de la API, le dan a los desarrolladores la posibilidad de interactuar con Twitter Search. La API de Streaming proporciona acceso en tiempo real a grandes volúmenes de Tweets.

Llamadas limitadas a la API REST

El límite de llamadas a la API REST varía en función del método de autorización que se esté utilizando.

- Las llamadas anónimas se basan en la IP de la máquina y se les permite hasta 150 peticiones por hora.
- Se permiten hasta 150 peticiones por hora utilizando la autenticación básica.
- Se permiten hasta 350 peticiones con autenticación por OAuth.

Los límites son aplicados a los métodos que soliciten información, como el comando GET de HTTP. En general, los métodos de la API que utilizan HTTP POST para enviar datos a Twitter no son limitados. Si la aplicación está siendo limitada por la API REST, se recibirá el código HTTP 400.

Limite en la paginación

A la API REST

Los clientes pueden acceder a un máximo teórico de 3.200 estados a través de la página y los parámetros para el recuento de los métodos de la API REST user_timeline. Otros métodos de control temporal de un máximo teórico de 800 estados. Las solicitudes de más que el límite se traducirá en una respuesta con un código de estado de 200 y un resultado vacío en el formato solicitado. Twitter aún mantiene una base de datos de todos los tweets enviados por un usuario. Sin embargo, para asegurar el rendimiento del sitio, este límite temporal artificial en su lugar.

A la API de búsqueda

Los clientes pueden solicitar hasta 1.500 estados a través de la página y los parámetros de RPP para el método de búsqueda. En la respuesta a una solicitud que exceda este límite será un código de estado de 200 y un resultado vacío en el formato solicitado. Este límite artificial para garantizar el desempeño del sistema de búsqueda. También limitar el tamaño del índice de búsqueda mediante la colocación de un límite de fecha de las actualizaciones que le permiten realizar la búsqueda. Este límite es actualmente alrededor de 1,5 semanas, pero es dinámico y está sujeto a disminuir el número de tweets por día sigue creciendo.

Plugins sociales

Constituidos en forma de widget, proporcionan mecanismos de interacción social con la red que se pueden incorporar en sitios de terceros.

Estos plugins pueden ser alguno de:

- Widget de Perfil: Muestra tus tweets más recientes en cualquier página.
- Widget de Búsqueda: Despliega resultados de búsqueda en tiempo real. Ideal para eventos en vivo, transmisiones, conferencias, programas de TV o incluso sólo para mantenerte al día con las noticias.
- Faves Widget: Revela tus tweets favoritos. También en tiempo real, este widget arrastrará los tweets que has marcado como favoritos. Es genial para administrar.
- Widget para listas: Pone en una lista a tus twitteros favoritos y los muestra en un widget. Muy bueno también para moderaciones.

Por ejemplo, para poder embeber el widget de búsqueda será necesario copiar el siguiente fragmento de código:

```
<script
src="http://widgets.twimg.com/j/2/widget.js"></script>
<script>
      new TWTR.Widget({
            version: 2,
            type: 'search',
            search: '',
            interval: 6000,
            title: 'Un titulo',
            subject: Enriquecer sitio',
            width: 250,
            height: 300,
            theme: { shell:
                        {background: '#8ec1da',
                        color: '#ffffff' },
                        tweets: { background: '#ffffff',
                              color: '#444444',
                              links: '#1985b5' }
                        },
            features: { scrollbar: false,
                        loop: true,
                        live: true,
                        hashtags: true,
                        timestamp: true,
                        avatars: true,
                        behavior: 'default' }
      }).render().start();
</script>
```

La figura 3.11 nos muestra la página de configuración del widget. En la parte izquierda vemos las propiedades de configuración, mientras que en la sección derecha de la imagen, se muestra la vista preliminar del widget a la derecha de la imagen.

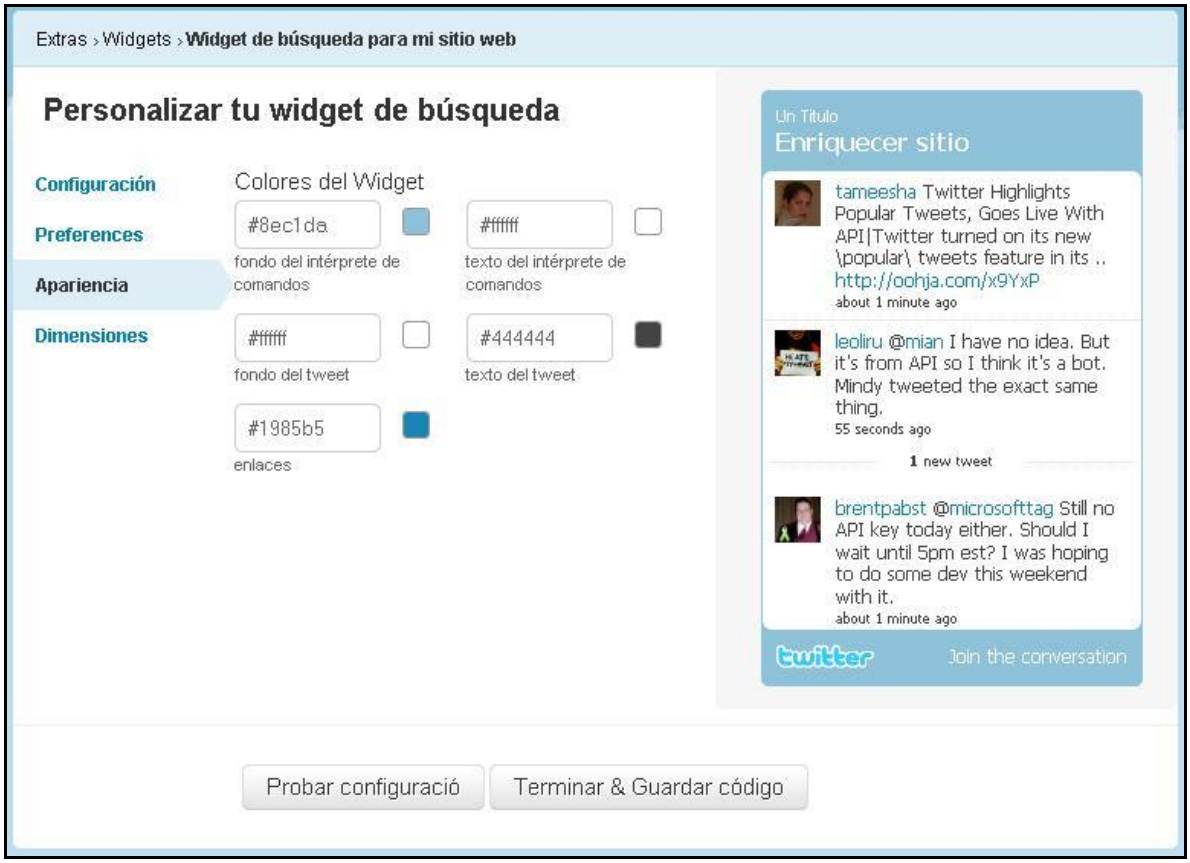


Figura 3.11 - configuración del widget.

Socialización por capas

En el presente capítulo procedemos a describir lo que proponemos como solución para abordar los problemas descriptos anteriormente en el documento. En esencia, vamos a implementar un "Enriquecedor social de sitios" que estará compuesto por un conjunto de componentes o unidades independientes, que interactúan entre sí, cada una de las cuales tiene un objetivo específico. A continuación presentaremos una breve descripción de cada uno, indicando que parte de las problemáticas planteadas se está intentando resolver.

Especificación de la API estándar del socializador

Antes de proceder a enumerar y reseñar cada componente, queremos dedicar algunos párrafos a explicar el proceso de abstracción que hemos realizado de los diferentes mecanismos de socialización que ponen a disposición los SNSs. Este aporte consiste en contemplar un conjunto estándar de acciones sociales que serán incluidas, constituyendo lo que representaría la API base del socializador que proponemos. Esta API común tiene como objetivo definir un conjunto unificado y estándar de métodos o acciones sociales para abstraernos del servicio social en el cual se promoverán tales acciones o contenidos. Entonces, esta API funcionará como integrador de diversos servicios sociales. Para llevar a cabo esta definición e identificación, nos enfocamos en los preceptos definidos por la programación orientada a aspectos (POA).

La Programación Orientada a Aspectos (POA) es un paradigma de programación propuesto por G. Kiczales [KICZ], que tiene como objetivo capturar los crosscutting concerns (aspectos cruzados) de un sistema. Se denomina crosscutting concerns a aquellos requerimientos secundarios o no-funcionales, compartidos por varios módulos de un programa y cuya implementación no puede ser ubicada en un único punto del sistema. Son por lo tanto, responsables directos del enmarañamiento de código. La seguridad, el caching, la encriptación, el logging son ejemplos típicos de este tipo de requerimiento. POA complementa las metodologías existentes tales como POO (Programación Orientada a Objetos) y la programación procedural, con conceptos y construcciones que permiten modularizar crosscutting concerns. POA incorpora el concepto de aspecto, el cual constituye una unidad de encapsulamiento de los crosscutting concerns. De esta manera, es posible utilizar POO como metodología base implementando los requerimientos principales como clases y utilizar POA y encapsular en aspectos los requerimientos secundarios [PAP] [LAD]. Este enfoque, donde los aspectos son considerados como complementos de una aplicación base, recibe el nombre de POA asimétrica, en contraposición a otros enfoques, denominados simétricos, en donde los aspectos cumplen un papel central en la construcción de la aplicación [FRAINE]. En lo que sigue, utilizaremos el término POA para referirnos a POA asimétrica. Los aspectos pueden ser clasificados de acuerdo a las características del crosscutting concern que implementan en:

- Aspectos ortogonales (independientes de la aplicación): son usados para agregar código a una aplicación de manera ortogonal. La aplicación es independiente del aspecto, por lo que permanece totalmente funcional aún sin el mismo. Estos aspectos son incorporados de manera automática en una aplicación sin necesidad de realizar cambios manuales en ella. Un aspecto de debugging pertenece a esta categoría.
- Aspectos no-ortogonales (dependientes de la aplicación): hacen un crosscutting profundo. El estado, estructura o lógica de la aplicación influye en el código de estos aspectos de manera tal que son sólo aplicables en el contexto de una aplicación específica. Los aspectos de esta clase forman una parte integral de la aplicación. [DEWIN]
- Aspectos semi-ortogonales (cooperantes con la aplicación): no son ortogonales, pero tampoco son totalmente dependientes. Estos aspectos apuntan a concerns no-funcionales que necesitan alguna cooperación de la aplicación para funcionar. Es el caso de los aspectos que implementan seguridad. [DEWIN] Actualmente, existen diferentes lenguajes de programación que incluyen extensiones para soporte de aspectos; en la familia de los orientados a objetos, podemos mencionar a: AspectJ [LAD1] [ECL1] (extensión de Java), AspectC++ (extensión de C++), Apostle (extensión de Smalltalk) y dentro de los procedurales se encuentra AspectC y Aspicere (extensiones de C).

Entonces, al analizar la funcionalidad que proveen las distintas redes sociales (teniendo en cuenta además lo expuesto en el capítulo de Estado del Arte), podemos identificar cierto comportamiento que se pone de manifiesto en la mayoría. Dicho comportamiento es fundamental para poder enriquecer sitios web mediante el uso de aspectos sociales. De este modo, vemos una analogía directa entre el concepto de "aspecto cruzado" y las operaciones o aspectos que están presentes en los proveedores de servicios sociales. En tal sentido hemos definido los siguientes aspectos comunes que conformarán la API unificada:

- Login: identificación social en la red destino.
- Comentar: dado un contenido asociado, se comenta o reflexiona al respecto. Esto es compartido y difundido en la red social.
- Compartir: publicar o difundir un contenido determinado en la red social destino.
- Calificar (Gustar/Votar/Rankear): expresar interés de diversas formas sobre un contenido particular. De acuerdo a la acción, tal condición podrá manifestarse como una opinión cuantitativa (rank) o cualitativa (voto o gustar).

La figura 4.1 que se muestra a continuación, ilustra el proceso de abstracción realizado, relacionando los aspectos o concerns sociales tanto con aquellos proveedores que disponen una implementación como así también con los diferentes tipos de contenidos a los que se puede aplicar.

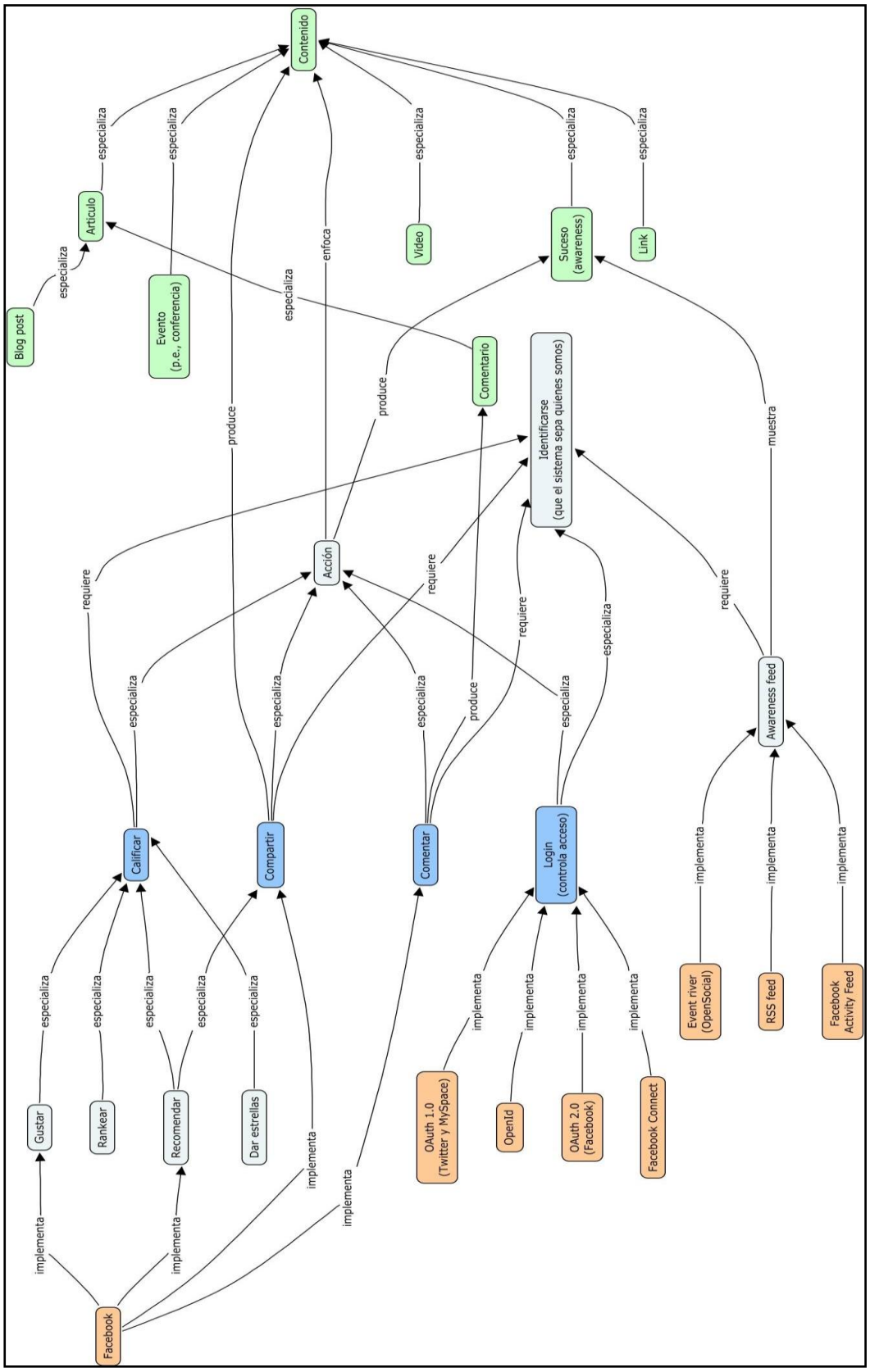


Figura 4.1 - Abstracción de aspectos sociales

Como se ve en la figura 4.1, los cuadros sombreados en azul representan los aspectos abstractos identificados. Aquellos cuadros naranjas corresponden a diferentes implementaciones existentes en diversos proveedores sociales. Luego, los cuadros sombreados en verde claro representan diferentes tipos de contenidos (tanto sociales como estáticos). Finalmente, los cuadros grises representan conceptos intermedios que actúan como conectivos entre los tipos de conceptos definidos previamente.

¿Cómo se implementan los aspectos definidos en las redes sociales estudiadas?

Ahora nuestro desafío es resolver la forma de adaptar cada aspecto que nos interesa a la red social objetivo, teniendo en cuenta que ésta puede o no tener una implementación nativa del aspecto social pretendido por nuestra interface.

Para llevar a cabo esto, analizamos y estudiamos distintas implementaciones en diversas redes sociales para los aspectos definidos, quedándonos con un subconjunto de ellas, las cuales tienen un soporte más adecuado a los aspectos con que queremos enriquecer nuestro contenido social. A continuación se mencionan los casos de estudio seleccionados, y en cada uno de estos se describen si las implementaciones tienen soporte nativo por parte de la red social, si es posible construir alguna adaptación o simulación por parte nuestra o si no es posible implementarlas.

Llamamos soporte nativo al caso en el que un proveedor social brinda una funcionalidad específica y dedicada para algún aspecto social que hemos identificado. En estos casos ocurre que el proveedor le da por nombre a dicha funcionalidad el mismo nombre del aspecto que identificamos y desarrollamos en nuestra API. Un claro ejemplo de esto ocurre en Facebook, que provee un mecanismo que nos permite comentar y otro totalmente distinto que nos permite compartir. Una consecuencia de esta situación, es que el aspecto en cuestión tiene el efecto esperado en el proveedor social.

Por otro lado, entendemos por soporte simulado, el caso en que el proveedor de aspectos sociales no brinde tantos servicios como funcionalidades proponemos en nuestra API, se analiza el caso de poder adaptar el uso de la funcionalidad que si nos provea de modo de dar un soporte al aspecto social que identificamos.

Facebook

- Login: Nativo. Inicialmente, Facebook proveía un mecanismo para implementarlo asincrónicamente, usando la API JavaScript provista por la red social. Para este caso, es necesario implementar la interacción extra que permite intercambiar los datos de la sesión de usuario entre el cliente y el servidor al momento de la identificación, e invalidarlos en el logout. Otra alternativa existente al momento de comenzar a escribir este informe era usar los métodos REST disponibles en Facebook para la autenticación a través de la API en Java. Recientemente

Facebook, junto con otras incorporaciones, adoptó el nuevo mecanismo o estándar para autenticación OAuth 2.0, convirtiéndose en uno de los primeros servidores en implementarlo.

- Comentar: Nativo.
- Compartir: Nativo
- Calificar (Gustar/Votar/Puntuar): Al momento de escribir este documento, la forma que adopta este aspecto es la de "gustar" un contenido y no se encuentra disponible para ser implementado a través de la API REST o Graph API. Para el mismo, solo es posible utilizar el widget provisto por la red social, con el objetivo de expresar este tipo de opiniones sobre enlaces externos. Entonces, el aspecto de gustar es simulado usando comentarios, es decir, agregar un comentario que indique que al usuario le gusta el contenido. Dicha acción de comentar se realiza en modo silencioso, es decir, no se notifica del comentario en el stream social del usuario, sino que se envía una notificación diferente, ilustrando esta acción. De esta misma forma, se puede simular la votación, usando como texto de comentario el valor asociado a la misma.

MySpace

- Login: Nativo usando OAuth 1.0.
- Comentar: Simulado, usando la operación de cambio de estado provista por la red social para mostrar la opinión expresada y el link asociado.
- Compartir: Simulado, usando el cambio de estado para mostrar el link a compartir.
- Calificar (Gustar/Votar/Rankear): Simulado, usando nuevamente la operación de cambio de estado. En este caso simplemente se expresa el interés de gusto adjuntando el link del contenido relacionado.

Twitter

- Login: Nativo, usando OAuth 1.0.
- Comentar: Simulada usando twitts. Para poder modelar este aspecto, deberíamos buscar los twitts que coincidan con una determinada cadena, o en la terminología de esta red, un determinado hashtag (por ejemplo #1234). Entonces, utilizamos como identificador para recuperar los comentarios asociados a un contenido determinado, las mencionadas etiquetas.
- Compartir: Simulada, usando twitts. El link del contenido a compartir forma parte del texto asociado al twitt.
- Calificar (Gustar/Votar/Rankear): No disponible. El aspecto gustar se puede simular usando twitts, indicando como texto del mismo que al usuario le gusta un contenido determinado. Con el objetivo de recuperar aquellos usuarios a los cuales les gusta un contenido particular, es posible utilizar hashtags para identificar el contenido y realizar las búsquedas de twitts que contengan dicho link junto con el texto asociado al aspecto gustar (que será igual en todos los casos).

Enfoque general mediante separación de capas

Con la implementación propuesta, se prevé brindar un servicio que simplifique el enriquecimiento de contenido web mediante la especificación de aspectos de interacción social virtual, de modo de facilitar y unificar la forma en que el usuario administre dichos aspectos. Así, el usuario será capaz de especificar independientemente con que aspecto socializar cierto contenido, así como también el proveedor del cual será consumido, abstrayéndonos tanto de la especificación e implementación utilizados por dicho proveedor.

Se han considerado, en el marco de la generación de contenidos web enriquecidos con aspectos de redes sociales, la siguiente separación de capas, ilustradas en la figura 4.2:

- Generador de contenidos: permite la definición de contenidos, haciendo explícita la semántica de los mismos. Así mismo, soporta la definición de contenidos independientemente de la forma en la que serán visualizados y socializados.
- Socializador: permite la especificación de la socialización a aplicar. Dicha especificación se compone de dos partes: los aspectos de socialización y la implementación de los mecanismos de socialización.
- Visualizador: permite especificar cómo serán mostrados los contenidos socializados sin ensuciar los contenidos ni las implementaciones de aspectos sociales. Permite incluir los contenidos socializados en sitios web con estrategias de presentación diferentes.

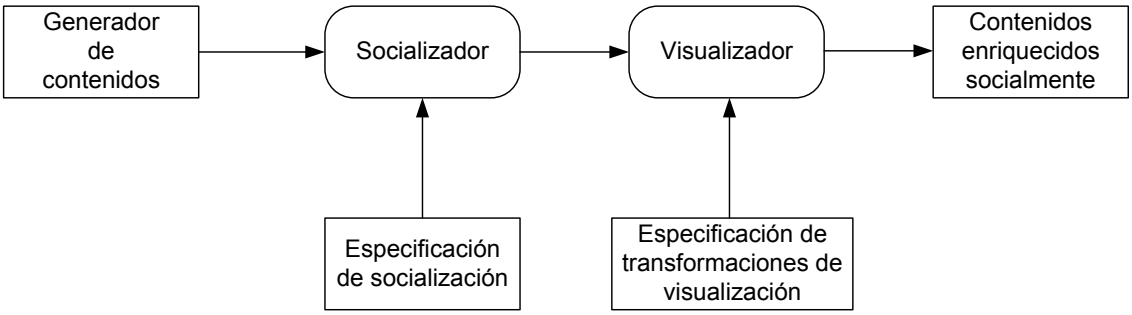


Figura 4.2 - Enfoque general de separación de capas.

La interrelación entra las distintas capas se ilustra en la figura 4.2. A partir de un origen que representa una fuente de contenidos, dicha separación se expresa por medio de una estructura de interconexiones o dependencias tipo tubería (pipeline). Cada capa consiste en un conjunto de especificaciones que serán aplicadas a la salida de la capa anterior, y servirán de entrada a la capa siguiente. Entonces, el funcionamiento en conjunto de cada capa permitirá enriquecer los datos provenientes del generador de contenidos con información generada por medio de la interacción social. Esta interacción social se pone de manifiesto a través de los aspectos identificados anteriormente en este capítulo.

Nuestro objetivo es enriquecer contenidos con una capa social. Esto es, enriquecer el contenido provisto por un servicio con funcionalidad para socialización (por ejemplo,

comentar, compartir, etc.), y con contenido generado por los usuarios (por ejemplo, comentarios agregados al contenido original). Además, nuestro plan es lograr este objetivo de forma transparente y separada, sin que el contenido original provisto por REST requiera ningún tipo de modificación o alteración.

Arquitectura general propuesta

La separación del contenido y su presentación, es una tendencia reconocible actualmente en el desarrollo web. Esto fue inicialmente motivado por la necesidad de lograr un bajo costo de mantenimiento y una evolución permanente en los portales web. Esto nos da la posibilidad de proveer servicios de publicación de información (por ejemplo, sin ninguna representación gráfica particular como ser HTML), que es la base para la creación de mashups web y la integración de servicios.

Teniendo como marco la figura 4.2, procederemos a describir la arquitectura general propuesta en etapas incrementales, enfocando el desarrollo en cómo se suman las componentes y cómo cada una agrega valor a la anterior.

Generador de contenidos

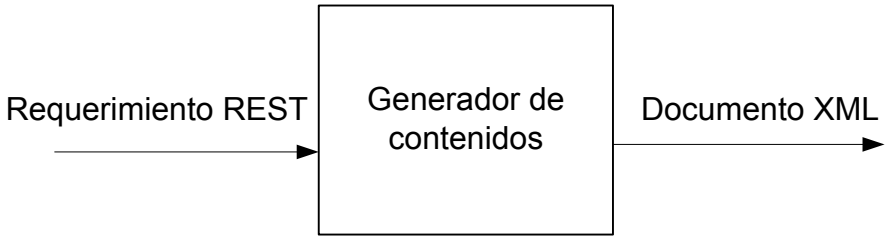


Figura 4.3 - Generador de contenidos.

Nuestro escenario de trabajo comienza con una fuente de datos, que es utilizada como el origen de información, tal como lo muestra la figura 4.3. Esta componente de generador de contenidos recibirá solicitudes REST, cuyas respuestas son documentos XML. Estos datos almacenados podrán ser consumidos tanto por un servicio de entrega de noticias o newsletter, un portal o distintos mashups web.

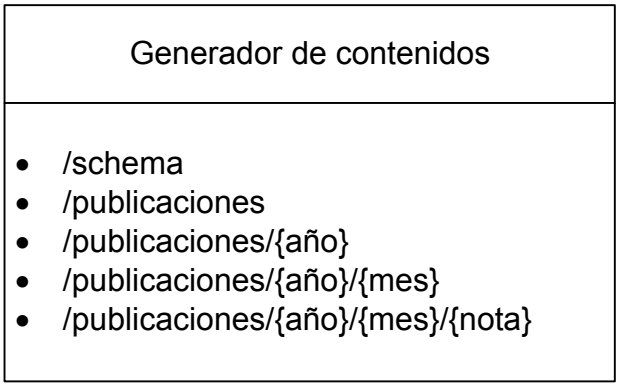


Figura 4.4 - Ejemplo de una posible API para el Generador de contenidos.

Entonces, el generador de contenidos es un servicio que provee una API REST para poder acceder a dicha información. La figura 4.4 ilustra un posible ejemplo de API REST para un generador de contenidos que representa un servicio de newsletter. En el ejemplo, la API está especificada genéricamente, indicando los parámetros encerrados entre llaves. A partir de esta API, el servicio aceptaría, por ejemplo, alguna de las siguientes solicitudes REST:

- /schema: retorna el esquema xml.
- /publicaciones: retorna todas las publicaciones.
- /publicaciones/2009: retorna todas las publicaciones correspondientes al año 2009.
- /publicaciones/2009/Julio: retorna la publicación correspondiente al mes de Julio de 2009.
- /publicaciones/2009/Julio/0: retorna la primera nota publicada el mes de Julio de 2009.

Además, y no menos importante, existe una restricción clave que es que la API exporte un método para obtener el esquema correspondiente a las respuestas XML (/schema en la API de la figura 4.4). Dicho esquema define la estructura de los documentos XML y será utilizado para validar los distintos requerimientos.

A partir de un requerimiento REST dado, por ejemplo "/publicaciones/2009/Marzo/1", el Generador de contenidos nos retornara el documento XML asociado, tal cual lo muestra la figura 4.5.

Figura 4.5 - Ejemplo de respuesta XML del Generador de contenidos.

Socializador

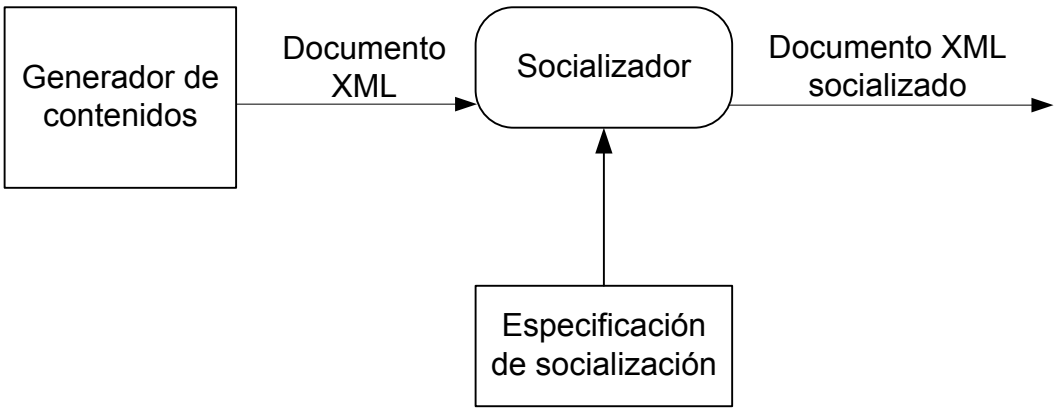


Figura 4.6 - Conectando el Generador de contenidos con el Socializador.

Un servicio adicional, diferente del que almacena nuestro contenido, actuará como decorador. Este servicio de socialización, soporta los mismos requerimientos REST que el Generador de contenidos. Entonces, la API REST del socializador es idéntica a la definida por el Generador de contenidos. Los requerimientos al socializador son reenviados al Generador de contenidos,

para que dicha respuesta sea extendida con el contenido social. La figura 4.6 ilustra esta situación.

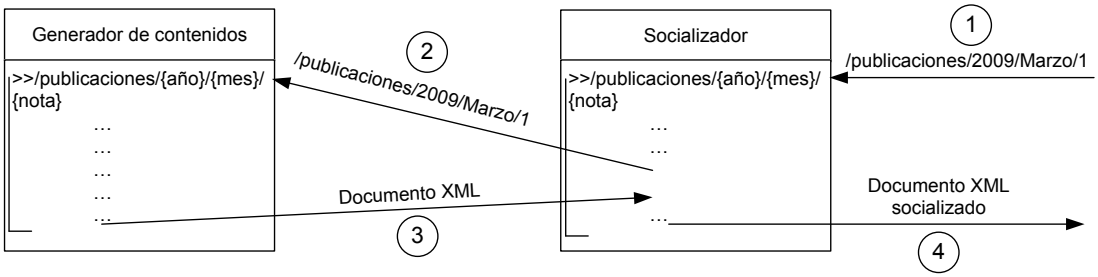


Figura 4.7 - API REST en el Socializador.

La API REST del Socializador es equivalente a la API REST descripta para el Generador de contenidos. Como vemos en la figura 4.7, supongamos que se realiza un pedido "/publicaciones/2009/Marzo/1" al servicio Socializador, este requerimiento será traspasado al Generador de contenidos, como hemos mencionado. Este retornará una representación XML estática del articulo solicitado y se lo pasará al servicio Socializador que recuperará, de ser posible, toda la información social asociada al contenido solicitado y la insertará en su representación, tal cual lo muestra la figura 4.8. Finalmente este documento XML socializado es retornado al usuario que realizó el requerimiento.

En consecuencia, si accedemos al Generador de contenidos obtendremos una respuesta XML que representa el contenido estático de un recurso, como vimos en la figura 4.5. En tanto que si realizamos la misma solicitud al servicio Socializador, obtendremos su representación enriquecida.

Figura 4.8 - Ejemplo de respuesta XML del Socializador.

Para completar la descripción del componente Socializador, lo que nos resta explicar es cómo este componente enriquece socialmente los contenidos. Como muestra la figura 4.6, este componente trabaja en función de una especificación de socialización, que básicamente es información expresada por medio de un lenguaje construido sobre XML. Como mencionamos cuando listamos las componentes de la arquitectura al principio de esta sección, esta especificación proporciona información sobre los aspectos sociales empleados y las alternativas de implementación utilizadas de los mismos. Por lo tanto, podemos explotar el componente Socializador en dos subcomponentes que operan sobre esos tipos de información, respectivamente:

- Tejedor de aspectos sociales: permite especificar los aspectos de socialización (comentar, compartir, votar, etc.) a ser integrados en los contenidos. Permite la evolución de un "lenguaje de especificación de aspectos sociales" con independencia de los contenidos a ser socializados y de la forma en que serán implementados.
- Tejedor de implementación: permite seleccionar entre distintas alternativas de implementación de los aspectos de socialización. Independiza las implementaciones de los contenidos y la forma en la que son socializados. Permite la publicación de los contenidos en distintas plataformas de socialización, así como la evolución independiente de las implementaciones de aspectos sociales.

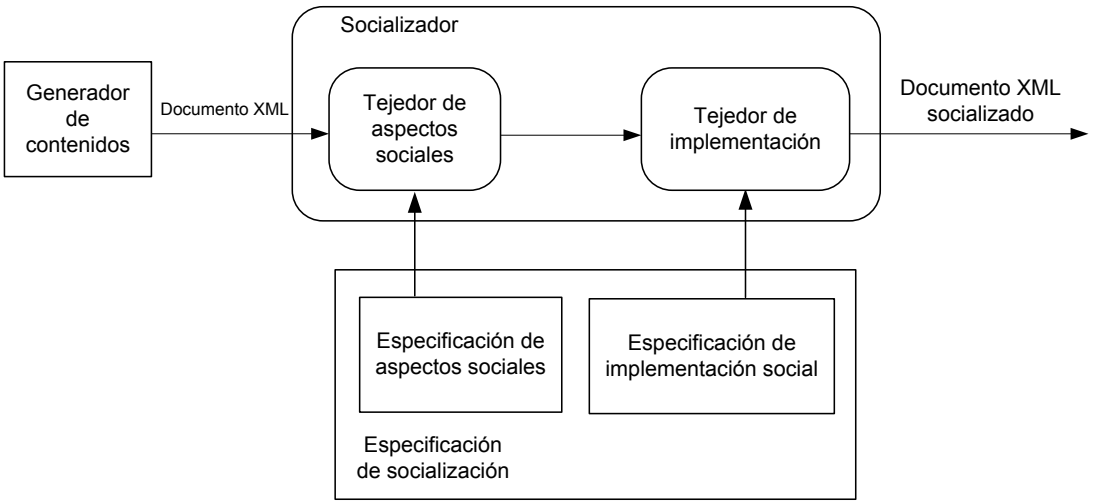


Figura 4.9 - Refinamiento del Socializador.

La figura 4.9 muestra como se desglosa el componente Socializador, de acuerdo a las nuevas subpartes introducidas anteriormente. Además, dicha figura ilustra cómo ahora se fracciona la información de especificación de acuerdo a las subcomponentes afectadas. Por un lado, la especificación de aspectos sociales determinará cuales son los aspectos que deseamos al momento de enriquecer el contenido con información social. Por otro lado, la especificación de implementaciones sociales consistirá en información correspondiente a proveedores sociales, en los cuales se delega la operatividad de los aspectos sociales indicados en la especificación de aspectos.

En resumen, el servicio de socialización es una puerta de enlace a la información e interacción social, referenciando el contenido desde el Generador de contenidos. Esto no significa que el servicio almacena información social, ni que implementa la funcionalidad de socialización. El servicio de Socialización delega esta responsabilidad en un servicio social externo, por ejemplo, Facebook. Esto significa que para enriquecer el contenido provisto por el Generador de contenidos, el servicio de Socialización interactúa con la plataforma de una red social externa para obtener, por ejemplo, los comentarios de un contenido dado.

Visualizador

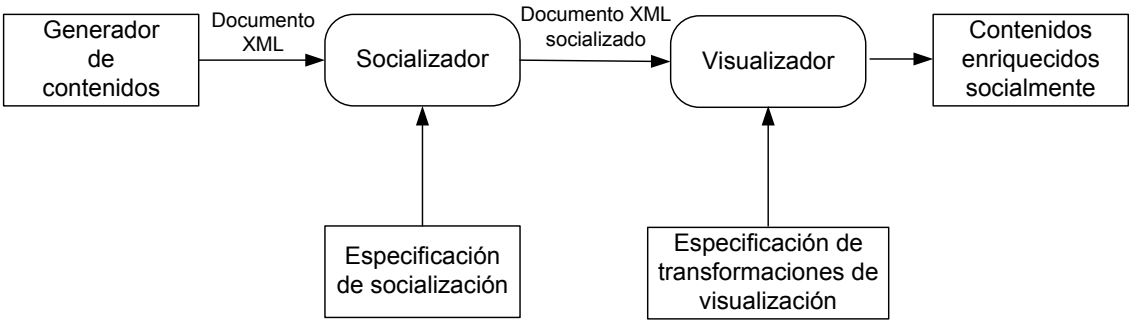


Figura 4.10 - Conectando el Visualizador al Socializador.

De modo de facilitar el procesamiento y presentación de la información socializada y simultáneamente mantener separada la información de los aspectos de la presentación, es que se provee un servicio de Visualización. La figura 4.10, muestra como se conecta la componente de Visualización al pipe (o tubería) de socialización por capas que se propone.

La API REST del Visualizador es idéntica tanto a la definida por el Generador de contenidos como a la del Socializador. Tomemos como ejemplo el requerimiento "/publicaciones/2009/Marzo/1", donde el Visualizador reenviará la solicitud al Socializador. Es este último, como se menciono anteriormente, quien realiza tanto la tarea de comunicación con el Generador de contenidos (de modo de obtener el recurso XML solicitado (ver figura 4.5), así como la transformación para retornar al Visualizador el documento XML socializado (ver figura 4.8). Ya con el recurso socializado, el Visualizador se encarga de transformarlo por medio de una plantilla XSLT y CSS para la organización y estilos de la presentación. La figura 4.10 muestra como quedaría el resultado de dicha transformación.

Figura 4.11 - Visualización de un recurso socializado.

Estrategia de Solución

Para conseguir los objetivos propuestos construiremos una herramienta donde se aplicarán estrategias de separación de concerns similares a las utilizadas en la construcción de mashups por medio de la composición de pipes que transforman/enriquecen servicios web. Cada pipe en el mashup consume un servicio, lo enriquece en base a una especificación, y lo vuelve a publicar como servicio.

Desglose de componentes

Generador de contenidos:

En nuestro caso el origen de datos será REST (Transferencia de Estado Representacional), utilizando XML como formato de intercambio de información y HTTP como protocolo de transferencia de estos archivos. Se optó por un origen de datos accesible vía REST, dado que se enmarca en el siguiente contexto:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente

ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis universal para identificar los recursos: En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Socializador:

Esta componente tiene como responsabilidad interactuar con los proveedores de redes sociales para difundir las acciones sociales en cada una de ellas, como así también gestionar el consumo y enriquecimiento de contenidos provenientes del origen de datos. El componente socializador será definido, diseñado y construido a partir de la especificación de API única elaborada anteriormente.

A partir de la configuración especificada, se encargará de tejer el contenido del que se dispone con los aspectos especificados en la configuración.

Este componente será diseñado e implementado para lograr:

- En primer lugar, independencia de la tecnología en la que están hechas las redes sociales con las que puede interactuar, reduciendo el esfuerzo y encapsulando lo específico de cada red.
- Segundo, extensibilidad, con el objetivo de poder incorporar nuevas redes. Esta componente se desarrolla en el marco de una arquitectura flexible y previendo situaciones en las que pueda ser necesario extenderla de modo de incorporar nuevos proveedores de aspectos de socialización.
- Tercero, punto central en la implementación de este componente, capacidad de introducir la capa de socialización sin la necesidad de modificar los contenidos originales. Esto, como previamente se menciono, se realiza gracias a una etapa de weaving en el que se tejen los aspectos y el contenido a mostrar.
- Cuarto, trabajar con una o muchas redes sociales al mismo tiempo, de manera transparente, y a través de la especificación de configuración.
- Quinto, reducción del esfuerzo del desarrollo del sitio. La herramienta deberá resultar no intrusiva, evitando así la alteración o modificación del contenido original.

- Sexto, transparencia y homogeneidad del código, siguiendo como premisa el empleo de estándares. Logrando esto, los aspectos relacionados con la interacción server-to-server con un sitio de red social en particular deberán quedar encapsulados, exponiendo solo el punto de conexión entre el aspecto social y la red elegida.

Visualizador:

Debido a que los sitios deben soportar tanto una visualización socializada como una transparente a esta, se hace necesario desarrollar una componente que se encargue de la vista del sitio.

En cualquier caso, cuando se acceda al sitio no se proveerá ningún tipo de socialización, pero si, se ofrecerá la posibilidad de iniciar sesión con alguna de las redes sociales soportadas, seleccionadas al momento de la configuración de la herramienta. De este modo, al iniciar sesión con las credenciales de alguna red social, el visualizador consumirá el contenido desde el origen de datos y mostrara un conjunto de widgets o componentes gráficos que permitirán la interacción del usuario, utilizando los aspectos detallados en la configuración especificada en su momento.

Configurador:

Ya sea por la gran variedad de SNS que actúan como proveedores de diversos aspectos sociales, como por la variedad de mecanismos que éstas utilizan para enriquecer sitios, resulta evidente la necesidad de proveer una herramienta de alto nivel, cuya tarea fundamental será facilitar el proceso de generación de la información de especificación de socialización a través de un mecanismo interactivo. En este contexto, el Configurador permitirá rellenar parámetros que describen la siguiente información:

- Generador de contenidos: Se deberá especificar la fuente de contenidos, es decir, desde donde se consumirá la información que luego se podrá enriquecer.
- Proveedores sociales: A partir de la lista de proveedores sociales soportados, se seleccionarán aquellos que estarán disponibles para iniciar sesión en el sitio.
- Aspectos sociales: A partir de la lista de aspectos sociales identificados, se seleccionarán aquellos con los que podremos enriquecer nuestro contenido.

Este componente no aparece especificado en el diagrama de la figura 4.2 como parte de la arquitectura, dado que no representa un nuevo elemento que resuelve una parte de la problemática. Se introduce en este capítulo como utilitario o aplicativo para simplificar la realización de una parte de la arquitectura, más precisamente la especificación de socialización.

Arquitectura de socialización por capas

A partir de la arquitectura general propuesta y los componentes introducidos en el capítulo anterior, nos enfocaremos en el presente a describir, detallar y especificar cado uno de ellos. La figura 5.1 muestra el enfoque general de separación de capas ya propuesto que iremos desglosando. El desarrollo estará organizado de la misma forma que el capitulo anterior, presentando los componentes de forma progresiva. En lugar de utilizar la notación genérica ya empleada, vamos a emplear notación UML para construir diagramas de clases, casos de uso, secuencia y componentes.

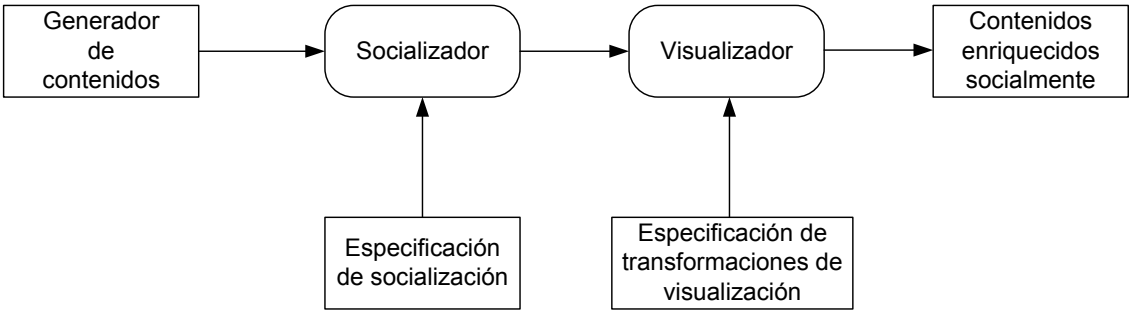


Figura 5.1 - Enfoque general de separación de capas.

Generador de contenidos

Propósito

Tal como se menciono en el capítulo 4, esta componente es introducida a la arquitectura con el objetivo principal de servir los contenidos a ser socializados. Es importante resaltar que su inclusión es necesaria para completar, ilustrar y entender el objetivo de nuestro diseño general, pero en realidad el diseño, especificación y construcción de la misma no es responsabilidad nuestra. Por esta razón, realizamos un tratamiento diferenciado para esta componente en relación con las restantes, enfoscándonos en los requerimientos que debe cumplir para ser capaz de interactuar con nuestro socializador.

Los dos requerimientos fundamentales que debe cumplir esta componente son:

- Proveer una API REST para acceder a los contenidos, utilizando XML como lenguaje base para el intercambio de la información.
- Contar con un punto de acceso al esquema XML que define la estructura o tipo de los objetos consumibles a través de la API anterior.

Diseño general y funcionamiento

Como mencionamos, esta componente es la que se encarga de servir los contenidos que serán consumidos. Este repositorio podrá ser accedido directamente por medio de solicitudes HTTP REST, o a través del socializador que se propone, proporcionando una vista enriquecida socialmente del contenido accedido. Básicamente, en este aspecto la componente socializadora soporta todas las solicitudes que el origen de datos es capaz de servir.

Tenemos un repositorio de contenidos del sitio que se querrá socializar. Como se menciono, al repositorio se podrá acceder sin intermediarios, por ejemplo

```
http://tesispapudany.dyndns.org/RestfulCorriere/publicaciones
```

Será un acceso sin mediadores a la fuente de contenidos. Este requerimiento retornara un recurso en formato XML.

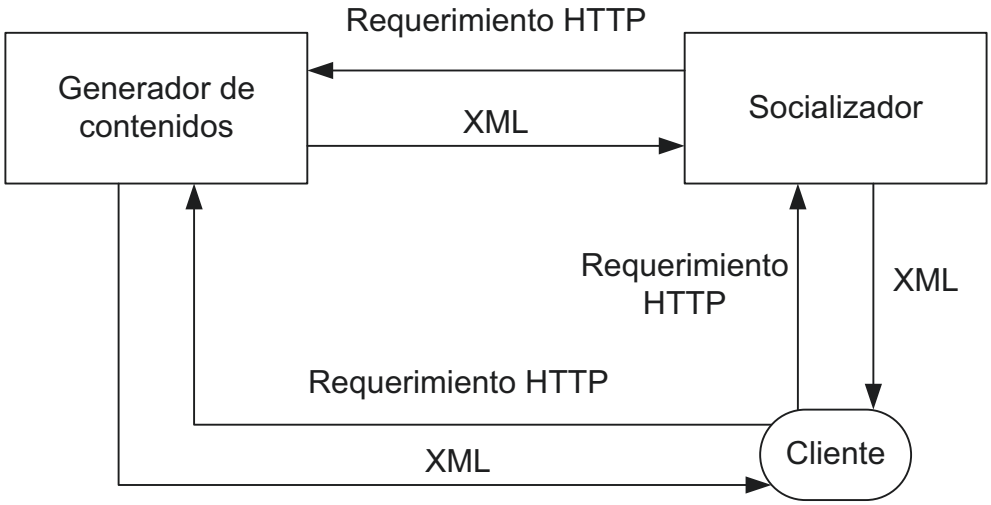


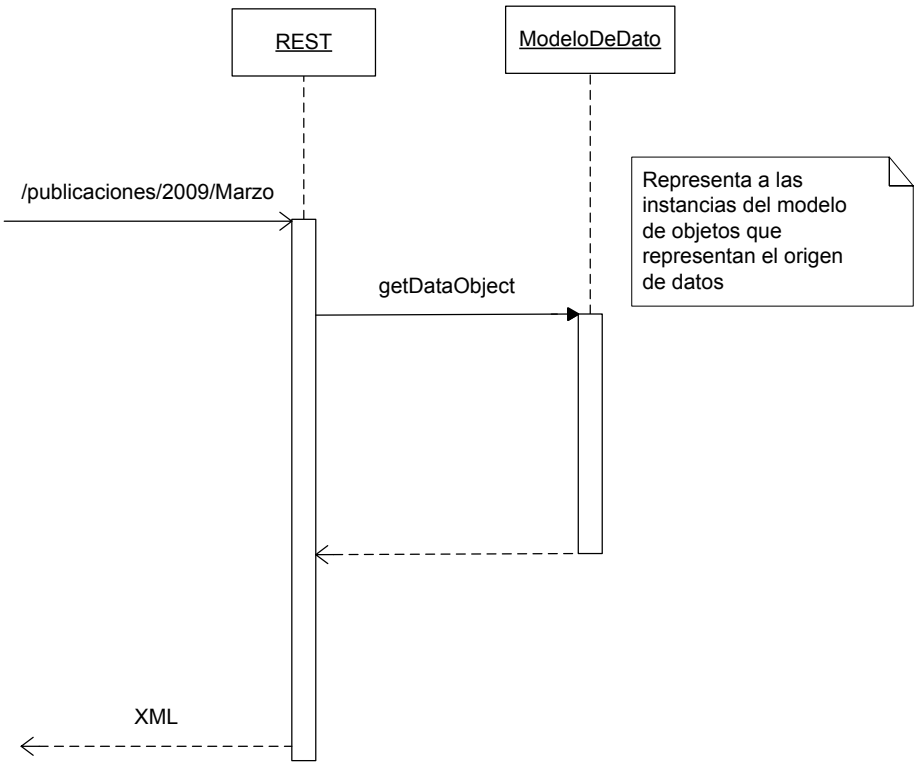
Figura 5.2 - Vista del funcionamiento del Generador de contenidos.

La figura 5.2 refleja esta situación, donde del mismo modo que se puede acceder al Generador de contenidos, se puede acceder a la información mediante requerimientos REST HTTP a través del intermediario o Socializador. De esta forma, el repositorio retornara los mismo datos, pero serán interceptados por esta ultimo componente.

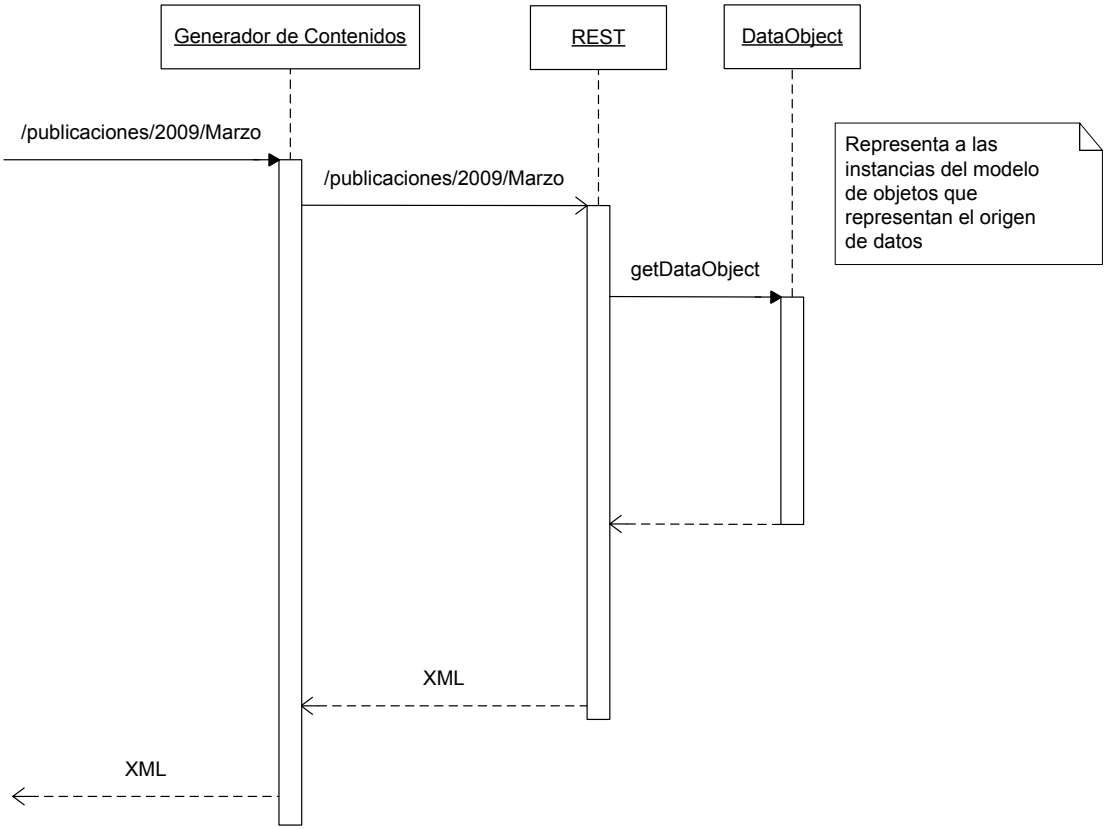
Diagramas

Diagramas de secuencia

El diagrama que se muestra a continuación, detalla cómo responde el generador de contenidos a una petición mediante su API REST.



A continuación, se muestra el diagrama que detalla la interacción REST a través del socializador. Además, se indican los diferentes componentes con los que este interactúa.



Aspectos de su implementación

La utilización de esta componente para el desarrollo de la herramienta es completamente reemplazable. Esto quiere decir, que la implementación aquí propuesta define los lineamientos a seguir para implementar un origen de datos con las capacidades para interactuar con nuestro socializador. De este modo, cualquier otro servicio web que cumpla con los requerimientos previamente expuestos podrá reemplazar al origen de datos que aquí proponemos.

Uno de los requerimientos fundamentales de esta componente es la de dar soporte a una API REST, se analizaron y estudiaron las distintas herramientas disponibles, optando por un framework web REST para Java, en este caso nos inclinamos por Restlet. Resulta de igual modo indispensable, poder contar con un punto de acceso al esquema XML que define la estructura y tipo de los objetos consumibles a través de la API anterior, la figura 5.3 muestra el esquema XML que se utilizo para realizar el presente trabajo.

Por otro lado, para el objetivo que nos planteamos con el desarrollo de esta herramienta, no solo es necesario contar con un origen de datos con soporte a un API REST, sino que se debe inicializar al mismo con cierto contenido, de modo de probar nuestra herramienta con un

conjunto valido de datos. Para tal fin, se pensó la API Rest en función a un modelo de objetos de un newsletter.

```
− <xs:schema targetNamespace="http://localhost:8080/RestfulCorriere/schema">
  − <xs:element name="sitio">
    − <xs:complexType>
      − <xs:sequence>
          <xs:element name="historia" type="xs:string"/>
        + <xs:element name="staff-estable"></xs:element>
        + <xs:element name="ficha"></xs:element>
        − <xs:element name="publicaciones">
          − <xs:complexType>
            − <xs:element name="publicacion">
              − <xs:complexType>
                − <xs:element name="ejemplar">
                  − <xs:complexType>
                    − <xs:sequence>
                        <xs:element name="anioPublicacion" type="xs:string"/>
                        <xs:element name="numero" type="xs:string"/>
                        <xs:element name="mes" type="xs:string"/>
                        <xs:element name="año" type="xs:string"/>
                        <xs:element name="mailto" type="xs:string"/>
                      − <xs:element name="notas">
                        − <xs:complexType>
                          − <xs:element name="nota">
                            − <xs:complexType>
                              − <xs:sequence>
                                  <xs:element name="id" type="xs:string"/>
                                  <xs:element name="titulo" type="xs:string"/>
                                  <xs:element name="resumen" type="xs:string"/>
                                  <xs:element name="contenido" type="xs:string"/>
                                + <xs:element name="autores"></xs:element>
                                </xs:sequence>
                              </xs:complexType>
                            </xs:element>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:complexType>
              </xs:element>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Figura 5.3 - Esquema XML utilizado para la implementación propuesta.

Restlet

Básicamente es una técnica utilizada en el desarrollo de arquitecturas del software. Restlet [Restlet] es el framework que decidimos usar, construido en el lenguaje Java. A continuación se listan algunas de las características de este framework, que motivaron su elección:

- Los conceptos centrales de REST tienen artefactos Java equivalentes (Recurso, Representación, Conector o Componente por ejemplo)
- Es apropiado para aplicaciones web tanto clientes como del lado del servidor. La innovación es que utiliza la misma API, reduciendo la curva de aprendizaje y uso del framework.
- Soporta el concepto de "URIs como UI", con base en los estándares de plantillas URI. Esto resulta en un ruteo muy flexible y a la vez simple con la extracción automática de las variables de la URI en atributos de la solicitud.
- Soporte incorporado para representaciones XML (JAX, JibX, DOM o SAX) con una API XPath simple basada en el motor XPath que incluye la JDK.
- Adaptadores de servlets provistos que permiten desplegar cualquier aplicación Restlet en contenedores compatibles como Tomcat.
- Servicio de túnel que permite a los navegadores ejecutar cualquier método HTTP (PUT, DELETE, MOVE, etc.) a través de solicitudes HTTP POST. Este servicio es transparente para las aplicaciones Restlet.
- Preparado para la Web Semántica, con soporte completo para RDF.

XML Schema

El propósito del estándar XML Schema es definir la estructura de los documentos XML que estén asignados a tal esquema y los tipos de datos válidos para cada elemento y atributo. En este sentido las posibilidades de control sobre la estructura y los tipos de datos son muy amplias.

Al restringir el contenido de los ficheros XML es posible intercambiar información entre aplicaciones con gran seguridad. Disminuye el trabajo de comprobar la estructura de los ficheros y el tipo de los datos.

XML Schema tiene un enfoque modular que recuerda a la programación orientada a objetos y que facilita la reutilización de código.

Los tipos de datos tienen en XML Schema la función de las clases en la POO. El usuario puede construir tipos de datos a partir de tipos predefinidos, agrupando elementos y atributos de una determinada forma y con mecanismos de extensión parecidos a la herencia. Los tipos de datos se clasifican en función de los elementos y atributos que contienen.

Los tipos de datos en XML Schema pueden ser simples o complejos.

XML Schema incluye el uso de espacio de nombres. Los "espacios de nombres" permiten definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se

anteponga un prefijo al nombre del elemento. El uso del espacio de nombres también evita confusiones en la reutilización de código.

Es posible agrupar atributos, haciendo más comprensible el uso de un grupo de aspectos de varios elementos distintos, pero con denominador común, que deben ir juntos en cada uno de estos elementos. Los ficheros XML Schema se escriben en el propio lenguaje XML [XML Schema].

Socializador

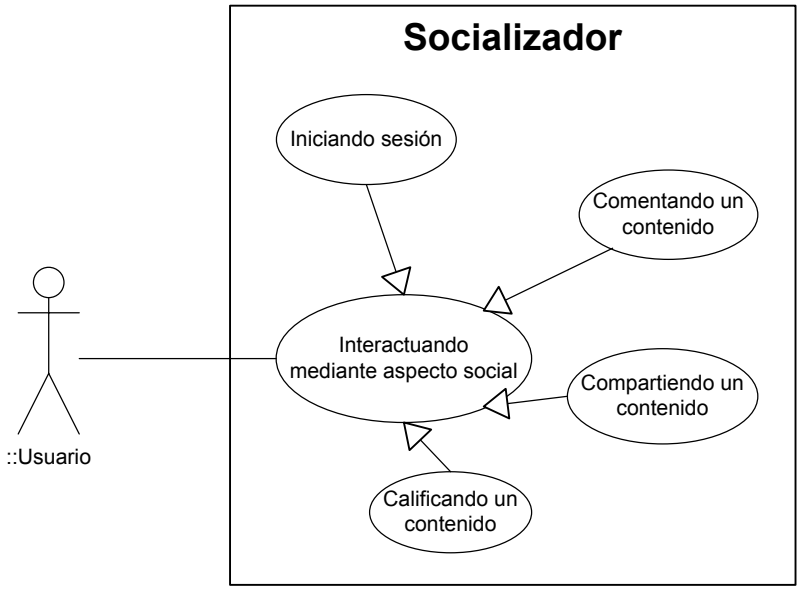
Propósito

Es el componente núcleo o central de la arquitectura. Básicamente, es el responsable de delegar las solicitudes REST al origen de datos configurado, recibir sus respectivas respuestas XML, y enriquecer este contenido con aspectos de redes sociales. Para llevar a cabo esto último, es capaz de interactuar transparentemente con diversos proveedores de redes sociales. Instanciado mediante la herramienta de configuración, su estructura de diseño tipo framework permite que el mismo sea extendido para agregar nuevos proveedores de redes sociales con facilidad.

Diagramas

Diagrama de caso de uso

A continuación se presenta un diagrama mostrando los distintos casos de uso involucrados en el socializador



Para simplificar el detalle de cada caso de uso, procedemos a documentar la descripción del caso de uso genérico 'Interactuando mediante aspecto social'

Nombre: Interactuando mediante aspecto social
Descripción: Un usuario ejecuta un aspecto social relacionado a un contenido determinado.
Actores: Usuario (interacción con el sitio enriquecido)
Precondiciones: El socializador está configurado con: • Un conjunto de proveedores sociales seleccionados durante la configuración con los cuales interactuará. • URL del origen de datos. • Aspectos seleccionados para tipos de contenido determinados.
Flujo normal: 1. El usuario accede al contenido enriquecido socialmente con distintos aspectos. 2. El usuario lleva a cabo una acción social (comentar, compartir, etc.) a través de los widgets clientes. 3. El sistema recibe dicha petición, la procesa y la promueve en el proveedor social configurado para ese aspecto. 4. El sistema responde indicando el resultado de la acción.
Flujo anormal: 2. La sesión del usuario expiro: a. El sistema presenta un mensaje indicando tal condición. b. Se renderiza el widget de Inicio de Sesión.
Postcondiciones: La acción social es promovida en el proveedor social destino configurado.

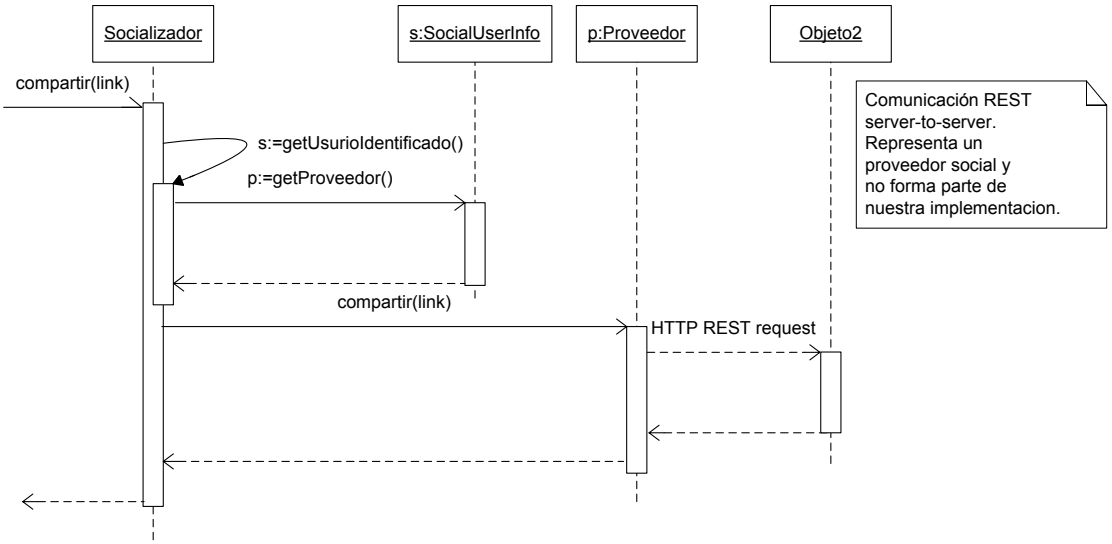
A continuación, desarrollaremos la descripción del caso de uso correspondiente al 'inicio de sesión', ya que su especificación involucra un tratamiento diferenciado a los aspectos sociales restantes.

Nombre: Iniciando sesión
Descripción: Un usuario se identifica en alguno de los proveedores sociales configurados.

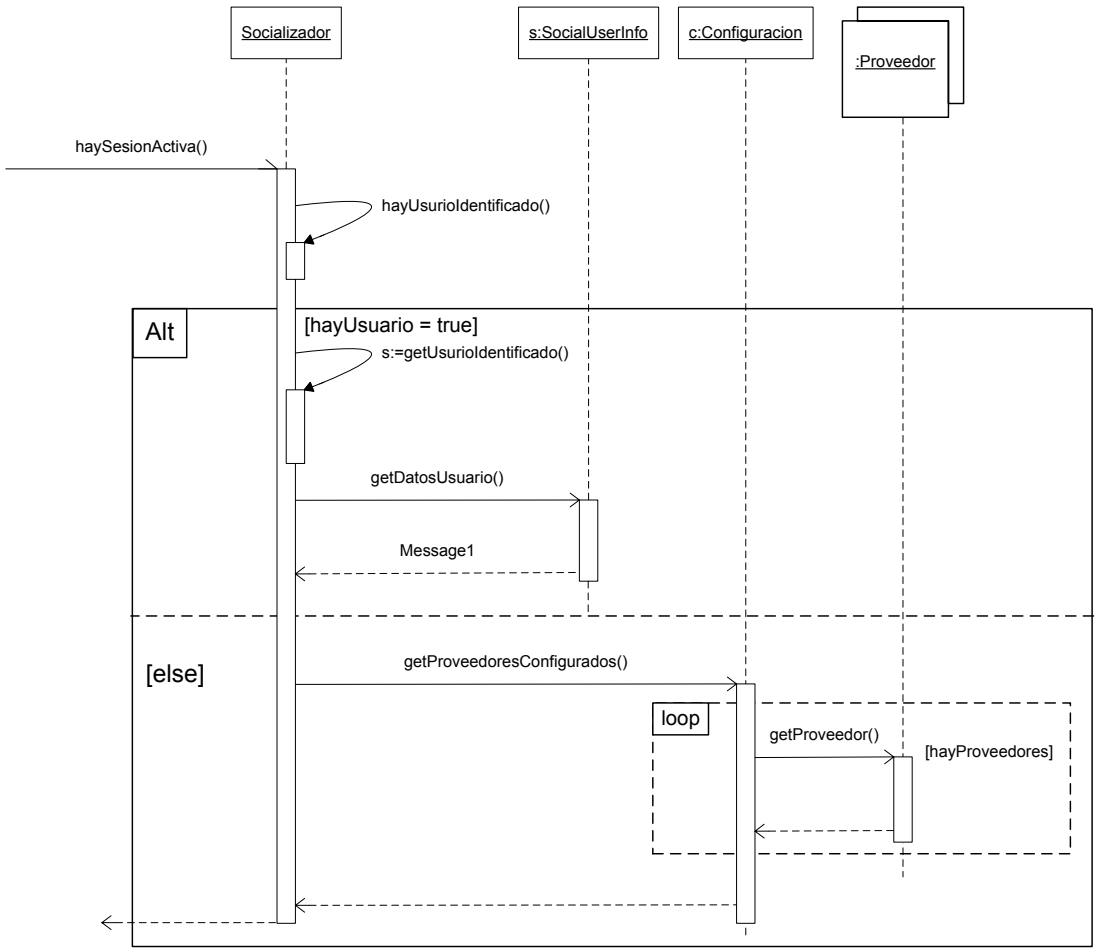
Actores: Usuario (interacción con el sitio enriquecido)
Precondiciones: • No existe sesión activa para el usuario. El socializador está configurado con: • Un conjunto de proveedores sociales seleccionados durante la configuración con los cuales interactuará. • URL del origen de datos. • Aspectos seleccionados para tipos de contenido determinados.
Flujo normal: 1. El usuario selecciona el proveedor social en el cual se desea identificar. 2. El sistema redirige al usuario para que continúe el proceso de identificación en el proveedor. 3. El usuario completa con sus credenciales el formulario de inicio de sesión y lo envía. 4. El proveedor redirige nuevamente el control al sistema. 5. El sistema guarda los datos de la nueva sesión.
Flujo anormal: 1. Existe una sesión activa para el usuario: • El sistema recupera los datos de la sesión • Se muestra el botón de fin de sesión para el proveedor 3. Se muestran los aspectos configurados para ese proveedor. Las credenciales ingresadas no son correctas: • El proveedor notifica dicha condición • Solicita el reingreso de los datos
Postcondiciones: El usuario se identifica en el proveedor social seleccionado

Diagramas de secuencia

A continuación se muestra el diagrama de interacción que ilustra la secuencia de enriquecimiento social, así como la interacción con un proveedor dado.

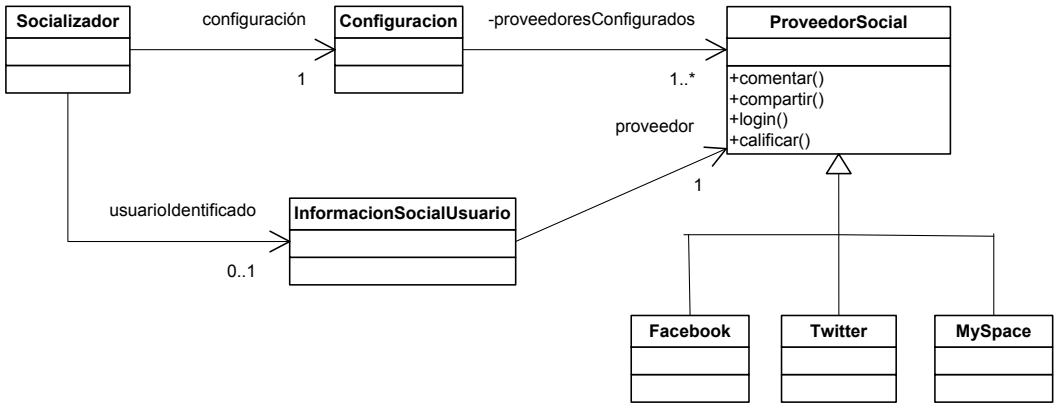


En el siguiente diagrama se muestra la secuencia de identificación de usuario y registro de sesión en el socializador



Diagramas de clase

El diagrama detallado a continuación describe las clases involucradas en la componente del socializador.



Aspectos de su implementación

Patrones considerados en el diseño y construcción de la componente

Procederemos a documentar los patrones de diseño orientado a objetos que hemos tenido en cuenta a la hora de diseñar y construir esta componente, buscando cumplir los objetivos que hemos explicitado en capítulos anteriores en el documento.

El diseño general expresado mediante el diagrama de clases anterior contempla la utilización de los conceptos del patrón *Puente* (o Bridge). El mismo tiene como propósito desacoplar una abstracción de su implementación, de modo que ambas puedan variar de forma independiente. En nuestro caso particular, lo que hemos buscado desacoplar es nuestra especificación de API Social de los distintos proveedores sociales donde se pueden implementar. De esta forma, evitamos un enlace permanente entre una abstracción y su implementación, por ejemplo cuando debe seleccionarse o cambiarse la implementación en tiempo de ejecución (al permitir interactuar con diferentes proveedores sociales al mismo tiempo). Además, posibilita que los cambios en la implementación de una abstracción no tengan impacto en los clientes.

Entonces, aplicar los conceptos de diseño asociados al patrón "Bridge" tiene las siguientes consecuencias:

1. Desacopla la interfaz y la implementación: No une permanentemente una implementación a una interfaz, sino que la implementación puede configurarse en tiempo de ejecución. Incluso es posible que un objeto cambie de implementación en tiempo de ejecución.
2. Mejora la extensibilidad: podemos extender las jerarquías correspondientes a la abstracción e implementación de forma independiente.
3. Oculta detalles de implementación a los clientes: podemos aislar a los clientes de los detalles de implementación.

Es importante mencionar que el conjunto de métodos que implementan el mecanismo de inicio de sesión siguen la estructuración indicada por el patrón *Método Plantilla* (o Template Method). Este patrón tiene como propósito definir en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, permitiendo así que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura. Entonces, para el caso del inicio de sesión, el esqueleto del algoritmo es especificado a partir de las operaciones definidas por el estándar OAuth (en sus versiones 1.0 y 2.0). Por lo tanto, a través de este patrón capturamos las partes de un algoritmo que no cambian y dejamos que las subclases implementen el comportamiento que puede variar, evitando además el código duplicado.

Manipulación y tratamiento de la configuración

Como hemos mencionado durante el desarrollo de este capítulo y el anterior, nuestra herramienta cuenta con un componente de alto nivel que permite la instanciación del

socializador. Mediante esta herramienta es posible especificar aspectos sociales a utilizar y proveedores con los cuales interactuar enfocados en contenidos particulares.

Toda la información generada por dicha componente de configuración será almacenada en el componente socializador en lo que se denomina "alcance aplicación" en la terminología web Java EE. Dichas acciones son llevadas a cabo utilizando la interface ServletContext, la cual define una vista para los servlets de nuestro socializador y permite, entre otras cosas, ligar objetos a dicho alcance.

La información almacenada involucra:

- Las instancias de los proveedores sociales seleccionados por el usuario configurador por cada aspecto social.
- Los aspectos sociales seleccionados por el usuario configurador.
- Los diferentes contenidos a los cuales se asocian o enfocan los aspectos sociales elegidos.

El mecanismo común elegido para representar esta información sigue la estructura de tablas de hash, es decir, colecciones del tipo clave-valor.

Entonces, los proveedores seleccionados durante la configuración serán consultados en el momento de desplegar el widget de inicio de sesión. Durante el proceso de inicio de sesión, se recuperará la instancia particular correspondiente al proveedor con el cual se intenta acceder.

Por otro lado, los aspectos sociales enfocados a diferentes tipos de contenidos serán consultados al momento de desplegar los widgets que permitirán interactuar socialmente. Cuando dicha interacción sucede, el socializador recupera la información que establece que proveedor social utilizará para ejecutar y promover la acción social requerida.

Identificación de usuario y mantenimiento de sesiones

Es nuestra intención destinar los siguientes párrafos a describir como hemos implementado el mecanismo de identificación de usuarios y mantenimiento de sesiones. En primer lugar, es importante explicar que por razones de seguridad y privacidad no es posible que nuestra componente socializadora sea un servicio intermediario que redirecciones las credenciales de identificación entre el usuario y el proveedor social. Por estas razones, los proveedores sociales populares proveen variados mecanismos para lograr la identificación, con el objetivo común de delegar dicha tarea al proveedor particular (ver Capitulo 3, Estado del arte).

Con el objetivo de construir un mecanismo de identificación en nuestra componente socializadora que sea genérico, y permita conseguir extensibilidad, adoptamos la utilización del estándar OAuth (ver Capitulo 3, Estado del arte). Repasando brevemente, este mecanismo permite llevar a cabo la identificación de un usuario, delegando el ingreso y la validación de los datos del mismo al proveedor social específico. Entonces, básicamente lo que permite esto es realizar una transferencia de los datos de sesión entre el proveedor social y nuestra

componente socializadora. Como hemos mencionado al tratar el estándar, se busca un token de acceso autorizado el cual formará parte de los requerimientos REST realizados sobre el proveedor.

Para soportar esta interacción, nuestra componente socializadora cuenta con un conjunto de servlets destinados a realizar el login, consultar si hay una sesión activa, y en caso afirmativo, poder invalidarla. Cada una de estas actividades es llevada a cabo por un servlet diferente, descriptos a continuación:

- El servlet de login está construido y estructurado como un método plantilla que sigue los lineamientos y pasos planteados por el estándar Oauth, e interactúa con la instancia específica del proveedor social con el cual se está intentando la identificación (cada instancia implementará los respectivos métodos que forman parte de la estructura de la plantilla). Como resultado de su ejecución, el usuario será redirigido al proveedor para que ingrese sus credenciales (usuario y contraseña), y si ambas son correctas, el control vuelve a nuestra componente, la cual registrará y almacenará los datos de dicha sesión. Estos datos involucran información social del usuario (nombre, apellido, id de usuario en el proveedor, etc.) y la instancia del proveedor con el cual se identificó. Dichos datos están representados por instancias de la clase SocialUserInfo, las cuales serán ligadas al alcance sesión de la arquitectura web JEE.
- El servlet de verificación de sesiones activas tiene el doble propósito de posibilitar la invalidación de una sesión activa si es que existe, o recuperar los proveedores sociales con los cuales se puede acceder en caso de que no exista una sesión activa. En el primer caso, se recuperará del alcance sesión los datos del usuario y del proveedor en el cual está identificado, para poder desplegar el correspondiente mecanismo de finalización de sesión. En el segundo caso, se recuperará y consultará la configuración actual del socializador (desde el alcance aplicación) para establecer los posibles proveedores a los cuales podemos intentar acceder.
- El servlet de invalidación de sesión permitirá eliminar del alcance sesión la instancia actual de SocialUserInfo e invalidar la sesión también en el proveedor social (mediante una solicitud REST). Esta última acción es delegada en la instancia correspondiente al proveedor social.

Es importante resaltar que estos servlets interactuarán solamente con el cliente JavaScript que hemos desarrollado (tratado más adelante). Además, otra cuestión no menos importante es que nuestro mecanismo de identificación de usuarios y mantenimiento de sesiones no lleva a cabo la tarea de determinar si el usuario ya se encuentra identificado en algún proveedor social al momento de interactuar con el socializador cuando no hay una sesión activa en la componente. En este caso, el mismo flujo del estándar presentará al usuario un diálogo en el cual confirmará que desea acceder desde nuestra componente, y no necesitará completar un

formulario de login con los datos de su cuenta, dado que ya había iniciado sesión previamente. Por esta razón, es que podemos pensar en nuestro mecanismo de autenticación como en un proceso de transferencia de datos de sesión desde el proveedor como mencionamos anteriormente.

Implementación especifica del socializador y del mecanismo de delegación en los proveedores sociales

En este apartado procederemos a explicar el funcionamiento e implementación de la clase Socializador. Básicamente, se trata de un servlet Java que atiende todas las solicitudes asociadas a llevar a cabo acciones sociales. Por esta razón, es que el mismo interactuará con el cliente Javascript que implementa nuestros widgets sociales. Dicho servlet recibirá y procesará solicitudes HTTP, determinando la acción social requerida y <extrayendo los parámetros específicos de la misma. Es importante tener en cuenta que este servlet podrá actuar solo cuando exista una sesión activa, dado que deberá ser capaz de recuperar la instancia del proveedor social y delegar en él la operación. Una vez que la acción social es llevada a cabo por la instancia específica del proveedor, el Socializador generará la respuesta correspondiente a la acción solicitada, indicando el estado de la misma, junto con datos adicionales específicos al tipo de operación que se llevó a cabo. Este proceso se ilustró anteriormente con el diagrama de secuencia correspondiente.

Si pensamos en los aspectos específicos de su implementación, este servlet está diseñado para interactuar con cualquier instancia de Proveedor, dado que espera solicitudes para las acciones sociales definidas en nuestra API, la cual está expresada a través de dicha jerarquía. Entonces, al recuperar los datos de la sesión de usuario, conocemos que instancia específica de Proveedor debemos usar. Luego, utilizando el mecanismo de "Reflection"[1] de Java, delegamos la operación solicitada en dicha instancia, junto con los parámetros específicos asociados a la misma. Utilizando este mecanismo de Reflection logramos que si la API de aspectos sociales sufre algún cambio, se extiende o reduce, tales cambios solo afecten la jerarquía Proveedor que representa nuestra API.

La instancia específica del proveedor será responsable de promover la acción social requerida en la red social destino. Recordemos que la interacción (server-to-server) con los proveedores sociales se lleva a cabo por medio de solicitudes REST. El lenguaje Java provee un framework completo para interactuar por medio de solicitudes HTTP de alto nivel, mediante las cuales construimos y ejecutamos tales requerimientos REST. Por lo tanto, cada operación implementada en la jerarquía Proveedor consistirá básicamente en construir la solicitud HTTP REST con los parámetros esperados, y dirigida a la URL definida y provista por el proveedor de red social para efectivizar dicha operación.

- Reflection: La API Java de Reflection permite examinar las clases, interfaces, atributos y métodos en tiempo de ejecución, sin conocer el nombre de las clases, métodos, etc. en tiempo de compilación. También posibilita la instanciación de nuevos objetos y la invocación de métodos por medio de este mecanismo de Reflection.

Descripción de la API en el diseño

En el capítulo 4 hemos descripto la abstracción de aspectos sociales que hemos identificado al analizar y estudiar los proveedores de redes sociales más populares, consiguiendo construir de esa forma una API genérica estándar para nuestro socializador. Es nuestra intención explicar cómo dicha API se pone de manifiesto en el diseño propuesto. Como lo muestra el diagrama de clase del socializador, existe una clase abstracta denominada "ProveedorSocial", que representa la definición de dicha API en la arquitectura. Cada uno de esos métodos está construido de acuerdo a los parámetros de diseño que dicta el patrón *Método Plantilla* (Template Method, mencionado anteriormente en este capítulo). Entonces, cada subclase será responsable de algunos de los métodos que forman dicha plantilla.

Sin embargo, la clase "ProveedorSocial" no es el único lugar donde se expresa la API genérica en la arquitectura propuesta. Como explicaremos en el componente Visualizador, existe un conjunto de widgets cliente que permiten interactuar con los proveedores sociales. Es posible interpretar a dichos widgets como una API cliente que interactúa con el Socializador. En otras palabras, cada uno de esos widgets representa los aspectos abstractos identificados, y su funcionalidad describe la acción correspondiente al aspecto que representan. Entonces, la abstracción de aspectos realizada estará presente también del lado cliente, en los scripts respectivos que gobiernan la funcionalidad de tales widgets. Dichos scripts gestionan, entre otras cosas, la comunicación asíncrona con el Socializador.

Volviendo al diseño del lado servidor, en el componente Socializador, y considerando los aspectos específicos del lenguaje seleccionado, claramente la API se implementó por medio de una jerarquía de clases. Teniendo en cuenta la implementación lograda y el mecanismo de interfaces que provee el lenguaje Java, existe la posibilidad de mejorar el diseño especificado en el diagrama de clase del socializador evolucionando dicha jerarquía a la definición de una interface, la cual declara la signatura de los métodos correspondientes a los aspectos genéricos. El objetivo que persigue esta evolución es lograr que la arquitectura resultante más genérica, y que la adición de un nuevo proveedor sea aún más transparente, evitando en lo posible programar. Esto es algo interesante que trataremos en detalle más adelante en el apartado de trabajos futuros.

Visualizador

Propósito

El visualizador es el responsable de dos tareas centrales en la visualización del sitio. Por un lado, es el encargado de disponer el contenido en la vista final, transformando el XML del contenido a socializar en un documento HTML. En segundo término, es el responsable de insertar los widgets de los aspectos sociales, seleccionados al momento de configurar la aplicación.

Diseño general y funcionamiento

Las acciones que lleva a cabo esta componente, pueden resumirse en los siguientes puntos:

- Recepción de requerimiento: Recibir requerimientos REST para mostrar una vista HTML (socializada o no) de los contenidos XML almacenados.
- Forward de requerimiento: Forward del requerimiento REST a la componente de Origen de Datos.
- Consultar la configuración: De modo de determinar la configuración seleccionada por el usuario administrador del sitio.
- Aplicar la transformación: A partir de la configuración leída y el recurso a mostrar, se aplicara la transformación correspondiente.

Del mismo modo, se pueden agrupar dichas acciones por funcionalidad provista, pudiendo ser estas:

- Presentación de la vista.
- Inserción de los widgets sociales.

Presentación de la vista

Cuando llega un requerimiento REST a esta componente, la misma lo reenvía al origen de datos. Por ejemplo, si se quiere visualizar alguna nota en particular de una publicación dada, podríamos acceder a la misma del siguiente modo

```
http://tesispapudani.dyndns.org/Visualizador/publicaciones/2009/Julio/1
```

Para comenzar el procesamiento, se toma la parte REST del requerimiento, en este caso '/publicaciones/2009/Julio/1', se arma el requerimiento y se lo reenvía al origen de datos. Quedando como

```
http://tesispapudani.dyndns.org/RestfulCorriere/publicaciones/2009/Julio/1
```

El origen de datos responderá con el recurso solicitado, tal cual lo muestra el fragmento de respuesta XML de la figura 5.4.

```
− <nota>
    <id>2009Marzo1</id>
    <titulo>Cooperación con Portugal en Orientación a Aspectos</titulo>
  + <resumen></resumen>
  + <contenido></contenido>
  − <miembros>
    + <miembro></miembro>
    + <miembro></miembro>
    </miembros>
  </nota>
```

Figura 5.4. Fragmento de respuesta XML.

En este punto, el visualizador ya puede transformarlo utilizando el archivo XSLT que se ubica en la URL que fue ingresada al momento de la configuración. Esto incluye:

- Aplicar las reglas de transformación a las distintas etiquetas del archivo XML.
- Consultar al configurador para determinar cuáles fueron los proveedores de aspectos sociales seleccionados al momento de la configuración.

Inserción de widgets sociales

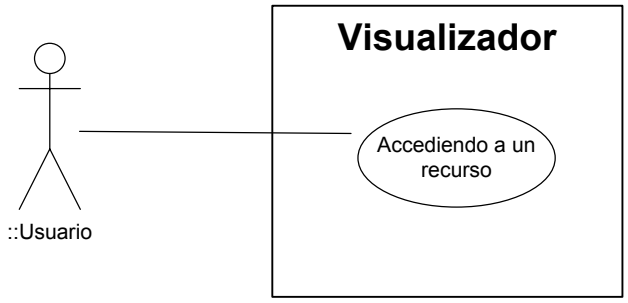
Para la inserción de los widgets sociales, el visualizador debe realiza dos tareas, a saber:

- Consultar al configurador: De este modo el visualizador pueda determinar cuáles fueron los aspectos seleccionados al momento de la configuración.
- Aplicar la transformación: En este punto el visualizador podrá, a partir de la configuración leída, insertar los widgets sociales.

Diagramas

Diagrama de caso de uso

El siguiente caso de uso detalla las acciones involucradas al acceder a un recurso.

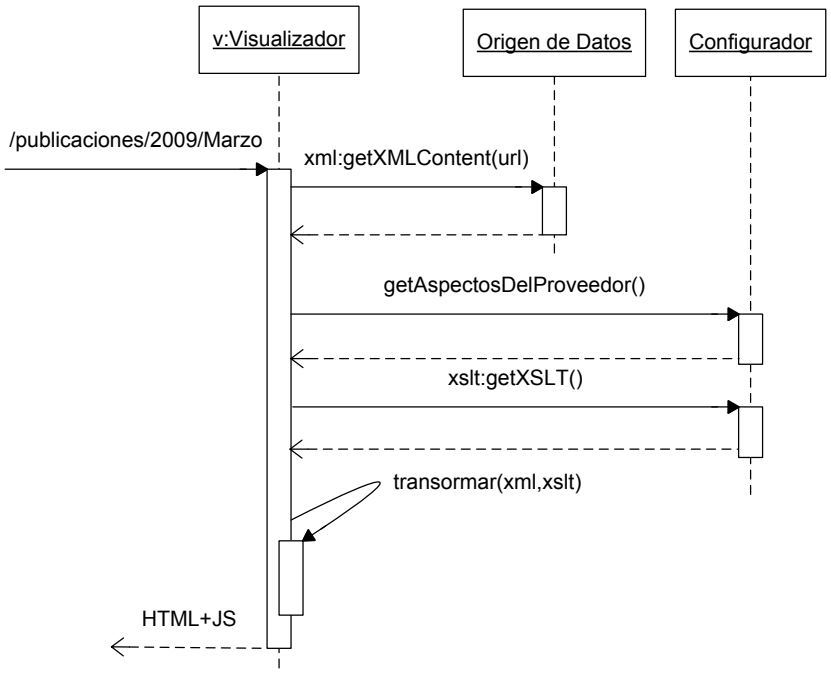


Nombre: Accediendo a un recurso

Descripción: Un usuario accede a un recurso del sitio
Actores: Usuario.
Precondiciones: • El sitio ya cuenta con una configuración, provista por el usuario administrador. • El usuario cuenta con un perfil en al menos uno de los proveedores cargados.
Flujo normal: 1. El usuario selecciona la opción de login a uno de los proveedores listados. 2. Se le solicita su nombre de usuario y contraseña. 3. El Usuario completa los datos solicitados y confirma. 4. Se verifica los datos del usuario. 5. Se carga la vista socializada del sitio
Flujo anormal: • La contraseña no es correcta: 1. Informa el error 2. Solicita que reingrese la contraseña
Postcondiciones: Se muestra una vista socializada del sitio.

Diagramas de secuencia

A continuación se muestra el diagrama de interacción que ilustra la secuencia completa de visualización de un recurso dado.



Aspectos de su implementación

La implementación del Visualizador se desarrollo utilizando un Serlvet Java. Tal como se menciono, este recibe y procesa requerimientos. Para finalizar su tarea, esta componente debe llevar a cabo una transformación del contenido XML requerido, recuperado del origen de datos, de modo que pueda presentarse una vista HTML al usuario final. Para llevar a cabo dicha transformación, se utilizaron las siguientes APIs.

Dada la lógica de esta componente, no fue necesaria la utilización de ningún patrón de diseño particular.

Dada la naturaleza de esta herramienta como aplicación web, y en particular para la ejecución tanto de esta componente como la de los Servlets en general, fue necesaria la utilización de un contenedor web [Contenedor web]. Dada esta situación, se analizaron las herramientas existentes, optando por la utilización del contenedor web Tomcat de Apache.

Contendor Web

En la plataforma Java 2 Enterprise Edition, un contenedor web es la implementación que hace cumplimiento del contrato de componentes web de la arquitectura J2EE. Este contrato especifica un entorno de ejecución para componentes web que incluye:

- Conectividad con la red.
- Capturar los requerimientos HTTP: Traduce a objetos que el servlet entiende, entrega al servlet dichos objetos quien los usa para producir la respuesta HTTP.

- Manejar el ciclo de vida del servlet.
- Dar soporte al manejo de errores.
- Dar soporte de seguridad.

Servlet Java

Los Servlets son programas que se ejecutan en el contexto de un servidor o contenedor JEE, especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML [Servlet Java]. La especificación original de servlets fue creada por Sun Microsystems, terminando la versión 1.0 en junio de 1997.

Un servlet implementa la interfaz `javax.servlet.Servlet` o hereda alguna de las clases más convenientes para un protocolo específico, por ejemplo `javax.servlet.HttpServlet`. Al implementar esta interfaz el servlet es capaz de interpretar los objetos de tipo `HttpServletRequest` y `HttpServletResponse` quienes contienen la información de la página que invocó al servlet.

Entre el servidor de aplicaciones o contenedor web y el servlet existe un contrato que determina cómo han de interactuar.

El ciclo de vida de un Servlet se divide en los siguientes puntos:

1. El cliente solicita una petición a un servidor vía URL.
2. El servidor recibe la petición.
    a. Si es la primera, se utiliza el motor de Servlets para cargarlo y se llama al método `init()`.
    b. Si ya está iniciado, cualquier petición se convierte en un nuevo hilo. Un Servlet puede manejar múltiples peticiones de clientes.
3. Se llama al método `service()` para procesar la petición devolviendo el resultado al cliente.
4. Cuando se apaga el motor de un Servlet se llama al método `destroy()`, que lo destruye y libera los recursos abiertos.

Diseño y estructura de los widgets cliente

La idea de utilizar un mecanismo no intrusivo para llevar a cabo la socialización de un determinado sitio derivó en la utilización de tecnologías que sean estándar para la construcción de los widgets sociales. De este modo, cuando nos referimos a widgets sociales estamos hablando de módulos o fragmentos de código que modelan la vista de un aspecto social determinado. En nuestra implementación hemos utilizado diversos mecanismos dinámicos que se emplean en la diseño de sitios web. Entonces, construimos una librería cliente en Javascript que incluye:

- Un conjunto de funciones responsables de la instanciación y manipulación de los componentes visuales.

- Un conjunto de funciones responsables de interactuar con el componente Socializador, utilizando AJAX como mecanismo de comunicación.

Para construir la interfaz gráfica correspondiente a los widgets que implementan los distintos aspectos, utilizamos ExtJs, que es una librería de código abierto multinavegador en Javascript para construir aplicaciones de internet ricas. Básicamente, dicha librería proporciona un conjunto de widgets gráficos configurables, que facilitan el desarrollo y personalización de cada componente visual que permitirá llevar a cabo cada aspecto. Luego, cada widget hará uso del conjunto de funciones para interactuar con el Socializador, y así, a través de él, delegar la manifestación del aspecto en la red social correspondiente. Como mencionamos, esta interacción está gobernada por el mecanismo AJAX.

En conjunto, todas esas funciones representan lo que denominamos anteriormente en este capítulo al describir el componente Socializador como API cliente. En particular, en esas funciones estará expresada la API abstracta que definimos en el capítulo 4 (ver figura 4.1), y que se ilustra en el diagrama de clases del socializador.

Por último, esta API cliente se encuentra en un archivo de extensión ".js" que se incluye al realizar la transformación visual mediante las hojas de estilo XSLT.

Configurador

Propósito

El configurador define el modo en el que se aplicará la socialización al sitio. Mediante esta componente, se configuran las redes sociales o proveedores con los que se podrá enriquecer el contenido, así como los aspectos y a que elementos del contenido se le aplicaran.

Diseño general y funcionamiento

Al acceder al configurador, se mostraran tres solapas que presentan los distintos aspectos que se deberán configurar. A continuación se detallan cada una de estas.

Origen de datos

La figura 5.5, presenta la sección que tiene como finalidad proveer la información relacionada al origen de datos. Para esto, se deben proveer los siguientes datos:

- Ubicación del esquema XML: Esta URL especifica la ubicación del XMLS que se usara para describir la estructura del contenido del sitio.
- Ubicación del Origen de Datos: Esta URL especifica el origen de datos, desde la cual se consumirá el contenido a visualizar y enriquecer.

Figura 5.5 - Sección de Origen de datos del Configurador.

Socializador

En una segunda solapa, vemos la información relacionada a la socialización del sitio. La figura 5.6 muestra los distintos aspectos a configurar entre los que se encuentran:

- Proveedores sociales: Son leídos desde un directorio dentro de la aplicación, como archivos de propiedades. En estos archivos, se guarda la información asociada tanto al proveedor social, la 'secret key' y 'consumer key' de la aplicación creada en dicha red social.
- Aspectos: Son aquellos aspectos sociales que se identificaron como parte de nuestra API.

A partir de la información obtenida del esquema XML, se podrá proveer una configuración. Esta tarea consiste en seleccionar:

1. Los proveedores sociales que estarán disponibles en el sitio.
2. Los aspectos sociales que se le aplicaran a cada elemento socializable del contenido.

Figura 5.6 - Sección de socialización del Configurador.

Vista

Por último, la figura 5.7 nos presenta la solapa relacionada a la configuración de la vista del sitio. Para esto deberemos proveer la siguiente información:

- Ubicación del XSLT: Esta URL especifica la ubicación del archivo para la transformación del XML (conteniendo este, referencias a otros archivos, por ejemplo, JavaScript).
- Estilo: Hoja de estilo que se utilizara para la presentación de la vista final del sitio.

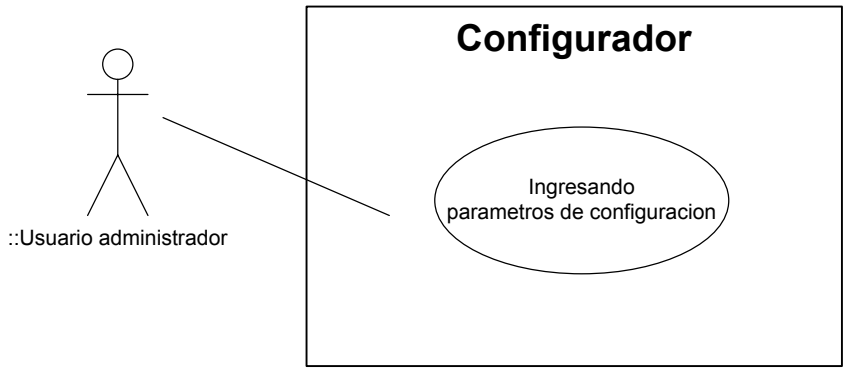
Figura 5.7 - Sección de la Vista del Configurador.

Para finalizar, en el presente trabajo y a partir del esquema XML propuesto, el elemento socializable para la presente herramienta es el elemento 'nota'. Por ende, será a este elemento al cual se le aplicaran los aspectos seleccionados.

Diagramas

Diagrama de caso de uso.

El siguiente diagrama muestra las acciones que intervienen al ingresar los parámetros de configuración al configurador.

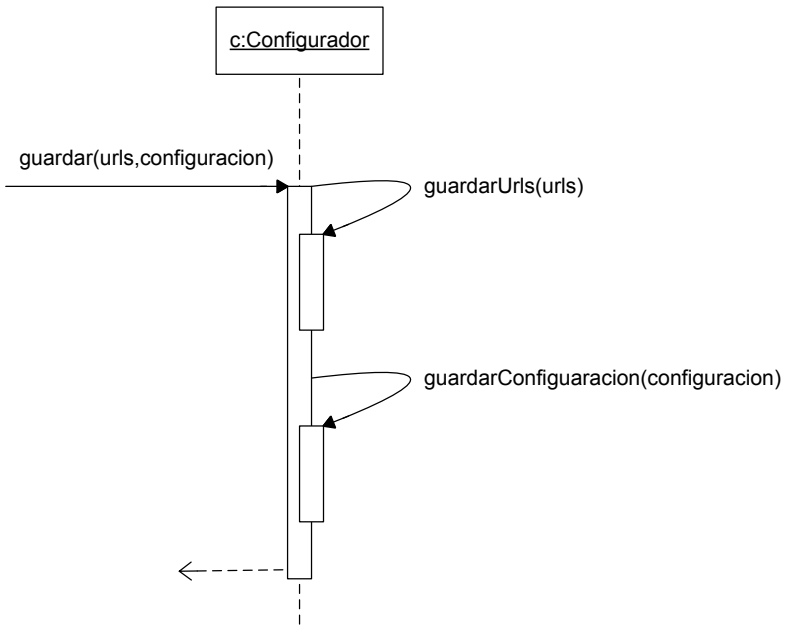


Nombre: Ingresando parámetros de configuración
Descripción: Un usuario administrador ingresa la información al configurador
Actores: Usuario administrador
Precondiciones: Existe un conjunto de proveedores y aspectos precargados
Flujo normal: 1. El usuario administrador accede al configurador. 2. Completa los campos requeridos y define que aspectos se aplicaran a los elementos socializables. 3. El usuario administrador confirma. 4. Se verifica los datos y presenta al usuario las opciones de usuario registrado
Flujo anormal: 2 Se ingreso algún parámetro incorrecto.

a. Se informa del error. b. Solicita que reingrese el parámetro incorrecto.
Postcondiciones: El configurador quedo inicializado correctamente.

Diagrama de Secuencia.

A continuación se presentan las acciones involucradas al guardar la información de la componente de configuración.



Aspectos de su implementación

Por motivos de accesibilidad y disponibilidad, el configurador se desarrollo utilizando una arquitectura cliente-servidor accesible vía web, de modo que el administrador pueda actualizar la configuración sin necesidad de instalar ninguna aplicación localmente. Al mismo tiempo, el configurador debe poder ser accedido desde cualquier localización.

Dadas estas características, el configurador fue desarrollado utilizando un Applet Java para el desarrollo de la vista y un Servlet Java que atienda los requerimientos en el servidor.

Applets

Un applet es un programa que puede incrustarse en un documento HTML, es decir en una página web. Cuando un navegador carga una página web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse en el contexto de dicho navegador, en el caso de los applets Java, esto sucede gracias a la Java Virtual Machine (JVM), o el Applet Viewer de Sun [Applet].

Ventaja:

- Son multiplataforma (funcionan en cualquier sistema operativo para el cual exista una JVM).
- El mismo applet puede trabajar en todas las versiones de Java. Sin embargo, si un applet requiere una versión posterior de la JRE, el cliente se verá obligado a esperar durante la descarga de la nueva JRE
- Puede ser almacenado en la memoria cache de la mayoría de los navegadores web, de modo que se cargará rápidamente cuando se vuelva a cargar la página web.
- Puede trasladar el trabajo del servidor al cliente.

Desventaja:

- Requiere el plugin de Java, que no está disponible por defecto en todos los navegadores web.
- No puede iniciar la ejecución hasta que la JVM esté en funcionamiento, y esto puede tomar tiempo la primera vez que se ejecuta un applet.
- Si no está firmado como confiable, tiene un acceso limitado al sistema del usuario.

Vista general de la solución propuesta

En el presente capitulo se detallo cada una de las componentes involucradas en la solución propuesta. La figura 5.8 que se muestra a continuación, nos presenta un diagrama de componentes que ilustra la interacción entre estos.

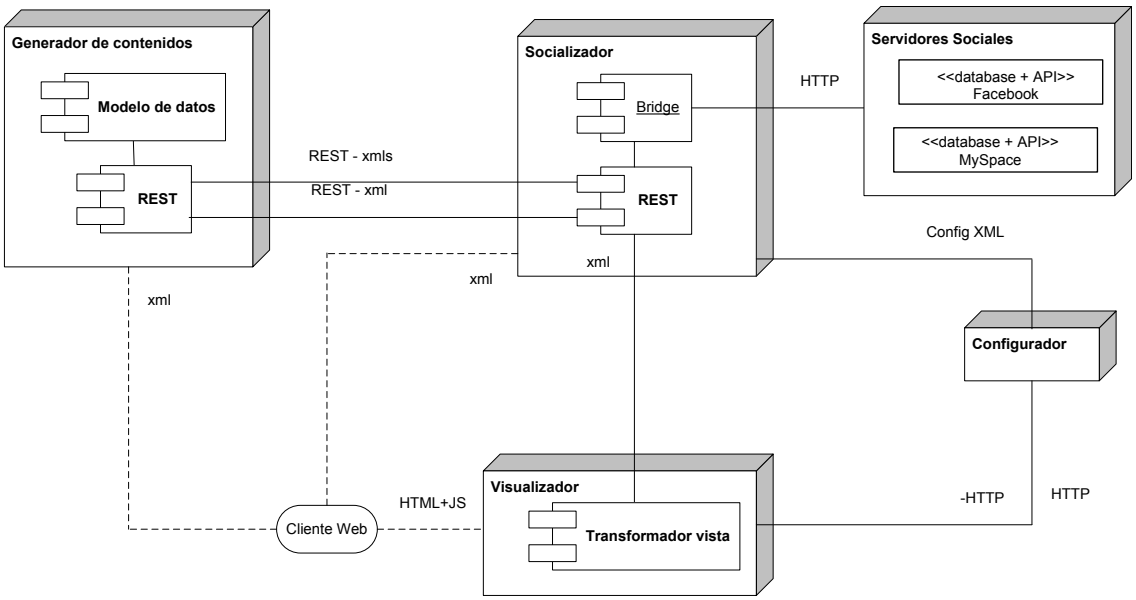


Diagrama 5.8 - Diagrama de componentes de la arquitectura.

Los componentes se encuentran interconectados, especificando el tipo de datos que intercambian. El Socializador consumirá por medio de solicitudes REST, documentos XML que definen los contenidos provenientes del Generador de contenidos, como así también el XML esquema de los mismos. De la misma forma se comunican los componentes Visualizador y Socializador (este último, agrupando al 'Tejedor de aspectos sociales' y al 'Tejedor de implementación').

Por otro lado, el nodo de Configuración permite inicializar los componentes Socializador y Visualizador, indicando aquellos parámetros necesarios para el funcionamiento de cada uno. Dicha configuración quedará almacenada y se intercambiara por medio de archivos XML.

Para finalizar, vemos como el cliente web puede acceder vía requerimientos REST, a los mismos recursos XML (en las distintas etapas que se detallaron), como la vista HTML de los mismos.

Enriqueciendo un sitio mediante la socialización por capas

Luego de presentar, analizar y describir en detalle la herramienta que proponemos en este trabajo, es nuestro objetivo destinar el presente capítulo a reseñar el conjunto de factores y elementos relativos a la interacción del usuario con la misma. En el desarrollo siguiente queremos resaltar los factores relativos al diseño, tales como interacción, usabilidad, utilidad, accesibilidad, entre otros.

Se abarca el proceso de enriquecimiento de un sitio mediante la socialización por capas desde dos visiones posibles, a saber:

- usuario administrador
- usuario final

Dado este enfoque, nos centramos en detallar cada una de las vistas propuestas. Avanzaremos en el desarrollo a partir de una instanciación de la herramienta utilizando los aspectos de "comentar" y "compartir", enfocada en los contenidos provistos por un origen de datos REST que representa un newsletter. Además, utilizaremos los proveedores sociales precargados Facebook y Twitter.

De aquí en más, se utilizara el siguiente vínculo como sitio socializado de ejemplo

```
http://tesispapudani.dyndns.org/Socializador
```

Usuario administrador

Para poder socializar un determinado sitio, es necesario acceder primeramente al configurador. El usuario encargado de esta tarea es el administrador de sitio, quien configura la aplicación, estableciendo los parámetros de socialización que se usaran y a partir de los cuales se mostrara la vista que verán los usuarios finales.

Para acceder al configurador del sitio debemos acceder a

```
http://tesispapudani.dyndns.org/Socializador/configurador
```

Luego de ingresar a la URL dada, se cargara la componente de configuración del sitio, tal cual lo muestra la figura 6.1.

Figura 6.1 - Componente configurador.

Tal como se ha mencionado en capítulos anteriores, la componente de configuración presenta distintas solapas de acuerdo a los distintos aspectos a configurar. Los parámetros que deberemos especificar consisten en:

- URL del esquema del origen de datos: que presenta el esquema de que deberá respetar nuestro contenido XML.
- URL del origen de datos: desde el cual se consumirá el contenido a presentar y socializar.
- URL del XSLT: que presenta la plantilla con la que se realizara la transformación del contenido en la vista.

Luego de proveer las ubicaciones de los archivos requeridos, será necesario proveer una configuración que incluye:

- Seleccionar los proveedores: a partir de la lista de proveedores precargados se seleccionaran aquellos con los que se podrán iniciar sesión.
- Seleccionar una configuración: de modo de especificar qué aspectos se le aplicaran a que elementos socializables.
- Seleccionar una vista: será la hoja de estilo que utilizara el sitio para presentar la vista.

Para finalizar, se deberá guardar esta configuración. En este punto, ya tendremos nuestro sitio preparado para que sea accesible con la vista socializada, por ejemplo, al realizar un requerimiento al recurso:

```
http://tesispapudani.dyndns.org/Socializador/Visualizador/2009/Marzo
```

se muestra la vista tal cual lo presenta la figura 6.2.

Figura 6.2 - Vista que presenta el sitio al acceder a un recurso dado.

Habiendo realizado esta tarea, el usuario administrador completo todo lo que se requiere para enriquecer el sitio mediante la socialización por capas.

Usuario final

Entendemos por usuario final a aquel que consume los contenidos desde una vista enriquecida socialmente, resultado de combinar los aspectos sociales con el proceso de formateo visual de los datos. Básicamente, dicho usuario deberá identificarse en alguna de las redes sociales seleccionadas en el proceso de configuración (Facebook o Twitter). Luego, será capaz de interactuar a través de los aspectos sociales enfocados a los contenidos de acuerdo también a la configuración seleccionada.

La vista enriquecida socialmente se accede utilizando URLs como la que se muestra a continuación

```
http://tesispapudani.dyndns.org/visualizador/2009/Marzo/1
```

donde el prefijo de la URL corresponde al componente encargado de tejer la vista consumible por un navegador web. El postfijo de la URL del ejemplo ("/2009/Marzo/1") corresponde a la ruta REST del origen de datos, y representa el recurso que disponemos a visualizar. Dicho postfijo es equivalente al que se usara para acceder al contenido original en formato XML, cuando el prefijo de la URL corresponde al componente de Origen de Datos

```
http://tesispapudani.dyndns.org/restfulcorriere/2009/Marzo/1
```

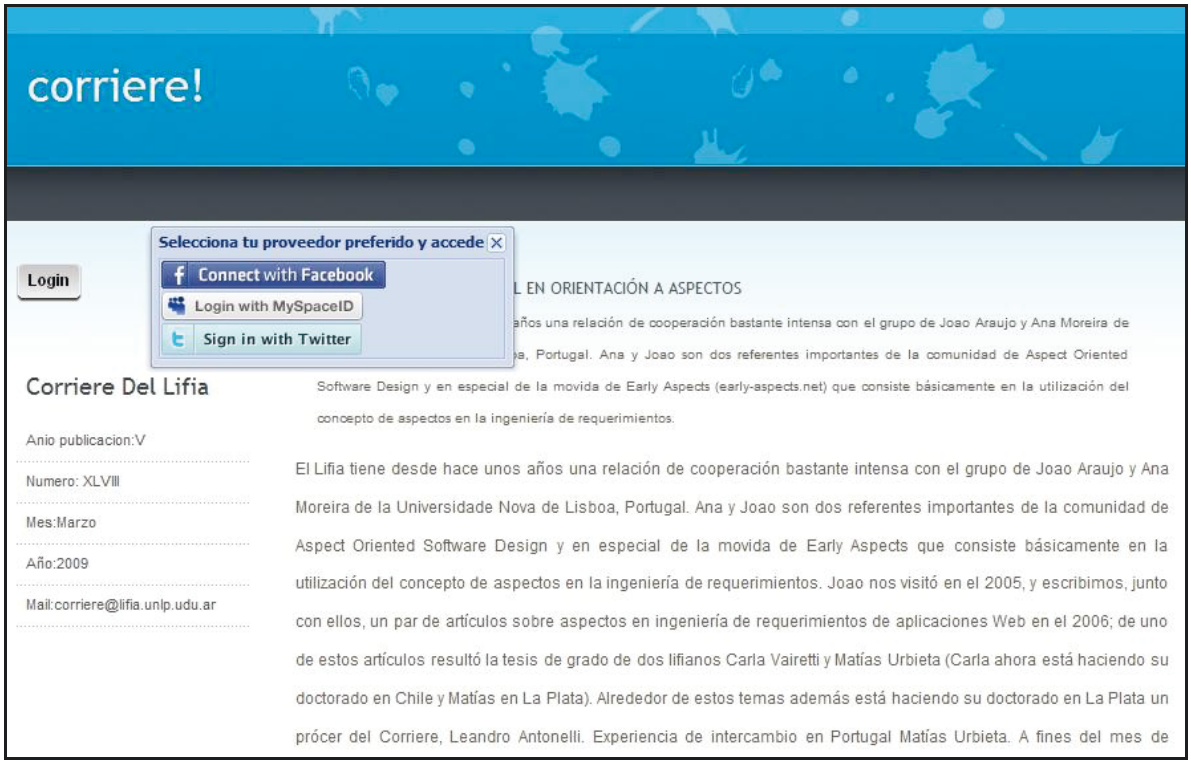


Figura 6.3 - Vista del widget de inicio de sesión del sitio socializado.

En el primer acceso, se desplegará el widget de inicio de sesión, tal cual lo muestra la figura 6.3. El mismo listará los botones de acceso a los proveedores sociales seleccionados durante la configuración para interactuar. Al pulsar en alguno de ellos, se desplegará el mecanismo de identificación delegado en el proveedor seleccionado. Una vez logrado el acceso, la vista mostrará los widgets de los aspectos que habíamos seleccionado durante la configuración, enfocados en los contenidos que definimos también durante la configuración. En este punto podemos argumentar que la vista enriquecida tendrá dos disposiciones, aquella que se muestra cuando no hay sesión activa para el usuario que accede, y aquella en la que sí. Esto

permitirá que el usuario pueda consumir los contenidos de su interés, y tenga la posibilidad de expresar opiniones, sensaciones, e interactuar socialmente al identificarse socialmente.

Figura 6.4 - Vista del sitio socializado con los widgets configurados

Cada widget provee el mecanismo para exponer la acción social en el proveedor con el cual nos autenticamos. A continuación se mostrara tanto la vista que tendrán los widgets, en este caso los de los aspectos sociales comentar y compartir, así como de la forma que se publica dicho aspectos en la red social correspondiente.

Comentar

La figura 6.5 muestra la apariencia del widget de comentar, que proveerá el mecanismo para completar con el texto de nuestro comentario.

Figura 6.5 - Widget de comentar

A su vez, dicho comentario podrá ser visualizado en la red social en la cual estamos identificados. Así por ejemplo, al identificarnos con Facebook, dicha acción social se mostrará en la red social como nos presenta la figura 6.6.

Figura 6.6 - Vista de un comentario en Facebook

Como notamos, aparece el comentario del usuario en su muro, junto con un enlace al contenido comentado. Al ser un aspecto social que este proveedor implementa de forma nativa, su promoción en el mismo corresponde a los conceptos de interacción definidos en esa red. Es decir, el comentario en el sitio enriquecido se manifiesta de la misma forma que en el proveedor social.

Además, cuando procedemos a publicar un comentario, para el caso particular de Facebook, veremos que en el sitio enriquecido se listaran las entradas correspondientes a los comentarios más recientes para el mismo contenido, junto con el usuario que los realizó.

Ahora bien, si nos identificamos en Twitter, dicho comentario se mostrará de acuerdo al mecanismo provisto por dicha red para exponer los twitts de cada usuario. La figura 6.7 refleja esta situación.

Figura 6.7 - Vista de un comentario en Twitter.

En este caso, además de que el aspecto no tiene soporte nativo en Twitter, su promoción en la red social está acotada a los aspectos específicos de su experiencia de uso.

Compartir

La figura 6.8 nos muestra la apariencia que tiene el widget de compartir. Permitirá exponer el enlace del contenido asociado en el proveedor social en el cual estemos identificados, junto con un comentario breve.

Figura 6.8 - Widget de compartir

Si tomamos como ejemplo Facebook, dicha acción social se manifestará en este proveedor, como nos muestra la figura 6.9. Tal como sucede con el aspecto de comentar, compartir un enlace en Facebook tiene soporte nativo, por eso su aspecto en la red social sigue los preceptos de interacción definidos en ese proveedor. Además, el enlace compartido se publicará en el muro del usuario identificado, con una imagen en miniatura opcional.

Figura 6.9 - Vista del aspecto compartir en Facebook.

Si nos identificamos en Twitter, la promoción del enlace se ajusta al formato del twitt de dicha red. No hay soporte nativo para este aspecto. El enlace aparecerá en formato de URL corto. La figura 6.10 nos muestra una vista de la red social, luego de haber compartido un enlace.

Figura 6.10 - Vista del aspecto compartir en Twitter.

Conclusiones, contribuciones y trabajos futuros

Conclusiones

En esta tesina hemos analizado y estudiado el problema de incorporar aspectos de interacción social a contenidos diversos, con el objetivo principal que dicha inclusión sea no intrusiva y permita a los usuarios interactuar con diferentes proveedores sociales de manera transparente.

Comenzamos el trabajo desde un enfoque teórico, con el objetivo de encontrar soporte conceptual al fenómeno actual de redes sociales. Hemos construido una definición de lo que consideramos e interpretamos, a partir de su análisis y estudio, como servicio de red social. También, hemos tenido en cuenta las tecnologías actuales vinculadas con la proliferación de los servicios sociales, y como se ponen de manifiesto en sitios específicos, dando soporte al argumento de su naturaleza viral.

A partir de este marco introductorio, abordamos el análisis de los problemas que encontrábamos y a los cuales buscábamos solución en torno al objetivo central de enriquecimiento social de contenidos y el estado actual del fenómeno de redes sociales. En sí, podemos pensar esos problemas como objetivos particulares en torno a o dependientes del objetivo central ya mencionado.

Entonces, procedimos a estudiar la construcción de una herramienta que permitiera lograr resolver tales cuestiones. Comenzamos por un análisis exhaustivo de los mecanismos y tecnologías de integración que ofrecen las redes sociales (estado del arte). Luego, procedimos a seleccionar aquellas tecnologías, priorizando estándares, que servían para resolver parte o la totalidad de cada uno de los problemas planteados. Además, este análisis sirvió de base para generar nuestro primer aporte de identificación de aspectos de redes sociales, su interrelación, y conexión con posibles contenidos. Con este aporte como marco preliminar, explicamos la arquitectura general de nuestra solución y luego, más adelante en el desarrollo, especificamos detalladamente cada uno de los elementos de dicha arquitectura, denotando la parte de la problemática que se estaba atacando.

Por lo tanto, analizando el trabajo realizado, expresado a lo largo de este documento, encontramos que:

- La herramienta construida permite dar resolución a los planteos iniciales que motivaron la realización del trabajo y alcanza los objetivos propuestos, argumentando esta afirmación con lo expuesto y desarrollado en los capítulos 4 y 5. En el capítulo 1, habíamos planteado como hipótesis de trabajo que se puede aumentar el impacto de

las SNS si se simplifica la inserción de funcionalidad de redes sociales en sitios existentes de contenidos. Como hemos desarrollado hasta aquí, es claro que cuanto más simple y homogéneo sea el mecanismo de socialización diseñado, más sencilla y transparente será su incorporación a un sitio de contenidos. Entonces, el aumento del impacto de las SNS provendrá de ocultar consideraciones específicas de cada una, y utilizarlas por medio de un lenguaje común de acciones sociales (la API propuesta en el presente trabajo).

- Al proponer una API unificada genérica obtenemos un enfoque abstracto común para abordar el problema de cuáles son los aspectos se pueden considerar y que valen la pena para la interacción social.
- Dicha API genérica es necesaria para abstraernos de los proveedores sociales. La api se conforma de un conjunto de métodos que representan aspectos de interacción social: comentar, compartir, calificar y login. Además, dicha especificación genérica puede servir para construir implementaciones para otros tipos de aplicaciones, como por ejemplo, para dispositivos móviles.
- El mecanismo de integración como servicios web independientes permite la evolución en forma separada de cada componente. Esto es porque cada componente está acoplado débilmente a los demás, y todo vínculo o dependencia entre ellos se expresa mediante parámetros que se establecen durante la configuración. Por ejemplo, será posible agregar una nueva implementación de la API genérica correspondiente a un proveedor social que se desea agregar en el componente Socializador, y este cambio no afectará al resto de los componentes. Más aún, dicha modificación en particular involucrará solamente a una parte del Socializador, específicamente al Tejedor de Implementaciones visto en la descripción general de la arquitectura en el capítulo 4.
- Con la estrategia de solución propuesta en este trabajo, proveemos un mecanismo para permitir la interacción social a partir de servicios de datos existentes de un modo transparente. En primer lugar, esto se consiguió al definir una API genérica de acciones sociales de interés, por lo que la interacción con un sitio enriquecido será igual para distintos proveedores sociales. En segundo lugar, cada implementación de la API correspondiente a cada proveedor contempla todos los aspectos específicos del mismo. Esto significa que los detalles técnicos están ocultos a los usuarios y permite que las distintas implementaciones sean intercambiables. En consecuencia, lo que se logra es homogeneizar los proveedores de servicios sociales para que la participación de cada uno en la arquitectura propuesta esté predefinida por un lenguaje común de acciones, nuestra API social.
- La separación de la representación visual y del contenido (tanto social como plano) es útil para que pueda ser consumido por otra entidad web y a su vez visualizado para poder interactuar socialmente. Esto significa que el XML provisto tanto por el Generador de Contenidos (plano) como por el Socializador (con contenido social) puede formar parte de otro flujo de tratamiento de los datos en él, teniendo en cuenta la tecnología de mushups mencionada en el capítulo 2. Por ejemplo, un servicio web

que relacione contenidos puede tomar el XML proveniente del Generador o del Socializador y vincular el contenido existente en él con otras fuentes de información. Además, y no menos importante, el esquema visual puede evolucionar y cambiar fácilmente al ser independiente de la definición de los datos.

- La integración es una necesidad creciente que tienen los sitios de redes sociales para llegar a usuarios que consumen contenidos fuera de ellos. Nuestra herramienta socializadora aprovecha este fenómeno abarcando varios proveedores. Esta tendencia creciente se pone de manifiesto con los servicios y herramientas de integración cada vez más diversos que ponen a disposición los proveedores sociales. Los más populares consideran e incluyen categorías del tipo "Desarrollador" o "API", en las cuales proporcionan documentación con ejemplos prácticos acerca de las herramientas disponibles para ser incorporados desde una ubicación externa (como hemos visto en el capítulo 3).
- La integración no solo sirve a los sitios de redes sociales sino también a los proveedores de contenidos que, por medio de los mecanismos de integración tratados en este trabajo, llegan a más usuarios en cada red social. Entonces, existe un beneficio mutuo que puede ser potenciado con una herramienta como la construida. En resumen, lo que estamos logrando es aumentar el espectro de usuarios alcanzables de manera transparente. Entre las motivaciones que hemos mencionado en el capítulo 1, explicamos que las redes sociales proporcionan un excelente mecanismo para difundir e incrementar conocimientos. Esto persigue el efecto de contagiar o estimular a los usuarios a consumir contenidos diversos, expandir su conocimiento o intercambiar experiencias, factores importantes para aumentar el espectro de usuarios. Por otro lado, lo expresado en esta conclusión representa un argumento más que da soporte a lo expresado en nuestra hipótesis de trabajo del capítulo 1 (también mencionada anteriormente en las conclusiones).

Contribuciones

- *Definición del problema de enriquececimiento e identificación de los requerimientos clave:* Lo central de esta contribución es el aporte de un marco conceptual al fenómeno actual en el cual las redes sociales se integran en sitios de terceros, y viceversa. Cuando comenzamos a analizar y estudiar el tema, encontramos que no existía un desarrollo descriptivo del mismo. Lo interesante de esta contribución es que podría utilizarse como punto de partida para encarar un proceso de automatización, teniendo en cuenta los requerimientos que hemos detectado.
- *Análisis exhaustivo de tecnologías para enriquecer contenidos con aspectos de redes sociales:* cabe destacar que por tratarse de un dominio tan flexible y en constante evolución, haciendo los últimos retoques a este trabajo Facebook introdujo un nuevo desarrollo, denominado Graph API, la cual define un modelo de datos social, estructurado como su nombre lo indica en forma de grafo. Otro ejemplo que podemos mencionar es que MySpace incorporó nuevos widgets, incluyendo un mecanismo de

mensajes en tiempo real, junto con la API correspondiente para poder consumir el servicio, equivalente al que ya existe en Facebook.

- *Un diseño general de solución que puede ser implementado en diversos lenguajes y puede ser extendido para dar soporte a nuevos proveedores sociales:* la solución diseñada no impone restricciones específicas de desarrollo. Si bien está pensada para su funcionamiento como servicios web que interactúan entre sí, puede ser construida en cualquier leguaje de scripting del lado servidor (PHP, ASP.NET, etc.). Además, su estructura de diseño tipo framework, basada en los principios y patrones de la programación orientada a objetos, permite que la adición de un nuevo proveedor sea sencilla y transparente.
- *Una prueba de concepto para la evaluación de la efectividad del diseño y su implementación de referencia*: lo importante de esta contribución es que demuestra que el diseño general propuesto en el capítulo 4 se ha podido plasmar utilizando diferentes aspectos de diseño y distintas tecnologías de servicios web (REST, XML, XSLT, etc.).
- *Una API genérica de aspectos sociales, expresada a través de nuestro diseño, que permite abstraernos de los proveedores sociales*: lo atractivo de esta contribución es que representa un mecanismo que posibilita la interacción social a partir de proveedores sociales de datos existentes de un modo transparente, constituyendo un lenguaje social homogéneo. Como mencionamos antes en este capítulo, este aporte puede ser utilizado en la construcción de otras aplicaciones, que requieran consumir contenidos sociales.
- *Un mecanismo de configuración de alto nivel interactivo para enriquecer contenidos:* Lo interesante de esta contribución es que construimos un medio mediante el cual es posible personalizar las características de los componentes Socializador y Visualizador de forma sencilla y clara, incluyendo las distintas opciones de enriquecimiento social (aspectos, proveedores, etc.). Esta herramienta es clave para permitir que los diferentes componentes de la arquitectura puedan interactuar, y a su vez, que los mismos sean intercambiados (por ejemplo, consumir datos desde otro Generador de Contenidos). Esto último es posible debido a que los mismos en el diseño propuesto están débilmente acoplados.
- *Conjunto de widgets estándar para implementar visualmente los mecanismos para interactuar socialmente:* esta contribución tiene como atractivo la utilización y evaluación del framework multinavegador "ExtJs" para construir aplicaciones ricas de internet. Mediante este conjunto de librerías Javascript y pequeñas porciones de código HTML de muy pocas líneas, logramos construir widgets atractivos para los aspectos sociales identificados en nuestra API a un bajo costo, empleando el diseño de la vista provisto por dicho framework.

Trabajos Futuros

Granularidad del configurador

La configuración del servicio social es un mecanismo que define de ante mano y para todos los usuarios, el modo en el que se realizara la socialización, por ejemplo elegir al proveedor de los aspectos sociales. Al mismo tiempo, hemos incluido en la componente de visualización un mecanismo que retrasa al momento del inicio de sesión, la chance de decidir que plataforma usar. En tal sentido, el contenido puede ser socializado en diferentes plataformas por diferentes usuarios.

En esta sección se introduce la posibilidad de realizar una especialización de la granularidad en la configuración. El escenario que actualmente planteamos e implementamos, propone una única configuración para todo el sitio, por ende, única para todos los usuarios finales que accedan al sitio socializado. De este modo, el nivel de la configuración le aplica todos los aspectos seleccionados a todos los elementos socializables. Para nuestro ejemplo, estos serian los elementos 'nota'.

Lo que se propone es desagregar esto de modo de hacerlo más flexible. A continuación se proponen distintas configuraciones posibles, yendo de la más restrictiva a la más flexible:

- Para un proveedor, ciertos aspectos se aplican a ciertas notas: En este caso, se selecciona un aspecto que no se le aplicara a todas las notas, sino que se podrá ligar dicho aspecto a las notas que se quieran enriquecer, para el proveedor elegido. Por ejemplo, una nota se enriquece con el aspecto comentar y otra nota con el aspecto compartir.
- Para ciertos proveedores, ciertos aspectos se aplican a ciertas notas: Como mejora del punto anterior, vemos que se pueden enriquecer las notas individualmente pero con la posibilidad de seleccionar para cada nota, que proveedor se encargara de aplicarle el aspecto seleccionado. Por ejemplo, seleccionamos que una determinada nota se enriquece con el aspecto comentar de un determinado proveedor, mientras que otra nota la decidimos enriquecer con el aspecto compartir, en este caso con otro proveedor.

De este modo logramos personalizar el enriquecimiento que se le pueden aplicar a los elementos socializables, tanto a nivel de proveedor, como a nivel de aspecto.

Para poder lograr esto se proponen los siguientes puntos a tener en cuenta a la hora de abordar esta mejora:

- Esquema que modele las notas.
- Mecanismo de persistencia de la información de la socialización aplicada a cada nota.

Soporte a la Navegabilidad

Tal como se menciona a lo largo de la documentación, el acceso a los recursos del sitio tiene el soporte REST para los requerimientos solicitados. Esto nos permite acceder a distintos niveles de recursos. En nuestro sitio de ejemplo, contamos con publicaciones, donde cada una de estas está compuesta por notas, donde podemos:

- Acceder a todas las notas del sitio.
- Acceder a todas las notas de un determinado año.
- Acceder a todas las notas de un determinado año y mes.
- Acceder a una determinada nota de un año y mes dado.

Supongamos que queremos recuperar el recurso

http://tesispapudani.dyndns.org/Socializador/Visualizador/2010/julio/1

Esto nos retornara, transformación de por medio, una vista HTML del contenido XML de la primera nota del mes de Julio de 2010. En general, y dado el soporte provisto por REST, actualmente no se da soporte a la navegabilidad vía links entre distintas notas. La propuesta de este inciso sugiere incluir este soporte para la lectura de diferentes recursos vía los distintos links que pudiera haber en las distintas notas.

Para dar soporte a la navegabilidad planteada, se debería tener en cuenta:

- Ajustar el XML esquema.
- Ajustar las reglas de transformación del XSLT.

Acceso REST y weaving dinámico de contenido socializable con su correspondiente contenido socializado

Tal como se ha mencionado a lo largo de la documentación, la solución que planteamos propone un origen de datos que nos provea los recursos que utilizaremos para llevar a cabo la socialización.

El efecto que tiene el aplicar alguno de los aspectos sociales sobre los recursos originales resulta nulo, por lo tanto no intrusivo para estos. Por otro lado, toda la información que envía el usuario final y que resulta de la interacción con los widgets de los aspectos sociales con que se enriqueció el sitio, no son manejados por la herramienta que proponemos. De esta forma, la componente socializadora actúa como puente entre nuestra aplicación y las redes sociales, que mediante su API REST, se encargan del manejo de la información que el usuario final envía.

Esta situación tiene dos consecuencias en cuanto a la socialización:

- El manejo de la información, tanto de la base de datos como de los medios para acceder a esta, resultan responsabilidad de la red social en cuestión.

- Como consecuencia del primer punto, la herramienta se encuentra sujeta a ser tan flexible con la información socializada como los proveedores enriquezcan su API.

Esto impacta directamente en la solución propuesta. Como ejemplo práctico y concreto, tomemos el aspecto 'comentar' y veamos como lo manejan algunos de los proveedores sociales.

MySpace

Al comentar una nota, vemos que se refleja en nuestro perfil, pero este proveedor no guarda dicho comentario en ninguna tabla o base de datos. Por lo tanto, resulta imposible poder recuperar los comentarios asociados a una nota en esta red social.

Facebook

A diferencia del resto de las redes sociales, esta si hace un manejo integro de los comentarios que hagamos. En este caso, el proveedor guardara cada comentario asociándolo a un identificador dado.

Surge como consecuencia de lo anterior, los siguientes puntos:

- La posibilidad de recuperar la información que surgió a partir de la socialización resulta posible, siempre y cuando la red social nos provea una API con soporte para tal fin. Siendo esto hoy posible, únicamente para el caso de Facebook.
- Una eventual implementación de esta propuesta, implicaría tener un esquema XML para el servicio de socialización, que resultaría en una extensión del esquema XML actualmente en uso por el origen de datos. De este modo, se conseguiría extender la API propuesta, de modo de poder darle un soporte de acceso REST al contenido que se genere con esta propuesta.

Para finalizar, cabe mencionar que en la presente herramienta no se ha podido llevar esto a cabo por una cuestión del soporte que nos dan los proveedores sociales, siendo Facebook el único que nos posibilitaba esto. Entendemos que esto es una limitación tecnológica y temporaria.

Relación con Web semántica

Como hemos mencionado en el capítulo 2, la relación entre las redes sociales y la web semántica es un tema emergente y muy interesante, que involucra diversos proyectos, y representa una veta de investigación posible para la continuación de este trabajo.

Básicamente, la Web Semántica es un grupo de métodos y tecnologías para entender el significado, o "semántica", de la información en la WWW. Estas tecnologías incluyen el framework de descripción de recursos (RDF), una variedad de formatos de intercambio de información (como XML) y notaciones como el Schema RDF y el lenguaje OWL.

Uno de los proyectos existentes que vincula los conceptos de las redes sociales con los preceptos de la Web Semántica es el denominado FOAF (Friend of a Friend). Se trata de la definición de un vocabulario RDF para expresar metadatos acerca de las personas, sus intereses, relaciones y actividades. Consiste en una comunidad específica dentro del objetivo global de la Web Semántica de crear una web de datos procesables.

La idea básica detrás de FOAF es simple: la web se trata de establecer conexiones entre elementos. Este proyecto provee un mecanismo básico que nos ayuda a expresar en la web las conexiones entre los elementos que nos interesan.

Entonces, nuestra propuesta de trabajo futuro en torno a la Web Semántica involucra estudiar y analizar los objetivos que persigue con un enfoque social, considerando los proyectos que vinculan ambos temas (como FOAF) y los aportes introducidos en el presente trabajo.

Dicho proceso de investigación puede involucrar aspectos tales como analizar y estudiar de qué forma es posible representar mediante los lenguajes de la Web Semántica la información social generada por nuestra herramienta, y que impacto tendría teniendo en cuenta los modelos actuales de datos sociales definidos por los proveedores sociales analizados. Otro aspecto interesante para evaluar es el mecanismo que permite consumir información social expresada en esos lenguajes semánticos.

Anexo

Manual del desarrollador

Hasta aquí en el presente documento se hizo hincapié en detallar la implementación de la herramienta, comenzando por una vista general de la solución planteada, avanzando en el detalle de cada componente de la misma. Se analizaron las tecnologías disponibles para llevar a cabo la aplicación propuesta, planteando como solución una arquitectura separada en capas. Luego, se profundizó en cada uno de los componentes identificados y el mecanismo mediante el cual dichos componentes interactúan, detallando por ejemplo, los patrones de diseño utilizados.

El presente capítulo está destinado a aquellos usuarios avanzados que deseen conocer la estructura del trabajo con el fin de alcanzar la capacidad para extenderlo o modificarlo. Es por eso que resulta casi imprescindible tener nociones básicas de ciertas tecnologías para poder llevar a cabo un desarrollo eficiente.

El lenguaje de programación elegido fue Java. Hay varias razones que fundamentan esta elección, como por ejemplo el hecho de que Java es un lenguaje orientado a objetos, potente, que incluye un conjunto de características nativas que facilitan la interacción servidor-a-servidor preponderante en el trabajo realizado. Específicamente, hacemos uso de J2EE, plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

En este trabajo en particular, además del lenguaje Java, se utiliza el lenguaje XML para el intercambio de información de contenidos e información vinculada con las configuraciones. Por otro lado, a lo largo del documento utilizamos el lenguaje de modelado UML, que deberá adoptarse para documentar cualquier nueva funcionalidad incorporada a la herramienta.

Entorno de desarrollo

Eclipse

Para el proceso de construcción de la herramienta utilizamos Eclipse, que es un entorno de desarrollo integrado de código abierto multiplataforma y un sistema extensible vía plugins. Está escrito principalmente en Java y puede ser utilizado para desarrollar aplicaciones en Java, como así también en otros lenguajes por medio de plugins, como C, C++, PHP, etc. En particular, utilizamos la versión 3.5 del IDE, específica para desarrollo EE [Eclipse].

Además, el entorno se configuró para interactuar con un sistema de revisión de versiones de software (SVN), con el cual se interactúa mediante el plugin SvnKit, que se puede descargar e

instalar desde el mismo Eclipse. Dicho sistema de revisión de versiones, se utilizó para llevar a cabo el desarrollo colaborativo de la herramienta.

Apache Tomcat

Tomcat, también llamado Jakarta Tomcat o Apache Tomcat, funciona como un contenedor de aplicaciones desarrollado bajo el proyecto Jakarta, en la Apache Software Foundation. Tomcat implementa las especificaciones de los Servlets y de Java Server Pages (JSP) de Sun Microsystems. La versión utilizada para el desarrollo ha sido la 6.0.20. Entre sus características, se destaca la implementación para Servlets 2.5, JSP 2.1 y diseño compatible para trabajar con Java SE 5.0 y posteriores [Tomcat].

Restlet

Como hemos mencionado en capítulos anteriores, Restlet es un framework REST liviano y de código abierto para la plataforma Java. Es apropiado tanto para aplicaciones clientes como servidor. Soporta la mayoría de los mecanismos de transportes sobre Internet, formatos de datos y especificaciones de servicio como HTTP, HTTPS, XML, JSON. La versión utilizada para el desarrollo es la 1.1.10 [Restlet].

Para la inclusión de Restlet en el proyecto desarrollado, se tuvo que descargar dicha herramienta, teniendo que importar en nuestro desarrollo un conjunto de archivos (de extensión .jar) necesarios para el correcto funcionamiento de dicho framework.

Puntos de extensión

En esta sección, nos centramos básicamente en brindar un soporte para guiar a quienes quieran llevar a cabo alguna extensión o modificación en la solución desarrollada. De este modo, se mencionan los 'hotspot' o puntos de extensión de la herramienta, de modo de dar un marco al desarrollador y detallar los pasos a seguir para completar alguna de las siguientes acciones:

- Agregar un aspecto social.
- Agregar un proveedor social.

Agregar o modificar un aspecto social

La herramienta se desarrollo proveyendo los aspectos de comentar, compartir y opinar. No obstante, se desarrollo teniendo en cuenta que los aspectos sociales deberían ser fácilmente configurables. Por lo tanto, para la inclusión de un nuevo aspecto social, se deben llevar a cabo los cambios que se detallan a continuación.

Cambios en el modelo

Una vez identificado el aspecto que se quiera agregar, se deberán llevar a cabo los siguientes cambios en el modelo de la herramienta:

- Agregar el método con el nombre del nuevo aspecto, en la superclase de proveedores.

A continuación se muestra la signatura de los métodos que se encuentran en dicha clase.

```
public abstract class Provider {
...
public abstract void loadAuthenticatedUserInfo(UserSocialInfo
authenticatedUser);


public abstract String comment(UserSocialInfo userSocialInfo,
String parameters);


public abstract String share(UserSocialInfo userSocialInfo,
String parameters);

public abstract String expressOpinion(UserSocialInfo
userSocialInfo, String parameters);
...
}
```

- Proveer una implementación al nuevo aspecto social en cada una de las subclases de la jerarquía 'Provider', de modo que todos los proveedores sociales puedan dar soporte al nuevo aspecto.
- Luego, debemos completar el enumerativo 'AtributosSociales' que contiene la representación de cada aspecto social identificado. Esto nos provee tanto de expresividad en nuestro código, como de seguridad, pudiendo evitar errores en tiempo de ejecución encontrándolos en tiempo de compilación debido a los chequeos de tipo.
- A continuación se debe actualizar la clase 'Configurador'. Esta última, se implemento extendiendo a HttpServlet y maneja toda la lógica de la componente de configuración. En la misma se almacenara y guardara para cada aspecto, si se deberá mostrar o no, de acuerdo a lo seleccionado en el configurador por el administrador.

```
public class Configurador extends HttpServlet {
...
HashMap<AtributosSociales,String> selectedAspects = new
HashMap<AtributosSociales,String>();
selectedAspects.put(AtributosSociales.COMENTAR, comentar);
selectedAspects.put(AtributosSociales.COMPARTIR, compartir);
selectedAspects.put(AtributosSociales.OPINAR, opinar);
...
context.setAttribute("selectedAspects", selectedAspects);
}
```

Como puede verse en el fragmento de código extraído, dentro de la lógica de nuestro servlet creamos un mapa <AtributosSociales,String> que guardará para cada aspecto soportado, el valor elegido en la vista del configurador. De este modo, el cambio que habría que hacer sobre esta clase sería el de agregar al mapa el valor asociado al nuevo aspecto.

Cambios en la plantilla XSLT

Como se ha mencionado a lo largo de la documentación, para poder generar la vista HTML se utiliza una plantilla XSLT que transforma los recursos XML que obtenemos del generador de contenidos. Para la inclusión de un nuevo aspecto, se deberá ajustar el archivo 'plantillaTransformacion.xsl', agregando una regla en la plantilla XSLT que de soporte al nuevo aspecto. De esta forma, cuando se realice la transformación, por cada ocurrencia de dicha regla se agregara un widget del aspecto asociado. A continuación, se muestra la regla que corresponde al aspecto compartir.

```
<xsl:template name="widget_compartir">
<a class="button" id="compartir"
onclick="mostrarWidgetCompartir(this,location.href);"></a>
</xsl:template>
```

Cambios en el cliente JavaScript

Luego de haber ajustado la plantilla con la que se realizara la transformación y a la cual se le agrego la regla correspondiente al nuevo widget, es necesario ajustar la lógica que atenderá a los distintos eventos que este dispare.

Para la realización de los widgets, se utilizo ExtJS, un framework JavaScript, para desarrollar aplicaciones web multinavegador. Dado que la tarea de enriquecer un sitio social debía ser no intrusiva, resultaba imposible incrustar los widget con los distintos formularios de interacción en la misma página en la que se muestra el contenido. Dada esta situación, se aprovecha toda la potencia de ExtJS para que la inclusión de los widget sea lo más transparente posible. Es así, que al enriquecer un sitio con algún aspecto, solo veremos el botón a partir del cual se abre una ventana con el formulario correspondiente para interactuar socialmente.

Dada esta situación, para poder incluir un aspecto se deberá ajustar el archivo 'ajaxMethod.js' que contendrá tanto el código que representa la vista del formulario, como la lógica asociada a los distintos eventos. A continuación se muestra el fragmento de código que atiende el evento 'onclick' del botón asociado al aspecto.

```
function mostrarWidgetCompartir(domNode,enlaceACompartir){
if(compartir){
enlaceCompartido = enlaceACompartir;
enlace.setText("Enlace compartido: "+enlaceCompartido);
compartir.show(domNode);}
}
```

Como se puede ver, esta función es la encargada de abrir la ventana con la cual podrá interactuar el usuario y de pasarle el vínculo que este último haya compartido. El código asociado al evento que muestra la ventana y contiene la lógica de asociada se detalla a continuación.

```
if(!compartir){
compartir = new Ext.Window({
title:'Comparte el enlace',
layout:'auto',
...
items: new Ext.FormPanel({
labelWidth: 75, // label settings here cascade unless
overridden
frame:true,
bodyStyle:'padding:5px 5px 0',
width: 350,
defaults: {width: 230},
defaultType: 'textfield',
items: [ enlace = new Ext.form.Label({
text: 'Etiqueta',
style: 'font-style: italic; color: #4682B4;'
}),
{
fieldLabel: 'Comentario',
name: 'comentarioEnlace',
allowBlank:false

}
],
buttons: [{
text:'Compartir',
disabled:false,
handler: function(){
enviarRequestCompartir(enlaceCompartido,compartir.items.item(0)
.items.item(1).getValue());
compartir.items.item(0).items.item(1).setValue("");
compartir.hide();
}
},{
text: 'Cerrar',
handler: function(){
compartir.hide();}}

]})});
}
```

Dentro de la sección 'ítems', se declaran los componentes que mostrara la ventana. Dentro de esta misma sección, se declaran los botones con la correspondiente lógica asociada, que para nuestro caso, consiste en llamar a otra función javascript que será la encargada de realizar la comunicación servidor a servidor.

Cambios en la vista del Configurador

En cuanto a la vista de la componente de configuración, vale decir que se actualizara automáticamente con todos los aspectos presentes en la herramienta. El mecanismo utilizado, se basa en recorrer los métodos abstractos de la superclase, cargándolos en la vista de la componente Configurador.

Agregar o modificar un proveedor social

A continuación explicaremos el mecanismo para agregar y dar soporte a través de nuestra herramienta a un nuevo proveedor social. En primer lugar, es necesario listar los requisitos que debe cumplir dicho proveedor para poder interactuar con nuestro socializador, teniendo en cuenta lo desarrollado en el capítulo 3 de estado del arte:

- Un mecanismo de integración que permita consumir sus contenidos desde un servidor externo, tal como sucede con los proveedores que utilizamos en este trabajo. En ese caso, se trata de aplicaciones que los usuarios de cada red pueden crear, las cuales representan puntos de entrada que posibilitan la interacción con el proveedor desde un servidor externo o aplicación de escritorio. La creación de dicha aplicación permite obtener claves de API y secretas que serán utilizadas en el intercambio con el proveedor.
- Un mecanismo de interacción servidor-a-servidor orientado a solicitudes REST. Los proveedores sociales configurados y utilizados durante el desarrollo de este trabajo exportan APIs REST.
- El proveedor social debe soportar e implementar el mecanismo de autenticación delegado OAuth, en cualquiera de sus dos versiones (1.0 o 2.0).

Dicho esto, para la inclusión de un nuevo proveedor debemos realizar los siguientes pasos:

Crear una aplicación en el proveedor social

De modo de poder obtener las credenciales que permitirán establecer la comunicación servidor a servidor. Dicha aplicación será nuestro punto de acceso al proveedor y las credenciales de la misma serán las bases para el mecanismo de autenticación delegado OAuth. En el capítulo del estado del arte vimos como crear una aplicación en Twitter y obtener dichos valores.

Ajustes en la herramienta de socialización

Este paso consiste en agregar un archivo de propiedades en la carpeta 'configuración' del WebContent del proyecto realizado. Este archivo deberá tener las siguientes entradas:

- api_key: valor que identifica la aplicación que utilizaremos para interactuar con el proveedor social.

- secret_key: valor que permite identificarnos de forma univoca ante el proveedor social durante el proceso de comunicación y autenticación.
- login_image: URL de una imagen para desplegar el correspondiente botón de login para el proveedor.
- logout_image: URL de una imagen para desplegar el correspondiente botón de logout para el proveedor.

El archivo quedaría como el siguiente fragmento:

```
api_key=673f81e6d7e94923bae2c05cc49b8c00
secret_key=280b345c4ec547bd9a112e80a31c8a3c5aac21e0375b4c838882
43d0a6e1563b
```

Cambios en el modelo

Crear una clase que extienda la superclase 'Providers' e implemente los métodos abstractos correspondientes a los aspectos definidos en esta. Dicha clase contendrá la implementación específica para cada aspecto de acuerdo al soporte que provea el nuevo proveedor para cada uno. Cada aspecto está implementado como un método plantilla, donde cada componente estructura de la plantilla será reimplementado en la clase específica. Dichos métodos estructura de la plantilla corresponderán a recuperar características específicas del proveedor, como por ejemplo el punto de entrada REST que permitirá ejecutar el aspecto correspondiente que queremos implementar en el proveedor social nuevo. A continuación se muestra la implementación del aspecto comentar para el proveedor Twitter.

```
public String share(UserSocialInfo userSocialInfo, String
parameters) {
Map<String, String> parametersValues =
parseParameters(parameters);

Map<String,String> args = new HashMap<String,String>();
Map<String,String> bodyArgs = new HashMap<String,String>();

String status = "Enlace compartido:
"+parametersValues.get("link")+".
"+parametersValues.get("comentario");
args.put("oauth_consumer_key", "a7Zcu8yDG1CgtRM50fqtQ");
args.put("oauth_token", userSocialInfo.getAccessToken());
args.put("status",status);
bodyArgs.put("status",status);
String urlStr =
Util.generateRequestUrl("http://api.twitter.com/1/statuses/upda
te.xml",args,userSocialInfo.getAccessTokenSecret(),"POST",
bodyArgs.keySet(),"G57HLAkSoIJDcAaYF5AhA1BrSUh9Y15t5PQbuyEXs");

Map<String,String> result = Util.doHttpRequest(urlStr, "POST",
bodyArgs, null, null, null);

// Armamos la respuesta que luego recibira el JS que invoco al
ActionListener
String respuesta = "";
respuesta+= "<respuesta>";
respuesta+="<estadoRespuesta>"+getStatusMessage(result.get("Res
ponseCode"))+"</estadoRespuesta>";
respuesta+="<uid>"+userSocialInfo.getUserId()+"</uid>";
respuesta+= "</respuesta>";

return respuesta;
}
```

Cambios en la vista del Configurador

Al igual que al agregar un aspecto, cuando se agrega un proveedor, la tarea de ajustar la vista de la componente de configuración se realiza de modo transparente. Nuevamente, el mecanismo utilizado, se basa en recorrer los archivos de propiedades que se encuentran en la carpeta 'configuración' del WebContent. Así, se actualiza la vista con todos los proveedores sociales presentes en la herramienta.

Referencias bibliográficas

Capítulo 1

- Clemons, E. K., Barnett, S., & Appadurai, A. (2007). The future of advertising and the value of social network websites: Some preliminary examinations. ICEC '07: Proceedings of the Ninth International Conference on Electronic Commerce, Minneapolis, MN, USA. (pp. 267-276).
- Skiba, D. J. (2007). Nursing education 2.0: Poke me. where's your face in space? Nursing Education Perspectives, 28(4), 214.
- Miltiadis D. Lytras y Patricia Ordóñez de Pablos. Social Web Evolution. Capítulo 5 - Web 2.0 Social Networking Sites
- Stanley Wasserman y Katherine Faust. Social Network Analysis, Methods and Applications. Cambridge University Press, 1994. ISBN 0-521-38269.
- Boyd, d. m., & Ellison, N. B. (2007). Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication, 13(1), 210-230.
- Ellison, N. B., Steinfield, C., & Lampe, C. (2007). The benefits of facebook "Friends:" social capital and college students' use of online social network sites. Journal of Computer-Mediated Communication, 12(4), 1143-1168.
- Tong, S. T., Van Der Heide, B., Langwell, L., & Walther, J. B. (2008). Too much of a good thing? The relationship between number of friends and interpersonal impressions on facebook. Journal of Computer-Mediated Communication, 13(3), 531-549.
- Lampe, C. A. C., Ellison, N., & Steinfield, C. (2007). A familiar face(book): Profile elements as signals in an online social network. CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, San Jose, California, USA. (pp. 435-444).
- Hathi, S. (2008). Billions lost from social networking. Strategic Communication Management, 12(2), 9.
- John Scott. Social Network Analysis, A Handbook. SAGE Publications, 2000. ISBN 0-7619-6338-3.
- http://edelmandigital.com/2010/05/20/social-networks-become-social-entertainment/

Capítulo 2

- [Web Semántica] URLs:
  - http://en.wikipedia.org/wiki/Semantic_Web
  - http://microformats.org/
  - http://www.w3.org/RDF/

  - http://www.w3.org/TR/owl-ref/
  - http://www.foaf-project.org/
  - http://www.xml.com/pub/a/2004/02/04/foaf.html
- [MushUps]:
  - Agnes Koschmider, Victoria Torres, Vicente Pelechano. Elucidating the Mashup Hype: Definition, Challenges, Methodical Guide and Tools for Mashups.
  - http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29
- [REST] URLs:
  - http://en.wikipedia.org/wiki/Representational_State_Transfer

Capítulo 3

- [Autenticación en Facebook]:
  - http://developers.facebook.com/blog/post/108
  - http://wiki.developers.facebook.com/index.php/Anatomy_of_a_Facebook_Connect_Site
  - Wayne Graham. Facebook API Developers Guide.

- [OAuth] URLs:
  - http://oauth.net/
  - http://geeks.ms/blogs/gtorres/archive/2010/05/30/autenticacion-oauth.aspx
  - http://hueniverse.com/oauth/
  - http://dev.twitter.com/pages/auth
  - http://dev.twitter.com/pages/basic_to_oauth
- [REST]:
  - Leonard Richardson, Sam Ruby. RESTful Web Services. O'Reilly, 2007. ISBN-10: 0-596-52926-0.
  - http://en.wikipedia.org/wiki/Representational_State_Transfer
  - http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html
  - http://java.sun.com/developer/technicalArticles/WebServices/restful/

- [OpenSocial] URLs:
  - http://www.error500.net/google-opensocial
  - http://code.google.com/intl/es-AR/apis/opensocial/docs/
  - http://wiki.opensocial.org/index.php?title=Articles_%26_Tutorials
  - Espacio de nombre OpenSocial - http://code.google.com/intl/es-AR/apis/opensocial/docs/reference.html
  - Guía de introducción - http://code.google.com/intl/es/apis/opensocial/gettingstarted.html

  - Tutorial OpenSocial - http://code.google.com/apis/opensocial/articles/tutorial/
  - http://code.google.com/intl/es-AR/apis/opensocial/docs/0.8/restfulspec.html
  - http://www.ietf.org/rfc/rfc4287.txt
  - http://oauth.net/core/1.0/
- [Twitter] URLs:
  - Vista general- http://dev.twitter.com/pages/auth
  - API - http://dev.twitter.com/pages/api_overview
  - Desarrollador - http://dev.twitter.com/pages/every_developer
  - Plugin Sociales - http://twitter.com/goodies/widget_search
  - http://dev.twitter.com/pages/rate-limiting

Capítulo 4

- Kiczales G., Lamping J.: ECOOP'97 - Object-Oriented Programming. Aspect-oriented.
- Ramnivas Laddad . I want my AOP!. January 18, 2002. http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html
- Odysseas Papapetrou and George A. Papadopoulos . Aspect Oriented Programming for a component based real life application: A case study. Department of Computer Science. University of Cyprus
- Bart De Win, Bart Vanhaute and Bart De Decker. Building Frameworks in AspectJ. Department of Computer Science , K.U. Lenven . 2001.
- De Fraine, B., Vanderperren, W., Suvée, D.: Motivations for Framework-based AOP. Vrije Universiteit Brussel. Belgium.
- Florian Leibert. A Service-Oriented Architecture for Social Networks. VDM Verlag Dr. Mueller e.K. (July 21, 2008). ISBN 978-3639061635.

Capítulo 5

- [XML]:
  - Extensible Markup Language (XML) 1.1 (Segunda Edición).http://www.w3.org/TR/2006/REC-xml11-20060816/
  - XML Schema:
    - http://www.w3.org/XML/Schema
    - http://www.w3schools.com/schema/default.asp
  - XSL:
    - http://www.w3.org/Style/XSL/
    - http://www.w3.org/TR/xslt20/
- [Applet] URLs:
  - http://java.sun.com/applets/
  - http://en.wikipedia.org/wiki/Applet
- [Contenedor web] URLs:

  - http://en.wikipedia.org/wiki/Comparison_of_web_server_software
- [Restlet] URLs:
  - http://www.restlet.org/
- [Servlet Java] URLs:
  - http://en.wikipedia.org/wiki/Java_Servlet
  - http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/javaservlets/servlets.pdf
- [ExtJs] URLs:
  - http://www.sencha.com/products/js/
  - http://dev.sencha.com/deploy/dev/docs/
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Patrones de Diseño. Pearson Educacion, S.A. Madrid. 2003
- Axel Guicking, Thomas Grasse. A Framework Designed for Synchronous Groupware Applications in Heterogeneous Environments.
- Brett D. McLaughlin, Justin Edelson. Java and XML, 3rd Edition. O'Reilly, 2006. ISBN 978-0-596-10149-7.
- Binildas CA, Malhar Barai, Vincenzo Caselli. Service Oriented Architecture with Java. Packt Publishing, 2008. ISBN 978-1-847193-21-6.
-

Capítulo 8

- [Eclipse]
  - http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliosr
- [Tomcat]
  - http://tomcat.apache.org/