UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros de Telecomunicación

A FRAMEWORK FOR BUILDING DISTRIBUTED SOCIAL NETWORK WEBSITES

TESIS DOCTORAL

Antonio Tapiador del Dujo
Ingeniero de Telecomunicación

Madrid, España
Junio, 2013

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

A FRAMEWORK FOR BUILDING DISTRIBUTED SOCIAL NETWORK WEBSITES

TESIS DOCTORAL

Autor: Antonio Tapiador del Dujo
Ingeniero de Telecomunicación

Director: Joaquín Salvachúa Rodríguez
Doctor Ingeniero de Telecomunicación

Junio, 2013

Tribunal nombrado por el Magnífico. y Excelentísimo. Sr. Rector de la Universidad Politécnica de Madrid, el día 22 de Mayo de 2013.

Presidente: D. Juan Quemada Vives. Catedrático de Universidad. ETSI Telecomunicación. Universidad Politécnica de Madrid (UPM)

Vocal: D. Jesús M. González Barahona. Profesor Titular de Universidad. ETSI Telecomunicación. Universidad Rey Juan Carlos (URJC)

Vocal: Dña. Effie Lai-Chong Law. Profesora Titular. College of Science and Engineering. University of Leicester. Reino Unido

Vocal: D. Xabiel García Pañeda. Director de Área de Gestión de I+D+I. Vicerrectorado de Investigación. Universidad de Oviedo (UNIOVI)

Secretario: D. Gabriel Huecas Fernández-Toribio. Profesor Titular de Universidad. ETSI Telecomunicación. Universidad Politécnica de Madrid (UPM)

Suplente: D. Luis Ángel Galindo Sánchez. Profesor Asociado. Ingeniería Telemática. Universidad Carlos III de Madrid (UC3M)

Suplente: D. David Larrabeiti López. Catedrático de Universidad. Ingeniería Telemática. Universidad Carlos III de Madrid (UC3M)

Realizado el acto de defensa y lectura de la Tesis el día 14 de Junio de 2013 en Madrid, habiendo obtenido la calificación de

El presidente, El secretario, Los vocales,

Abstract

Networks are the substance of human communities and societies; they constitute the structural framework on which we relate to each other and determine the way we do it, the way information is diseminated or even the way people get things done. But network prominence goes beyond the importance it acquires in social networks. Networks are found within numerous known structures, from protein interactions inside a cell to router connections on the internet.

Social networks are present on the internet since its beginnings, in emails for example. Inside every email client, there are contact lists that added together constitute a social network. However, it has been with the emergence of social network sites (SNS) when these kinds of web applications have reached general awareness. SNS are now among the most popular sites in the web and with the higher traffic. Sites such as Facebook and Twitter hold astonishing figures of active users, traffic and time invested into the sites.

Nevertheless, SNS functionalities are not restricted to contact-oriented social networks, those that are focused on building your own list of contacts and interacting with them. There are other examples of sites that leverage social networking to foster user activity and engagement around other types of content. Examples go from early SNS such as Flickr, the photography related networking site, to Github, the most popular social network repository nowadays. It is not an accident that the popularity of these websites comes hand-in-hand with their social network capabilities

The scenario is even richer, due to the fact that SNS interact with each other, sharing and exporting contact lists and authentication as well as providing a valuable channel to publize user activity in other sites. These interactions are very recent and they are still finding their way to the point where SNS overcome their condition of data silos to a stage of full interoperability between sites, in the same way email and instant messaging networks work today.

This work introduces a technology that allows to rapidly build any kind of distributed social network website.

It first introduces a new technique to create middleware that can provide any kind of content management feature to a popular model-view-controller (MVC) web development framework, Ruby on Rails. It provides developers with tools that allow them to abstract from the complexities related with content management and focus on the development of specific content. This same technique is also used to provide the framework with social network features. Additionally, it describes a new metric of code reuse to assert the validity of the kind of middleware that is emerging in MVC frameworks.

Secondly, the characteristics of top popular SNS are analysed in order to find the common patterns shown in them. This analysis is the ground for defining the requirements of a framework for building social network websites.

Next, a reference architecture for supporting the features found in the analysis is proposed. This architecture has been implemented in a software component, called Social Stream, and tested in several social networks, both contact- and content-oriented, in local neighbourhood associations and EU-founded research projects. It has also been the ground for several Master's theses. It has been released as a free and open source software that has obtained a growing community and that is now being used beyond the scope of this work.

The social architecture has enabled the definition of a new social-based access control model that overcomes some of the limitations currenly present in access control models for social networks.

Furthermore, paradigms and case studies in distributed SNS have been analysed, gathering a set of features for distributed social networking.

Finally the architecture of the framework has been extended to support distributed SNS capabilities. Its implementation has also been validated in EU-founded research projects.

Resumen

Las redes son la esencia de comunidades y sociedades humanas; constituyen el entramado en el que nos relacionamos y determinan cómo lo hacemos, cómo se disemina la información o incluso cómo las cosas se llevan a cabo. Pero el protagonismo de las redes va más allá del que adquiere en las redes sociales. Se encuentran en el seno de múltiples estructuras que conocemos, desde las interaciones entre las proteínas dentro de una célula hasta la interconexión de los routers de internet.

Las redes sociales están presentes en internet desde sus principios, en el correo electrónico por tomar un ejemplo. Dentro de cada cliente de correo se manejan listas contactos que agregadas constituyen una red social. Sin embargo, ha sido con la aparición de los sitios web de redes sociales cuando este tipo de aplicaciones web han llegado a la conciencia general. Las redes sociales se han situado entre los sitios más populares y con más tráfico de la web. Páginas como Facebook o Twitter manejan cifras asombrosas en cuanto a número de usuarios activos, de tráfico o de tiempo invertido en el sitio.

Pero las funcionalidades de red social no están restringidas a las redes sociales orientadas a contactos, aquellas enfocadas a construir tu lista de contactos e interactuar con ellos. Existen otros ejemplos de sitios que aprovechan las redes sociales para aumentar la actividad de los usuarios y su involucración alrededor de algún tipo de contenido. Estos ejemplos van desde una de las redes sociales más antiguas, Flickr, orientada al intercambio de fotografías, hasta Github, la red social de código libre más popular hoy en día. No es una casualidad que la popularidad de estos sitios web venga de la mano de sus funcionalidades de red social.

El escenario es más rico aún, ya que los sitios de redes sociales interaccionan entre ellos, compartiendo y exportando listas de contactos, servicios de autenticación y proporcionando un valioso canal para publicitar la actividad de los usuarios en otros sitios web. Esta funcionalidad es reciente y aún les queda un paso hasta que las redes sociales superen su condición de bunkers y lleguen a un estado de verdadera interoperabilidad entre ellas, tal como funcionan hoy en día el correo electrónico o la mensajería instantánea.

Este trabajo muestra una tecnología que permite construir sitios web con características de red social distribuída.

En primer lugar, se presenta una tecnología para la construcción de un componente intermedio que permite proporcionar cualquier característica de gestión de contenidos al popular marco de desarrollo web modelo-vista-controlador (MVC) Ruby on Rails. Esta técnica constituye una herramienta para desarrolladores que les permita abstraerse de las complejidades

de la gestión de contenidos y enfocarse en las particularidades de los propios contenidos. Esta técnica se usará también para proporcionar las características de red social. Se describe una nueva métrica de reusabilidad de código para demostrar la validez del componente intermedio en marcos MVC.

En segundo lugar, se analizan las características de los sitios web de redes sociales más populares, con el objetivo de encontrar los patrones comunes que aparecen en ellos. Este análisis servirá como base para definir los requisitos que debe cumplir un marco para construir redes sociales.

A continuación se propone una arquitectura de referencia que proporcione este tipo de características. Dicha arquitectura ha sido implementada en un componente, Social Stream, y probada en varias redes sociales, tanto orientadas a contactos como a contenido, en el contexto de una asociación vecinal tanto como en proyectos de investigación financiados por la UE. Ha sido la base de varios proyectos fin de carrera. Además, ha sido publicado como código libre, obteniendo una comunidad creciente y está siendo usado más allá del ámbito de este trabajo.

Dicha arquitectura ha permitido la definición de un nuevo modelo de control de acceso social que supera varias limitaciones presentes en los modelos de control de acceso para redes sociales.

Más aún, se han analizado casos de estudio de sitios de red social distribuídos, reuniendo un conjunto de caraterísticas que debe cumplir un marco para construir redes sociales distribuídas.

Por último, se ha extendido la arquitectura del marco para dar cabida a las características de redes sociales distribuídas. Su implementación ha sido validada en proyectos de investigación financiados por la UE.

Acknowledgments

La realización de una tesis doctoral supone un largo y duro camino en el que inevitablemente se encuentran compañeros de viaje. Cada uno de ellos ha constituido una oportunidad para aprender algo nuevo. Sin duda, esta tesis doctoral no hubiera podido llevarse a cabo sin sus aportaciones.

Los principales compañeros de viaje han sido el grupo de compañeros de laboratorio, llamados *Isabelinos* a causa a la herramienta de videoconferencia, ya abandonada, con la que nos tocaba iniciarnos a todos. De aquellos comienzos soy capaz de recordar aún muchos nombres, no sin algo de esfuerzo, que me acogieron en el grupo de investigación en mis primeros pasos, antes aún de empezar el proyecto fin de carrera. Maria José, J, Lailo, Álvaro, Miguel, Tito o Germán son algunos de ellos. Todos dejaron el grupo hace mucho tiempo. Otros se quedaron un poco más, Vicent, Jaime, Barce, Emilio o Fernando no estaban interesados en recorrer el camino o abandonaron. Pero sin duda fueron una fuente de aprendizaje en muchos aspectos, tanto técnico como humano.

El grupo ha ido creciendo y menguando, y muchos han sido los compañeros que he tenido el placer de conocer y disfrutar. Sin duda, el buen ambiente del grupo ha endulzado los sinsabores de tantos años de esfuerzos. La bondad de Iván Rojo, el furor de Rafa, el optimismo de Marina, la pasión de Álvaro Martín, la templanza de Carletes, el carácter de Isabel, la fuerza de Gabo, el dinamismo de Diego Moreno, el compañerismo de David Prieto, la despreocupación de Alberto, la destreza futbolística de Ivancete, la alegría de Irena, el pragmatismo de Pedro, la nobleza de Alvaro Alonso, el genio musical de Abel, la amabilidad de Antonio Mendo, el ingenio de Néstor, la frescura de Juancar, el humor de Víctor Hugo, la labia de Víctor, la serenidad de Adriana.

Un recuerdo especial merecen los compañeros realmente involucrados en sus tesis, quienes nos hemos arropado mutuamente y empujado constantemente en el arduo camino lleno de altibajos, celebrando el éxito de cada publicación y compartiendo los sinsabores de un entorno singularmente difícil en el que llevar a cabo nuestros objetivos. De esta etapa, Sandra fue la primera que consiguió cumplir sus sueños y regresar a su tierra como una señora doctora. Javi es ya un señor doctor mientras escribo estas líneas, gracias a su gran capacidad de trabajo y su gran optimismo. Dani va más que sobrado en sus objetivos, en su determinación y en la gran hazaña de sacar adelante una empresa y una tesis, al igual que Kike, que ha añadido a sus notables capacidades de gestor el último empujón que le faltaba. Aldo y Álvaro lo tendrán mucho más fácil, aunque capacidades no les faltan en absoluto para andar este camino del doctorado.

Un agradecimiento muy especial y cariño merecen los compañeros que han trabajado conmigo en Social Stream y que han contribuído de manera significativa a llevar a cabo un

proyecto tan ambicioso. Diego Carrera fue el primer compañero de trabajo intenso, no carente de discusiones calientes y discrepancias, aunque con un gran entendimiento mutuo que hizo que la primera versión de Social Stream viera la luz. Esa versión no hubiera sido posible sin el trabajo de Eduardo y Alicia, a quienes tuve el gusto y el privilegio de tutorizar oficialmente, disfrutando de sus grandes personas. Aldo construyó un módulo entero de Social Stream, su espíritu crítico es admirable y le abrirá muchas puertas. Rafa y Raquel aportaron su amplia experiencia y versatilidad digna de las bellas personas que son. Víctor Sánchez pasó fugazmente, a la velocidad que le es propia intelectualmente, para dedicarse a sus propios proyectos en los cuales encontrará el éxito sin lugar a dudas. Carolina supo dominar Social Stream y construir su propio proyecto haciendo alarde de su gran capacidad tanto técnica como humana. Jaime, Kike y Hao también pusieron su grano de arena para hacer Social Stream más grande. Y por último, y no menos importante, Crispín trabajó muy duro para darle la apariencia a Social Stream 2.0, aprendiendo rápidamente un montón de tecnologías y dando el toque fino como sólo él sabe hacer, con la dedicación y el mérito de trabajar en unas condiciones totalmente adversas. Un placer y un privilegio trabajar con vosotros.

Especial reconocimiento merecen los revisores de la tesis, Chema y Antonio Fumero quienes hicieron el esfuerzo de leersela y revisarla dándome la tranquilidad de haber sido revisada por otros compañeros. El propio Antonio, Víctor y Joaquín asistieron a los ensayos de la defensa, aportando los conocimientos de amplia valía como oradores.

Este trabajo no pudo haber sido desarrollado sin la dirección de Juan Quemada, que ha sabido formar y mantener uno de los grupos más grandes de la universidad y conseguir los recursos necesarios en múltiples proyectos de investigación, tanto nacionales como europeos. Joaquín Salvachúa supo despertar en su doctorando una gran capacidad de autonomía y una amplia visión que indudablemente a contribuído a la basta extensión de esta tesis. Profesores del departamento han sido ejemplos en otros ámbitos, como la rigurosidad de Gabriel Huecas o la humildad de Santiago Pavón.

He tenido la suerte de disfrutar de un gran personal administrativo que ha facilitado enormemente los tediosos e interminables trámites burocráticos. La cercanía y competencia de Angelines fue sustituido por la gran predisposición de Alicia, que ha hecho un gran esfuerzo por ponerse al día en este laberinto de papeles. Lola siempre ha estado disponible para cualquier trámite y hacer malabares para sacar las cuentas del grupo adelante. Este trabajo no podría haberse llevado a cabo sin el apoyo del personal del Céntro de Cálculo. Gabi, Judith y Omar siempre han estado dispuestos a echar un cable cuando se les necesitaba.

I keep an special affect from my stay in Belgium, where I could contrast how other reseach groups work in Europe and meet great people both personally and professionally. Starting with Seda Gürses, who opened the doors of the stay to me and with whom I have worked in other areas, or Dave Clarke, who welcomed me into his research group and was a professional example. Collegues as Leandro, Rula and Rafael took me both professionally and personally. I had the

privilege of meeting the great family made up Koosha, Dimitrios, Theo, Francesca, Carlo, Vittoria and many others, because man does not live by work alone. Natalia and Josh made my landing a gentle and sweet experience and I opened the doors of their world. Without any doubt, I would have stayed long in Leuven.

El Centro Cultural La Piluka ha sido un espacio de formación y experimentación constante, acogiendo los desarrollos de una manera objetiva y pragmática, y constituyendo una gran fuente de inspiración. Nombrar a todas las personas de esta gran familia requiría un apartado de agradecimientos aparte.

Este trabajo, o cualquier otro, no podía haberse llevado a cabo sin el soporte de mi familia, la cual supone un refugio permanente, sólido y seguro. Mis padres, Antonio y Angelines, que han soportado los múltiples bandazos vitales, siempre respetando mis decisiones, y han estado siempre dispuestos para lo que hiciera falta en una dedicación sin igual. Mi hermano David, el cual ha sido un compañero fiel tanto en los múltiples ámbitos de nuestras ajetreadas vidas, como confidente personal ante cualquier experiencia. Y Alicia, mi isla de cielo con la cual he compartido no solo amplias discusiones al adentrarme en un campo, el de la sociología, en el que siempre ha estado dispuesta a instruirme. He tenido el privilegio de disfrutar de una intimidad y un apoyo sin igual. Os quiero.

Contents

Illustrations

Tables

Acronyms

API Application Programming Interface

CAS Central Authentication Service

CGI Common Gateway Interface

CMS Content Management System

CMF Content Management Framework

IETF Internet Engineering Task Force

IP Identity Provider

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

LID Light-Weight Identity

MVC Model View Controller

PC Personal Computer

PoCo Portable Contacts

PuSH PubSubHubBub

RDF Resource Description Framework

REST Representational State Transfer

RP Relay Party

RSS Really Simple Syndication

SNA Social Network Analysis

SNF Social Network Framework

SNS Social Network Site

SWAT Social Web Acid Test

TDD Test Driven Development

URI Uniform Resource Identifier

URL Uniform Resource Locator

UI User interface

UX User experience

VCC Virtual Conference Center

ViSH Virtual Science Hub

WWW World Wide Web

W3C World Wide Web Consortium

XML eXtensible Markup Language

XRDS eXtensible Resource Descriptor Sequence

XRI Extensible Resource Identifiers

YAML YAML Ain't Markup Language

Chapter 1

Introduction

> *If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas.*
>
> *George Bernard Shaw*

1.1 Motivation

The main motivation that has driven all these years of hard work has been to build software that enhances human relations in all contexts, from local associations and neighbourhood cultural centres to workplaces or companies.

This aim was contextualized by the Web 2.0 realm, which led to a shift in the perception of the World Wide Web. Right before the web bubble burst, media companies were betting on web portals where their users would find sufficient and appropriate content. Afterwards, a new wave of technology empowered users and enabled them to participate in a more horizontal way. The barriers separating content creators from consumers became blurred and a new landscape that overcame the limitations drawn by the mass media establishment could be devised. This was without doubt a promising social change as well as something to work on and see where it led.

The first objective of this work was to create a framework that would easily provide this kind of content management technology to web developers and make more available the capability of users to supply content in the web.

After the content management shift, there was another one to come: social networks. The findings of Barabási in network science [Barabási 2003] showed a fascinating set of rules that managed to explain human relations. Pareto distributions became a law for certain aspects of them. Social network sites (SNS), such as Twitter and Facebook, became popular, as well as other content-oriented networks like Github. Social networks became a powerful grounding on the basis to build telecommunication tools. There seem to be uncountable niches in human activities that could be enhanced by social networks tools, neighbourhood associations, clubs dedicated to all the hobbies we can imagine, small, medium size enterprises, or big companies. This is where a framework that builds these kinds of social network websites, easily and rapidly, becomes a key.

However, we can take it a step further. Current social network platforms are isolated, they are data silos. They leveraged their appeal based on their number of users. Metcalfe's law states that the value of a telecommunications network is proportional to the square of the number of

connected users of the system (n^2) [Carl Shapiro 1999]. The ideal scenario showed social sites communicating with one another. An example of this is the academia ecosystem, where research institutions would manage their own social sites in order to enhance the collaboration between their members. These institutions include universities, research institutes and agencies. They would be the primary SNS providers for their researchers. However, they would like to federate so that researchers can follow the work of their partners across their institutions. Researchers may change work between institutions or perform stays in other institutions, so they would want to have their profile added. In addition, there is an ecosystem of services that could benefict, including conferences, journals or online learning platforms. This distributed scenario could bring advantages for all of them.

This scenario can also be applied to other fields in human society, regarding every hobby and every organisation's site. I believe such a shift could bring a change to human relations as they are known nowadays.

1.2 Research methodology

The work carried out in iCamp[i], a research project within the European Union Seventh Framework Programme for Research and Technology Development, was crucial to identify these scenarios. In particular, the involvement of the author in the Work Package (WP) 3, that studied interoperability issues between different eLearning platforms across Europe. iCamp was able to describe the problem of different institutions that cooperate together. Three different scenarios were identified: institutional, personal and augmented. The institutional scenario showed different institutions providing educational tools and interoperability between them. The personal scenario showed a similar setting with personal tools. Finally, in the augmented scenario (figure 1.1), tools provided by a learning institution interoperated with the personal tools used by individuals in their own scenarios.

At the same time, the participation of the author in social movements and local cultural centres showed a very similar scenario in the field of collaboration between citizens, where different groups of people, who are scattered throughout the city get to meet, work in similar topics and cooperate with each other.

The arrival of the GLOBAL project was an opportunity to test the ideas regarding code reuse. The Station middleware was designed and built as a kind of plug-in, called *engine*, for the popular Model-View-Controller web development framework Ruby on Rails. It provided a generic component for content management and it met requirements in different scenarios, such as the Virtual Conference Center (VCC) developed in the GLOBAL project, but also organisational issues in the above mentioned scenarios.

[i] `http://www.icamp.eu/`

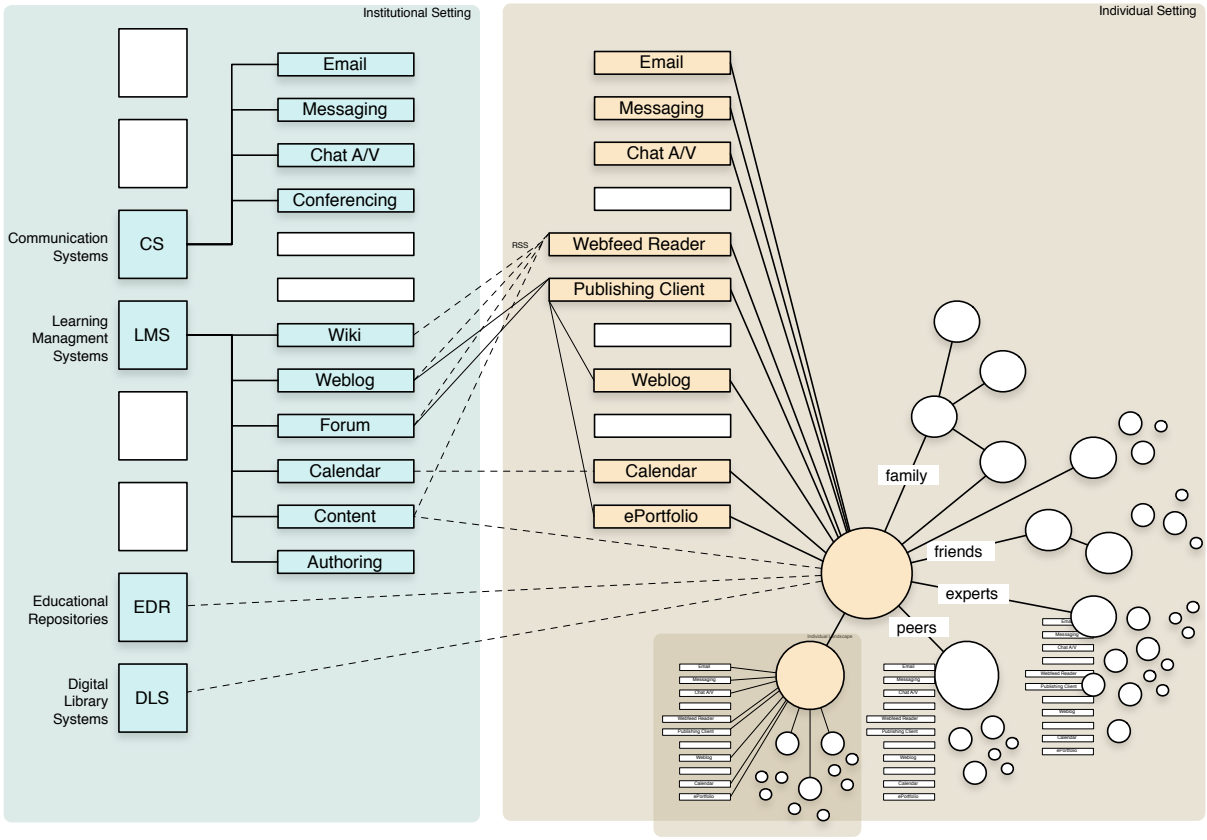

Figure 1.1 : Augmented landscapes defined in the iCamp project

The development of Station coincided with a debate about the utility and viability of Rails engines, into the core of Ruby on Rails. This was an opportunity for the author to contribute to the Rails framework itself, adding internacionalization support to Rails engines.

Although the VCC, the outcome of the GLOBAL project, was successfully built, Station did not gain the community it was intended to. A lesson was learned here as smaller engines eventually provided the functionality Station included and they were the ones that were broadly adopted by Rails developers.

After these events took place, there was a shift towards social networking, along with the seminars on social networks science organized by Orange Catedra in the UPM, which showed the importance of research on social networks. The work on Social Stream began, in collaboration with Diego Carrera. The work included defining the architecture for supporting the features of SNS which were later formalized in a study about the most popular SNS platforms.

The Global Excursions project brought the opportunity to use Social Stream for building an excursions-oriented site, while it was already being tested in a local cultural centre, *"La Piluka"* to gather user experience and improve the framework from the user's point of view. The outcome of the Global Excursions project was the Virtual Science Hub (ViSH), which was developed quite rapidly. The reuse measures showed that Social Stream became a valid framework for building social network websites. This was also confirmed by external developers all around the world that started to use Social Stream in their own projects.

Along with the definition of Social Stream's architecture, Tie-RBAC access control model was developed in order to transfer the advantage of the RBAC model, which was successfully used in Station, to the field of social networks. The work on access control model was reinforced by a foreign stay at the DistriNet group in KU Leuven and by participating in the SPION project, founded by the Belgium government, where the author had the opportunity to introduce his work and obtain valuable feedback from the participants.

Simultaneously, the work on distributed social networking was carried out and was mainly focused on tracking the state of the art, due to the inability to contribute until the framework was available. The attendance to the Federated Social Web Europe meeting in Berlin [i] was crucial, since the author had the opportunity to hold discussions with other projects and developers, as well as rising privacy problems in current solutions. The formal study of distributed social network had to take a different approach from the SNS case, where there are already well-established social network platforms to study. Consequently, the study of distributed features was centred in a review of different paradigms and the analysis of two particular cases of study. Distributed techniques were used in the ViSH with the ViSH Editor, in the context of the Global Excursion project, and the federation scenario in the context of the FI-Content project, which concluded the work towards building a distributed framework for social network websites.

[i] `http://d-cent.org/fsw2011/`

The amount of this work as a whole has provided the author a wide European scientific context and has established his work on an international basis, which is the main reason for the author to apply for the Doctor Europeus Mention.

1.3 Structure of this document

Following this chapter, which provides the motivation for the work and the research approach that has been followed, chapter 2 details the results of the analysis of the state of the art, gathering a review of the different fields that are relevant to this work, such as web applications, web services, formats and APIs, social software, online social networks, authentication and authorization, and software reuse.

Chapter 3 defines the objectives of the work under this thesis, where the main goal is to obtain a framework that allows to build social network websites. Chapter 4 ensures the viability of this framework in the case of content management, building a content management framework and assessing its validity in an application for a EU-founded project through code reuse metrics. Chapter 5 analyses the features present in social network websites, in order to obtain the requirements a social network framework should meet. Chapter 6 introduces Social Stream, a framework for building social network websites, along with its architecture and the sites that have proven its validity, as well as code reuse metrics. Chapter 7 introduces a new access control model for social networks based on Social Stream's architecture which overcomes some of the limitations found on the models reviewed in the state of the art. Chapter 8 reviews paradigms of distributed social network websites, as well as the cases of study of OpenID identifiers and popular SNS APIs. All of them will define the requirements for a distributed social network framework. The last chapter of contributions is chapter 9 which introduces Social Stream's extended architecture for building distributed network websites, as well as two EU-founded projects where this architecture has been validated.

Chapter 10 gathers a summary of the validations of all the contributions, which include Master's theses, some of them tutorised by the author of this dissertation in addition to several EU-founded research projects and dissemination results in international conferences and a book chapter.

Finally, chapter 11 details the main conclusions of this work, as well as a description of the most interesting future research activities.

Chapter 2

State of the Art

“If I have seen further it is by standing on ye sholders of Giants.”

Isaac Newton

2.1 Web applications

2.1.1 The World Wide Web (WWW)

Web applications are inherent to the development of the World Wide Web (WWW). It should not be neccesary to stress the importance that this technology has brought to humanity. The computer system conceived by Tim Berners-Lee [Berners-Lee 2000] has produced a deep impact in our society, at all levels: education, health, economy, politics, etc. And also in human relations through web-driven social software.

The WWW is a system of interliked hypertext documents accessed via the internet, which is other invention so important and crucial in last years, without which the WWW could not have been developed. The WWW is made up of three main technologies, which sit on the top of the internet stack, at the application level [Fielding 1999] (figure 2.1)

- a system for unique identifiers or Uniform Resource Locator (URL)[Berners-Lee 1994], later extended to non-dereferenciable resources by Uniform Resource Identifier (URI) [Berners-Lee 2005]
- a markup language for text documents; the Hypertext Markup Language (HTML) [Jacobs 1999a]
- an application level protocol for the interchange of resources between computers, the Hypertext Transfer Protocol (HTTP) [Fielding 1999]

The architecture follows a client-server design [Berners-Lee 1990]. It is composed of a client program, the **browser**, that runs in the client machine and displays the hypertext to the user. The browser is also resposible of transversing the hypertext links, remembering the browser history, while the client just responds to client requests. Finally, the browser is responsible of the negociation of resource formats, in dialog with the server. On the other hand, the **server** is an application running in the server computer. It is responsible of managing a web of nodes in the machine and negotiating content format with the client, providing the requested resource.

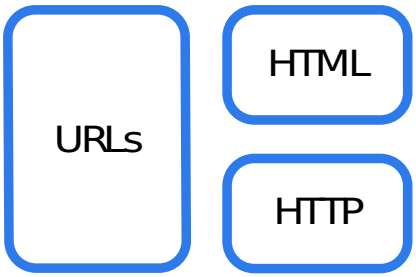

Figure 2.1 : Main web technologies

2.1.2 Server-side web programming

In the very begining of the web, HTML pages were static. The person or group that wanted to publish information had to edit the HTML files by themselves using text editors and upload the new files to web servers. However, dynamic content generation appeared promptly with the invention of common gateway interfarces. Later, more powerful scripting languages were designed. The last step in server-side web programing has been settled by model-viewer-controller frameworks.

Common Gateway Interfaces

Common Gateway Interface (CGI) [Robinson 2004] is a standard method for web servers to delegate content generation to scripts. These are executable files, often stored in a `cgi-bin/` directory. Scripts interpret the URL, including parameters, etc., run and return the data with the HTTP headers. CGIs are usually scripts, so they can be ported and executed in multiple platforms.

Scripting languages

Server-side programing brought the birth of specific web server languages. The paradigmatic case is PHP, one of the most popular language in web servers nowadays. Several top web sites have been built using PHP, such as Facebook or Wikipedia. Other languages has been used to program server-side websites and applications. These come from the more classic backgrounds, such as Java, or even C, to more modern ones such as Ruby or Python. Ruby is an object-oriented, interpreted programming language [David Flanagan 2008], created by the japanese programmer Yukihiro "Matz" Matsumoto, which is the base of the popular MVC framework Ruby on Rails.

Model-View-Controller frameworks

Beyond server-side scripting, the last tendencies in web developing show developers using frameworks, usually with model-view-controller (MVC) patterns. These patterns provide multiple benefits [Johnson 2005], including complexity hiding, giving structure and consistence and promoting best practices. They also provide frameworks for test driven development (TDD) [Astels 2003], which has become a standard for giving quality to web platform code.

MVC frameworks support role-separation between backend-, database-oriented developers and frontend-, design-oriented ones. Both of them have their work scopes separated and can focus on their field of expertise.

Ruby on Rails in an example of a popular web development framework. It is developed to increase productivity. It implements MVC architecture, and relies on "convention over configuration" and "don't repeat yourself" [Bachle 2007]. Other MVC have appeared for other languages, such as Django for Python, Springs for Java or Symphony for PHP. Existing MVC frameworks are reviewed in [Ignacio Fernández-Villamor 2008]

2.1.3 The WWW as application platform

Web applications are gaining popularity along with the web. The WWW is becoming the main platform for applications, continously replacing desktop applications. These became popular in the 80s with the appearance of the personal computer (PC) and window user interfaces (UI).

The *web application* concept was introduced in the Java language in the Servlet Specification version 2.2 [i]. Web applications are computer applications access through the WWW. They are based on the browser as thin client, leveraging the deployment of them in several platforms. They do not require other software to be installed besides the web browser, which is currently present in every personal computer and smartphone. Other advantage is that it can be deployed once, in the server side. Users do not have to upgrade to receive new versions.

Web applications reside in a web server. They usually have a code that is executed remotely. They also have a persistence layer provided by the filesystem and a database. They render HTML, CSS and Javascript to the client. The Javascript code is executed in the browser and provides Rich Internet Applications (RIA). These are enchanced applications that provide better user experience, enabling sophisticated user interactions, client-side processing, asynchronous communications and multimedia [Fraternali 2010].

2.2 Web services and APIs

Web applications implement webservices for other applications to use them. A web service is defined by the W3C as "*a software system designed to support interoperable machine-to-machine*

[i] `http://www.jguru.com/faq/view.jsp?EID=129328`

interaction over a network" [Haas 2004]. Web services describe a Web application programming interface (API), the interface that systems may support in order to interoperate. The basic HTML over HTTP can be seen as a webservice for the browser application.

Web service technologies suffered an architectural war between REST and SOAP architectural styles [Benslimane 2008], in other words, resource-oriented (ROA) versus service-oriented (SOA) architectures [Guo 2010].

Web APIs have been used in last years to create mashups, a composition of services that facilitate the design and development of modern Web applications [Benslimane 2008].

2.2.1 Representational State Transfer (REST)

Web applications exchange resources. The resource request and transfer follows an architecture principle called Representational State Transfer (REST), developed by Roy Fielding as part of his doctoral thesis disertation, who was active in the principles of the WWW design.

REST is a style of software architecture for distributed systems, such as the WWW. REST principles are addressability, stateleness and representations [Fielding 2000, Richardson 2007]. Because REST is an architectural style, it is not tied to any particular protocol, even HTTP.

REST set are the foundations for web APIs designing in last years. They have become the preferred architectural style versus XML-RPC and SOAP. A comparison between them is addressed at [Richardson 2007]

2.2.2 Web formats

Web formats describe how resources are described and trasferred in REST. Besides HTML, the original format in the Web, many other formats have been described and used these days.

Hypertext Transfer Markup Language (HTML)

Hypertext Transfer Markup Language (HTML) is the markup language used by most of the web pages in the WWW. An HTML is a text field made up of HTML tags, such as `<html>`. Tags can be nested between them. Each HTML tag is equivalent to an element, and the anidation of elements builds the element tree. For instance, each HTML document is composed by a `<head>` part and a `<body>` part, each one declared with its tags. Listing 2.1 shows an example of HTML code.

One of the relevant parts for this work is the `<link>` header, present in the `<head>` part of the HTML document. The `<link>` tag is defined in the HTML specification [Jacobs 1999b] as a way to define a relationship between current resource and an external resource. The list of link relation types is maintained by IANA [Registry 2011], however web developers may use custom types.

Listing 2.1: Example of HTML

```
<html>
  <head>
    <title>A framework for building distributed social network websites</
        title>
    <link type="alternate" href="http://dit.upm.es/atapiador/thesis">
  </head>
  <body>
  </body>
</entry>
```

Microformats are a set of simple, data formats that are the result of describing other familiar formats using HTML elements [Allsopp 2007]. They are semi-structured information embedded in the HTML markup. Examples of microformats include personal cards (*hCard*), which follows the vCard specification [Dawson 1998a], events (*hCalendar*) following the iCalendar specification [Dawson 1998b] and tags (*rel-tag*). Other object types are in the definition process.

eXtensible Markup Language (XML)

eXtensible Markup Language (XML) is a textual data format designed by the W3C to be both machine-readable and human-readable [Bray 2008]. It defines the rules to encode documents with strong support for every language in the world through Unicode. It supports the use of different schemas at the same time, incentivating the distributed definition of the data structure. XML documents include syndication formats, such as RSS and Atom, office documents, such as Microsoft's Office Open XML and Libreoffice.org's OpenDocument, and even communication protocol messages, such as in XMPP [Saint-Andre 2011].

Really Simple Syndication (RSS) is a family of document formats used to subscribe to updates in a webpage, such as blog entries, news, audio and video. RSS has had different versions. It started using RDF, but last versions use XML as serialization format [RSS 2009].

Atom syndication format is a stardard for web syndication [Nottingham 2005]. Atom was developed in the IETF in an open way, to overcome RSS limitations and flaws [Wild 2007]. Atom has become a popular format rivalizing with RSS. Due to the modular design based on XML capabilities, it has been extended to support threads [Snell 2006], copyright licenses [Snell 2007] and feed paging and archiving [Nottingham 2007]. It was the first serialization format for Activity Streams, described below.

YAML Ain't Markup Language (YAML)

YAML is other human-redable serialization format [i]. It takes concepts from several programing languages and XML. YAML is easier to write than XML, so it is becomming more popular in the last days, specially its subset JSON.

JSON format [ii] is a subset of YAML. JSON has become very popular as it is a native format in Javascript. Browsers can easily parse and generate it without additional add-ons. There are parsers for most of the programing languages as well, and it is the preferred format in web APIs nowdays, as will be shown in this document.

Activity Streams

Activity Streams [iii] is an specification for the serialization of social activities. Its preferred format for serialization is JSON (listing 2.2). Besides, it exist an specification for serializing activity streams in Atom (listing 2.3).

Listing 2.2: Example of Activity Streams in JSON

```
{
  "published": "2012-12-10T15:04:55Z",
  "actor": {
    "url": "http://dit.upm.es/atapiador",
    "objectType" : "person",
    "id": "tag:dit.upm.es,2002:atapiador",
    "image": {
      "url": "http://dit.upm.es/atapiador/avatar",
      "width": 250,
      "height": 250
    },
    "displayName": "Antonio Tapiador"
  },
  "verb": "post",
  "object" : {
    "url": "http://dit.upm.es/atapiador/thesis",
    "id": "tag:dit.upm.es,2012:thesis",
    "objectType": "article",
  }
}
```

[i] http://www.yaml.org/

[ii] http://www.json.org/

[iii] http://activitystrea.ms/

Listing 2.3: Example of Activity Streams in Atom

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://
    activitystrea.ms/spec/1.0/">
  <id>tag:dit.upm.es,2012:thesis</id>
  <title>A framework for building distributed social network websites</
      title>
  <published>2012-12-10T15:04:55Z</published>
  <link rel="alternate" type="text/html" href="http://dit.upm.es/atapiador/
      thesis" />
  <activity:object-type>article</activity:object-type>
</entry>
```

Activity Streams describe social activities made by users or other agents in a web site. Each activity is composed by the publication date, a description of the actor that made the activity, the activity verb (e.g. *post*, *like*, *follow*) and the object of the activity. The target of the activity may also be included. Further extensions have been defined in order to include audience targeting[i] and responses in activity streams[ii].

Resource Description Framework (RDF)

RDF is an standard to exchange data in the Web [RDF 2004]. It allows structured and semi-structured data to be exposed, mixed and share accross web applications. RDF documents are set of statements or triples, in the form of subject-predicate-object expressions. RDF triples can be serialized in different formats, including XML. RDF is the foundation of the Semantic Web.

An ontology represents a set of contents from an specific domain of knowledge, besides the relationships among these concepts. RDFS and OWL show how ontology languages can be built upon RDF.

RDFa is a set of extensions for embedding RDF metadata in XHTML documents [Birbeck 2008]. It is an attempt to bridge the gap between the web for humans and the web for computers.

Friend Of A Friend (FOAF) is an ontology describing persons and their relationships built on the top of RDF and OWL [FOA 2010]. People are able to export their social data in a distributed way, so a machine can collect the data and use FOAF profiles to find the list of all people two friends know, for example.

[i] http://activitystrea.ms/specs/json/targeting/1.0/

[ii] http://activitystrea.ms/specs/json/replies/1.0/

eXtensible Resource Descriptor Sequence (XRDS)

XRDS [XRD 2008] is an XML format for discovery of services associated with a certain resource. It was originaly developed by the OASIS consortium as a format for the resolution of Extensible Resource Identifiers (XRI). Besides XRI resolution, XRDS is used for the discovery of OpenID and OAuth.

2.2.3 Web protocols

Although HTTP is the standard protocol for the WWW, other protocols have been defined on the top of it. These protocols focus on an specific task, such as content publishing or identifiers discovery. These section introduces them.

HTTP

The Hypertext Transfer Protocol (HTTP) is the application protocol for distributed information systems building the WWW [Fielding 1999]. It was developed by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

HTTP is a client-server protocol that works on the top of TCP. It is composed by requests followed by responses. It is founded on the REST architectural principles, transfering representation of resources, primary in HTML, but any other format can be managed as well.

Listing 2.4 shows an example of HTTP exchange.

Listing 2.4: Example of HTTP request and response

```
GET /atapiador/thesis HTTP/1.1
User-Agent: curl/7.26.0
Host: dit.upm.es
Accept: */*

HTTP/1.1 302 Found
Location: http://irss.dit.upm.es/users/atd/documents/35
Connection: close
```

HTTP exposes several verbs [Fielding 1999, Dusseault 2010] in order to perform different operations on resources; such as `GET` for reading, `POST` for adding resources, `PUT` for replacing a given resource, `PATCH` for changing an existing resource or `DELETE` for removing resources.

Atom Publishing Protocol (AtomPub)

The Atom Publishing Protocol (AtomPub) [Gregorio 2007] is an HTTP-based protocol for creating and updating resources. It uses the Atom format as resource representation. It provides support for collections, which are represented as Atom feeds, services for collections discovery and the necessary steps for creating, editing and deleting resources.

Yadis

Yadis [Yad 2006] is an specification for discovering services from URIs. It was developed in the beginnings of OpenID. In fact, it was the name of the first OpenID specification. Later, it was adopted by OASIS as part of the specification of XRI Resolution for discovering XRDS documents from URIs.

OpenSocial

OpenSocial [Häsel 2011] is a standard promoted by Google, Yahoo!, MySpace and others. It comprises a set of common APIs oriented primary to social network applications. The main advantage is that developers need to understand the API only once, then they are able to develop applications that work in any site compatible with OpenSocial, and general components working with OpenSocial can be reused across sites.

OpenSocial has been changing to embrace other technologies such as OAuth 2.0, Activity Streams and PoCo. It is based on well-known technologies such as HTML, XML, JSON and Javascript.

OStatus

OStatus [E. Prodromou 2010] is an open standard enabling users in different social network platforms to follow each other. It uses several protocols for the different stages of federation, including **Webfinger** for discovering users from their IDs, **Portable Contacts** for describing user's social network, **PubSubHubBub (PuSH)** for suscribing to users' activity and **Salmon** for notification updates.

OStatus was designed with public feeds in mind, and it is not suitable for settings where privacy restrictions are needed.

Webfinger is a protocol for discovering information from an email-like URI[i], such as `user@example.net`. It is inspired in the old UNIX finger service, which allowed to get information about a user in any computer that ran the service.

The protocol is based on the Web Host Metada specification [Hammer-Lahav 2011], a method for retrieving well-known information on a web site, such as PuSH hubs, authors or copyright licenses. Webfinger resources are described as URI templates [Gregorio 2012] with a link relation value of `lrdd`. The template is resolved using the webfinger URI, and dereferencing the resulting URL provides the document describing the webfinger account. Through the initial specification proposes XRD as the default format, latests discussions move towards using JSON as default format.

[i] `http://code.google.com/p/webfinger/`

Webfinger is used in OStatus as a way to discover remote users in distributed social network sites.

Portable Contacts (PoCo) Portable Contacts (PoCo) [i] provide a way to access user's address books and friend lists. They define a common access pattern and contact schema for retrieving both the social network and contact details.

PoCo is used in OStatus to retrieve user's contact list.

PubSubHubBub (PuSH) is a protocol for subscribing to web feeds[ii]. It tries to solve the inefficient pattern associated with web syndication. Web feeds consumers that are subscribed to web feeds need to periodically request the feed to check if there are new updates in it. PuSH enables these consumers to subscribe to updates providing a public URL to the PuSH hub. When the website has a new update, it pings the hub server, which fetches the new feed and multicasts all the subscribers with the updates. Subscribers can discover the PuSH hub address as a link entry in the feed.

In the context of the OStatus suite, social networks sites use PuSH to subscribe to the activities feed of remote users on behalf of their local users. This way, remote users can be followed. The activities feed is published within the webfinger information of a user. This activities feed must have a link to its PuSH hub.

Salmon is a message exchange protocol running on the top of HTTP[iii]. Its goal is to decentralize comments and annotations notifications made against Atom feed articles, such as blog posts. It uses the Atom format along with its threading extensions to describe the notification content, its author and related information. Salmon notifications are base64 encoded and signed to assert the authenticity of the origin.

OStatus makes use of Salmon notifications for enabled distributed support of activity follow-ups and user's follow notifications.

2.3 Social software

Social software in a broad sense, is software that people use to interact with other people [Anderson 2005]. Some authors restrict social software to leisure activities. However, we use it to name the software that supports every kind of social interaction.

According to Stewart Butterfield, "*Social software ... is software that people use to interact with other people, employing some combination of the following five devices: identity, presence, relationships, conversations and groups*". [Lihua 2010]

[i] `http://portablecontacts.net/`

[ii] `https://code.google.com/p/pubsubhubbub/`

[iii] `http://www.salmon-protocol.org/`

2.3.1 Computer Supported Collaborative Work (CSCW)

Computer Supported Collaborative Work (CSCW) is a discipline which aim is to support requirements of cooperative work arrangements [Schmidt 1992]. It studies how work collaborative activities and their coordination can be supported by means of computer systems.

Since its beginnings in the 1980s, Computer Supported Collaborative Work (CSCW) has been focused on formal organisations. CSCW emerged as a multidisciplinary field, involving technologists, economists, sociologists, psychologists, anthropologists, organisational theorists, educators, and anyone else who could shed light on group activity. [Grudin 1994]. It goes beyond building tools to support these activites, and studies how this tools impact on work performance and organisation dynamics.

CSCW emerged at the time of corporate and academic networks, and the collaboration supported by computers only made sense in the framework of big organisations. Later on, network connections and PCs become more popular. CSCW field grew covering more organisational aspects, ranging from the individual to organisation [Grudin 1994]. Many groupware solutions have been developed for supporting these organisation-centered scenarios. Nevertheless, they were far away from being completely successful. Some authors show groupware and intranets drawbacks [Stenmark 2002], which include: difficulties finding relevant information, lack of coherent design and structure, inconsistent vocabulary, unclear ownership, becoming one-way communications channels for corporate information instead of sharing knowledge tools on a peer-to-peer level.

Groupware

Wilson makes a distinction between CSCW and groupware. On the one hand, CSCW refers to the generic term, which involves understanding how people work in groups with the enabling technologies. On the other hand, groupware addresses the set of tools of computer networking, and associated harware, software services and techniques that enable that support. [Wilson 1991]

Examples of groupware tools include electronic calendars, project management systems, online spreadsheets, instant messaging, telephony, videoconferencing, e-mail, wikis, content management systems and social network sites. There are several attempts to classify these tools, such as according to the synchronous or asynchronous space and time [Baecker 1995] or the level of collaboration (communication, conferencing and co-ordination) [Corporation 1995].

2.3.2 Content management

"*Content management is the process behind matching what your organisation has to what your audience wants*" [Boiko 2001]. It comprises collection, management and publishing content to any outlet. Traditional content management systems support several outlets, from printed to electronic media.

Boiko [Boiko 2004] establishes differences between *data*, *information* and *content*. On one hand:

> *Data consists on small snippets of computer information - numbers, words, images, sounds - that have much of the human meaning queezed out of them.*

Compared to content, data becomes mechanical, uninteresting and hard for consumers to understand. These is because data alone lacks from context and human interpretation. On the other hand:

> *Information is what human beings transform their knowdledge into when they want to communicate it to other people. It is knowledge made visible or audible, in written or printed words or in speech.*

The difference between information and data is that information is contextualized in someone's mind, who provides with context and significance.

Finally, raw information becomes content when it is useful. Information must be given an usable form for one or more purposes. The more usable a piece of information is, the more valuable content it is.

> *Content, is information that you tag with data so that a computer can organise and systematize its collection, management, and publishing.*

A content management system should help their users to provide their information with that interest and meaning along the way.

Web Content Management Systems

Web content management is the result of delivering content to the web. Web content management became popular with the growth of the web as media [McKeever 2003]. Publishers were first interested in delivering content to the web as one of the multiple media. Afterwards, specialized web content management systems supporting only web publishing appeared.

McKeever [McKeever 2003] and Liduo [Liduo 2010] provide a list of features in the field of content management systems (CMS). They are classified according to the lifecycle or process of web content management; content collection, workflow, content delivery and control or administration.

Content collection Content collection includes content creation or introducing content to the system from an existing source. It may involve diffent people inside the organisation, each one with a distinct role in the process, such as editor or reviewer.

- Standard tools for content creation, with standard user interfaces. These interfaces should be easy enough for non-technical users.
- Multi-user support & authorship. Multiple users should access to the system at the same time from different locations.
- Separation of content from presentation, with a centralized definition of the look and feel, so editors do not need to pay too much attention to presentation and can focus on content.
- Support for content syndication, users should be able to suscribe to feeds.
- Content preview capability
- Content versioning for authors, they should be able to tag different content versions, review changes between versions and roll back to previous ones.
- Relevant content types; images, articles, or other type of formats.
- Form support for catalogue type data
- Localization and multiple languages
- Shared database for content storage
- Thin client for easy rollout
- Real time access to content management functions

Workflow Workflow enables the organisation of edition and aproval process on content, from development to publication. It may have a single threaded planification or may allow seveal tasks in paralel. Besides, informal and strict workflows may supported, with loose and tightly defined roles.

- Flexible, multi-threaded workflow. It should support several simultaneous tasks, while concurrency issues are solved.
- Workflow monitoring and control features, for the supervision of workflow stages.
- Support for workgroups

Content delivery Also called, publishing, deployment or distribution, content delivery is the act of making the content available to web users. It involves extracting relevant information from content repository and building web pages.

- Support for both static content and dynamic content

- Automatic web page link checking, so there are not dead links because of removed or changed content.
- Data error checking, such as validations in produce price values which shold be positive.
- Separate environments for development and production, even test in the last times.
- Content version rollback to previous versions.
- Multi-channel support for multiple user devices.
- Scheduling of automatic site changes
- Content personalization, allowing users to customize it.

Control and administration Finally, some features should support the supervision of the system to assert security in processes.

- Predefined role definition or the ability to customize roles, managing permissions among different functions in the system. These include content editing, auditing and publishing process.
- Support for the definition of information architecture through taxonomies.
- Audit trail that supports the recording of content related activities
- Reporting functions and metrics, such as site visitors, their activity and deployment history.

Content management frameworks (CMF)

Mooney and Baenziger describe a content management framework (CMF) as low level solution consisting on an extended programming language and an application programing interface (API) supporting content management features for website development. CMFs can provide efficient development of any CMS feature [Mooney 2008] However, they do not make a difference between web application frameworks and CMF. Zope, the popular application written in Python, is one of the examples with CMF support [Thiruvathukal 2004], however it is not implemented using the modern web development framework paradigm.

Content management and MVC frameworks

Given the popularity that MVC frameworks have gained between web developers, it would be interesting studiying how these kind of applications can be supported in MVC frameworks. However, there is almost no work in literature that studies the gap between MVC frameworks and content management. There is recent work that introduces the implementation of a web content

management system using J2EE MVC technologies [Liduo 2010]. It presents a successful case using a 3-tier (MVC) architecture and collects the requirements for web content management. However, it does not explain how content management features are supported by a MVC framework. There is not literature covering CMFs and their support in MVC frameworks.

2.3.3 Web 2.0

At the beginings of the 2000s, there was huge hype about the next WWW generation, the Web 2.0. The term "Web 2.0" is a controversial one. There isn't full consensus about what the "Web 2.0" is. We will adopt here the Wikipedia's definition[i], as the Wikipedia is one of its paradigmatic products:

> *Web 2.0 generally refers to a second generation of services available on the World Wide Web that let people collaborate, and share information on line. In contrast to the first generation, Web 2.0 gives users an experience closer to desktop applications than the traditional static Web pages.*

The qualitative leap from Web 1.0 is the ability of users to *easily change remote content*. The implications of this new ability are crucial, since communication channels become bidirectional. Everybody has the potential to participate on *"anything"* on the web.

Some papers try to collect this new framework characteristics[Valdes 2005, Mcafee 2006]. There are four main features that characterize Web 2.0. The first two of them coming from the original Web:

- **Decentralization**. As a consequence of being a real network. Every node and user has the ability to act as emitter and receptor of information.
- **Openness**. Using standards in communication, free licenses on content, promotes collaboration.
- **Dynamism**. Applications are developed and deployed quickly. User suggestions are attended and supported.
- **User orientation**. Easier and better user interfaces facilitates participation.

These four features just match with social software application problems, raised by Eric Gradman in [Gradman 2004]. As they are issues don't fully solved yet, they keep applications for being "Web 2.0 compliant".

There is a significant shift in the Web 2.0 that comes along with significant changes in the way organisations work. Groupware tools were designed for *tree organisations*. But in the last

[i] `http://en.wikipedia.org/wiki/Web_2.0`

years, the organisation environment has become dynamic, organisations themselves need to adapt to changes. Organic structures emerge, and the networks science approaches are gaining momentum within the management science, changing the way we understand the organisation itself. Hierarchies loss momentum and networked approaches for the modern organisation are considered [Cross 2005].

As organisations flatten, become more organic, collaboration patterns turn semilattices. In these new environments, centered communication becomes a key factor [Farmer 2005]. Users must be able to customize presence and operation to suit individual needs, represent themselves as unique individuals and select and control the medium and manner in which they access and participate in the environment. The 'new organisation' must be tackled as a series of virtual communities interacting for achieving a set of well-defined shared objectives and goals. Such a scenario change in the real world can be devised; In considering the species populating the digital economy landscape, we are shifting from "corporation man" to "networked (knowledge) professional" [Economist 2006]. Such a change directly impacts the traditional top-down view which usually has shaped the CSCW systems design and deployment.

Web 2.0 technologies

There is a set of technologies which were developed along with the emergence of the Web 2.0. These technologies supported the user participation and collaboration shift that characterizes this stage of the Web.

Blogs are one of the best examples on how the Web 2.0 raised user participation to the next step. Non-technical users found in blogs a powerful tool for expressing their thoughs and they became first class publishers to an audience as potentially wide as the whole world.

A blog is a periodical updated website that gathers text and articles by one or several authors. The more recent article appears the first. Blogs often allow commenting to their post, enabling a conversation between the author and her readers. Blogs possibilities have been explored in a wide range of fields, including education [Duffy 2006], business [Sprague 2007] and clinical practice [Boulos 2006].

Wikis are other paradigm in Web 2.0 collaboration. A wiki is a web site whose pages can be edited by its users through their web browser. In the same way that blogs, the key shift in wikis is that their authors do not require technical skills to collaboate on web content. Their potencial has also been widely explored [Boulos 2006, Duffy 2006]

The paradigmatic wiki-based web site is the Wikipedia, a collaborative, multilingual enclyclopedia that has become one of the most popular sites in the Web.

Folksonomies are another technical feature emerging from Web 2.0. Content management web sites allow their users attaching key works to the media they store. These words are called *tags*, and provide semantics to content. Tags compose *folksonomies*, a categorization system built by principles [Mathes 2004]. Opposite to ontologies, this type of categorization is user oriented, which follows Web 2.0 features. Tag use facilitates content search. Search web sites like *Technorati*[i] base their search on blog posts in this tags. It also allows subscription to certain tags.

REST APIs are the preferred technology used by Web 2.0 applications to build web services. REST doesn't have SOAP extra abstractions what facilitates services implementation and deployment. This is contributing to a fast spread and adoption by many web sites.

With lightweight, easy-to-deploy web services, web sites become service providers. Every organisation or individual is able to built or host its own service and offer it to the world.

Mashups are a composition of services that facilitate the design and development of modern Web applications [Benslimane 2008]. One step further on server side collaboration is building web server tools using web services from external sites. Web APIs have been used in last years to create mashups. It brings the opportunity to create a complex mesh of web services offering, use and dependency.

2.4 Online Social Networks

Online social networks can be considered networks in a broad sense, beyond social network websites described below. Online social networks came along with interpersonal communications. Agendas in a mobile phones can be considered to constitute a social network in the broad sense. Social networking tools are populating the Internet penetrating even the enterprise and corporate world. Social capital is being used for assessing the value of networks, and Social Networks Analysis (SNA) techniques are being added to the corporate human resources toolkits because of the importance of "worthy networks" for the business models sustainability in a digital economy [Majumdar 2006].

2.4.1 Social Network Analysis (SNA)

Social network analysis (SNA) provides the foundations for understanding the linkages between social entities and the implications of these linkages [Wasserman 1994]. It provides researchers with a full set of methods for the analysis of social networks, as well as a collection of solid concepts.

[i] `http://www.technorati.com/`

Social network definition

A social network is a finite number of actors and the relations defined between them. The term was originally atributed to Barnes [Barnes 1954]. The presence of information about relations is a characteristic of the network.

The concepts behind a social network are actors and their relations[Wasserman 1994]. These are the basic elements that need to be present to start talking about a social network.

- **Actor**: Social entities are referred to as actors. They can be people of course, but also collective social units such as corporations, organisations, institutes, cities or even events. The term actor does not mean that the entity has the ability to act by itself, but it may usually treated in that way, for instance, *"The University has signed an agreement"*, though is the head of the University which in fact performs the action in behalf of the institution.

- **Tie** Actors are linked by social ties. A tie is made up of two actors and the type of the tie. Actors are ordered, the first of them is the sender of the tie and the second one is the receiver of the tie. A social network with ties between only one type of actors is called a one-mode network, such as the network of exportation exchanges between countries. Social networks that consider more than one type of actor are called two-mode, three-mode, etc.

- **Relation** The most common types of ties between actors include evaluation of one person by other, for setting friendship or respect, transfer of material resources, association or affiliation, attending to a social event or a social club, interaction component: talk or exchanging messages, moving between two places, like migration or physical mobility, physical connection, like a path, a river, formal relations, like authority or biological relation, such as descendants or parents. A relation is the set of all ties of the same type between actors in the network. Examples are the set of friendships in a classroom or the set of commercial transactions between nations in the world. Two actors can have or not ties in different relations. For example, two children in the same class may be friends but not be seated together at the same desk.

  Relations can be reciprocal. If there is one tie with a reciprocal relation, there must be another tie with the actors in reverse order. This is the case of a friend relation managed by Facebook, where both parts must accept the tie before it is established. If there is a tie of friendship between Alice and Bob, there will be other between Bob and Alice. Other relations are not reciprocal, the reciprocal tie may or may not exist. This is the case of the follower relation on Twitter. Bob is following Charlie, but Charlie may or may not be following Bob.

Social network structure

Since Granoveter [Granovetter 1973], research in social network literature shows that relational ties are not equal. People organise their relations around weak and strong ties. Strong ties are the relations with people we care the most, we interact with more often and share more content with. Weak ties are people we know less, we interact with more occasionally and share less content with.

Some studies [Adams 2011] have also found that we belong to several social groups, which come from different contexts in our life: *family*, *school*, *college*, *work*, *hobbies*, etc. We manage from 4 to 6 groups at a time. However these groups change with life. Each group has from 2 to 10 members. We have strong and weak ties with people of each group. There are also differences in how we name relations (*friends*, *collegues*, *partners*, *buddies*, etc).

Social structure is related to privacy management. We do not share the same content with our strong ties than with our weak ties. Besides, we do not share the same content with our family than our working colleagues. The image we show to others, also in the case of group or organisations, is an important part of the social network. People care much about what they do in presence of others. Actors show different types of profiles, depending on the social group. Privacy also has to do with profile visibility in Facebook: 8% of the profiles are public, while a 64% are only shared with friends [Adams 2011]

Network growing

One of the most researched aspects in social networks is how networks grow. There are some motivations that drive people to participate and engage in a network. Examples are the need to belong to social groups, the reward of being popular, social prestige, social network dynamics related to structural balance, transitivity, clique, subgroups, social cohesion, social position and role, reciprocity, mutuality, interchange, influence, dominance conformity [Heider 1946, Barnes 1954, Nadel 1957]. Clustering is caused by sociological tendencies such as cohesion and influence, based on transitivity [Harary 1955]

All of these reasons are from the real world. Boyd studied that network growing is not only driven by the interest to meet strangers, but above all the need to articulate and make visible their social networks [boyd 2007]

Relation evolvement

Research is also focused in how relations are built over time [Adams 2011]. Studies show relations build around content exchange. Engagement is driven by trust. When actors have reliance on others, they are more likely to build relations, share content and points of view with them.

2.4.2 Social Network Sites

Social network platforms have become among the most popular websites [Mislove 2007], though their history is very recent [boyd 2007].

boyd and Ellison [boyd 2007] define SNSs as web-based services allowing individuals to do three basic tasks, i.e. construct a profile, articulate a lists of users with whom they share connection and view and traverse the lists. Besides, they talk about other elements that SNSs may have: avatars, privacy settings, customization of relation names (beyond the "friend" cliché), posting resources to user's wall, private messages and any type of content sharing (photos, videos, etc.)

On the contrary of content management systems (section 2.3.2), there is not a formal study of social networks features. King et al. surveyed computational approaches in social computing [King 2009]. They look at social platforms, such as social networks and other social media. Their work is focused on computational tasks and techniques, such as social analysis, ranking, query log processing, or spam detection, without approaching the field of functional features, which are the ones of our interest.

2.4.3 Social Network Sites APIs

Popular social network platforms are continously increasing their web service offer. They provide authentication commodities, relieving third-party websites, applications and services from managing user data. They are also an increasingly important media. Using already existent identity providers can improve the engagement of users in those third-party sites. Through APIs, these providers allow third-party applications to connect with the activity streams of the users and insert new stories created by the use of the third-party application or service. Nowadays it is not strange to see brands, companies and products offering strong integration with Facebook, Twitter and others providers to improve engagement, visibility and impact.

Ko et al. review social network connect services from Facebook, Google and Myspace [Ko 2010]. They show how these services provide social network features to third-party websites, which do not have to build their own social network. They provide easy sign in and enrich user data and experience by mashing up their own data with the pieces retrieved from the API, e.g. finding friends in the platform. On the other hand, this is also interesting for the provider that now introduces new kinds of activity to its streams.

An attempt to provide a unified connect service was made by OpenSocial, introduced in section 2.2.3, which has been adopted to some extend.

2.4.4 Distributed Social Network Sites

In the last years, there have been an increasing awareness on the importance of distributed social network sites. Even Tim Berners-Lee, the Web inventor, published a position paper describing current problems of centralised SNS, i.e. they constitute information silos; information on one

site is not usable in the others, and they do not allow users much control over how their personal information is disseminated; which results in potential privacy problems [man Au Yeung 2009].

The decentralization problem has been thoroughly studied within the W3C by several groups, i.e. the Social Web Incubator Group [i], the Federated Social Web Incubator Group [ii], which later transitioned to the Federated Social Web Community Group [iii]. Among their main achievements there is the report *A Standards-based, Open and Privacy-aware Social Web* [Appelquist 2010]. This document reviews several problems, including identity, profiles, social media, privacy and activity, along with a summary of the main technologies trying to solve them.

Besides, it gathers a list of decentralized social networking projects that are alreading providing solutions to these issues, such as StatusNet[iv], OneSocialWeb[v] or Diaspora[vi]. Though they are already providing solutions for federated social networks, they are final applications, but not frameworks for building any kind of SNS.

The W3C also defined social web acid test (SWAT) level 0 and 1 (SWAT0 and SWAT1), a set of use cases that social network platforms must fulfil in order to test interoperatiblity. SWATs are not tied to any particular protocol or technology, they are top level actions.

2.5 Authentication in web applications

2.5.1 Identity in the WWW

Identity in a broad sense referes to the collective aspect of the set of characteristics by which a thing is definitively recognizable or known[vii]. Identity is a key aspect in social networks, because each node is unique with a set of characteristics that identifies them and also a set of connections with other nodes. Identifiers let to locate individuals among the rest of the nodes in an unambiguous way. Identity has been intensively studied in social sciences by psychology, sociology, philosophy and anthropology.

Since the Web 2.0 realm is giving people the opportunity of taking several actions based on the individual (publication of news, photos, comments, personal information, etc...), identity becomes a key issue, not only from a sociological, but from a technological point-of-view. Trustability and credibility emerge as key issues as well.

Up to now, there isn't a solid identity model in the web. Users have had to register themselves every time they access a web site which constitutes and awkward and not much handy process. An identity mechanism should allow users log in using one single URI.

---

[i] `http://www.w3.org/2005/Incubator/socialweb/`

[ii] `http://www.w3.org/2005/Incubator/federatedsocialweb/`

[iii] `http://www.w3.org/community/fedsocweb/`

[iv] `http://status.net/`

[v] `http://onesocialweb.org/`

[vi] `http://joindiaspora.com/`

[vii] `http://www.thefreedictionary.com/identity`

Some initiatives tried to tackle the identity problem within the Web 2.0 framework, including Identity 2.0 [Hardt 2006], mIDm [Downes 2005] and OpenID [Fitzpatrick 2005]. While OpenID was spreading, it did not reached mainstream. It was replaced with authentication services provided by major social network platforms, based on OAuth. However, a consistent representation of identity in the Web is still missing today. When the identity problem is solved, organisations and individuals will define and consolidate their web presence. Some proposals like PIF [Gradman 2004] point in this way; beyond the traditional web page, organisations and individuals web sites will gather news and media publications, subscription to several services, just as their own content search services, which will be also based on tags. Following this way, latest work on web identity on the client side, will probably provide easy access to personal and organisational content and services.

2.5.2 Classic authentication

Web applications have developed several authentication mechanisms in order to assert the identity of users. The most common of them is user and password credentials. Users are presented with a registration form where they introduce their name, email, password and password confirmation. After some checks that assert the validity of information, the user is registered in the web application. In subsequent visits, the user is presented with a web form with two fields, username (or email) and password. This form is sent to the web application server, that checks the validity of the credentials or through HTTP authentication headers such as basic or digest authentication [Franks 1999]. Afterwards, an encrypted cookie may be stored in the client browser to authenticate future requests.

An extension of this mechanism is cookie token. After successful authentication, a long-live cookie is stored in the client browser. This cookie contains a token that will authenticate the user in future visits.

Cryptographic client certificates can also be used in web authentication [Dierks 2008].

2.5.3 Central Authentication Server

The Central Authentication Service (CAS) [Jasig 2004] is a single sign-on (SSO) solution developed by Yale University. It is made up three parties, the web application the user wants to log-in, the client browser and the CAS server. When the user wants to sign in the web application, she is redirected via the client browser to the CAS server, where the user provides the credentials in order to authenticate. After a succesful authentication, the client is redirected with a security to the web application. These application checks if the token is valid with the CAS server.

The drawback of the CAS server is that web applications can use only one CAS server, which was overcomed by the OpenID identity framework.

2.5.4 The OpenID identity framework

OpenID [Recordon 2006b] is a user-centric identity standard built and promoted by some of the advocates of the Open Web [Ope 2011]. It specifies how web sites called relay parties (RP) can authenticate users based on an URL, instead of the usual login and password credentials. It was developed as a solution to the multiple login and password problem. With the advent of web 2.0 and the participation of users in numerous sites, every user needs to create and maintain an account for each website where he or she wants to participate in. OpenID proposes that RPs relay authentication in the identity provider (IP), a web service that authenticates the user, providing him with an identity URL, called OpenID identifier. This unique identifier will be used by the user for the rest of the web sites where she wants to log in.

OpenID early version was published in May 2005 by Brad Fitzpatrick [Fitzpatrick 2005] under the name of Yadis. A year later, the community released a revised version, called OpenID 1.1 [Recordon 2006a], which is in fact the first version of the protocol. OpenID 1.1 specifies a flow to prove that a user owns an identity URL. The specification itself considers discovering more information from the OpenID identifier: *(it) does not provide any mechanism to exchange profile information, though Consumers of an Identity can learn more about an End User from any public, semantically interesting documents linked thereunder (FOAF, RSS, Atom, vCARD, etc.).*

OpenID 2.0 was published in December 5, 2007. It introduced some improvements like providing the URL of the identity provider, instead of the user's. The OpenID provider (former IP) helps users to choose their identifiers and gives them the opportunity to use opaque URLs and promote anonymity. It also supports XRI (Extensible Resource Identifier) [Wachob 2008], a new type of URI whose main goal is a standard syntax and discovery format for abstract, structured identifiers. It describes URL normalization. Finally, it specifies, along with HTML, the use of Yadis as a discovery protocol, which was developed in collaboration with the creators of Light-Weight Identity (LID) [LID 2004], the original URL-based identity system, later followed by OpenID. The former name of the first OpenID specification was reused to designate a protocol for the discovery of services.

OpenID has been adopted by multiple web sites. Several major providers have become OpenID providers, including Google, Yahoo, Myspace, Wordpress and AOL. A lot of users already have OpenID identifiers, though they have never noticed. Otherwise, OpenID is used in multiple personal sites. These places contain authoritative, first-hand information about people. In most of cases, they are about technologists that have the knowledge and motivation to build and maintain such web sites. They are also the primary adopters of web standards, the ones that will accept a new technology if it is valuable enough, echoing it and using it in their projects and works. In such cases, OpenID can be taken as a reference to determine whether a web page represents the identity for a person or not, because there will be a motivation to link identity to that web page.

OpenID says nothing about whether a web page must have any data attached to it or not. In

fact, there are people very concerned with anonymity, who would prefer to use opaque identities. Google is an identity provider that operates in this way; it uses a feature, supported in version 2.0 that provides a generic identifier. Then, the user logs in at the provider's site and so he chooses his identity. Some of Google's identifiers are not relevant for the user, meaning that they are not a reference of him, neither are they traceable.

But however, there are other cases where people do maintain a primary source of authoritative information about their personas. They use their pages as their profile page, linked from comments in other websites, where the identifiers were collected. It is expected that users build these pages with relevant information about themselves. It is a reference about them where they put identity-related information. They show their achievements, contact data, name, address, occupation or even localization. This way, they provide to the web page where they are logged a reference for others users who might want to learn more about them.

Some security problems [Oh 2008] have been identified in OpenID, which led to revisit its applicability in environments with demanding security constraints.

2.6 Authorization in web applications

System's security attends several issues on system's information [Joshi 2001]. These include the following aspects:

- **confidentiality or secrecy**: ensuring that information is not accessed by unauthorized people
- **integrity**: ensuring that information is not modified without authorization
- **information availability**: ensuring the information is available
- **accountability**: ensuring that actions can be traced back

The work in this dissertation focuses on access control, which refers to exerting control over who can interact with certain resource.

2.6.1 Classic Access Control

Several access control models have been proposed to achieve access control. They are broadly classified as Discretionarly Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC).

2.6.2 Discretionary Access Control (DAC)

In the DAC, access control policies are specified in a per-object basis. Every object in the system is enumerated and may have policies attached to it. These policies describe the actions that different

users are authorized to perform on the object. The common setting is that every object has an owner, which is the resposible of setting its policies.

DAC models are very common in web applications, because they are easy and straightforward to implement. However, they do not provide high security assurance. Having the users the control on resources, mistakes can lead to privacy leaks, which can be propagated through the system [Joshi 2001].

A classical representation of the DAC model was introduced by Harrison et al. in the access control matrix [Harrison 1976].

2.6.3 Mandatory Access Control (MAC)

Contrasting to the DAC model, MAC policies are system defined. Users and objects are classified in security levels. These levels are used as the basis for access control decisions. These model assures controling the information flow, ensuring confidentiality and integrity of information. In this case, the user has not that freedom to decide available actions on objects.

A well-known implementation of MAC model is the multilevel security mechanism that uses no read-up and no write-down rules, also known as Bell-LaPadula restrictions[Sandhu 1993]. It is used by the U.S. Department of Defense to formalize their multilevel security policy. Access levels range from the most sensitive (e.g "Top Secret"), down to the least sensitive (e.g. "Unclassified") [Bell 1998].

2.6.4 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) [Ferraiolo 2001] is a conceptual model that helps to manage access control information using the metaphor of roles. In RBAC, users are assigned to roles. Permissions, actions performed on an object, are also assigned to roles. Thus, users acquire permissions by being members of roles. This simplifies the management of privacy policies, by assigning them to roles instead of single entities.

The RBAC model is organised into four components, each providing more features to the model. Core RBAC captures the basic features of users, roles and permissions. Hierarchical RBAC, the second component, introduces role hierarchies that reflect the organisation's lines of authority and responsibility. Privileges are inherited through the hierarchy. When a user is assigned to a role at the top of the hierarchy, she will also have permissions belonging to roles further down in the hierarchy. The other two components are the static separation of duty and dynamic separation of duty.

RBAC provides several well-recognized advantages over the former access control models, i.e. Discretionary Access Control (DAC) and Mandatory Access Control (MAC) [Joshi 2001]:

- Arbitrary, organisation-specific security policies. RBAC is "policy neutral". A wide range of security policies can by defined, including DAC, MAC and user-specific.

- Simplification of security administration. When a user moves from one position to another inside the organisation, it supports moving the user between roles instead of revoking individual permissions.

- Permissions on other roles. Special administrative roles can be designed to manage other roles.

Parameterized RBAC (PRBAC) is an extension to RBAC in order to improve its flexibility. Abdallah and Khayat review existing related work and formalize it as an extension to RBAC [Abdallah 2005] that includes parameters in its elements, e.g., roles, objects and subjects. PRBAC is demanded because RBAC does not handle patterns appearing in similar roles. In RBAC, each role has a unique set of permissions. Users are assigned to a role, hence to a set of permissions. But it is usual to find patterns in permissions, e.g. an *Account_Holder* role in a online banking scenario. Every client must have rights on his own account. In RBAC, a new different role for every account is needed, because each account involves a different set of permissions, *Withdraw Account1* is different from *Withdraw Account2*. This behavior involves a huge burden on permissions management, an impact in the scalability of the systems in large organisations and an impact in the consistency of roles sharing the same pattern [Abdallah 2005]. PRBAC introduces parameters to role definitions. In PRBAC, each real role is an instance of a role with its parameters taking values. In the example, *Account_Holder* is replaced by *Account_Holder(a)*, where a is the parameter that takes the value of a specific account. So, a user opening an account Account1 is assigned to instantiated role *Account_Holder(Account1)*. This way, permissions of all the instances of *Account_Holder(a)* are managed in the same consistent way.

2.6.5 Access control in Social Network Sites

Privacy is one of the most controversial issues in social networks. Probably because of that, it is between the most popular issues in the research community on social networks. Privacy research fields in context of social networks are multidisciplinary, involving several disciplines, such as sociologists, lawyers , educators, designers, as well as computer scientists [SPI 2011]. In the case of information technologies, related aspects include storage security and graph anonymization, traffic data protection, accountability, audit and access control [SPI 2011].

Current practices in SNS access control are reviewed in literature [Carminati 2008, Carminati 2009, Hart 2007, Squicciarini 2010, Squicciarini 2009]. Most of present approaches adopt a Discretionary Access Control (DAC) point of view. This means that resources usually have an attribute used to control its access. This attribute takes one of the following values: *private*, *friends*, *friends-of-friends* and *public*. Although these are the most common values, SNSs may manage other types, e.g. *follower*, *colleague*, *classmate* and *business partner*. A full review of them is provided by Carminati et al.[Carminati 2009]. This popular

method is attributed to the straightforwardness and ease of its implementation, and the best balance between ease-of-use and flexibility [Carminati 2008, Carminati 2009, Hart 2007] . But it introduces the following limitations: it is too coarse and using the friends' relation for access control forces users to choose between protecting their privacy and appearing popular.

Some recent proposals try to overcome these limitations. Squicciarini et al. carries out extensive work in the field. Their contributions include a solution with automated ways of sharing images based on an extended notion of content ownership, using a game theory based mechanism [Squicciarini 2010], "web-traveler policies" attached to images in order to restrict access to them in SNS [Squicciarini 2009] and PriMa, a privacy protection mechanism which supports semi-automated generation for access rules for user profile information [Squicciarini 2010]. All the work is based on policy specifications on resources, which take a Discretionary Access Control (DAC) approach.

Ali et al. and Carminati et al. base their solutions on trust. Ali et al.'s approach [Ali 2007] define an evaluation of user reputation based on trust relationships with other users. Resources are given a confidence level. They will be accessed by users with more or equal reputation level. The Carminati et al. solution [Carminati 2009] is more sophisticated. They propose a complete semi-decentralized solution based on three parameters, i.e. type of link, path depth and trust value of the link. The model supports decentralized SNS with a centralized certificate authority that asserts the validity of the links. Both of them rely on trust, measured as a numeric value.

A different approach is taken by Fong et al. [Fong 2009]. They describe a full algebraic model for access control, claiming Facebook to be just an instance of it. Their model is abstract to large degree. Authorization is based on two issues. The first one is the communication history, the set of events that happen between each user, e.g., invitations to participate in the social network. The second one is acquaintance topology, the set of relationships between users stored in the persistent layer of the SNS. They introduce the latter to simplify the former. As the authors recognize, consuming all the communication history in order to evaluate an authorization request is intractable. However, significant communication events could also be saved as relationships, which would simplify the model considerably more. Relation-based access control is also suggested by Gates as a requirement for access control [Gates 2007].

Other work focused in relations is proposed by Giunchiglia et al. Their RelBAC access control model is different from RBAC models in that permissions are relations between subjects, and access rules an instantiation of permissions [Giunchiglia 2008].

In the role based access control model proposed by Li et al. [Li 2009], users are assigned roles and relations are established through these roles. For example, user A is assigned role A, user B is assigned role B and a relation is established between both roles.

2.6.6 OAuth

OAuth is the omnipresent *"open protocol to allow secure API authorization in a simple and standard method from desktop web applications"* [i]. It was developed by some web enthusiastics in the aim to develop a common standard for accessing APIs. Version 1.0 was proposed as a IETF request for comments [Hammer-Lahav 2010].

OAuth removes an anti-pattern that was appearing in web services applications. With the growth of those applications, there was a need of a web application to access protected resources from others. For instance, imagine some private pictures in a photo sharing service, and another web application that prints photos and send them to your home. You may want to let the printing service access to several private photos in order to print them for you. Before OAuth, the only way to achieve this was sharing the photo-sharing web service credentials to the printing service. The obvious drawback of this approach was that the printing service had full access to your account in that photo-sharing service.

OAuth proposes a solution to this problem. When the user wants to print the photos, the photo printing service will redirect the user-agent to the OAuth handler of the photo-sharing service. In this step the user will authorize the photo-sharing service to provide to the third-party app (printing service) access to their pictures. Finally the request will be redirected to the printing service including an access token parameter. Using this provided token the printing service will be granted access to the user private photos.

In April 29, 2009 a security flaw was discovered in the protocol [ii]. A session fixation attack allowed an attacker to pre-build an authorization request and inject it to a user to finish it. OAuth version 1.0a was released. This security announcement was about a year before the RFC publication, so the IETF's document already has the security fix. Nevertheless, most of the APIs still refer to their supported version of OAuth as 1.0a.

OAuth 2.0 is the next version of the protocol [Hardt 2012]. OAuth 2.0 simplifies the protocol. It describes four different grant types, i.e., *authentication code*, *implicit*, *resource owner password credentials* and *client credentials*. The first one, authorization code, is suitable for clients capable of maintaining the confidentiality of their credentials, i.e. web applications that reside in a web server. This flow authenticates the client, and the authorization token is granted directly to the client, it does not pass through the user-agent. The second one, implicit flow, is suitable for clients implemented in a browser, typically Javascript. The access token is directly granted to the client, it is not authenticated neither. The third is similar to the case OAuth tried to aliviate. The client uses the user's credentials to take an authorization token though they are used only once and should not stored. Nevertheless, this method is requested to be the last alternative to be used. Finally, client credentials allows a client to manage protected resources controlled directly by him in the

[i] `http//oauth.net/`

[ii] `http://oauth.net/advisories/2009-1/`

authentication server.

OAuth 2.0 also describes access token scopes. The scope parameter contains a list of space-delimited parameters defined by the authentication server. They are used to define to which resources the user is granting access to the third-party application or service.

2.7 Software reuse

Software reuse research is reviewed in [Frakes 2005]. It is defined as the use of existing software to construct new software. Its purpose is improving software quality and productivity. The key idea in software reuse is domain engineering or product-line engineering; most software systems are not new, but variants of systems already built. This idea fits completely within the objectives of this dissertation, the aim for building a component that can be reused in multiple SNS.

Reusability can be achieved at several levels, including domain engineering, where the domain knowledge is reused to build new products, the programming language level, that allow functions, modules, subroutines, etc. to be reused, at the libraries level and the architecture level, with reference, application and platform architectures [Frakes 2005].

Software reuse measurement is surveyed in [Frakes 1996]. They introduce several methods such as cost-benefit analysis, maturity assesment and amount of reuse. Cost benefit analysis is related with the amount of time required to build a software from scratch, compared with the amount of time required with reusable components. Components are classified according to their complexity as monolithic, polylithic, graph and menu, mask. Maturity assessment has to do with reuse culture in the organisation. Amount of reuse assess and monitor a reuse improvement effort. Levels of abstraction need to be defined in order to measure reuse. The abstraction hierarchy, the definition of external repositories and the definition of the "uses" relationships must be defined.

Measuring the amount of reuse in object oriented systems is introduced in [Bieman 1992] and [Karunanithi 1993]. Authors classify software reuse in verbatim and leveraged. Verbatim reuse refers to reuse without modifications. This classification includes creating instantiations of objects of a class and calling methods of that class. On the other hand, leveraged reuse refers to class inheritance, for instance, measuring the number of instances of a class.

Frameworks are a code reuse technique themselves. They provide developers with a library of existing components. The new components usually inherit from an abstract superclass. Besides, frameworks are a reusable design and they reuse analysis [Johnson 1997]. However, there is not literature that provides measures of code reuse in MVC frameworks.

Chapter 3

Objectives

> *Caminante, no hay camino, se hace camino al andar.*
>
> *Antonio Machado*

The main objective of this work is improving the building of distributed social network websites, by providing a framework that hides the burden associated with social network features providing them.

Social network sites are becoming quite popular, as they leverage social networks features, the mechanisms that make society work. Building such a framework would facilitate the deployment of these kind of websites, allowing them to communicate and creating a mesh that will improve human communication in disparate fields.

In order to achieve this general objective, the following steps must be covered:

- *Explore and assess the feasibility of code reusability in the context of MVC web development frameworks.*

  As reviewed in section 2.1.2, MVC frameworks are the state of the art in web development. However, there is not previous literature or experiences on how software reuse can be applied to MVC frameworks, beyond the frameworks themselves (section 2.7).

  The first objective is building a component that provides CMS features to web developers and MVC users, in the context of the Web 2.0 realm (section 2.3.3).

- *Gather a thorough enumeration of functional features present in social network sites (SNS)*, along with their description and the justification of their utility in a virtual social context.

  Despite SNS are between the most popular sites of the Web, and social network features are very valuable for web sites, there are still gaps on their methodological study. As reviewed in section 2.4.2, there is not a formal study about the features a website must fulfill to be considered a SNS, in a way as there are in content management systems (section 2.3.2).

  An analysis and evaluation of the implementation of these features in leading social network sites will validate the list of features.

  The analysis of such features will lead the design of a social network framework by setting the requirements for such a framework.

- *Design and implement a framework for building social network websites.*

  There is not literature about how social network sites can be built. Furthermore, there are not architectural guidelines that describe best practices in building SNS. The requirements for a framework for building social network websites will be collected from the set of features described in the former objective. An architecture that supports the set of requirements will be described. The framework will be implemented and validated by building customized social network sites.

- *Design a social network analysis-driven access control model for SNS.*

  Current access control practices in social networks, reviewed in section 2.6.5, have several limitations. We need a model that provides the following features:

  - The model should support fine-grained policies beyond sharing with friends, such as sharing with specific users. Besides, it should conciliate contact aggregation and privacy protection, leting users to add more contacts at the same time they grant them less permissions or without granting any permission at all.
  - They should facilitate permissions managing, allowing users to change permissions at the same time. For instance, when *Alice* unfriends *Charlie*, she should be able to revoke permissions at one time without changing each individual permission.
  - Besides user-defined policies, the system administrator should also be able to define suitable system relations, along with their associated permissions.
  - The access control model should be flexible enough to let users use their custom words for defined relations with their contacts, and using their own words beyond friend, such as *classmate*, *buddy*, etc.
  - Access control must not be reciprocal. The model should support Alice granting permissions to Bob while Bob does not grant permissions back to Alice.
  - The model should consider other social actors, such as groups, organisations and institutions as first-class citizens in policy definition, allowing them to grant and revoke permissions, as well as being granted and revoked them. Besides, it should consider users acting on behalf of these social actors.
  - Instead of assigning a numeric value of trust to links, we want to take an approach that is focused on the type of relationship as the channel for the transfer of information. Relations are ordered by their strength, assigned permissions to them in the same way as RBAC does.

- As in the case of social network features, a *study on the different paradigms and mechanisms involved in distributed social network sites* is needed, in order to gather the requirements for a framework that provides those features.

The study will analyse several paradigms in distributed social networking. Besides, it will study two specific cases that can shed light on the topic.

On the one hand, an analysis from the perspective of OpenID identifiers will provide the user-centric point of view. The objectives in this case are:

- Research on identity on the web via public identifiers. Find current tendencies around user-centric identity.
- Identify what are the most used technologies for exporting information used by OpenID providers in profile pages
- Identify what kind of information do people share in profile pages

On the other hand, an analysis on current SNS APIs will allow us to learn from current practices in social network distribution from the marketing, pragmatic point of view, and see what are the leader websites offering to their customers.

- *Design and implement a framework for building distributed social network websites.*

  As reviewed in section 2.4.4, there are already solutions that provide with distributed social networks. However, there are not suitable for building customized content-oriented sites. They are final applications and are not designed to provide a component that final applications will use.

  An analisys on the distributed social features will draw the requirements that a distributed social network framework must fulfill. According to these requirements, the architecture of the social framework should be extended to support these requirements. In the last place, this architecture should be implemented and validated by building distributed social network sites.

Chapter 4

A Framework for Building Content Management Systems

> *Good programmers know what to write.*
> *Great ones know what to rewrite (and reuse).*
>
> *Eric S. Raymond. The Cathedral and the Bazaar*

4.1 Introduction

Reusable components supply practitioners with well tested software and reduce development effort (section 2.7). After the appearance of the Web 2.0 (section 2.3.3), users have become to play a main role in the web. The technical shift that allows users to upload content without having to rely in the webmasters popularized CMS-systems (section 2.3.2). The state of the art in web development in recent years have been Model-View-Controller (MVC) frameworks, as introduced in section 2.1.2. In this age of agile web development, it seems quite interesting building a piece of reusable middleware that provides MVC users with capabilities to agile building websites with CMS features, which is not covered by current content management frameworks (section 2.3.2).

4.2 Requirements for a content management framework

Content management features have been gathered in literature, as reviewed in section 2.3.2. A set of these features were found to be most interesting to be implemented in a first instance:

- Management of user accounts and user profiles. User accounts are required for multi-user support and authorship, content personalization, user security and reporting features.
- Groups management, required for workgroups
- RBAC support (section 2.6.4), provides role definition and user security.
- Support for relevant content types, such as text posts, comments and files, with specialized management depending on the type of file (pictures, audios, etc.)
- Tags and categories, used in taxononies or the more loosely, user-defined folksonomies (section 2.3.3).

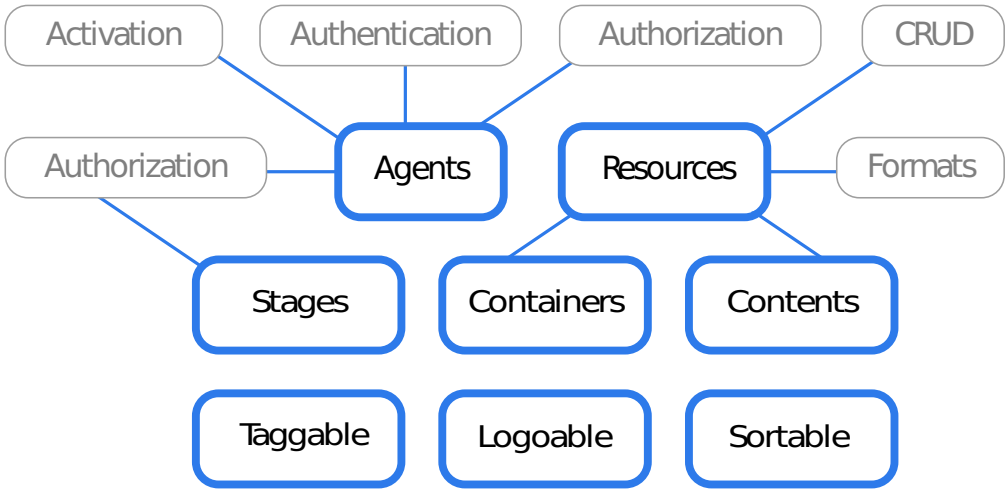

Figure 4.1 : Components of Station, the CMS framework

4.3 Station, a content management framework

Station is a content management framework (CMF) developed to achieve the objectives proposed in this chapter. It is a middleware for Ruby on Rails that provides developers with support for content management. It fulfils the requirements described in the former section.

Station's architecture is divided in the following components (figure 4.1):

4.3.1 Agents

Agents are entities that are able perform actions in the application. The paradigm of agents are users. However, Station is flexible enough to provide other entities with agent features. Using Station, any of the models in the application can have agent features. These features include authentication, authorization and account activation.

Authentication

Agents must provide with credentials to the web application in order to identify themselves. Station offers several methods to achive this task, from classic login and password to modern OpenID. Almost all the authentication methods reviewed in section 2.5 are supported by Station, i.e. login and password, OpenID, Central Authentication Service (CAS) and cookie token.

Authorization

Station provides access control by supporting role-based access control (RBAC) functionality (section 2.6.4) within the authorization framework. Agents perform a role within each stage they participate. The role defines the permissions the agent can perform in the scope of that stage. Stage functionality is described below.

Activation

Some web applications may require users to activate their account, in order to verify the email they provided is valid and ensure further communication. Besides, the verified email can be used to reset the password, in the case the user forgot it. Station also provides this functionality within the agent module.

4.3.2 Resources

Resources include any type of data, such as text, files, images, audios, etc. Resources functionality provides models with create, read, update and delete (CRUD) operations and the ability to be imported/exported in different formats, such as Atom or RSS (section 2.2.2).

This functionality goes beyond the scaffold included in Rails [i]. Rails scaffold allows users to create a model with an associated controller supporting CRUD actions, along with their views. Station includes these actions within a module, so they can be included by the resource's controller.

Station supports facilities for exporting resources in the following formats:

- **XML**: text document encoded using Extensible Markup Language (XML) (section 2.2.2). These include XHTML, Atom and RSS.
- **YAML**: text document encoded using YAML Ain't a Markup Language (YAML) (section 2.2.2). These include JSON.
- **HTML encoding**: usually the result of HTML Forms, uses `application/x-www-form-urlencoded` or `multipart/form-data`
- **Raw**: binary data, usually in files.

Besides, Station provides two kind of relations between resources, content and containers.

[i] `http://guides.rubyonrails.org/getting_started.html`

Content

The content functionality enables support for resources that are included in a container. Examples of contents are photos in an album, tasks in a project or points in a map. The content functionality provides the model with the needed plumbing for the relations to be saved in the model and URLs to be automatically generated, such as in `/projects/1/tasks`.

Container

On the other hand, a container is a model that has many other models that depend on it. Example of containers are a project with tasks, a forum with posts or an album with photos. As in the case of contents, Station facilitates the management of containers and the contents associated with them.

4.3.3 Stage

An stage is a model that defines an RBAC authorization context for agents. An example of stage is a group. A single user can perform the role of *admin* in a group and a different role in another group.

Permissions are parameterized per stage, following the PRBAC model described in section 2.6.4. For instance, the permission *Add members(stage)* grants to their holders the ability to add new members to a given stage.

4.3.4 Taggable

A model with a taggable functionality will have tags attached to it. Station is flexible enough to support both taxonomies and folksonomies (section 2.3.3), letting sites administrators or current users to define the tags.

4.3.5 Logoable

The logoable functionality allows attaching logos to any model. This functionality can be used for user avatars, group logos, book covers, etc.

4.3.6 Sortable

The sortable module lets models to be ordered by arbitrary attributes, such as title, creation date, etc.

4.4 Content Management websites built using Station

Station was used to build several content management websites, including a conference management system developed in the context of an EU-founded research project.

4.4.1 MOVE Organizational Virtual Environment (MOVE)

MOVE Organizational Virtual Environment (MOVE) [i] is a CMS platform developed on the top of Station. It manages user and groups, with invitations. Besides, it can manage the following contents: articles, images, audios, links, documents and tasks.

MOVE was deployed in the Departamento de Ingeniería Telemática (DIT) of the Universidad Politécnica de Madrid [ii], mainly as a personal page, having 13 registered users. Up to time, it is the base for the personal page of the author of this dissertation.

MOVE was also deployed in the context of the local cultural centre *La Piluka* in Madrid [iii], having up to 120 users.

4.4.2 Virtual Conference Center (VCC)

The VCC [iv] is the web application behind GlobalPlaza [v] a global a hosting service for publishing and discovering videoconferences. The VCC was developed in the context of GLOBAL, a FP7 EU-founded project whose aim was providing "*a virtual conference centre using advanced communication technologies and concepts to support the promotion of e-infrastructure topics in Europe and around the world*".

The VCC's main task is event management. It supports CMS features for the organisation of events with agenda entries, along with news, attachments and posts around each event. It also manages collaborative spaces. Every user performs a role in the context of each space, which uses Station's stage functionality. The VCC defined a second stage, events. Users had a role in each event, speakers, assistants, etc. The container/content functionality was also used for space's news, events, attachments and posts.

Global Plaza, reached 1030 registered users, 261 spaces, 724 videoconferences organised and 1102 videos stored. It was the outcome of the Global project.

4.5 A metric for measuring code reuse in MVC frameworks

Measuring the amount of code reuse is the way to assess the usefulness of a CMF. Code reuse measures show how much effort is saved by developers using the framework. Code reuse techniques are reviewed in section 2.7. There are code reuse measures on the level of object-oriented programming. However, there is not previous work on code reuse measures at the MVC level.

---

[i] `https://github.com/atd/move`

[ii] `http://irss.dit.upm.es/`

[iii] `http://antiguo.piglove.lapiluka.org/`

[iv] `https://github.com/ging/vcc`

[v] `http://globalplaza.org/`

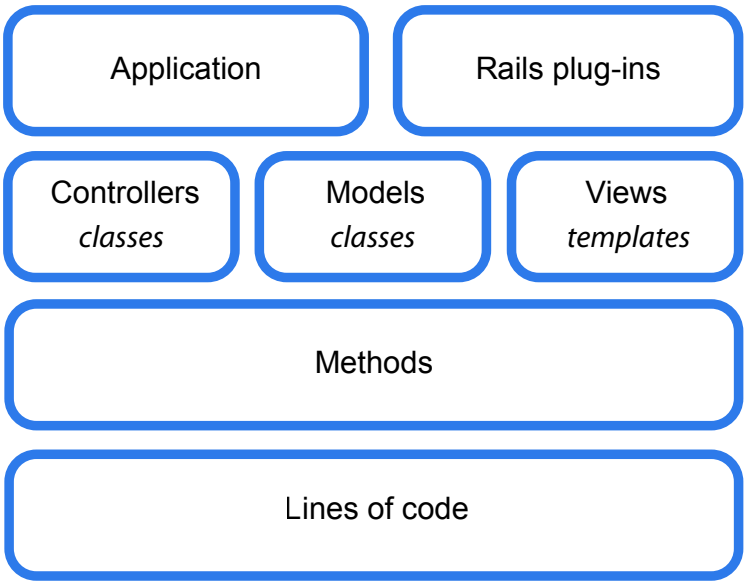

Figure 4.2 : Levels of abstraction in code reuse metrics

This section introduces a metric for measuring code reuse in a MVC framework. Frakes and Terry specify the requirements that must be fulfilled in order to measure reuse, i.e. the definition of the levels of abstraction and the abstraction hierarchy, the definition of external repositories and the definition of the "uses" relationships must be defined [Frakes 1996].

The level of abstraction of our measures is the library at the MVC level. In the case of Rails, this is called the Rails engine level, which comprises models, controllers and views. In the case of controllers and models, these are classes who define lower levels of abstraction, such as methods and lines of code. In the case of views, there are partials who may call helper methods and also lines of code. Figure 4.2 show the levels of abstraction and their hierarchy.

The content management framework is defined as the external repository, and the final application reuses its code. Rails engines provide models, controllers and views that are visible from the final application, and can be used by it without any modification. The total amount of code C in the final application is represented in:

$$C = V + L + N$$

Where V is the verbatim use of components (models, controllers and views) that are used by the final application without any modification This is the ideal case, where there is not extra effort required to the final application developers. L represents leveraged use, which include components that are present in both the framework and in the final application. They already

provided developers with architectural patterns and inital code, but yet required effort to adapt them for the final application. Finally, N represents the amount of code that is new in the final application and required effort from developers.

The amount of code directly saved by verbatim reuse C_v can be measured by:

$$C_v = \frac{V}{C} * 100$$

In the save way, the amount of code leveraged C_c reused can be measured by:

$$C_l = \frac{L}{C} * 100$$

Finally, the amount of new code C_n:

$$C_n = \frac{N}{C} * 100$$

Code reuse in the VCC

The above code reuse metric was applied to the VCC, in order to assess the validity of Station, the content management framework. Table 4.1 shows the results. It also includes the number of components provided by Station that were not used in the final application.

Table 4.1 : Measures of code reuse in Station and VCC.

	Station		VCC						
	Total	Not used	C	V	L	N	C_V	C_L	C_N
Controllers	11	2	47	2	7	38	4.25%	14.89%	80.85%
Models	26	1	70	18	7	45	25.71%	10%	64.28%
Views	34	15	367	6	13	348	1.63%	3.54%	94.82%

There is significant more code reuse in models, having C_V = 25.71% and C_L = 10%. These amount of reusability is smaller in the views where there is a C_V = 1.63% C_L = 3.54%.

Figure 4.3 shows the amount of V, L and N on models, controllers and views from Station used in the VCC. It also includes the amount of code provided by Station that was not used in the final application. Figure 4.4 shows code reuse normalized to the amount of files in each category of MVC.

In any case, it is worth to note that these measures reflect code reusability at the Rails engine level. This involves models, views or controllers that are used in the final application. This level is one of the highest, but there exists other types of code reusability that are present in the VCC, at lower levels. For instance, many of the models included some functionality from Station components, such as agents, resources and stages, including them as a module. Further study would be necessary to measure this kind of reuse.

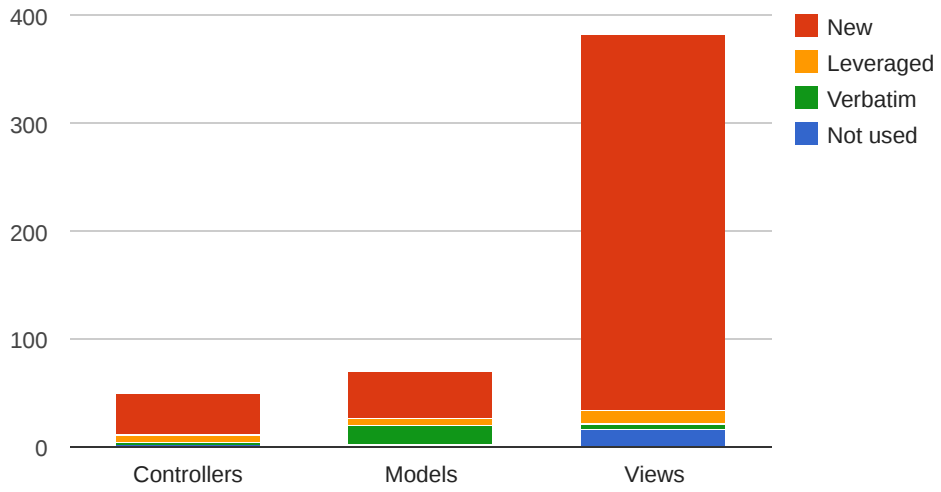

Figure 4.3 : Code reuse between the VCC and Station, models, controllers and views

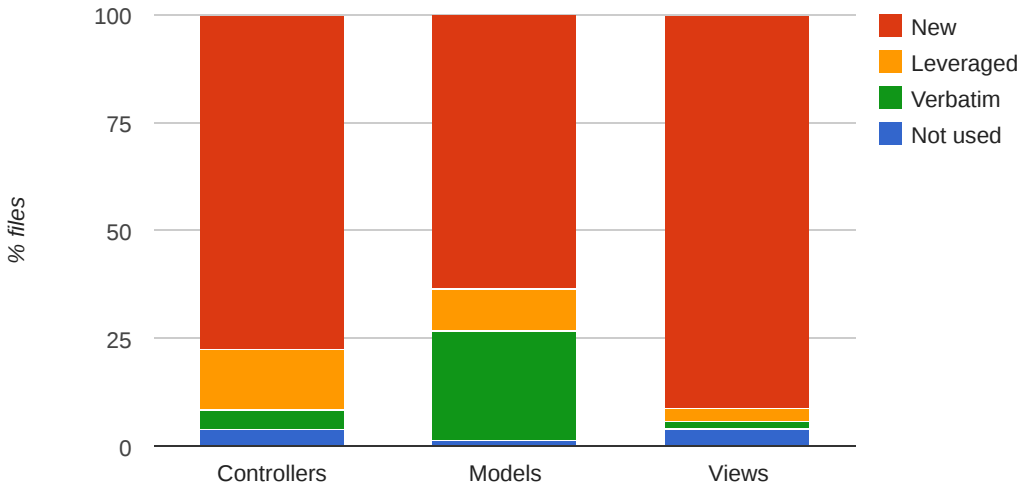

Figure 4.4 : Code reuse between the VCC and Station, comparison in percentages of files between models, controllers and views

4.6 Discussion

Despite its use at the core of the VCC, Station did not gained community, as it is expected in free open source software (FOSS). Maybe because it resulted in a rather big and complex piece of software. However, the idea of building a middleware that gathered CMS features proved to be valid. Along with Station's development, Ruby on Rails adopted support for engines, which fluorished and gained many users. But it turned out that most of the functionality offered by Station was started to be offered by other plugins that focused on specific parts of Station.

Appendix A gathers a review of content management features implemented by Ruby on Rails and its plug-ins. Table 4.2 shows the Rails plug-ins that implemented features offered by Station. *Devise* is a plug-in for authentication that provided solutions such as login and password, cookie authentication and password recovery, among other features. *CanCan* is a plug-in that offers authorization capabilities. *Inherited resources* provides the application with CRUD controllers for resources. *ActsAsTaggable* with tagging support and *RailsAdmin* a complete backed for model management, including model sorting.

Table 4.2 : Station's features supported by other Rails plugins

Station feature	Rails plugin
Agents' authentication	Devise
Authorization	CanCan
Resource's CRUD	Inherited Resources
Taggable	ActsAsTaggable
Sortable	RailsAdmin

Besides the popularity of social networks started to show up. Facebook, Twitter, LinkedIn became major actors in the web. It became more clear the interest on adding social network capabilities to Station. The outcome of this new work was called Social Stream (section 6.3).

4.7 Conclusions

This chapter explores and assess the feasibility of code reusability in the context of a MVC web development framework. It introduces Station, a middleware that provides a MVC framework such as Ruby on Rails with CMS features. Station was validated in the context of three deployments, i.e. two local content management platforms and a EU-founded project event management website. Besides, a code metric that assess code reuse in MVC components was defined. In spite of being in the right way, Station turned out to be too big to succeed. Their features were implemented by different focused plugins, which cover content management capabilities in Ruby on Rails. However, its validation as reusable component was used in the development of a distributed social network framework, which is introduced in next chapters.

Chapter 5

Features of Social Network Sites

> *Know [the] other, know [the] self, hundred battles without danger; not knowing [the] other but know [the] self, one win one loss; not knowing [the] other, not knowing [the] self, every battle must [be] lost.*
>
> *Sun Tzu*

5.1 Introduction

Though social network sites (SNS) have grown to be between the most important sites in the web, there is not a formal survey of their functional features.

In the same direction as content management, social networking is transversal to websites' technology. Content management websites provide a set of features that have been studied in literature (section 2.3.2). This collection is worth because it focuses the technological solutions towards specific issues, identifying underlying problems and solutions.

There exist informal lists of social network features, as well as recent surveys, more focused in the field of social computing, that were reviewed in section 2.4.2. However, there is a lack of a comprehensive review of social network features present in SNS. Moreover, there does not exist formal measure on how these features are adopted by current social network platforms. This study is needed in order to establish a base for further research on SNS.

5.2 Enumeration of SNS functionalities

This section introduces a comprehensive list of functional components that can be found in social network sites. Some of them were already present in [boyd 2007], while others are novel in the literature.

A social network is a finite number of actors and the relations defined between them. The term was originally attributed to Barnes [Barnes 1954]. The presence of information about relations is a main characteristic of a social network. The concepts behind a social network are actors and their relations [Wasserman 1994]. These are the basic elements that need to be present to start talking about a social network (section 2.4.1).

5.2.1 Social actors

Social entities are referred to as actors. They can be people of course, but also collective social units such as corporations, organisations, institutes, cities or even events. The term actor does not mean that the entity has the ability to act by itself, but it may usually treated in that way, for instance, *"The University has signed an agreement"*, though is the head of the University which in fact performs the action in behalf of the institution.

Social actors must be explicitly supported by a social network platform. This feature should be implemented by every site that claims to be a SNS.

User accounts; users must be allowed to register in the site. They should fill a form, provide some basic data, typically name, email and password, though this could vary from one site to the other. Later, in the log-in process, users provide credentials in order to be authenticated. The importance of authentication resides in that users perform actions in their own behalf, like establishing relations with other users, uploading content, etc. The requirement is that users must be univocally identified. The website also bases authorization on user authentication.

Some SNS may force the **email to be confirmed** and do not let users perform actions in the site until this step is completed. On the other hand, there are sites that let users continue and start exploring the site without email confirmation.

Nowadays, leading websites are providing authentication services to other websites. SNS may **delegate authentication** to them by using these services. This is a convenient and user-friendly way to authenticate, because users do not have to introduce their credentials just another time, and the SNS can extract profile data and contacts from the authentication service.

Some features intend to bring more users to the SNS. One of them is sending **invitations** to other users by email. This feature consists on a form in which users introduce their contacts' emails. An extended version of the former functionality is **contact import**. The SNS obtains the agenda from the user's email account or other SNS, and uses it to send invitations or set user's contacts in the site.

Other social entities should be also supported by the site, such as *groups*, *organisations*, *institutions*, etc. This is a way of extending coverage to other entities in the social spectrum. These entities may be able to maintain their own social image, which includes logo, contact information, etc. They must not be confused with contact groups or lists, which are reviewed below.

5.2.2 Social relations

In SNS, actors must be able to establish some kind of relations with other actors. This must be another mandatory feature for every SNS. Relations can be unidirectional o bidirectional.

Unidirectional relations are established by one of the parties and do not need confirmation by the other party. *Follow* is a popular case of unidirectional relation. When some actor *follows* another one, he is expressing his interest in the activity of the other actor. This usually implies that their activities reach them to their home wall, as we will see below, in section 5.2.9. The

paradigm of this relation is Twitter. A full functional social network can be build with unidirectional relations.

Bidirectional relations mean that both parts need to approve the relation to be established. Facebook is the paradigm of this relation type. Bidirectional relations are usually attached to privacy and permissions; e.g. when both users accept the relation, they are allowed to publish to their walls. However unidirectional relations can also provide them.

The way to establish relations is usually facilitated with a link or button next to their avatar, name and other data that identifies them.

Contacts management features help users to organize their contacts in several categories, lists or groups. It is a way to avoid the context collapse problem [boyd 2007]. Contacts management is in constant evolution and there has been a lot of research related to privacy and permission management. There are several models to manage contacts; Twitter uses lists, as well as Facebook. Google+ uses a more advanced and eye-candy interface of Google Circles. Contact management settings can also be used in **privacy settings** to grant permissions to certain groups of contacts at the same time.

5.2.3 Content

Contact-oriented SNS usually provide means for their users to share content. Besides, *content-oriented* SNS are built around some specific type of content. These features take the influence from the work in web content management systems (CMS), introduced in section 2.3.2.

Content types managed by SNS may be quite different. Text comments are the most basic type of content. Examples of SNS built around content are Flickr for photographs, YouTube for videos, Github for code repositories. Even contact-oriented sites like Facebook support some types of content such as pictures or events. One type of content that is very popular in contact-oriented networks is external links to other websites. Using this feature, contact-oriented sites become a forum for content exchange between affine people or groups.

Specific type of content help to focus the website on some specific type of user, e.g. software repositories or artistic photographs will attract software developers and photographers, respectively. The more types of content the social platform supports, the more channels actors have to interact between them, but also the more diluted the purpose and identity of the SNS becomes.

Content may have **threads** that allow discussions on topics. Content have a “reply” button that allows submitting new related content, such as **comments**. Afterwards, new content appears below and often tabulated to the right. Content threads encourage engagement between users.

Another functionality providing engagement is to **cite or tag other users** in content. This is typical in Twitter, when users are cited in tweets by writing their name preceded by @. Other types of citation are facial recognition and photo tagging. The last one is very popular on Facebook, and many people use it as some kind of heads up.

SNS may also implement **content search engines**, a feature that enables users to find content that was uploaded to the network. This feature also comes from the CMS world.

5.2.4 Communication tools

SNS sites may also support other *communication tools*. One example is **system notifications**, by which the system alerts users of some types of events, for example the posting of new content that may be of their interest. Other examples include **private messages** between users, **chat** and **video-conferences**. These communication tools are often connected to email.

5.2.5 Privacy and content visibility

Privacy in social networks is a hot research topic nowadays. Access control mechanisms allow content to be shared only with a given audience. When posting content to the site, users can choose which users or groups of people they want to share with.

There are different levels of information disclosure. **Private** content means that only the user that posted the piece of information to the SNS will have access to it. Other option may be sharing the content only with current **contacts**. An extended option may be using second degree relations, so sharing the content until **contacts of contacts**. Audience can also be picked up from groups or lists in **contact management settings**. For instance, Google+ allows choosing the audience of content from user-defined Google Circles. Another option is sharing with **specific users**, which may be chosen from existing contacts. Finally, content may be shared **publicly**.

The existence of different levels of content visibility may facilitate users to share more content. There are some sensitive pieces of information that would not be shared if the only option is publishing them to the public, but they might be shared with a restricted audience.

5.2.6 Ratings

In many social network sites, people are allowed to rate content and activities from other users. **Rating systems** take several forms, from like, binary like and dislike, 5 starts, etc. Rating systems are reviewed at Sparling and Sen [Sparling 2011]. This functionality is the base for popularity rankings. They can be used for building recommendations to users and promoting featured content.

5.2.7 Activities timeline

An activities timeline consists on a more-recent ordered list of actions performed by one or several actors. It may refer to posting some content, creating new friendships, rating content, etc. Examples of these actions are *"Alice uploaded a photo"*, *"Bob likes your comment"* or *"Charlie pushed three commits to a repository"*.

The activities timeline help users to be aware of the activity of others, and have a sense of what is happening in the social network. They can appear in several parts of the site, as we will see below.

5.2.8 Wall

We define a wall as the posting box that allows actors adding or creating any type of content to his own profile or to other actor in the SNS. The box is usually besides the activity timeline, so after the user posts a new content, a new activity appears in the timeline, a way of providing users with *instant gratification*.

The box usually supports posting several types of contents, such as text and files. It can also appear in several sections of the SNS, as we will see next.

5.2.9 Home

As users are authenticated, SNS may have a special page called home or dashboard. This is the first page offered to users after signing into the site. This page is used to present users a summary of the most important information in the user's social network.

Some features that may appear in the home page include a **wall**, where the user can post new content to the social platform, an **activities timeline** with the actions from the people in his network, and relevant **contact or content lists**.

5.2.10 Profile

Social network sites may offer a single web page per actor registered in the network. These pages are quite important, as they represent the external image that social actors project, they are the representation of users to the rest of the social network [boyd 2007]. They build a summary of their identity.

Profile pages may have the following sections:

- **Profile information** about the actor, such as location, age or birthday, contact details, etc. This may be a way to increase trust, helping users to recognize others and promoting interaction between them.

- An **avatar**. Actors can upload and crop an image that represents them in the website. This feature also may help to build trust, in the same way as the former.

- **Activities timeline** a list of more-recent ordered actions that includes only activities related to the profile owner. It provides a way to show a summary of recent profile owner's activity in the SNS.

- **Wall** allows the profile owner and other actors to post activities to him or her, and thus, improve communication between users.
- **Contact list**: a list of the contacts from the profile owner. It helps the network growing, letting the users to browse through the social network and finding more contacts.
- **Content list**: a list of relevant pieces of content shared with or by the profile owner.

5.3 Survey on popular SNS

16 top social network platforms were surveyed. The sites were chosen from Wikipedia's list of social networks websites [i], ordered by page ranking. We excluded non-English sites and sites that had restricted sign-up and needed invitation. Table 5.1 shows the lists of sites surveyed, along with their description, registered user number and Alexa's page ranking. SNS users are between almost 1 billion from Facebook to 10 million from SoundCloud. Page rankings are between 2nd from Facebook to 435th from Viadeo. SNSs' focuses include general, blogging and micro-blogging, photo sharing and art, business and music sharing.

Table 5.1 : List of surveyed SNS, along with their registered users and pange rankings. Source: Wikipedia

Name	Description/Focus	Registered users	Alexa's ranking
Facebook	General	908,000,000+	2
Twitter	General. Micro-blogging	500,000,000	8
LinkedIn	Business and professionals	160,000,000	12
Flickr	Photo sharing	32,000,000	48
LiveJournal	Blogging	17,564,977	115
Badoo	General Meet new people and dating	154,000,000	118
deviantART	Art community	22,000,000	131
StumbleUpon	Stumble through websites	20,000,000	146
Myspace	General	30,000,000+	161
Yelp, Inc.	Local Business Review and Talk		186
Taringa!	General	11,000,000	214
XING	Business	11,100,000	270
Tagged	General	100,000,000	288
SoundCloud	Music pieces	10,000,000	299
Orkut	General	100,000,000	319
Viadeo	Social Networking and Campus Networking	35,000,000	435

[i] `http://en.wikipedia.org/wiki/List_of_social_networking_websites`

Two user accounts were opened for the survey, using Gmail and Hotmail email providers. The survey was performed between July 10th and July 14th, 2012, and consisted in the following steps:

1. Go to the site's main page. Does the site allow signing up? Are there any other authentication services available to perform the registration?

2. After signing up providing an email, does the site allow performing actions? Or is email confirmation needed?

3. After signing up with both accounts, try to find the other user. Does the site have actor search? Can invite by email? Can use external agendas (email, other social networks) to search for contacts? If cannot search for the other account, or the search does not work, just paste the profile page from one browser to the other.

4. When the other account is found, add him as a contact. Does the site require confirmation from the other part (bidirectional) or not (unidirectional)? Can the contact be blocked? Is there any type of contact management available? Are privacy settings related to contact management settings?

5. What type of contents can be posted to the site? Can perform comments? Re-share? Tag or mention other users? Search for content?

6. Does the site provide system notifications? What about private messages, chat or video-conference?

7. When posting content, what type of access control is available? Options are: make the content private, share with all the contacts, with contacts of contacts (second degree), settings from content management, like user lists, share with specific users or make the content public.

8. Can users rate the content? Available options: like, like or dislike or any kind of scale

9. Is there a home page available? Does it include a contact list, content list, activities timeline or wall?

10. Is there a profile page? What kind of actor's attributes does it contain? Does it include an avatar? What about contact lists, content lists, activities timeline or wall (both in own profile and other's profile)?

11. Finally, is there any other type of social entities supported by the site?

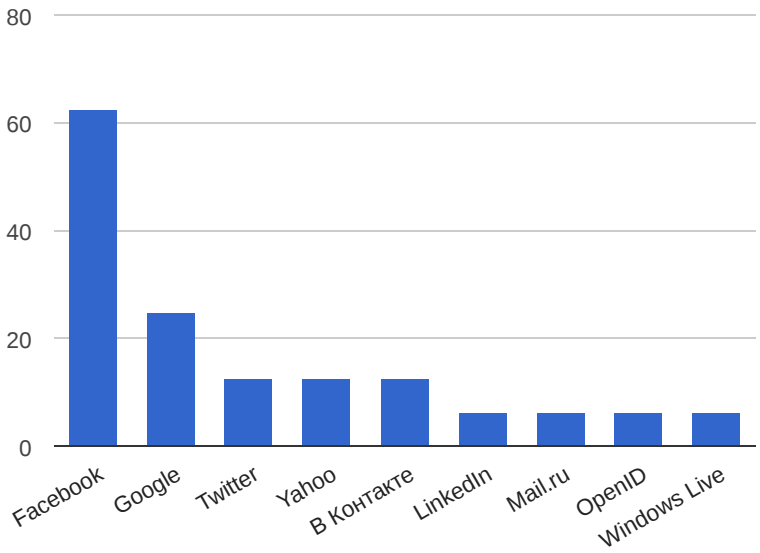

Figure 5.1 : Usage of external authentication services

5.4 Results of the survey on popular SNS

Every site allowed users to create an account. Some of the services provided the option of using **external authentication services**. Figure 5.1 shows the most popular services used by the surveyed social platforms. Facebook is the most popular service for external authentication, used by more than 60% of the sites analysed. Google follows, with more than 20%, then Twitter and Yahoo.

Regarding the **immediate use of the service**, the 68.75% of the analysed platforms allowed to perform operations in the site without confirming the email address. However, some of them restricted the actions, e.g. uploading content or establishing contacts. Others denied subsequent logins without the email account being confirmed.

Almost one third of the websites (31.25%) implement the feature of **invite external contacts** with a form by sending them an email. This figure contrasts with the high availability to **search in email agendas or other SNS** contacts. Figure 5.2 shows the support for searching in other services for contacts. Gmail is supported in 87.5% of the SNS for leveraging contact recognition. Yahoo and Hotmail services follow with 68.75% and 62.5% respectively. Note that other SNS such as Facebook is also used for this case in a 25% of the cases. Almost other 100 different services were supported by analysed SNS, notably Badoo and LinkedIn support more than 50 each.

Almost all the SNS analysed supports **searching for users**, being Taringa! the exception.

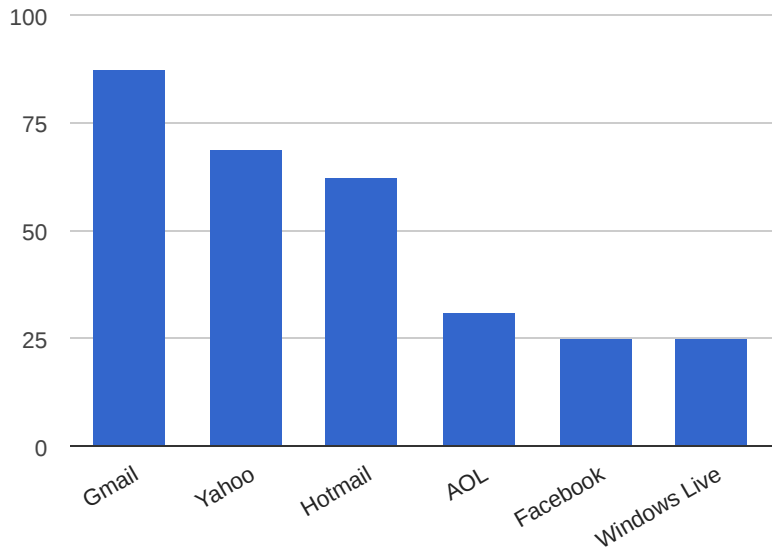

Figure 5.2 : Usage of external services to import friends to the SNS

Every SNS allows some kind of relationship between users. Yelp, Inc., was the only one that supports both unidirectional and bidirectional relationships, SNSs tend to support only one relation mode. Figure 5.3 shows the distribution of relation mode between SNS. **Unidirectional relationships are slightly more used by SNS**.

Some type of **contact management is supported in half of the cases** (50%). 25% of the sites provide their users with preset lists. Regarding permissions, privacy settings can be shared with contacts more than half of the cases (56.25%). Using specific content management settings for permissions is available in 12.5% of the SNS.

More than half of the sites (62,5%) support **blocking other users** among their features.

Figure 5.4 show the most popular **content types** that are posted at the begining of a thread. It excludes comments. Text posts are the most popular type (68.75%), followed by photos (56.25%), videos (43.75%), events and links (31.25%). There is a tail of other different type of contents supported by sites, depending of what the SNS is focused on. They include reviews, ratings, prints or games.

Comments do not appear in the graphic as they are not the start of the **thread**, but they are supported by every SNS. Reshare contents from other users is supported by half of the sites (50%). On the other hand, **tag or mention users** is supported by more than third of the SNS (37.5%). **Content search** is supported by every SNS.

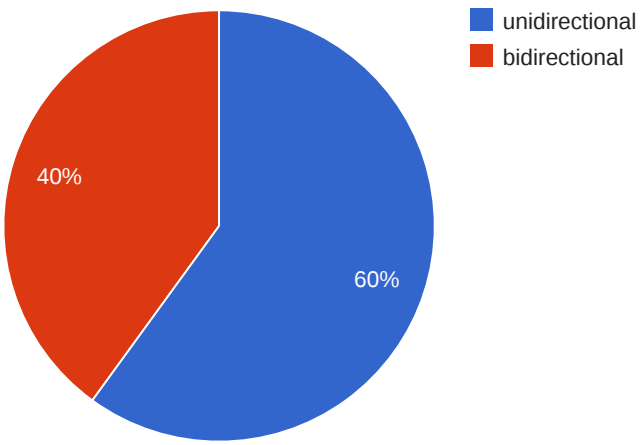

Figure 5.3 : Support for relationship types in SNS

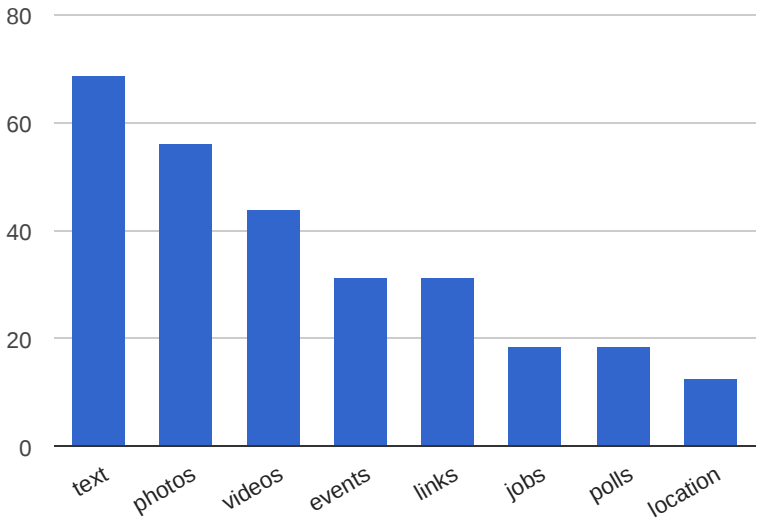

Figure 5.4 : Most popular first-post content types managed by SNS

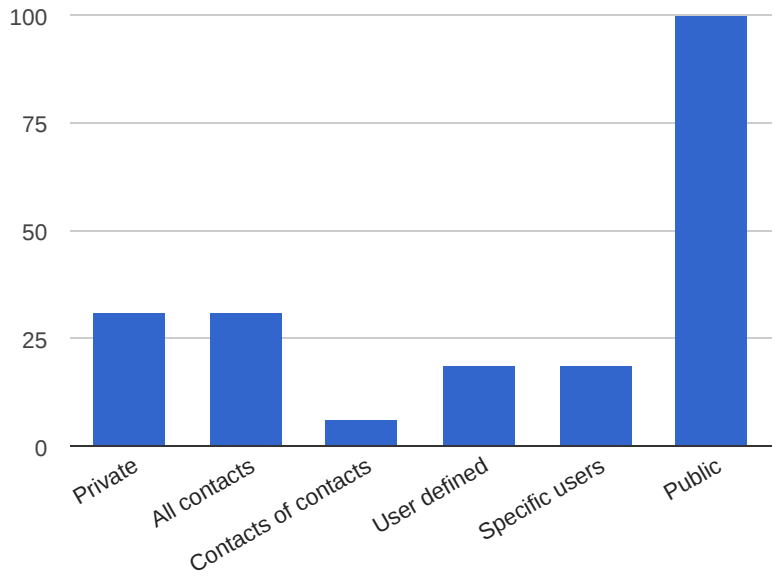

Figure 5.5 : Content privacy settings supported by SNS

Regarding communication tools, **system notifications** are present in every SNS. **Private messages** are supported in 93.75% of the sites, only not supported by Orkut. **Chat** is not so popular, supported only by 18.75% of the sites. Finally, **video-conference** is not present in any site.

Regarding **content visibility**, there is an irregular distribution between the available options. Figure 5.5 shows the popularity of options by SNS. Uploading content and keep it private, and sharing only with direct contacts both features are supported by 31.25% of the sites. Sharing with contacts of contacts is only supported by Facebook. Sharing with user-defined contact lists, or with specific-users is supported by few sites. Finally, **every SNS supports public sharing**.

Unary **rating** is the most popular one; it is supported by 68.75% of the sites. Like and dislike is only supported by one site. There is also only one case with a scale of 5.

Regarding the **home page**, this feature is present in almost every SNS, being devianART the exception. Figure 5.6 shows features present in home pages. A list of, or at least a link to, contacts and content is present in more than half the SNS, while the timeline is much more usual (80%). Finally, posting content from the homepage is well supported, almost as much as the activities timeline.

Every SNS has a profile page per user. Figure 5.7 shows popular features in profile pages. **Profile attributes** varies much from one site to another, with an average of 2 attributes per profile,

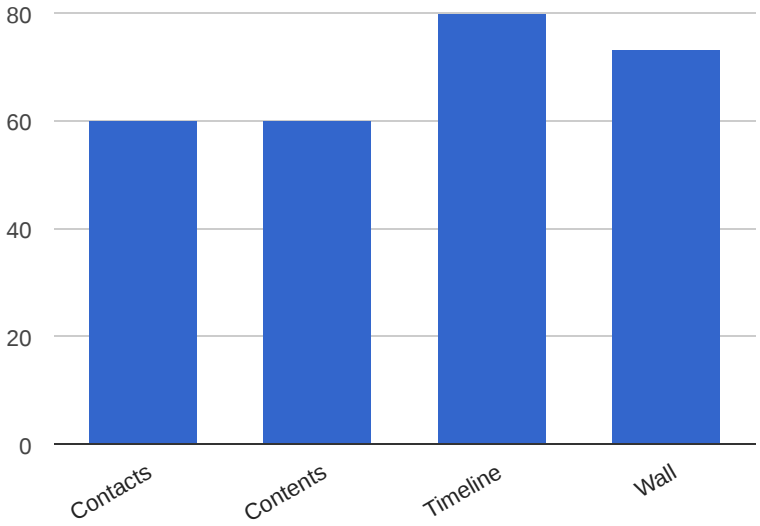

Figure 5.6 : Home page features present in SNS

and *location* being the most common. **Avatar** support is outstanding, being present in every site. **Contacts and contents lists** are also quite present in profile pages. Activity timeline is less popular, but still has a 75% of present in sites. Posting in self profile is less supported, only by 56% of the sites. Finally, posting to other users in their profile is only supported by 25% of the SNS.

Finally, figure 5.8 shows other types of social entities supported by SNS. Groups are the most popular social entity, being present in almost half of the sites (43.75%). Some sites like Facebook and XING support two types of social entities at a time (groups, pages and groups, companies respectively).

5.5 Discussion

Results show a list of functional features supported by every SNS. These features include user account creation and relationship establishment, as it was expected by the social network definition. This is also consistent with the idea that SNS base identity building and network creation on authentication. Other features present in every SNS are comments, some type of content publishing with public visibility, system notifications and a profile page with an avatar. Features present in almost every social network include user search, private messages and a home page. These results contribute to the set of features mentioned by boyd and Ellision [boyd 2007].

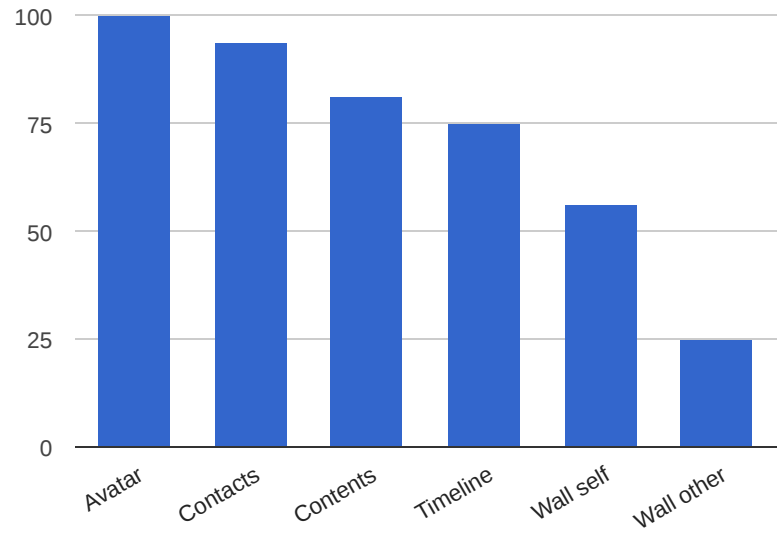

Figure 5.7 : Profile page features present in SNS

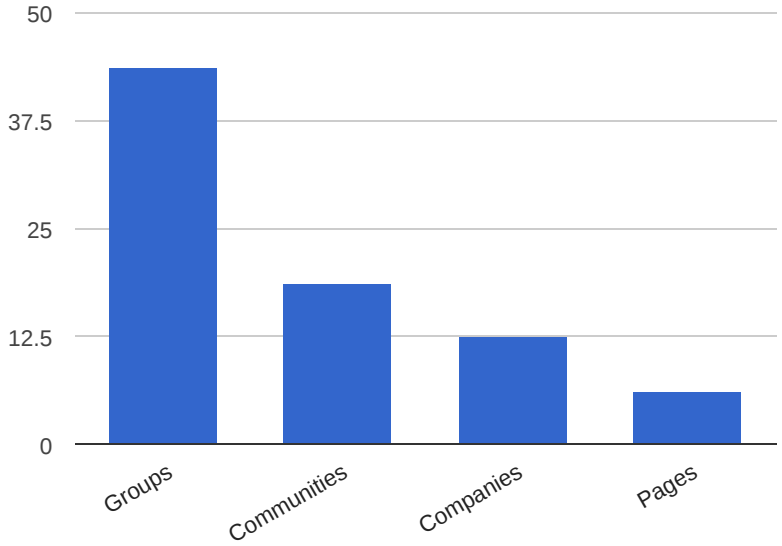

Figure 5.8 : Social entities supported by SNS

A very strong tendency to delegate authentication have being found. Being present among the top popular sites, which fight between each other for a higher user base, this feature was expected in not so popular platforms, which can take more advantage of it to leverage authentication.

Contact recognition using external services is other popular feature, with Gmail support present in 87.5% of the SNS. Results on Facebook's friend recognition may be higher if authentication were performed using Facebook, instead of using an email-based sign up.

Regarding relation types, unidirectional and bidirectional options are both almost equally popular. Blocking relation is significant, present in more of the half of the cases.

Another significant finding is the presence of comment support in every SNS. Content types have popular ones, such as text, images and videos, and other types are more specialized. These types are expected to be more diverse as analysing other SNS, as this sample is biased towards the most popular, general purpose SNS. This reinforces the opportunity for building social network frameworks that support easy creation of niche-specific SNS platforms, which would support different specific contents.

Regarding privacy access control, it is significant that every SNS support public sharing. Regarding other type of settings, there is a distribution of available options, but with limited support (25%), being significant the role of Facebook in this field.

Unary rating (like) is well supported along surveyed SNS. The same cannot be said of other types of rating, whose presence is anecdotal.

The analysis also reveals home pages as a place for publishing contents and following activity. On the other hand, profile pages are the place for content and contact lists, and less frequently activity timelines and walls.

Finally, other social entities such as groups and institutions are supported in half of the cases, not as many as their role in society might suggest.

5.6 Conclusions

This chapter introduces a formal study on social network features present in popular SNS. The analysis show some features being present in every site, i.e. user registration, relationship establishment, comments, public sharing of contents, system notifications and profile pages with avatars. User search, private messages and home pages are also very popular.

There is a strong tendency to use external services for authentication and contact recognition, despite the analysed sites compete with each other for user base.

Relation types are almost equally distributed between unidirectional (follow) and bidirectional (friend) contacts. Contact management and private content access control is not very popular.

There is a wide set of supported content types, being text, pictures, videos, events and links the most popular ones in that order. The set is probably biased by the sample, appearing more content types if more SNS were analysed.

The home page is primary a section to get the updates from followers and posting new things, whilst profile pages are the place to show relevant contents and contact lists.

These features define the requirements for a social network framwork.

Chapter 6

A Framework for Building Social Network Sites

> *Computer programming is tremendous fun. Like music, it is a skill that derives from an unknown blend of innate talent and constant practice. Like drawing, it can be shaped to a variety of ends - commercial, artistic, and pure entertainment. Programmers have a well-deserved reputation for working long hours, but are rarely credited with being driven by creative fevers. Programmers talk about software development on weekends, vacations, and over meals not because they lack imagination, but because their imagination reveals worlds that others cannot see.*
>
> *Larry O'Brien and Bruce Eckel, Thinking in C#*

6.1 Introduction

The success of social network sites (SNS) is not only restricted to *contact-oriented* social platforms, the websites focused on establishing and maintaning contacts. Social networking features are also present in *content-oriented* social networks, the ones that are organised around some type of content: trips, code repositories or conference attendance. These sites are also leveraging social features to gain in user base and popularity.

This scenario provides a wide variety of websites that can be enhanced and which can leverage the usefulness of social networking.

In the same way than content management frameworks (chapter 4), it seems quite interesting providing practitioners with a social network framework (SNF), a software with generic functionality to build social network websites easily and quickly. This framework aims to alleviate the overhead associated with common features present in social network platforms.

This chapter gathers the features described in chapter 5 to establish a set of requirements that a SNF should fulfill. Next, it introduces Social Stream[i], a social network framework for Ruby on Rails, the framework for building websites described in section 2.1.2. There is an overview of Social Stream's components that support those requirements. Finally, we present the case of use of Social Stream as contact-oriented network, besides its application to the Virtual Science Hub (ViSH), an excursions content-oriented social network platform in which this framework has been validated, along with the appropiate code reuse measures that assess its validity.

---

[i] `http://social-stream.dit.upm.es/`

6.2 Requirements for a social network framework

In chapter 5, a comprehensive list of common social network features present in popular social network platforms was introduced. They are summarized here, along with the actions a SNF should take to support them.

- **Social actors**, every user must be able to register an user account and log into the site. External services may be used for authentication and contact import. A SNF should provide an user account system together with hooks to connect with external authentication services. Additionally, it should provide ways to support several actor types besides users, such as groups, organisations or companies.

- **Social relations**, different types of relations can be found in SNS. Unidirectional (e.g. follow) do not need confirmation from the other party, while bidirectional (e.g. friend) do need confirmation. Other type of relations between social actors are also used, such as blocking other users. Futhermore, users might be able to manage their contacts, e.g. creating lists in their own terms (*buddies*, *working collegues*, etc.). A SNF should be flexible enough to let developers build their applications with any type of relations: unidirectional, bidirectional, system-defined or user-defined.

- **Content**, multiple types of content are managed by SNS. Examples include *text posts*, *pictures*, *videos*, *events* and *links* to external sites. The SNF may provide support for the most common content types. Besides, *content-oriented* social network platforms will probably be interested in an specific kind of content, such as trips or code repositories. The SNF must let developers to focus on the specific content each application demands. It must provide a base for those new content types to seamlessly integrate with the rest of the social features.

  Other popular features related to content include linking content pieces to users through *citations*, *mentions* or *taggings*. Unary *ratings* are also very popular. Binary or other type of ratings may be supported as well. The SNF should support these social features.

  Content visibility supports several options, i.e. keeping the content private, sharing with contacts, with contacts of contacts, with user defined list of contacts, sharing individually with specific users, only with registered users and sharing the content publicly, being the last the most popular. The SNF should support the public option, and as many of the other options as possible.

- **Activities timeline** is the stream of more-recent ordered actions performed by social actors, such as *"Alice uploaded a picture"*, *"Bob likes Alice's comment"* or *"Charlie changed his avatar"*. The activities timeline appears in several contexts, each one showing different activities. The most popular are the home activities timeline, made up of all the activities

in which the actor's followings are involved, and the profile activities timeline, made up of the activities related to the profile owner. The SNF must support activities generation from multiple events (e.g. contact establishment, content posting, ratings) and displaying the suitable set of activities in various contexts (home and profile, at least).

- **Home page** is the section when actors can access the more relevant information in the SNS. Sections appearing in the home page usually include the home activities timeline, a form for posting new content, a list of their closer contacts and relevant content items. The SNF must allow developers composing home pages with the different modules mentioned. In the case of new content types, it should be easy to develop new posting forms than integrate with the other types of content, if required.

- **Profile pages** are present in every SNS. They are a place for social actors to present an image of themselves to the rest of the community. Profile pages usually include an avatar, information attributes such as location or interests, an activities timeline that gathers actions related to the profile owner, a wall for other users to post content to the profile owner, the contact list associated with her and relevant content items from her. The SNS must provide support for uploading avatars and managing profile attributes. As in the case of the home page, developers should be able to easily arrange elements to fit a custom layout, while reusing code.

- **Notifications and private messages** seems to be two popular features to be supported as well. The SNF should provide them.

6.3 Social Stream, a social network framework

Social Stream is a Ruby on Rails library that supports the required features described above.

Figure 6.1 describes the main modules provided by Social Stream. It is based on two core components, **actor** and **object**. Both are polymorphic supertypes [Pierce 2002]; the functionality on these supertypes is valid for every subtype defined in each category. An example of supertype is *bird*. Subtypes of *bird* are *duck*, *cuckoo* and *ostrich*. Introducing supertypes in the framework provide final applications with the ability to define multiple additional supertypes that can seamessly integrated with the social network features.

6.3.1 Actors

Actor is the supertype for any social entity defined in the SNS. Actors subtypes have three main characteristics; a profile, with an avatar and other attributes, social relations with other actors and the ability to perform actions on objects.

Social Stream provides two prevalent subtypes of actors: user and group. All the code functionality on profile, social relations and actions works interchangeabily with user, group or

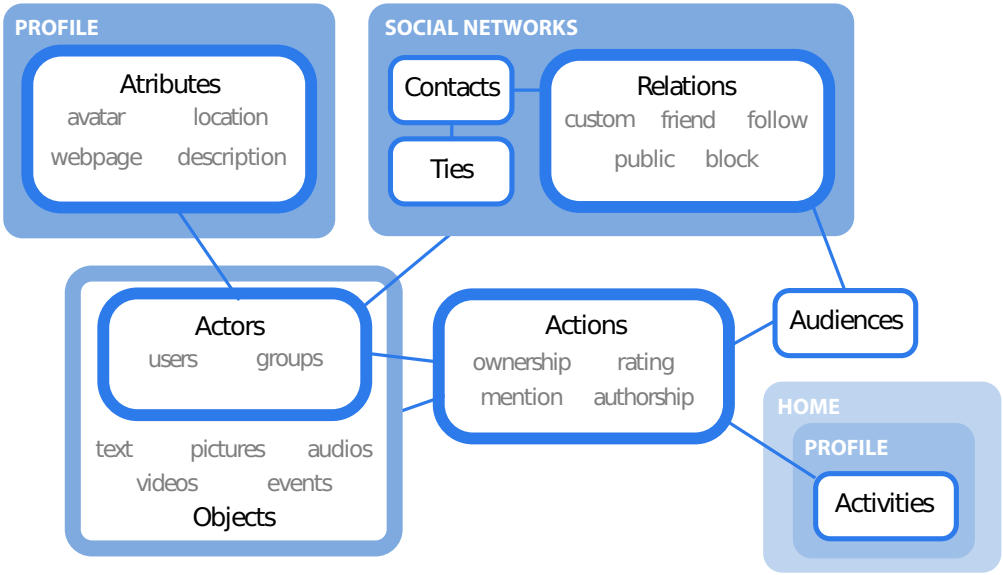

Figure 6.1 : Architecture components provided by Social Stream, a social network framework

other actor the developer may define.

In addition, Social Stream includes user authentication through the external library Devise[i], and authentication with external services through Omniauth[ii]

6.3.2 Profile

An avatar or picture represents the social entity. In the case of users, it is usually a picture of the person. In the case of other entities it is usually a logo. Social Stream provides support for avatars through the library *Avatars for Rails* [iii]

Common profile attributes present in all social entities include name, contact email, location, address, description, website, etc. All actor subtype instances have a profile entity attached that gathers these attributes.

6.3.3 The social network

The social network model built in Social Stream is powerful enough to support unidirectional (e.g. *follow*), bidirectional (e.g. *friend*), system-defined (e.g. *block*) and custom, user-defined relations.

[i] `https://github.com/plataformatec/devise`

[ii] `https://github.com/intridea/omniauth`

[iii] `https://github.com/ging/avatars_for_rails`

It is composed by three entities:

Contact

It represents an ordered pair of actors. For instance, the contact from Alice to Bob *{Alice, Bob}* is different than the contact from Bob to Alice *{Bob, Alice}*

However, the presence of a contact instance does not implies there is a established relationship between both actors. There must exist a tie for that, as we will see below.

Relation

It represents different modes of relationships between actors, such as *friend*, *follow*, *reject*, etc. Social Stream provides the following subclasses of relation:

- **System defined relations**. These are described by the application developer. A relation instance is unique for all the ties established in the system. Examples of these relations are *follow* or *reject*. These relations cannot be customized by users.

- **Custom relations**. These relations are defined by users. They can create them and give them convenient names. These relations can only be used to establish ties from the user that created the relation. In other words, for a tie to be valid, the first actor in the contact must be the same actor that created the relation.

Tie

It records the presence of a given relationship between two actors. It is made up of two elements, i.e. a contact and a relation.

When Alice adds Bob as friend, a new tie is created with the contact *{Alice, Bob}* and the relation *friend*, which in this case is user defined. The resulting tie is *{Alice, Bob, friend}*. Because *friend* is user-defined and *Alice* is establishing the contact, *Alice* must be the owner of relation *friend*. If *Alice* rejects *Charlie* as contact, the system-defined relation *reject* would be used. The tie *{Alice, Charlie, reject}* will be recorded in the SNS.

Let us note that there can be several ties per contact. In the case *Bob* creates two custom relations, *buddy* and *work colleague* and adds *Alice* with both of them, two ties will be created, sharing both of them the same contact *{Bob, Alice}*, but each one with a different custom relation: *{Bob, Alice, buddy}* and *{Bob, Alice, work colleague}*.

Ties have permissions attached to them through relations. This way, actors are able to grant permissions to their contacts. This novel access control model is called Tie-RABC and is further explained in chapter 7.

6.3.4 Objects

Object is the second supertype introduced by Social Stream's architecture. It represents every entity that can be object of an actor's action, and thus appearing in activities timelines.

Social Stream provides the following popular subtypes of object: *text posts*, *files*, *images*, *audios*, *videos*, *events* and *links* to external websites. Actor is also included in the list of subtypes, because every actor can be a suitable object of activities, for instance in *"Alice added Bob as contact"*.

Objects also have common attributes such as `title` and `description`.

6.3.5 Actions

Action is the entity that gathers all the interactions between actors and objects. Authorship, ownership and rating are currently supported by Social Stream. Other features such as mentions fall in the category and will probably implemented in the framework or as external plug-ins.

An action instance is made up of the actor, the object and a set of columns for each feature:

- **Authorship** a boolean attribute indicates if the actor is the one who created the object. When an user uploads a picture, for instance, a new action is created. This action includes the user, the picture and has the boolean attribute of authorship set to true.

- **Ownership**; a boolean attribute indicates that the object was posted to actor's wall. When Alice posts a picture to Bob, a new action is created between Bob and the picture. This action has the boolean ownership attribute set to true.

- **Rating**; an integer attribute indicating the score some actor gives to some object. Until the actor performs the rating action, it may be set to null. Afterwards, it is set to the suitable value. An integer attribute supports several rating systems, such as like, like-dislike and likert scales.

- **Mention** another boolean attribute records that the actor has been tagged in a picture or mentioned in a text post.

  Figure 6.2 shows the set of actions in the context of the SNF.

6.3.6 Audiences

Social Stream content visibility model is based on relations. When actors upload a new object, they choose the relations to share with. For instance, *Alice* uploads a picture to her wall. She chooses the *friend* relation for sharing. The picture will be available to all her friends, in other words, to every actor that has a tie from *Alice* with the *friend* relation.

Social Stream provides a system-defined *public* relation for sharing content publicly.

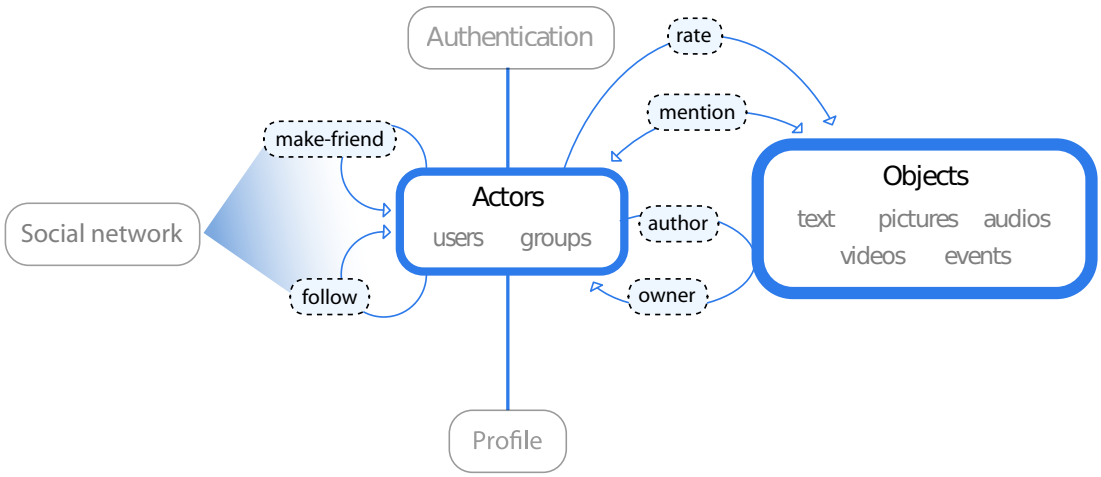

Figure 6.2 : Set of actions in the socal network framework

This model covers sharing with all the contacts, with user defined lists of contacts, and public content visibility options.

6.3.7 Activities timeline

The activities timeline component follows the Activity Streams standard, introduced in section 2.2.2.

Activities are made up of a subject, who is the author of the activity, a verb, (e.g. *post*, *make friend*, *like*) and an object. It may also include indirect objects or context.

The activities timeline has different views depending on where it is displayed and to whom. Social Stream provides queries for home and profile timelines. Besides, there is an authorization layer. Not all activities can be seen by all followers or profile checkers. For instance, a user posting some content to all her contacts should not be accessed by third-parties. Some activities must be filtered depending on who is viewing the web page.

Activities are recorded in the following events:

- A new tie is established, an example activity would be *"Alice added Bob as contact"*
- A new object is posted, such as *"Alice uploaded a picture"*
- A new rating is performed, such as *"Charlie likes Alice's picture"*

6.3.8 Home and Profile pages

Social Stream provides home and profile pages all to every actor subtype instance. Every page is built using modules, so developers are able to recombine them as needed. In addition, a template convention facilitates building forms for new content, just creating a file with required fields.

6.3.9 Private messages and notifications

Social Stream includes a messaging and notification component provided by Mailboxer[i] library, which was developed by Eduardo Casanova as part of his Master's Thesis [Casanova 2012].

6.4 Social network websites built with Social Stream

Social Stream has been published as a free and open source software. In its website[ii], the SNF is introduced and resources for developers are available. These include screenshots of the default look-and-feel of the framework, a link to the developer's forum, source code and documentation. The source code is uploaded and managed in Github[iii], along with an issue tracking and a wiki. The documentation is generated along with the code in new versions and it is available at Rubydoc[iv].

Social Stream is divided into several components. The reason is providing developers with the right modules suitable for the needs of the specific SNS to be built. Social Stream components are:

- **Base** provides with the basic functionality for building a SNS: the social network, the activities timeline, private messages and notifications. It provides two subjects, i.e. user and group, as well as two objects, i.e. post and comment.
- **Documents** introduces four new objects that are related with file management, i.e. document, picture, audio and video. It also provides an HTML5 player to play audio and video objects.
- **Linkser** adds external links to the list of social objects. Actors are able to paste URL to the posting box, which are dereferenced and the remote content is displayed for posting. This is quite attractive in the case of Youtube videos, which are embeded in the activities timeline.
- **Events** introduces event as new social object. Actors are able to create events in their calendar, as well as suscribing to them in other actor's calendars.

[i] `https://github.com/ging/mailboxer`

[ii] `http://social-stream.dit.upm.es/`

[iii] `https://github.com/ging/social_stream`

[iv] `http://rubydoc.info/gems/social_stream/frames`

- **Presence** provides with chat support to the SNS. It requires an external XMPP server for managing connections. This component was developed by Aldo Gordillo as part of this Master's Thesis [Gordillo 2012].

Social Stream has received external contributions. It has more than 30 contributors so far, ten of which belong to the local university group. However, more than half of the commits have been made by the author of this dissertation[i].

The SNF is available as a Ruby gem, a popular package system used by Ruby developers and the main mean to distribute Ruby libraries. Social Stream has reached 85,000 downloads and it is used for building several SNS. The most relevant cases are described below.

6.4.1 Social Stream's experimental platform

Social Stream is a Ruby on Rails engine, a kind of Rails library that contains a full application that can be mounted and accessed from the main web application. Thus, the SNF is accessible and provides a fully functional SNS out-of-the-box.

A demo deployment[ii] of Social Stream is available for testing and taking a quick overview of Social Stream's capabilities. 1833 users and 268 groups have registered in this site so far. It is a contact-oriented SNS. Uploaded content is shared with all the user's contacts by default. User-defined lists of contacts and public content are also available. Contacts are established using custom relations, in the same way Google+ works. Besides, other system relations such are *reject* are also available. Supported content types are the native from Social Stream, i.e. *text posts*, *files*, *pictures*, *audio*, *video*, *events* and *external links*.

6.4.2 Piglobe

Social Stream has also been deployed to the Piglobe[iii] in the context of the cultural centre "*La Piluka*"[iv], in a popular neighbourhood of Madrid, Spain.

The Piglobe has little changes on the original Social Stream's distribution, mainly focused on the site appeareance. Colors, logos and some of the applications messages has been customized. The rest of the functionality remains the same.

The Piglobe has reached 88 users and 14 groups. It has been a valuable source of feedback on the usability of the SNF. It also contributes to demonstrate the flexibility of the SNF building general contact-oriented SNS as well as focused content-oriented SNS, as the case of the ViSH described below.

---

[i] `http://www.ohloh.net/p/social-stream/contributors`

[ii] `http://demo-social-stream.dit.upm.es`

[iii] `http://piglobe.lapiluka.org/`

[iv] `http://www.lapiluka.org/`

6.4.3 Virtual Science Hub (ViSH)

Social Stream has been used to build the Virtual Science Hub (ViSH)[i], an excursions content-oriented SNS where scientists and teachers are able to exchange and establish collaborations, and where pupils are able to experience real e-science applications in areas of high relevance for the future, such as nano- and biotechnologies. The ViSH is developed in the context of the Global Excursion project[ii], funded by the European Commission under the FP7.

The ViSH uses some of Social Stream's components, i.e. base, documents and linkser. Most of its available capabilities have been reused in the ViSH. These modules include user account registration and sign-in, external authentication services, avatars and profile management, private messages and notifications. Furthermore, other features has been improved in the ViSH, starting from the functionality Social Stream was already offering. This is the case of Social Stream's file management, which is intensively used by the ViSH for resources management. The interaction has also been improved, changing some of the views in the platform.

One of the key parts that demonstrate Social Stream's flexibility and potential is the introduction in the ViSH of new object subtypes: *excursions*, *slides* and *embeds*. One excursion is made up of one or several slides, which includes one or several resources. Embeds provides with the capability of managing HTML embed objects, such as Java applets or Flash objects. Due to Social Stream, development has been focused on the definition and development of these types, achieving a seamlessly integration with existing social features. Authors, owners, ratings and timeline integration of the new object subtypes were given by the framework without additional work from developers.

In the ViSH, all the content is shared publicly. However, a new system-level relation was defined: (*follow*). This relation converts the ViSH into the Twitter-like relation model. The system relation *reject* is used as well.

Figures 6.3, 6.4, 6.5 and 6.6 show screenshots comparing the home and profile pages from Social Stream's experimental platform and the ViSH. They show the power of Social Stream for building totally different, contact- or content-oriented SNS.

Code reuse in the ViSH

Measures of the code reuse in the ViSH assess the success of Social Stream as SNF. Measurement metrics are the same as in the case of Station, described in section 4.5. Social Stream is a Rails engine. Models, controllers and views are visible from the final application, and can be used by it without any modification or verbatim use. Leveraged use include components provided by Social Stream that were modified in the ViSH.

[i] `http://vishub.org/`

[ii] `http://www.globalexcursion-project.eu/`

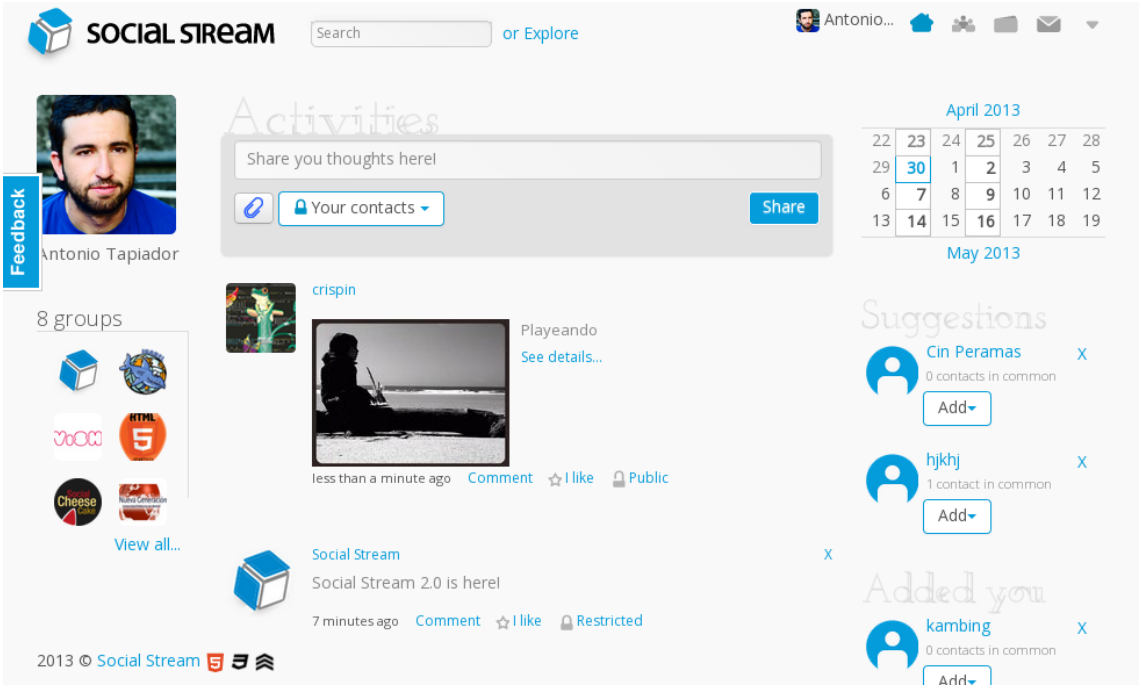

Figure 6.3 : Home page in Social Stream's experimental site

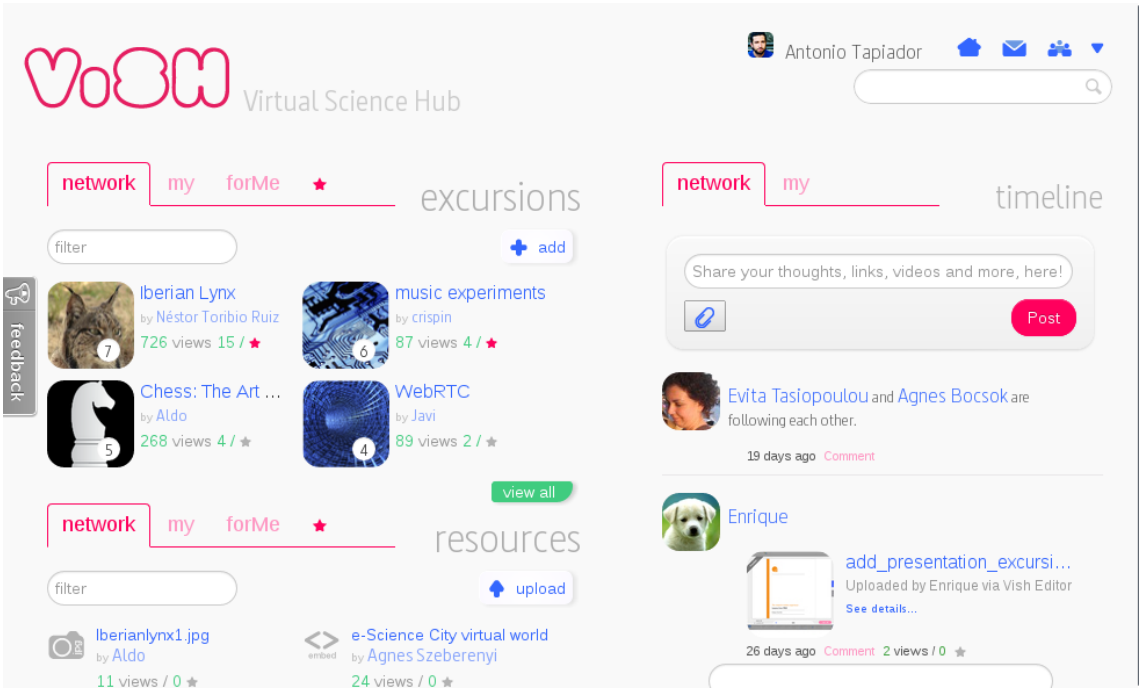

Figure 6.4 : Home page in the ViSH

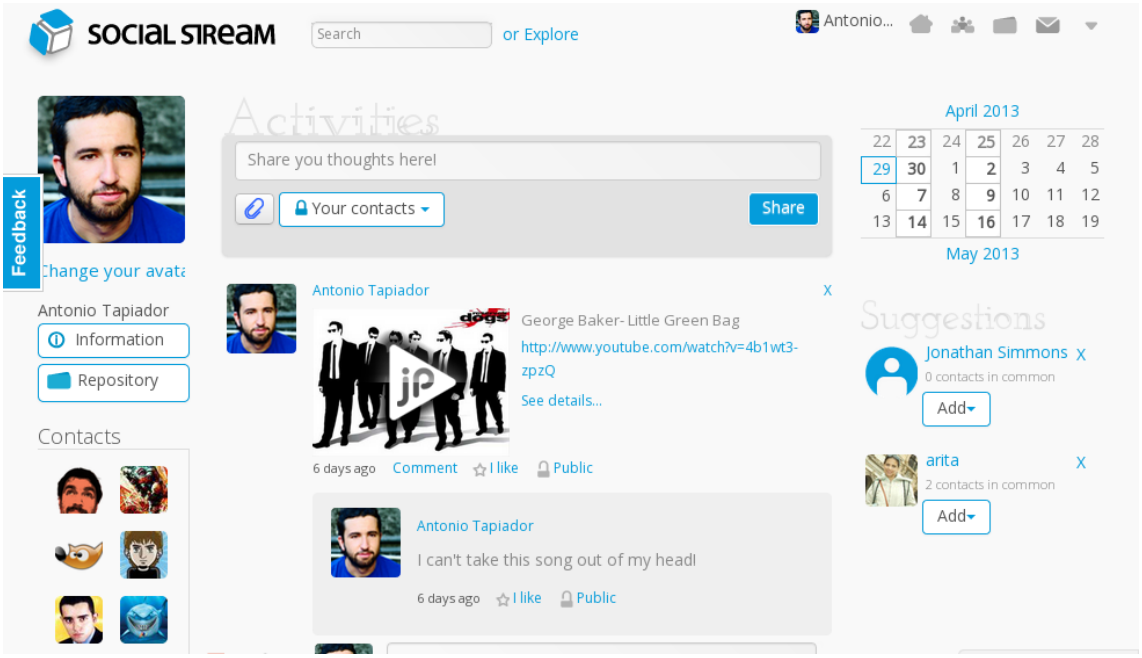

Figure 6.5 : Profile page in Social Stream's experimental site

Figure 6.6 : Profile page in the ViSH

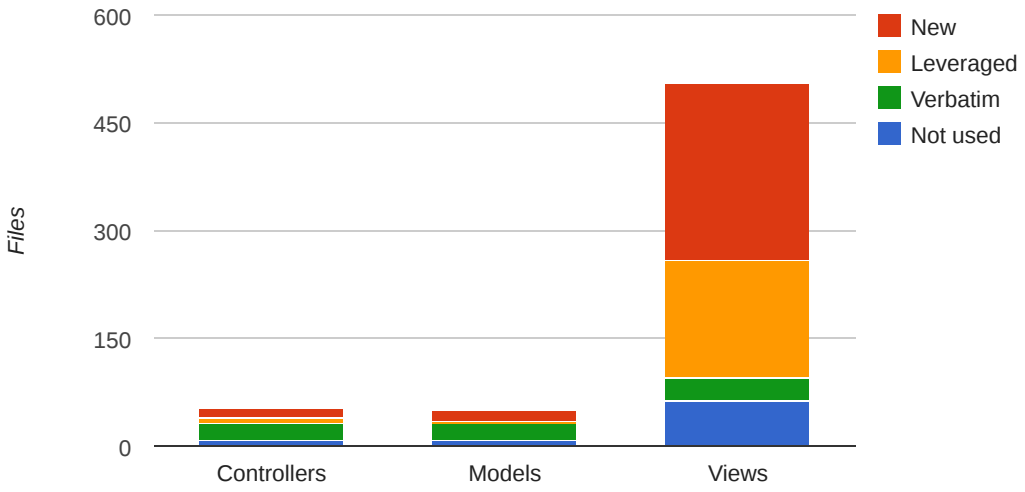

Figure 6.7 : Code reuse between the ViSH and Social Stream, models, controllers and views

Table 6.1 shows the results. Contrary to results in section 4.5, most of the code in controllers and models has been verbatim reused (C_V = 51.16% and 54.54%), without any amount of effort from developers. Besides, there is a notable leveraged use of components that sum to the usefulness of the framework (C_L = 18.60% in the case of controllers). The amount of reusability was significant less in the views, where most of the reuse is leveraged (C_V = 7.04% and C_L = 37.04%).

Table 6.1 : Measures of code reuse in Social Stream and ViSH

	Social Stream		ViSH						
	Total	Not used	C	V	L	N	C_V	C_L	C_N
Controllers	39	9	43	22	8	13	51.16%	18.60%	30.23%
Models	34	7	44	24	3	17	54.54%	6.81%	38.63%
Views	258	64	440	31	163	246	7.04%	37.04%	55.90%

Figure 6.7 shows the amount of models, controllers and views from the ViSH provided by Social Stream, while figure 6.8 shows a comparison of code reuse normalized to the amount of files each in controllers, models and views.

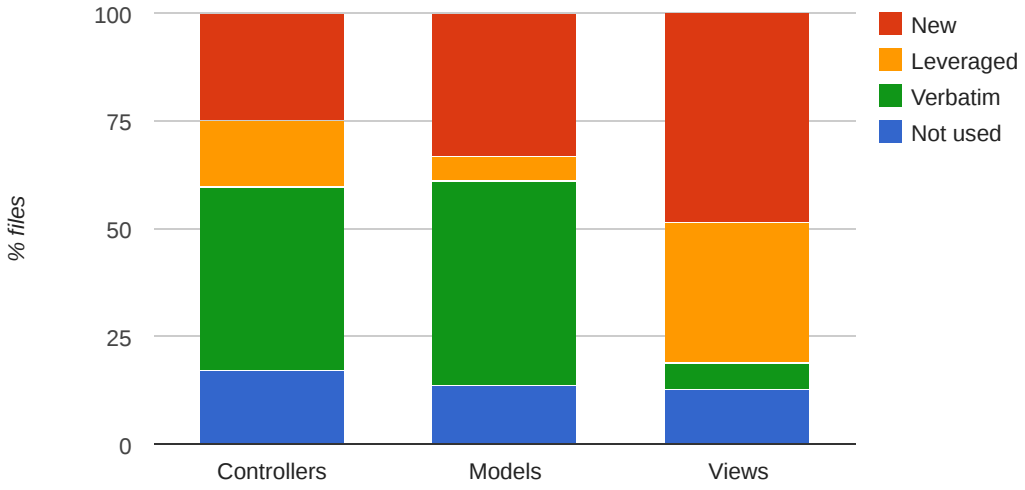

Figure 6.8 : Code reuse between the ViSH and Social Stream, comparison in percentages of files between models, controllers and views

As in section 4.5, these results do not take other kind of code reuse into account, as with libraries. For instance, excursions include the object code necessary for integration in timeline and other social features, which is known as leveraged reuse. An study on this would surely enforce the usefullness of the SNF introduced in this chapter.

6.4.4 Other social networks

Social Stream is been used in several networks beyond the ones introduced here.

At least one case is confirmed and publicly available. The production website WotWentWrong [i] is a community about relationships featured in the mass media built on the top of Social Stream.

The increasingly active community both in the forum and in Github points that more sites are using Social Stream as the base for building social network websites.

6.5 Discussion

The SNF has proved to be very effective for bulding SNS. However, there are still some issues to be solved.

[i] `http://wotwentwrong.com/`

Currently, Social Stream architecture model only supports sharing with relationships. This model should be extended to support other content visibility options, such as contacts of contacts or specific users.

Regarding the activities timeline, we have found two approaches when building it:

- **Lazy approach**. In this approach, all the information is just registered. The activities are selected in the moment the activities timeline's view is rendered. The advantage of this feature is that is dynamic and can be adapted to network changes. For instance, *Alice* publishes a post to her contacts. Later on, *Alice* adds *Bob* as contact. When *Bobs* sees *Alice*'s profile, the post will be available for him, although the contact was established after the post was shared. Another advantage is that writing new activities is a cheap operation in terms of computation. The drawback of this approach is that reading is more expensive. When calculating the home profile, the SNF must get the followers and then apply privacy restrictions.

- **Eager approach**, activities are preprocesed and timelines are build just after the activity is generated. In this approach, read is cheaper, because the records are already calculated and ready for delivery. However, in the case mentioned before, *Bob* will not see *Alice*'s posts, because the post generation activity was delivered to *Alice*'s contacts before *Bob* was among them. Another drawback is that writing is more expensive. Deliveries must be calculated and written to the persistance layer in the time the activity is created. A performance improvement can be achieved using background jobs to deliver activities to timelines, dettaching them from the main thread of the web sever. However, this approach brings a penality in the system responsiveness to users, because activities may take some amount of time to appear in actors' timelines.

Social Stream currently supports the lazy model for building the activity timeline. More experience and benchmarks should be required to prove which of the models is more effective for the management of the activities timeline.

Finally, there is a notable difference in code reuse results between models and controlers in the one hand, and views in the other. The next version of Social Stream will be based on Bootstrap [i], an easy framework for views development and Sass [ii], a technology for object-oriented CSS. These technologies will allow uncoupling the HTML from the views from CSS, providing more customization possibilities without changing the views, which should revert in a higher reuse rate in the views.

[i] `http://twitter.github.com/bootstrap/`

[ii] `http://sass-lang.com/`

6.6 Conclusion

This chapter introduces Social Stream as a succesful SNF for building SNS. It presents the framework architecture, which includes two supertypes, i.e. actor and object. Actor is the supertype for every social actor and provides with a profile, social network and actions on objects. Object is the supertype of any content, e.g. text, documents, photos, videos, events and external links.

Social Stream is a free and open source Rails engine that implements this architecture. It is gaining a growing community and being used beyond the scope of this work. Social Stream's architecture is flexible enough to support the customization of the embeded social network it provides to a content-oriented (excursions) one, the ViSH. It uses some of the components from Social Stream, i.e. base, documents and linkser, taking advantage of its modularity.

Most of the features of the actor subtype were reused without any additional work: user account registration and sign-in, external authentication services, avatar and profile management. The social network module supports either user-defined list of contacts (present in the demo site and Google+) and follow relation model (present in ViSH and Twitter). Regarding the object subtype, most of Social Stream content types could be reused. Besides, two new content types were defined (excursion and slides), which were seamlessly integrated with the social features (activities timelines and actions, such as authorship, ownership and ratings). Home and profile pages could been customized to high degree, reusing most of the modules such as profiles and activities timelines.

The code reusability metrics at the Rails engine level show how most of the ViSH's code is provided by Social Stream, which probes the validity of the SNF.

Chapter 7

Tie-RBAC, an application of RBAC to social networks

> *They who can give up essential liberty to obtain a little temporary safety, deserve neither liberty nor safety.*
>
> *Benjamin Franklin*

7.1 Introduction

Privacy in social networks is a quite controversial topic. We are living the time in human history with more public information from individuals available. We are probably assisting to a redifinition of the boudaries between the private and the public. This turns privacy into a fascinating research field, not only for human sciences, but also for computer scientists.

There is literature on how user privacy expectations in SNS do not match with real settings. Liu et al. measure the disparity between the desired and actual privacy settings in Facebook, finding that privacy settings match users' expectations only 37% of the time [Liu 2011]. We also have found that proposed access control models, reviewed in section 2.6.5, have several limitations.

Classical access control in social networks are too coarse. They allow sharing resources with friends, friends or friends or public, but they do not allow other type of fine grained policies, sharing with explicit users, for instance. Besides, using the friend relation for access control forces users to decide between adding users to be more popular, or protecting their privacy.

They do not support managing permissions for several contacts at the same time. DAC models support granting individual permissions to subjects, and this is the case for current access control model for social networks. For example, *Alice* may want to move *Charlie* from her friends group to acquaintances, revoking some of the permissions in the same way.

Besides, current solutions do not support system-defined and user-defined policies at the same time. Like RBAC, we want the access control model to be flexible enough to be policy neutral and be able to define policies from both approaches.

Users often does not use the word *friend* to express the relationship with other partners [Carminati 2009]. Other words, such as *colleague*, *classmate*, *business partner* or *acquaintance* are used. This is the same for organizations, which define positions such as *administrator*, *operator*, *organizer*, or *participant*. Ties are also established between social entities of different type, not only between two users or two organizations, but also between an organization and a user. The access control model should be flexible enough to let users define their own relations in

their own terms.

Relationships may not be reciprocal. There may be different sentiments between two users. While *Alice* may consider *Bob* as her *friend*, she could consider him just as his *partner*. This issue is more evident between social entities of different kind. *Alice* is nominated as *assistant professor* in a computer science department, but the opposite is not possible. As mentioned above, actors should be able to define their own relation names and properties assigned to them. These customizations should be unidirectional, so users are not constrained by the definitions of other partners.

Furthermore, there is no way to act from inside other social actors different from users. Groups and other actors are an important part of social behavior, but they are not the subjects of current access control models. In some solutions, groups are considered as part of the DAC policy definitions but no solution considers them as first-class actors.

Individuals and organisations share resources with other social entities with different levels of confidence, frequency, etc. This issue is modeled as weak and strong ties between them (section 2.4.1). The model should take in consideration this issue; relation levels should be associated to different levels of information disclosure, so users could use them to define the audience of their posts.

This chapter introduces Tie-RBAC, an access control model that solves these issues. Tie-RBAC results for combining the Tie model present in social network analysis (section 2.4.1) with role-based access control model (section 2.6.4).

7.2 Method

Tie-RBAC comes from the application of the well-known and succesful model of RBAC (section 2.6.4) to the discipline of social network analysis (section 2.4.1). As seen in social network literature, section 2.4.1, a tie is the formalization of a connection between two users with a relation attached. **In Tie-RBAC, the tie element defined in SNA is re-interpreted in the context of the RBAC model as the assignation of a user to a role**.

Figure 7.1 presents the classical RBAC model. When a user is assigned to the role `Admin` it adquires all the permissions attached to the role. This way, system administrators are able to decouple organisation roles competences from the users that are in that competence in a given time. On the other hand, figure 7.2 presents a social tie. The sender actor A establishes a contact with the receiver actor B using the relation `Friend`.

In Tie-RBAC, users are able to define their custom relations (e.g. *friend*, *buddy*, *colleage*). Users attach permissions (e.g. *read wall*, *post to wall*, *represent*, etc.) to these relations, in the same way an administrator assigns permissions to roles in RBAC (figure 7.3). When actors establish ties using these relations, they are assigning permissions to the receiver of the tie. Actors gain permissions from other actors who add them as contacts. This is equivalent to role assignation to users in RBAC. The sender of the tie is the entity which grants the privileges on

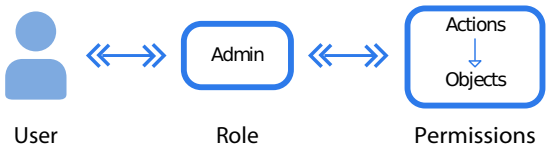

Figure 7.1 : RBAC model: users are assigned to roles. Besides, permissions are assigned to roles.

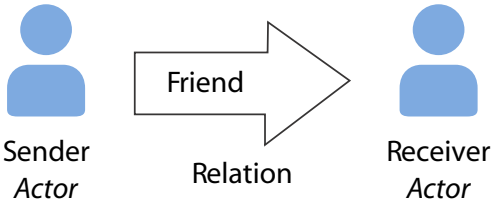

Figure 7.2 : A tie is made up by a sender actor, a receiver actor and a relation

their objects when establishing the tie. The receiver of the tie is the entity assigned to the role, which gains privileges on the sender's objects. The relation of the tie, that is defined by the sender, is the role. Note that both the sender and receiver are actors, so they can be users, but also groups, organisations or any other type of social entity.

Figure 7.4 shows permissions assignation through ties. Sender actor A uses the relation `Friend` she has defined to establish a tie with receiver actor B. Because the relation had a set of permissions attached, they are granted to B at the same time the tie is established.

Figure 7.5 shows the equivalence between the tie assignation and the role model in Tie-RBAC. The establishment of a tie between sender actor A and receiver actor B, using the relation `Friend` with attached permissions results in the assigment of actor B to the equivalent role B, with the same set of permissions.

For instance, *Alice* defines the relation *friend*. She grants *read wall* and *post to wall* permissions to *friend*. When she establishes the tie to *Bob*, she chooses *friend* as the relation of the tie. At the same time, she is granting *Bob* the permissions of reading and posting to her wall. On the other hand, the *computer science department* defines a relation *delegate*, and assigns the permission *represent* to it. Then, it adds *Charlie* as a *delegate*, so he can now represent the department in the application.

The last example has the chicken and egg problem. If there are not delegates, who will create the delegate relation and will establish the ties? One solution to this problem are default relations defined by the administrator of the application. Tie-RBAC model is general enough to support user-centric definition of relations. But administrator-defined relations can be supported in the system as well.

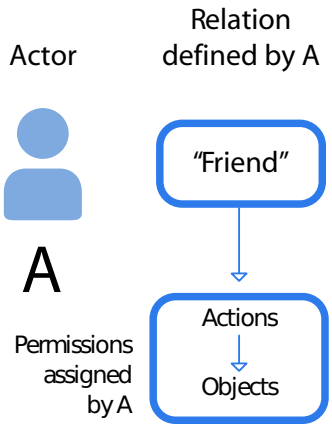

Figure 7.3 : Definition of a custom relation by actor A and assignation of permissions to that relation

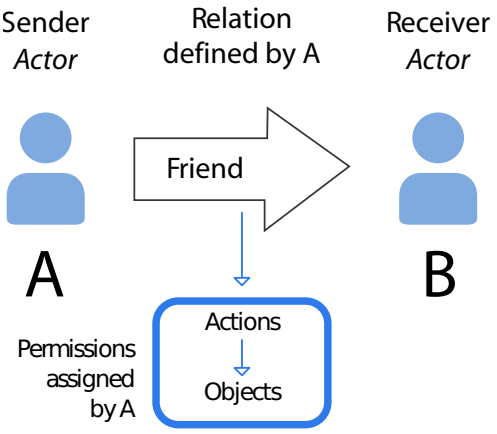

Figure 7.4 : Tie establishment in Tie-RBAC. Sender actor A establishes a tie with receiver actor B using relation `Friend`, which have permissions attached to it

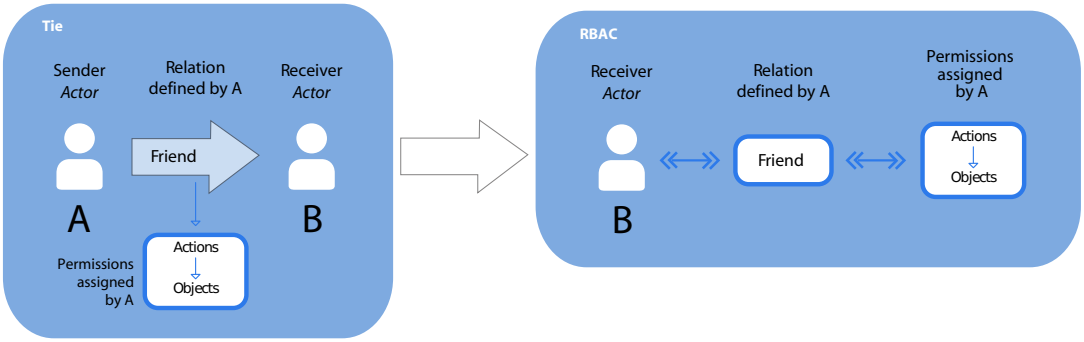

Figure 7.5 : Equivalence between tie establishment and RBAC model. When A established a tie to B with relation Friend and permissions, is A tie is made up by a sender actor, a receiver actor and a relation

The model proposed fulfills the objectives. The policy neutrality of RBAC provides the access control model with a user-centric approach, allowing users to define their policies.

It simplifies security administration, allowing users to change the permissions they granted just changing the ties, in the same way RBAC works.

Administrative roles provide a powerful feature never seen before in SNS. The capability of assigning representatives in a social network. An example are important and busy people which can delegate some of their administrative tasks on other users in the SNS.

The model allows different relations with different permissions, fulfilling the objective of the weak and strong ties approach.

The model works seamlessly with other social actors beyond users. All the model is based on the actor supertype, described in section 6.3.1, which can be substituted by users, groups or other social entities.

The model also supports custom relation management. Relations can be defined by users, which will attach them suitable permissions at their will.

The model is unidirectional. Actors are able to define their custom relations, assign permissions and establish ties in an independent way from the rest.

Finally, there is support for adminstrator defined, system relations.

7.3 Application of Tie-RBAC

Tie-RBAC is implemented in Social Stream (section 6.3), a social network framework.

Social Stream defines the following permissions:

- **create activity**: the ability to post to the actor's wall.

- **read activity**: the ability to read the posts in the actor's wall
- **follow**: the actor is interested in the updates from the contact's activity. Allowed updates from contact will appear in actor's home timeline.
- **represent**: the contact is allowed to act in behalf of the actor. This permission is common in group actors.

When a new user is registered in the application, three new relations are created for her, *friend*, *colleague* and *family*. These relations have the *create activity*, *read activity* and *follow* attached to them. This means that when a user creates a contact and uses any of these relations, she is granting her new contact with the ability to read her posts by default, to post to her wall and she will be interested in the activity updates from her contact.

When a group is registered, two relations are created: *member* and *partner*. The *member* relation has all the permissions attached to it, i.e. *create activity*, *read activity*, *follow* and *represent*. The *partner* relation only have the permissions *create activity*, *read activity* and *follow* attached. When a user creates a new group, a new tie from the group to the user is created. This tie has the *member* relation attached. This means that the group creator will be able to read group's wall posts and create new wall posts. The user updates will appear in the group's home wall. Finally, the group creator will have the ability to act in behalf of the group in the application.

Besides, Social Stream defines a system wide relation *reject*. When actors receive a contact request, they can reply it and add the requesting actor using one of the user-defined relations *friend*, *colleage*, *family* or any of the relation the actor might have defined. The actor also have the option of rejecting the contact. If that is the case, a new tie will be created, from the replying actor to the requester, with the *reject* relation. The *reject* relation has not any permission, so the rejected actor will have not any rights granted from the replying actor.

The model has also been used by the ViSH, the excursions-oriented social network websites that validates Social Stream, introduced in section 6.4.3. Because every content in ViSH is made public, *read activity* and *create activity* permissions have not sense. However, the *follow* permission is attached to the system-defined *follow* relation, so the follower's activities reach user's timeline. Besides, the *reject* relation is also reused from Social Stream, having no permissions attached to it.

7.4 Conclusions

Tie-RBAC model meets the objectives described at the begining of the chapter.

The model supports fine-graned policies. Users are always able to define custom relations for specific cases they want to deal with, and grating custom permissions to these relations. It also keeps away contact aggregation from privacy protecting. It lets users to define policies with no permissions, so users are able to add contacts at the same time they protect their privacy.

The model also facilitates permissions managing. By changing the ties of their contacts, they can change multiple permissions at the same time. It allows *Alice* removing *Charlie* from their friends, and maybe adding him to other relation without permissions. This way, she is revoking permissions at one time without changing each individual permission.

System relations can be defined by administrators. These should allow some users to be granted special permissions.

Custom relations are defined by users, which are able to use their own vocabulary, and using their own words beyond friend, such as *classmate*, *buddy*, etc.

Access control is not reciprocal. *Alice* is able to define the friend relation, granting some permissions to it and establishing a tie to *Bob*, while *Bob* does not grant permissions back to *Alice*, or maybe he adds her with his custom relation with custom permissions.

The model is generic enough to consider any kind of social actor, such as groups, organisations and institutions. The model supports users acting on behalf of these actors, allowing them to grant and revoke permissions.

The model has been validated in Social Stream, the social network framework described in section 6.3 and the content-oriented ViSH, section 6.4.3.

Chapter 8

Features of distributed social networks

> *Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts... A graphic representation of data abstracted from banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding...*
>
> *William Gibson, Neuromancer*

8.1 Introduction

Social network sites are not isolated web applications limited to give access to a web browser. Their distributed features go far beyond even mobile applications. There is a rich set of interactions between SNS and other applications at every level of their features.

Users develop their personal or professional activities in different platforms, many of them with social network features. Some person may use a social code repositories site, such as Github, to manage his projects, along with the ones from his organisation. Besides, he may use a blogging platform to express his thoughts, a photo-sharing platform to publish his images, and a social network platform to be in contact with his collegues. All these platforms may let other users like or republish the blog posts or the photographs to other social platforms. Besides, the blogging platform might be integrated with the photo service, in order to include images to ilustrate the articles, and last posts may appear in the social network user's profile. Interoperability issues at several levels in the social network architecture can be found. User authentication, profile sharing, resources distribution and activities distribution are examples of these interrelations.

Even an scenario of network federation is possible. This top kind of interoperability is essential in other distributed technologies in the internet. If we take a look at the email, we can see how proprietary solutions appeared first. These solutions did not interoperate between them. Eventually SMTP [Harrenstien 1982], an open solution, emerged and dominated the market. Instant messaging is other distributed technology. Every institution is now able to set up its own XMPP server and join the network of instant messaging. It seems plausible that it will happen the same with the social web. Now, current vendor solutions such as Facebook, Twitter or Google+ do not allow creating contacts between them. However, in a future it should be possible to add contacts between networks and follow user activity, no matter which their affiliation is.

These interactions have been studied by several W3C groups, as reviewed in section 2.4.4. The report *A Standards-based, Open and Privacy-aware Social Web* [Appelquist 2010] gathers several problems, including them in a different framework, i.e. identity, profile, privacy, social media, activity and emerging frameworks.

In this chapter, a different approach is taken. Patterns in distributed social networking at the different levels of the social network framework introduced in previous chapters are analysed, i.e. authentication and profile, social network, objects and actions. Furthermore, there is a depth analisys from two points of view. In the first place, a distributed identity framework, OpenID is analysed in order to see what can be learnt from users that individually set their own web site, to devise potencial interactions at site level, to discover what users are exporting publicly when define their profile and what can be learnt from web ids. In the second place, an analysis of social networks sites APIs is performed, so the information social sites are currently offering can be described.

8.2 Paradigms in distributed online social networks

Several patterns in the distribution of SNS are gathered in this section. Distribution is studied from several levels that correspond to the social network framework introduced in previous chapters, i.e at the level of actor identity, authentication and the construction of distributed profiles, at the level of the social network, how contacts and content can be exported to other networks, and the level of the activities, the actions that actors take on objects. Finally the case of full social network federation is analysed.

8.2.1 The authentication and identity fragmentation problem

One of the key functionalities for the interoperability of social network platforms is identity. As shown in section 2.5.1, identity is becoming a keystone in the Web.

Section 2.5.2 shows how most SNS use as de-facto standard the ancient user and password combination for identifying users. As a huge amount of different services appearing after the Web 2.0 explosion, users have to manage not only a lot of different passwords for a variety of services, but a growing number of different profiles in a series of distributed platforms all around the world.

Recently, authentication solutions appeared, such as OpenID (2.5.4) and OAuth (2.6.6), however, none of them has completely solved the full problem yet. In the case of OpenID, its popularity grew at the end of the last decade, but sites have gradually abandoned it. On the other hand, OAuth has become in current state-of-the-art solution for distributed web authentication. However it lacks of universal identifiers. It currently requires a sign in button for initiating authentication, such as "*Sign in with Facebook*". This requires the host to explicitly support providers. Users cannot authenticate using a different, independent identity provider that supports OAuth if the host has not registered and made it available.

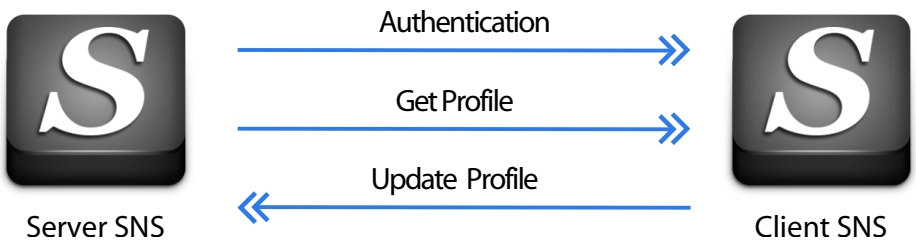

Figure 8.1 : Distributed authorization and profile features between SNS

Besides the burden associated with credential management, current situation involves a fragmentation of user activity along all the SNS. Because user profiles cannot be seamessly integrated, there isn't a coherence between different profiles created in every SNS. Furthermore, there is not a well-supported standarized way to combine digital contents created on every platform, unless web developers explicitly support one option. The identity fragmentation scenario can also be desirable for privacy reasons. Sometimes we don't want our activities traced and bound between every place we log in. But in other cases, specially when we want to build a coherent identity and reputation, such interoperability would make things easier. In these cases, from the user point-of-view, it will be desirable to have one single profile that could be validated against any service she would be accessing. Profile could be obtained from an external source so it has not to be filled in again. Furthermore, profile data could be synchronized when new attributes or changes in the actor status happen. An example could be a researcher publishing a new article, it can be linked to her profile in their research institution.

Figure 8.1 ilustrates a server SNS providing authentication services to a client site, as well as profile information about users. It also includes the synchronization of profile attributes, which happen in both ways; the server SNS notifies the client about new profile updates, and the client updates the server profile with a given interesting attribute, e.g current user's location.

This paradigm includes the identity and profiles frameworks introduced in the W3C report.

8.2.2 Leveraging social connections

Recent services provided by major social network platforms allow the exportation of the social graph. In other words, the connections one actor have with other actors in the site can be retrieved by other web application.

This information is used by remote sites for customizing social user experiences based on people acquaintances [Adams 2011]. We can find more and more examples of this issue. A couple of them are gathered here in order to illustrate the point.

The first of them is the case of *couchsurfing* services, a web page focused on offering users

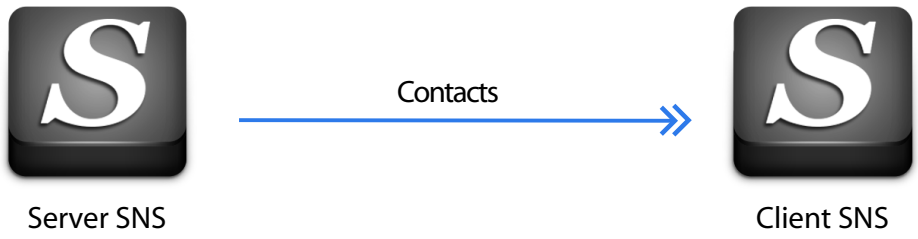

Figure 8.2 : Distributed contact access between SNS

hostage abroad for their trips. Trust is an important issue when people sleep with strangers. On the one hand, visitors will feel better if they find hints that the place they are visiting is save. On the other hand, hosts may have information that indicates their guests are reliable. The couchsurfing service grabs contacts from guests and hosts in the social network so both users can look at each other's connections. They may also find some connections in common, so it is more confident that hosting complete strangers.

Another example has to do with conference attendance. The conference scheduling service grabs the social graph from a social network site. This is a convenient way to know which conferences your followings are attending. If you are following an interesting person in the social network, you will probably want to attend a conference this person is attending or even speaking.

Figure 8.2 shows the server SNS exporting the contact list or social graph of users to the client SNS.

This paradigm is included in the profile framework described in the W3C report.

8.2.3 Content-oriented and related services

There are other cases where the social sites may be interested in some specific resources created by the user.

We introduce two examples of this pattern in the case of open source development and code repositories. Github [i], the popular social network for code sharing, exports the list of repositories one person manages. Travis [ii] is a continous integration server [Paul M. Duvall 2007] that is in charge of passing application tests and notify users when they have broken the application functionality in their last commit. Travis use distributed authentication to let users sign in using Github account. They grab the list of repositories one user has, allowing easily set up of repository testing. Finally, Github notices Travis when a repository changes, allowing Travis to update it and pass the tests.

[i] `https://github.com/`

[ii] `http://travis-ci.org/`

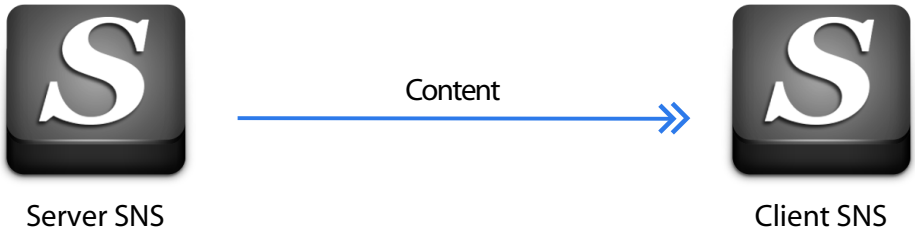

Figure 8.3 : Distributed content access between SNS

Another example is a photographers-oriented social network along with a printing service. A user may want to print some of the photos he manages in the social network. After authenticated in the printing service, this can grab the photographs associated to the user, so it is easier for them selecting the appropriates one to print.

Figure 8.3 shows a SNS serving user's content collections to a client SNS.

This paradigm is included in the Social media framework introduced in the W3C report, thus the report is more focused on provenance and accountability issues.

8.2.4 The new media

Major SNS are becoming a outstanding communication media. Service providers are looking forward to users talking about them to their contacts. Successful viral campaigns are gold. And the more influence people are the one closer to one person [Adams 2011]

Service providers implement buttons so it is easier for users to post about one service to their social networks, or report that they like it. Generic share, *Facebook's like* or *Twitter's tweet* buttons have become very popular. They allow users to post specific content or perform a like action on a remote resource. Besides, sites usually grab the contact users that like that content and shows them besides the content. This practice reforce the positive feelings of users towards the product, because their friends like it too [Adams 2011]

Figure 8.4 shows the client SNS grabbing user's activities from the server SNS. Besides, the client SNS creates new activities, such as posts or likes, in the server SNS.

This paradigm is gathered in the activity framework described in the W3C report.

8.2.5 On equal terms

Finally, there are cases where two institutions or networks interoperate on equal terms. This is the case between institutions such as universities or companies. They may want to share profile information, and their members may be interested in following each other. Contrary to former

Figure 8.4 : Distributed activity generation between SNS

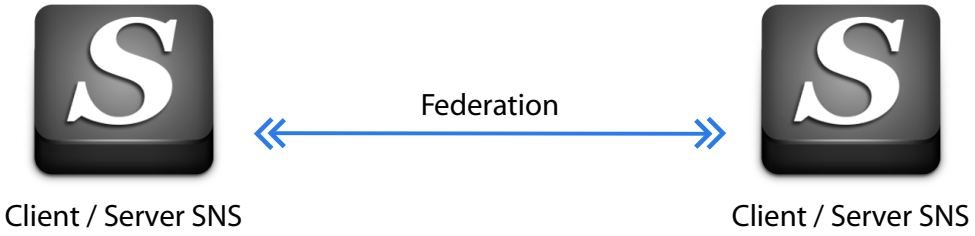

Figure 8.5 : Distributed federation of SNS

cases, there is reciprocity between both sites, so both of them have a role of client and server at the same time.

There are already protocols such as OStatus (section 2.2.3) that support this kind of interactions between SNS.

Figure 8.5 show federation between SNS.

8.3 Analysis of OpenID identifiers

OpenID is a user-centric distributed scheme for web identity (section 2.5.4). The OpenID framework is taken as a case study for analising distributed identity and profile information in the web. The idea behind this approach is the qualitative leap that entails a dereferenceable IDs such as OpenID.

Until OpenID appeared, using a new SNS platform implied providing a login and a password, as described in section 2.5.2. In most cases, an email address is also required. The address will be verified by the web site, which sends an email including a link that will confirm the validity of the address. If we analyse the information the SNS platform has about users at this point, it is limited to some credentials to authenticate in the web site, and an email address to contact with the user.

On the other hand, an authentication framework such as OpenID provides the server with a dereferenceable ID, that is, an URI representing a resource that can be located and fetched. If the platform could dereference the ID, then it will be able to obtain extended information from it. Upon an OpenID URI we are able to get an HTML document, which can contain a lot of information about the user.

The interest in this analysis is obtaining current practices behind OpenID practitioners, learning how they are, what OpenID providers they are using, what technologies are used to provide profile information, and what kind of information they are providing.

Appendix B introduces the survey on OpenID identifiers.

Results show practitioners deploying their custom OpenID domains, and more frequently delegating the OpenID service to a well-known OpenID provider. The last version of OpenID is not so popular, which announces the decline of this technology. Profiles are often linked to activity syndication feeds, and Microformats are a popular mean to export information, more popular than Semantic Web technologies. Personal and contact information is present in third and fourth of the sample, respectively, which shows practitioners willing to share this kind of information linked to their profile.

8.4 Analysis of SNS APIs

The other in deep approach for studiying distributed social networks consists on an analysis of current social network connect services.

Appendix C presents the review of the APIs of seven popular SNS platforms. Results show OAuth as the standard authentication service, with an increasing adoption of OAuth 2.0 and usage of OAuth scopes. OAuth authentication services force webmasters to explicitly support OAuth providers, in contrast with OpenID, which was an universal service.

There is an strong tendency to provide Javascript APIs allowing services to be integrated in client side. Regarding REST APIs, JSON is clearly the preferred format. However, resource representations are very disparate and their are all focused in the social network field of providers, e.g. code repositories or location places. This is consistent with content type analysis in section 5.4, which showed popular SNS implementing a diversity of content types. Disparate resource representations are present even a common resource: the user, and also for the pagination extensions along all the social network services. A standard should emerge in this field, maybe the Activity Streams (section 2.2.2) standard could be an option where OpenSocial (section 2.2.3) has not been.

8.5 Features of a distributed social network sites

This section analyses the cases described above and gathers a set of features present in distributed SNS. This work extends the theoretical framework developed in chapter 5.

8.5.1 Actors

Actors, which represent social entities (section 5.2.1), are present in a distributed fashion across SNS. Actors were introduced to have three main characteristics, i.e. a profile, with several attributes such as avatar, location, contact addresses; social relations with other actors and the ability to perform actions on objects. We will explore the distributed requirements of these characteristics in each section below.

Authentication

Some kind of actors, typically users, register and authenticate themselves into the site. When users are authenticated, they can perform actions in the site on their behalf. Authentication can be distributed beween sites using protocols such as OpenID (section 2.5.4) or OAuth (section 2.6.6). According to their authentication mode, actors may be **native**, **alien** and **foreign**.

- **Native actors** are entities that are registered in a given SNS in the first place. An example of native actors are users that sign up into a SNS using one of the classic authentication methods described in section 2.5.2. We call this server the **identity site**. The identity site can also be provider of an authentication service to other sites. In that case, users will be able to authenticate in remote sites, enabling the foreign actor case described below. Remote authentication is always performed after actors are authenticated in their identity site.

- **Alien actors** are entities that are known to exist in an external identity site, but have never authenticated in a SNS. The SNS having references to alien actors is called the **remote site**. An example of alien actors can be found in a site that analyses the social network of a microblogging application, but does not support any kind of authentication. Actors appearing in this site are alien actors. The actions performed by those remote actors in their identity site become remote actions in a remote site. Because these actions are not performed in the site, the remote actor's timeline must be filled with notifications from remote sites.

  Figure 8.6 shows a remote site with alien users that are referencing native users in their identity site.

  Remote actors need an identifier for them to be discovered and referenced in remote sites. We call this identifier the social ID (see next section).

- Finally, **foreign actors** are alien actors that use the distributed authentication to login into the remote site. They gather most of the characteristics from both native and alien users. They are able to perform activities in their identity site as well as in the remote site they have signed in. Therefore, their activities may be communicated between servers in both directions. Remote sites may provide local authentication means as well, allowing their foreign actors to become native. An example of this proccess would be a remote site

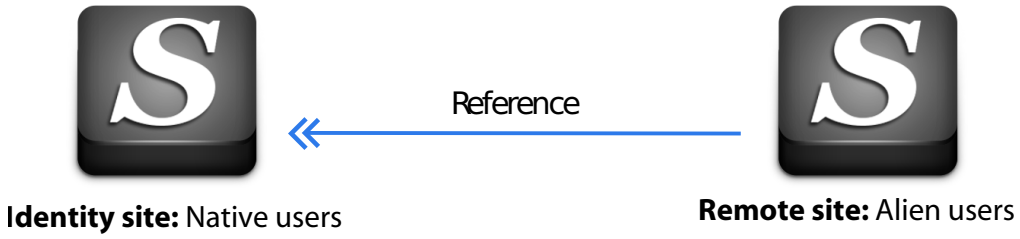

Figure 8.6 : Native and alien users in identity and remote sites, respectively

Figure 8.7 : Native and foreign users in identity and remote sites, respectively

allowing their foreign actors to validate their email. Classical user and password authentication could be established afterwards using this mean.

Figure 8.7 shows foreign users in a remote site that are authenticated by their identity sites, where they are native users.

Social ID

The social ID is used to reference actors from any site in a distributed social web. Some protocols already propose social IDs, such as OpenID (section 2.5.4 or Webfinger 2.2.3. The two most popular schemes for social IDs is the email scheme (*user@example.edu*) and http URL (*http://user.example.edu*). There is still an intense debate on the pros and cons of each one.

Distributed SNS are able to dereference and obtain information from the Social ID. This way, given a Social ID, users must be able to search and locate those actors from a given site in the distributed system. Besides, Social IDs are useful for identifing actions performed by the same actor in different remote sites.

Actors' representation: profile

The profile is made up of data attached to an actor, such as a name, an avatar, contact emails, location address, description or personal websites (section 5.2.10). Remote sites query the identity site to obtain more information about an alien actor when they discover her Social ID. Identity sites export the actors' representation, in one or several formats, including hCard and FOAF in the case of OpenID identifiers, JSON and XML in SNS APIs. In the case of OStatus, there are several means to learn about alien users, including Webfinger, ActivityStreams, HTML and Portable Contacts. In streaming APIs, identity sites may notify remote sites on profile updates.

Besides, actor's profiles may include collections of content attached to them. These include the set of actor's contacts that define the actor's social network and other kind of content such as images, videos or code repositories. The collections may be related to several actions, such as content created by her, owned, liked, etc. In the case of SNS APIs, these related collections are usually defined off-line, in API specifications, which does not satisfy REST principles (section 2.2.1). The analysis of OpenID identifiers show collections described in different formats, such as HTML links and Microformats inside the profile. Finally, OStatus uses the webfinger representation and HTML for linking content collections inside the profile representation.

In popular SNS APIs, remote sites are able to set or update profile attributes from and to the identity site. The profile representation might also be changed from activities performed in remote sites, as explained below.

Users control access to their resources and collections stored in the identity site using a protocol such as OAuth and its scopes. Popular SNS usually provide mechanisms to manage different profile data sets easily. Users typically want to show different kinds of profile to different SNS. They may want to provide the minimum required information to some remote sites, but detailed information to other trusted remote sites.

Actor's collections: the social network

The list of actor's connections is available in distributed SNS for retrievement. OpenID idenfiers use Microformats (XFN) and FOAF, popular SNS use JSON and XML, while OStatus proposes PoCo. SNS could publish both received connections (*followers*) and sent connections (*followings*).

Relation types are reviewed in section 5.2.2. In a distributed environment, unidirectional follow relations have a better fit, because they can be established between a native and an alien actor without any protocol. In OStatus, identity sites notify the remote site on contact establishment to alien actors and subscribe to their activity. However, in bidirectional relations establishment the identity site should wait for the notification from the remote site acknowledging the correspondence of the relation establishment. A relation establishment protocol is probably needed.

In the case of foreign actors, both identity and remote sites can communicate contact updates to each other, because the foreign actor is able to perform actions in the remote site, such as

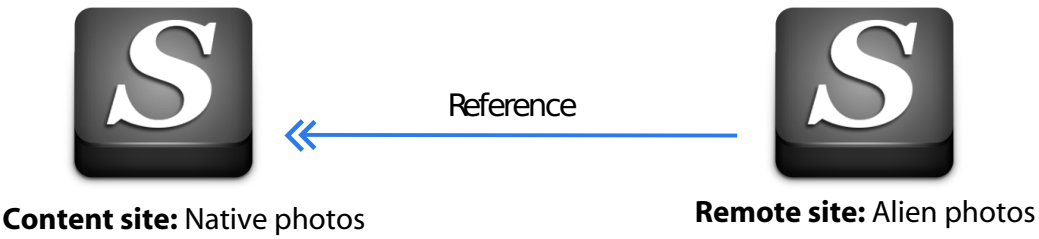

Figure 8.8 : Native and alien photographs in content and remote sites, respectively

establishing new contacts. These contacts should be communicated to her identity site, so her contact list is updated.

8.5.2 Objects

Native and foreign users are able to create native objects in a site, such as text, files, images, audios, videos or other kind of content. The server where the objects are created in the first place is called the **content site**.

Native objects may be referenced through notifications or activities from the content site to another site, called the remote site. For instance, there is the case where a native actor *Bob* in site *B* follows native actor *Alice* in site *A*. When *Alice* posts a photo to site A, this photo becomes a native object in site *A*, which becomes the photo's content site. The notification about the photo creation should reach site *A*'s alien actor *Bob* in remote site *B*. This notification should have a reference to *Alice*'s photo in site *A*. Therefore, each content object should have an object ID so it can be referenced from different SNS.

Alice's photo becomes an alien object in remote site *B*, which will have information about the photo obtained from server *A*. The remote site (site *B*) may just reference the object or retrive a representation and cache it. The advantage of referencing is that there are not synchronization issues to be taken into account, less storage space is required, and user's browsers retrieve the object directly from the original site. The advantages of caching is that it can be seamelessly integrated with the rest of the site, searched and will be persistent if it is deleted in the content site. For instance, in the case of images, specific thubnails can be generated, in order to improve efficiency. Objects may become obsolete or deleted in the content site, which might be able to notify the remote site on object changes.

Figure 8.8 shows native photographs in their content site referenced as alien objects in a remote site.

A foreign object is defined as content that can be updated from the remote site. In the example, it could be possible that *Bob* was able to modify the photograph, whose changes may reach back

to site *A*.

Content sites export representations in several formats. In the case of OpenID identifiers, which are tied to the blogging community, blog posts are often exported as HTML pages and RSS / Atom feeds. Popular SNS APIs export content in JSON and XML.

Objects can also be a source of activity streams. Different native and foreign actors may modify the objects, whose activities may be a vehicle for alien actors suscribed to object changes to discover and connect to them. Besides, there may be activity related to alien objects in the remote site, such as ratings, comments or mentions that could be propagated to the content site. This case is supported in OStatus by Salmon notifications.

Content visibility

Actors are also able to impose privacy controls in their native objects, through OAuth scopes. However, these objects may be cached in authorized remote sites. These servers should ensure that privacy restrictions are applied. In the example above, *Alice*, a native user in site *A*, may want to share her photo with *Bob*, a native user in site *B*, but not with *Charlie*, who is also a native in site *B*. Site *B* should ensure that *Charlie* is not able to retrieve *Alice*'s photo.

Ths is the same case with email. Email servers ensure that email messages sent to a user cannot be reached by others. Work addressing content re-sharing policies and their enforcement can be found in literature [Squicciarini 2009, Appelquist 2010]

8.5.3 Actions and activities

Identity sites record actions from their native actors. These activities are available as feeds. Also, an streaming API may allow remote sites to subscribe to activities.

In popular SNS, users may allow remote sites to post activities on their behalf. This is the case in Facebook applications. Facebook even lets remote sites to define new activity verbs in their application. User must have granted permissions to this remote sites through an specific OAuth scope. This is the case of foreign actors, where activities may be exchanged in both ways between identity and remote sites. In one hand, remote sites retrieve actor's activities. In the other hand, remote sites post new activities to the identity site.

In the case of federated social networks and OStatus 2.2.3, identity severs may update the contact graph of their native users or attributes of native content with the activities performed by alien actors, such as contact establishment, mentions, etc.

Activity sources may be focused both on actors and objects. Activites from actors gather their actions performed both in identity and remote sites. Activities focused on objects also include actions in both sites, such as creation, mentions or comments.

Examples of distributed activities are present in Activity Streams specification (section 2.2.2). These include:

- **Follow**: a native actor starts following an alien or foreign actor. The followed actor's identity site should be notified so the *followers* collection is updated. This action may trigger a subscription from the remote site to the followed actor's activities published by his identity site.

  Follow activity have also the opposite, unfollow. The contact collection in the identity site is updated and subscription is cancelled.

- **Authorship**: a native or foreign actor creates a new object in a content site. The new object can be accessed by her followers in remote sites. Alien actors may subscribe to the nofications of the new object.

- **Ownership**: a new object is created in the wall of an actor. As the case above, the activity may be propagated to the followers via the content and the identity sites.

- **Reply**: an actor creates an object that is related with another one previously posted. Object authors may be notified and the comment may be agreggated to the content relation collection of replies in its content site.

- **Rating**: the native actor rates some alien object from his identity site. As in the case above, object authors may be notified. The rating may be processed in the content site.

- **Mention**: an alien actor is mentioned in the context of an object, tagged in a photograph for example. The actor may be notified in his identity site.

- **Reshare**: the actor repeats a copy of an activity. Authors may be notified.

Table 8.1 shows how features of distributed social networks are implemented from three different points of view, i.e the OpenID framework and the information attached to its identifiers, current popular SNS APIs, and the OStatus federation protocol.

The figure shows the diversity of technologies up to date for solving the same problems. There is not a consensus not even in the Social ID. The only consensus seems to use URL as the identifier for object resources.

8.6 Conclusion

This chapter reviews the features of distributed SNS in the several levels described within the social network framework introduced in previous chapters. These are the user authentication and profile export, with a fragmentation of user profiles across different SNS, the distribution of actor's contacts, that can be used to easily build social experiences in external sites, the distribution of content representations attached to users, the leverage of social channels to distribute information about external interests, such as services, products or political campaigns,

Table 8.1 : Technologies implementing distributed distributed in OpenID identifiers, popular SNS APIs and OStatus federation protocol.

		OpenID identifiers	SNS APIs	OStatus
User Authentication		OpenID	OAuth	✘
Actor	Social ID	OpenID identifier (HTTP URI)	HTTP URI	Webfinger, HTTP URI
	Representation (Profile)	hCard, FOAF	JSON, XML	Webfinger, ActivityStreams, HTML, PoCo
	Collections (Contacts)	XFN, FOAF	JSON, XML	PoCo
	Collections (Objects)	Feeds, Microformats	Custom	Webfinger, Atom feeds
	Activities feed	RSS Atom	JSON Atom	ActivityStreams (Atom)
	Notifications	Pingback	PuSH, Streaming	PuSH, Salmon
Object	Object ID	URL	URL	URL
	Representation	HTML	JSON, XML	ActivityStreams
	Collections	✘	Custom	✘
	Activities feed	✘	JSON Atom	✘
	Notifications	✘	PuSH, Streaming	✘

and finally, the distribution of SNS as a whole. These levels are gathered by different use cases in the W3C report [Appelquist 2010].

The analisys of OpenID identifiers show practitioners building their own systems, more often as the front service that provides their profile and delegating the authentication service to external providers, but yet deploying their own services. Users often link their profiles to other SNS, in order to reduce the profile fragmentation problem and exporting personal information embeded in the HTML using Microformats. The information mainly is focused on keywords and personal information using hCard. They often provide a means to syndicate their activities.

The APIs analysis show SNS becoming identity providers and communication media. Their web services are increasingly being used by third-party web applications. We can see OAuth as an emerging standard for authentication and authorization, giving support to control profile information sharing. Javascript plug-ins are increasingly popular and provide a way to build distributed SNS architectures that move the integration layer to the client-side. REST and

streaming APIs are more interesting for social network federation point of view. Through JSON is the preferred format, there are different kind of resources and representations, which shows a missing gap in this scenario that could be filled by Activity Streams.

The analysis of the distributed extends the features introduced in chapter 5.

Actors are classified in native, alien and foreign. Native actors authenticate in their identity site using classic authentication. Alien actors are referenced in remote sites by their Social ID. Foreign actors use distributed authentication to log in remote sites. Actor profiles include attributes, object collections and the social network. These may be updated between sites and associated actions can also be propagated between distributed sites.

Objects are also classified in native, alien and foreign. Native objects are created within a content site. Alien objects are referenced from a remote site, using their object ID. Finally, foreign objects are modified across sites. There are issues related to content visibility issues , including re-sharing policies and their enforcement.

Activity actions are generated and distributed across sites. There is a description on how different activities, i.e. follow, authorship, ownership, reply, ranting, mention and re-share are generated by distributed actors on distributed objects and propagated across distributed sites.

Chapter 9

A framework for building distributed social network sites

> *— ¿Por qué tantos obstáculos entre mi objetivo y yo?*
>
> *El niño se encoge de hombros y me contesta:*
>
> *— ¿Por qué me lo preguntas a mí? Los obstáculos no estaban antes de que tú llegaras... Los obstáculos los trajiste tú.*
>
> *Jorge Bucay, Obstáculos*

9.1 Introduction

This chapter extends the SNF introduced in chapter 6 to support the distributed social networking features gathered in chapter 8. It describes an architecture that supports distributed actors with authentication, profile synchronization, contact and content exchange, distributed objects and activity interoperability.

There are already applications that provide solutions for federated social networks, reviewed in section 2.4.4. However, they are final applications, but not frameworks for building any kind of SNS. Trying to build a customized content-oriented website, a fork of the project is probably needed, because the customized code will clash with the social network project at the same level. Some of them may provide plugin and them support, but they require the development to think about modifying the project to feed their needs, instead of focusing in their application and having a framework that provides with required features.

The distributed framework provides with a powerful infrastructure in the top of which a dense mesh of social network applications could be build for different domains. The paradigm case shows research institutions providing SNS to their researchers, which are able to follow the activities of remote partners and connecting with a variety of services, including conference sites and learning platforms. Its application include every aspect regarding social networks, including business, education, health or politics.

9.2 Requirements of a distributed social framework

The analysis below describes a set of requirements that distributed SNFs need to meet in order to support distributed social networks.

9.2.1 Actors

In a distributed scenario, actors are scattered along SNS, becoming native, foreign or alien to each SNS (section 8.5.1). A distributed SNF must support alien actors at least, i.e. it must reference actors that have been registered in remote sites. Optionally, it may support foreign actors authentication, binding native and alien information in order to show the actor as a single entity and solving profile fragmentation.

Authentication

The SNF may provide distributed authentication capabilities both as client and server, using OpenID, OAuth or other distributed identity protocol. These capabilities can be implemented as client and as server.

- **Client authentication** capabilities enable the site to become a remote site that allows the entrance of foreign actors. These will be able to perform activities in their behalf, which the remote site may notify back to their identity sites of origin.
- **Server authentication** capabilities enable identity sites to become exporters of foreign users. They may support OAuth scopes or other kind of mechanism for allowing remote sites to update their native user's profiles and activities timelines on their behalf. They also may allow their foreign users to validate some kind of local classic authentication means in order to bypass remote authenticating and become native users.

Social ID

The distributed SNF must provide some kind of Social ID, such as OpenID, Webfinger or URI. The ID should be dereferenciable, in order to obtain more information on the remote actor and displaying this information to local users.

Besides, the Social ID enables SNS to combine actions performed by the same actor in different sites, solving the profile fragmentation problem.

Actors' representation: profile

The SNF should provide actor's profile representation besides HTML profile pages. This is a way for remote sites to learn about alien actors, their attributes and their collections, and build aggregated services on them. The profile representation should be linked to the Social ID, in order to facilitate information discovery.

A distributed SNF may allow profile synchronization in both ways, from the identity site to remote sites and back. This will enable remote sites to subscribe to profile changes in the identity site, but also pushing updates. A subscription API described in the former chapter could be used here.

Profile representations may include any kind of content collection, such as the contact list, photo albums or blog posts. This can also by used for building aggregated services in the top of the distributed social network infraestructure. In the case of the academia, scholar's profile representations would link to their published articles, for instance.

Because native actors may be also foreign actors and perform activities in remote sites, or even alien actors may perform activities related to a native actor, such as mentions, profile representation may be updated from activities in remote sites.

Finally, identity sites may use access control, such as OAuth scopes, for their users to limit and control the profile information that is distributed to remote sites.

Actor's collections: the social network

Distributed SNFs must provide a representation of the contact list of their native actors, using a format such as XFN, FOAF or JSON. The site must include the contacts established by the actor, which is a requisite for the social graph to be built. They may also provide a list of received contacts. Unidirectional relations between alien actors must be supported, while a protocol for bidirectional relations may be defined.

Upon relation establishment, the SNF should notify the remote server about the contact establishment, and subscribe to remote actor's activity, so alien actor's activities are available to their users in the local site.

The distributed SNF may also support updating the received contact list from notifications of remote contacts made by alien or foreign actors.

9.2.2 Objects

The SNF should export representations of the content types it supports, using JSON, XML or other available format. This capability enables the site to become a content site (8.5.2).

Objects must be referenced in activities notifications using an object ID. The object ID should be dereferenciable, letting remote sites get the object representation. Object representations may be cached or not. The SNF may also have a mechanism to refresh cached representations, or notify remote sites on object updates or deletions. It may also support foreign objects, allowing their native and foreign users to modify remote objects and pushing the new representation back to the content server.

The distributed SNF may also provide activity streams from actions performed on some kind of objects, such as a photo. Because these activities reach remote sites, the SNF may also be able to discover and present information about alien actors that are performing those actions on alien objects.

Content visibility

The SNF should enable their native and foreign actors to impose privacy controls in their native objects, e.g through OAuth scopes. They also should respect user privacy, implementing the required mechanism to show information only to their intended recievers, and not to other users in the same SNS.

9.2.3 Actions and activities

Actor's activity feeds should be provided. Support for an streaming API would improve the service allowing remote sites to receive this activities in real time.

Besides, the distributed SNF may support aggregating activities performed in remote sites. Foreign users may allow these sites to do so through OAuth scopes in their identity site. Aggregated activities may also come from actions performed by alien actors on actors or content belonging to the local server. In one hand, remote sites retrieve actor's activities. In the other hand, they post new activities to the identity site.

The SNF may provide activities which source are actors or objects. They may create activities from following, authorship, ownership, reply, rating, mention or reshare, among other verbs defined in Activity Streams.

9.3 Social Stream, a distributed social network framework

Social Stream's architecture is modular and consistent enough to be extended in order to support distributed capabilities. Figure 9.1 shows the modules provided by Social Stream. It extends figure 6.1 with some modules that fulfil the requirements described above.

Social Stream implements the OStatus protocol and adds a new component to the list introduced in section 6.4, ostatus, which provides SNS with federated social network capabilities.

9.3.1 Remote subjects

Distributed Social Stream adds a new type of actor, **remote subject**. It represents alien actors as described in above. Taking advantage of Social Stream's modularity, remote subjects are seamelessly integrated in the framework, without additional effort in managing actions, profile data or their social network.

9.3.2 Authentication services

Social Stream implements OAuth client support through the Omniauth Rubygem[i], which is integrated with the authentication gem Devise introduced in section 6.3.1. Ominiauth provides

[i] `http://rubygems.org/gems/omniauth`

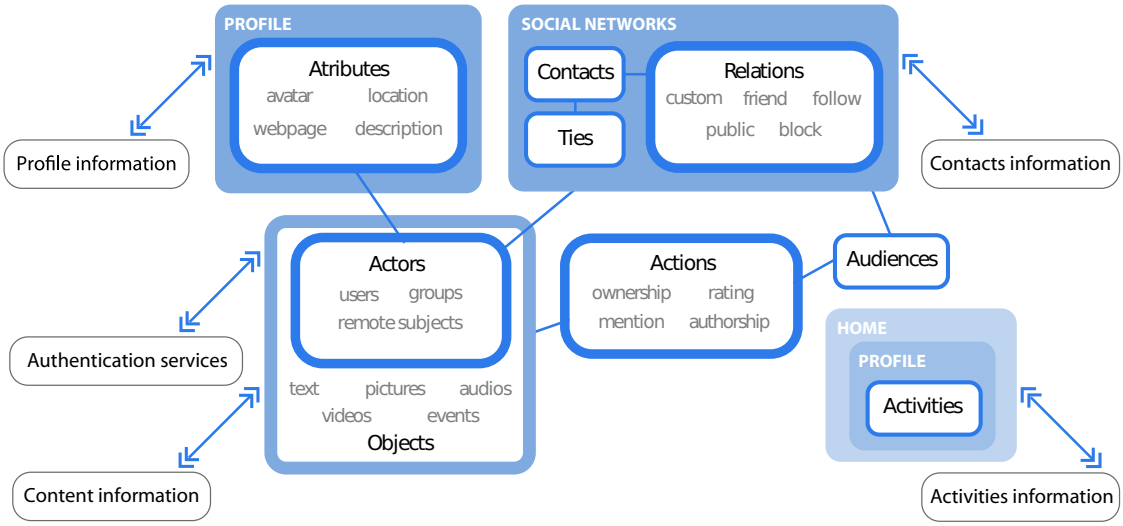

Figure 9.1 : Architecture components provided by Social Stream, a distributed social network framework

with support for several identity servers, such as Facebook, Twitter or LinkedIn. Besides, it provides with OpenID support.

Social Stream requires new foreign actors to provide with a valid email prior to formal registration. This ensures actors to retrieve a valid password and authenticate with classic credentials if the identity provider fails.

On the other hand, OAuth server support is being developed in the context of the FiWare project [i].

9.3.3 Profile representation

Social Stream exports actor profiles in JSON format. Besides, it supports the Webfinger protocol, providing information in the Webfinger representation such as avatar and the location of the activities feed.

9.3.4 Contacts representation

Actor's contact network is exported as JSON. Social Stream supports their native users to establish contacts with any type of actor, i.e. native, foreign and alien, through the common Actor model, described in section 6.3.1.

[i] http://www.fi-ware.eu/

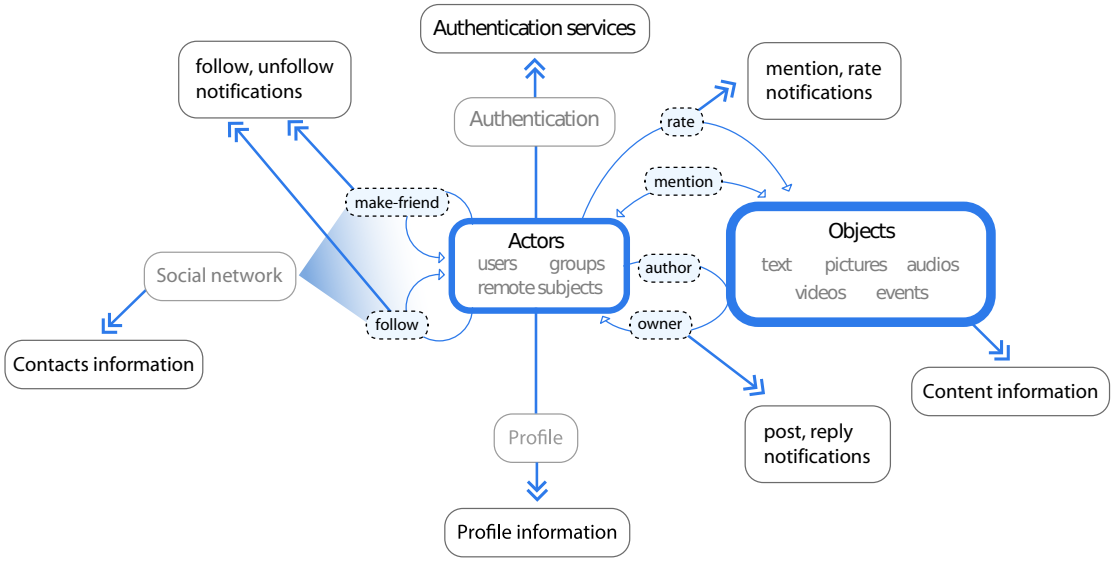

Figure 9.2 : Set of representations and actions exported in a distributed-enabled socal network framework

Incomming contacts can be established through the JSON API, but also upon contact activities from alien actors, i.e. when an new activity that consist on an alien actor establishing a contact to a native actor is received, Social Stream creates a new tie from the alien actor to the native actor.

9.3.5 Object representation

Representations of native content types are exported using JSON. The API also provides support for objects creation. Alien objects can also be registered through activities, in the same way new ties are created. Received representations of objects are cached in order to integrate them in the activities timeline.

9.3.6 Activities representation

Social Stream exports activities feed using the Activity Streams standard.

It also implements PubSubHubBub and Salmon for notification of public activities. Private activities are not shared in this way, due to privacy restrictions present in these protocols (see section 2.2.3).

Figure 9.2 upgrades figure 6.2 with the set of representations and actions that the distributed-enabled architecture exchanges with other sites.

9.4 Distributed social network websites built with Social Stream

The distributed framework has been validated in the following scenarios:

9.4.1 JSON API in the ViSH

Social Stream's object representations are used in the ViSH by the ViSH Editor. It is a Javascript, client-side based application developed in the context of the Global Excursion project, along with the ViSH. The ViSH Editor in focused on virtual excursion composing and display, allowing mashing up content from different sources, including the ViSH, but also content-oriented social networks such as Flickr or Youtube. The ViSH Editor can run stand-alone, independent of the ViSH and also have offline capabilities, allowing the download and display of virtual excursions offline.

The ViSH editor uses Social Stream's content exchange capabilities, both for content provided by Social Stream, such as documents and pictures, and also for ViSH specific content, such as excursions. The communication takes place through a JSON API. Every content created by the ViSH editor is seamessly integrated within ViSH's social network, author activities appearing in the timeline.

Besides, the ViSH mobile application is being developed. It currently supports uploading photographs and videos taken from the mobile. As in the case of the ViSH Editor, uploaded contents are seamesly integrated in the social network. In the future, the mobile application will use profile information and activities timeline exchange to improve user experience.

9.4.2 Social network federation in FI-Content

Social Stream's distributed capabilities have also been validated in the context of the FI-Content project[i], a two-year EU-funded research project in the framework of Future Internet Public Private Partnership (FI-PPP) programme.

A demonstration was taken along with Telecom Italia. The scenario (figure 9.3) comprised three different SNS, i.e the ViSH, Social Stream's demo site and Telecom's Italia's Teamlife [ii].

No extra effort was required to provide the ViSH with federation features. Adding the OStatus module was sufficient for the ViSH to obtain the required functionality from Social Stream.

The demo showed federation between all the sites. Remote contacts were establised between sites. Post made in the ViSH appeared in the others sites, and pictures uploaded from a mobile application to Teamlife appeared in the ViSH and Social Stream's demo site.

[i] `http://www.fi-content.eu/`

[ii] `http://beta.teamlife.it`

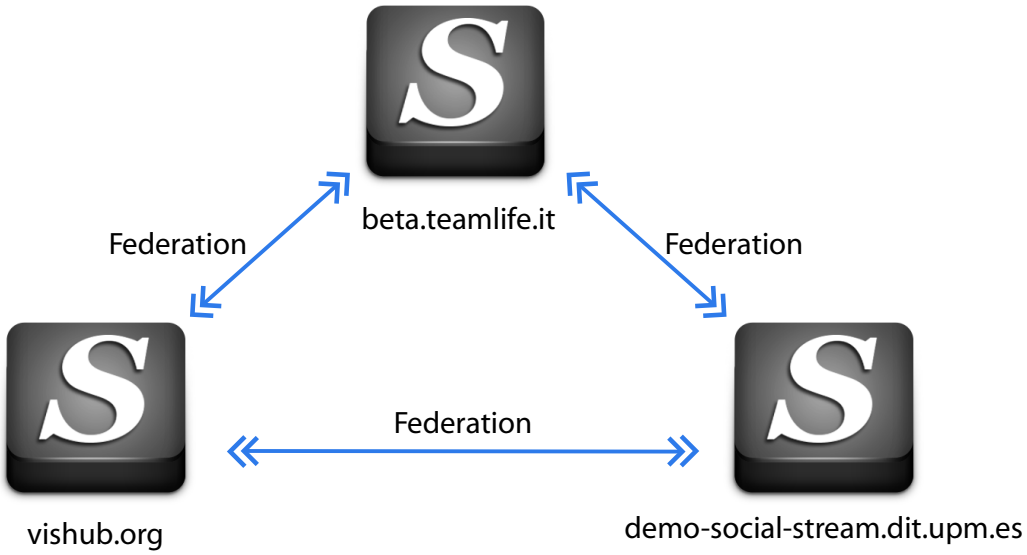

Figure 9.3 : Social network federation scenario in the FI-Content demonstrator

9.5 Discussion

Social Stream's implementation of the PubSubHubBub and Salmon standards showed that some of the functionality is overlapped. In both cases, the SNS is notified on new activities, on posting new content in the case of PubSubHubBub, and on replies, mentions and follows in the case of Salmon. A simpler, Activity Streams' JSON based protocol that unifies activities notifications might be designed. This protocol may use OAuth in order to identify remote servers, in the same way Salmon signatures are used now.

This protocol might also solve the privacy problem currently present in OStatus, which was designed for public activities. The combination of HTTPS and OAuth scopes might be sufficient to achive a privacy enabled distributed social network.

Social Stream does not currently passes SWAT0 tests, described in section 2.4.4. However, the support of the mention action will allow the full compliance with them.

9.6 Conclusions

This chapter introduces the requirements for a distributed social network framwork, defined upon the set of features presented in chapter 8. It extends the framework proposed in chapter 6 to fulfill them. Distributed features include authentication services, export representation of actor profiles,

the social network and content objects, as well as update notifications on actor's activities and object incidences.

The distributed framework has been validated in the context of two EU-funded projects, both providing support to the ViSH Editor, a client-side application for the creation of virtual excursions, and in a federation scenario along with an external application provided by Telecom Italia.

Chapter 10

Validation

> *We reject: kings, presidents and voting.*
> *We believe in: rough consensus and running code.*
>
> *Internet Engineering Task Force (IETF)*

This chapter presents the actions that the author has carried out in order to validate the ideas and contributions introduced in previous chapters.

The framework introduced in this work has served as the basis for several Master's theses, where the author has played a key role by tutorising them or by collaborating with the authors.

Additionally, the framework has been validated in the context of several EU-founded research projects. The concepts and ideas of the framework have played a crucial role in developing and achieving these projects.

Furthermore, the contributions have been used to provide scientific results by means of publications in international conferences and a book chapter.

There are also a number of contributions published as free and open source software (FOSS). They include the framework used to validate this work, which has a growing community and has been used in other projects beyond the scope of the author.

10.1 Master's theses

The author has worked as a member of the research staff in the Departamento de Ingeniería de Sistemas Telemáticos (DIT) of the Universidad Politécnica de Madrid (UPM), prior to the start of his PhD. During this time he has had the opportunity to collaborate and tutorise several Master's Theses of Telecommunications Engineering students who were finishing their degree, in addition to a Master's Thesis of the Master's degree in Computer Science in the same university.

- *Analysis, design and implementation of a core for social networks integrated with video-conferences [Carrera 2012].* Diego Carrera and I worked together to gather the requirements of social network sites and designing the social network framework. Diego Carrera applied the results for building a video-conference oriented social network.

- *Design and implementation of a private messaging system with notifications in a social network [Casanova 2012].* Eduardo Casanova designed and implemented the private

message system with support for notifications in the context of Social Stream. The system was another proof of the viability of reusable components at the Rails engine level. It was released as a Ruby gem and it is currently the most popular gem for private messaging in Rails. It has been used in multiple projects in addition to the scope of Social Stream.

- *Design of a user interaction model for its application in a social network [Díez 2012a].* Alicia Díez developed a UX-driven design of contact management system. It meant an innovative project regarding one of the first experiences in UX in the department.

- *Development of presence and instant messaging systems in social networks [Gordillo 2012].* Aldo Gordillo designed and implemented the presence and chat module in Social Stream.

- *Design and implementation of a recommendation system based on location on a distributed social network [García 2012].* Carolina García designed and implemented a places content-oriented social network site based on Social Stream. This is another example of the validity of the framework to manage new objects and focus on different content-oriented networks.

10.2 iCamp project

iCamp was a research and development project funded by the European Commission under the IST (Information Society Technology) programme of FP6. The project aimed at creating an infrastructure for collaboration and networking across systems, countries, and disciplines in higher education.

The iCamp project provided a framework for devising the objectives of this thesis, studying interoperation among learning platforms in Europe. It also gave the opportunity to the author to participate in his first European project and starting its training as international researcher.

10.3 GLOBAL project

GLOBAL was an FP7 EU-funded project with the aim of providing "*a virtual conference centre using advanced communication technologies and concepts to support the promotion of e-infrastructure topics in Europe and around the world*".

The main outcome of the GLOBAL project was the Virtual Conference Center (VCC), a conference-oriented content management system that supported the planning of events, in the context of virtual spaces, and extra-features such as forums and content repositories.

The VCC proved the validity of Station as a content management framework, as described in section 4.4.2

10.4 Global Excursion project

Global Excursion project[i], is a research project funded by the European Commission under the FP7.

The main outcome of the Global Excursion project is the Virtual Science Hub (ViSH) [ii], an excursions content-oriented SNS that allows scientists and teachers to exchange and establish collaborations, and where pupils are able to experience real e-science applications in areas of high relevance for the future, such as nano- and biotechnologies.

Social Stream has been used to build the ViSH. It has proved its flexibility and feasibility to build a content-oriented website with different contraints as compared to the default settings of Social Stream. Code reusability confirms the important contribution of Social Stream to ViSH functionality, as described in section 6.4.3.

10.5 SPION project

The SPION project, funded by the Flemish government, is an interdisciplinary project on privacy and security in online social networks. Its main objective is to mitigate the responsibility of individuals who use or are affected by social networking services by making the underlying social networking infrastructures and the organisations that run them more accountable.

The author participated in the SPION project as a visiting researcher in a three-month stay at KU Leuven. The Tie-RBAC model (chapter 7) was presented, obtaining valuable feedback. He also had the opportunity to experience and learn from the research activities in a foreign institution.

10.6 FI-Content project

FI-Content is a project within the Future Internet Public-Private Partnership Programme (FI-PPP), an ambitious effort to harmonise European-scale technology platforms and their implementation, as well as to integrate and harmonise the relevant policy and the legal, political and regulatory frameworks. FI-Content addresses five important content areas, spanning future uses of AV, games, Web, metadata and user created content.

A scenario showing social network federation was demonstrated in the context of the FI-Content. This scenario involved two platforms from the UPM i.e. Social Stream's demo site and the ViSH, and a platform from Telecom Italia. The scenario is described in section 9.4.2. It attracted much attention from a European Commissioner, who pointed this issue as an interesting matter among the political objectives of the EU.

---

[i] http://www.globalexcursion-project.eu/

[ii] http://vishub.org/

10.7 Dissemination of results

The results of this dissertation have been described and submitted for publication to relevant conferences. Additionally, the author has contributed to a book chapter in the context of the dissertation. The following list provides further details:

- *A Web Collaboration Architecture* [Tapiador 2006]. This paper was presented in the 2nd IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, held in Atlanta, Georgia, USA. It describes an early proposal of distributed architecture, which was further developed in the following years.

- *An Interoperability Infrastructure for Distributed Feed Networks* [Wild 2007]. The paper, which includes the outcomes of the iCamp project, introduces the interoperability issues related to blog feeds.

- *Resource Relationships in the design of collaborative Web applications* [Barra 2010]. This paper describes relationship issues present in REST architectures. It gathers some of the lessons that were learned when building the Virtual Conference Center.

- *Extended Identity for Social Networks* [Tapiador 2011b]. It proposes an architecture for distributed social networking based on attached information to a user-centric identity framework such as OpenID. This paper was presented in the Blogtalk conference and was later published as a chapter of Springer's Lecture Notes in Computer Science.

- *Integral solution for web conferencing event management* [Barra 2011]. This paper introduces the Virtual Conference Center, the conference-oriented content management system in which Station, the content management framework, was validated.

- *Tie-RBAC: an application of RBAC to Social Networks* [Tapiador 2011a]. The access control model proposed in this disseration was presented in a workshop of the 2011 IEEE Symposium on Security and Privacy held in Oakland, California, USA.

- *Social network federation with Social Stream and Social2social* [Sánchez 2011] This paper gathered mainly the privacy issues found when OStatus support in Social Stream was developed. In was presented in Federated Social Web Europe 2011, a world-wide submit that took place in Berlin, with positive feedback from the leader of the OStatus protocol lead designer.

- *Content Management in Ruby on Rails* [Tapiador 2011d] This international conference poster reviews content management support in Ruby on Rails, provided both by the MVC framework as well as by external plug-ins.

- *A survey on OpenID identifiers* [Tapiador 2011c]. The survey on OpenID identifiers was presented as a full paper in the 7th International Conference on Next Generation Web Services Practices, in 2011.

- *An analysis of social network connect services* [Tapiador 2012c] This international conference poster presents the results on the survey of the most popular SNS APIs.

- *Social Cheesecake: An UX-driven designed interface for managing contacts* [Díez 2012b] This poster introduces the work of an enhanced UX-driven interface for contacts management that was developed in the context of Social Stream.

- *A survey on social network site's functional features* [Tapiador 2012a] The analysis of the top SNS and their features was presented as a full paper in the IADIS WWW/Internet 2012.

- *Social Stream, a social network framework* [Tapiador 2012b] The framework for building social network websites was presented in the International Conference on Future Generation Communication Technology (FGCT 2012) as a full paper.

10.8 Free and open source software (FOSS)

The author has contributed to many free and open source projects which are currently used beyond the scope of his working group.

- *Ruby on Rails*: The author had the opportunity to make several contributions to the popular web development framework [i]. The most significant one was adding internationalization (i18n) support to engines, the advanced Rails plug-ins.

- *Station*, the content management framework was published as a Rails plug-in. It obtained up to 76 followings and 5 forks in Github.

- *Social Stream*, the reference implementation of the distributed framework introduced in this work was published as a Ruby gem [ii], a popular package system used by Ruby developers and the main means to distribute Ruby libraries. It has currently reached over 85,000 downloads. The code is available in Github[iii], where it currently has more than 650 favourites and 140 forks. Its website[iv] provides developers with resources that help them to build their own SNS. These include a get started guide, links to developer's forums, source

[i] `http://contributors.rubyonrails.org/contributors/antonio-tapiador-del-dujo/commits`

[ii] `http://rubygems.org/gems/social_stream`

[iii] `https://github.com/ging/social_stream`

[iv] `http://social-stream.dit.upm.es/`

code and documentation. The latter is generated along with the code in new versions and is available at Rubydoc[i].

Social Stream is divided into several components in order to provide developers with the right modules suitable for the needs of the specific SNS to be built. Current Social Stream's components include *base*, *documents*, *events*, *linkser*, *presence* and *ostatus*.

Social Stream has received external contributions, with nearly 30 contributors so far, 10 of whom belong to the local university group. However, more than half of the commits have been made by the author of this dissertation[ii].

Social Stream has been used for building several SNS. The most relevant cases include the ViSH, an excursion-oriented social network, AdviseOnly [iii], a commercial grade investors-based social network and the federated social network demonstrator in the Fi-Content project.

- Other projects published by the author in the context of this work include:
  - *rails-scheduler* [iv] a gem that provide Rails applications with support for scheduled events
  - *paperclip_waveform* [v] an audio post-processor to generate wave forms from audio files
  - *rails_engine_decorators* [vi] which adds support to easily extending Rails engines.

  Contributions to other projects include:
  - *proudhon* [vii] a Ruby implementation of the OStatus suite
  - *mailboxer* [viii] provides a Rails application with messaging support
  - *avatars for rails* [ix] provides a Rails application with avatar support
  - *linkser* [x] a link parser for Ruby
  - *inherited resources* [xi] which provides restful actions to Rails controllers

[i] `http://rubydoc.info/gems/social_stream/frames`
[ii] `http://www.ohloh.net/p/social-stream/contributors`
[iii] `https://adviseonly.com/`
[iv] `https://github.com/atd/rails-scheduler`
[v] `https://github.com/atd/paperclip_waveform`
[vi] `https://github.com/atd/rails_engine_decorators`
[vii] `https://github.com/shf/proudhon`
[viii] `https://github.com/ging/mailboxer`
[ix] `https://github.com/ging/avatars_for_rails`
[x] `https://github.com/ging/linkser`
[xi] `https://github.com/josevalim/inherited_resources`

- *acts-as-taggable-on* [i] a tagging plugin for Rails applications

[i] https://github.com/mbleigh/acts-as-taggable-on

Chapter 11

Conclusions

> *Change, be it evolutionary or revolutionary, is the essence of life.*
>
> *Manuel Castells*

This chapter provides a brief enumeration of the contributions that have been described in previous chapters and assess the achievement of the objectives introduced at the beginning of this work. Lastly, the chapter concludes by proposing a set of areas for further research.

11.1 Contributions

- *Explore and assess the feasibility of code reusability in the context of Model-View-Controller (MVC) web development frameworks*.

  This work proves that building middleware that provides any kind of content management system (CMS) features in an MVC framework like Ruby on Rails is feasible. An architecture supporting CMS features was designed. Futhermore, a CMS-framework was developed, which was validated in the context of three deployments, i.e. two local content management platforms and a conference management website developed in an EU-founded project. This was complemented by a subsequent study on the support on content management provided by Rails plug-ins.

  A new code reuse metric was defined, at the high level of Rails engines, which concerns the MVC architecture. This metric has been used to assess the validity of the middleware.

- *Gather a thorough enumeration of functional features present in social network sites (SNS), along with their description and the justification of their utility in a virtual social context*.

  This work introduces a formal study on social network features present in popular SNS, since formal studies are not available in current literature.

  The analysis has shown some popular features present in every or almost every SNS, i.e. user registration, user search, relationship establishing, comments, public sharing of contents, system notifications and private messages, home pages and profile pages with avatars.

  A strong tendency to use external services for authentication and contact recognition was found.

Relation types are almost equally distributed between unidirectional and bidirectional. Contact management and private content access control is present but it is not as popular as other features.

A wide set of content types are supported, with text, pictures, videos, events and links as the most popular ones in this order.

The home page has been found to be the primary section to get the updates from followings and post new content, while profile pages are the place to show relevant contents and contact lists.

These results provide the ground field for further research on SNS, such as gathering the requirements for building a social network framework (SNF).

- *Design and implement a framework for building social network websites.*

  This work introduces Social Stream as a successful framework for building SNS. Social Stream's architecture has proven flexible enough to support the customization of a contact-oriented social network, but also a content-oriented (excursions) one, the Virtual Science Hub (ViSH).

  The analysis of code reuse in the ViSH shows how most of the functionality in the application was provided by Social Stream without any additional work. There was a large part of the application that leveraged Social Stream's functionality with minimal work. Finally, developers were able to focus on new features and content types, such as excursions, which seamlessly integrated with social features provided by Social Stream.

  The framework was flexible enough to customize the relation model and the content visibility settings.

  Additionally, home and profile pages could be customized to a higher degree, reusing most of the modules such as the profile information and activity timelines.

- *Design a social network analysis-driven access control model for SNS.*

  Tie-RBAC is a new access control model for social networks based on RBAC advantages.

  It supports fine-grained policies. Users are always able to define custom relations for specific cases they want to deal with and grant custom permissions to these relations. Contact aggregation remains independent of privacy protecting. It allows users to define relations with no permissions in a way that users are able to add contacts while protecting their privacy.

  The model also facilitates the option of managing permissions. By changing the ties with their contacts, users are able to modify multiple permissions at once. It allows *Alice* to remove *Charlie* from *friends* relation, and add him to another relation without permissions.

This way, she is revoking permissions at once without having to change each individual permission.

System relations can be defined by administrators. This should allow some users to be granted special permissions. Custom relations are defined by users, who are able to use their own vocabulary and hence their own words beyond *friend*, such as *classmate*, *buddy*, etc.

Access control may not be reciprocal. *Alice* is able to define the *friend* relation, granting some permissions to it and establishing a tie with *Bob*, while *Bob* may not grant permissions back to *Alice*.

The model is generic enough to consider any kind of social actor, such as groups, organisations and institutions. It supports users acting on behalf of these actors, allowing them to grant and revoke permissions, as well as being granted or being revoked from them.

Lastly, the model has been validated in Social Stream, the social network framework described in section 6.3 and the content-oriented ViSH, section 6.4.3.

- *Study on the different cases and mechanisms involved in distributed social network sites*

  This works introduces a study of distributed paradigms taking place at several levels of SNS features, i.e. user authentication and profile exchange, the distribution of the actor's contacts, the distribution of content representations attached to users, the leverage of social channels and the federation of social networks.

  Additionally, there is an in-depth analysis from two points of view: the distributed identity framework OpenID and popular social networks sites APIs.

  The analysis of OpenID identifiers show practitioners building their own systems, more often than the front service that provides their profile and delegating the authentication service to external providers, but yet deploying their own providers. Users often link their profiles to other SNS in order to reduce the profile fragmentation problem and export personal information embedded in the HTML using Microformats. The information is mainly focused on keywords and personal information using hCard. Users usually provide a means to syndicate their activities.

  The APIs analysis shows that SNS are becoming identity providers and communication media. Their web services are increasingly used by third-party web applications. We can see OAuth as an emerging standard for authentication and authorisation, providing support to control profile information sharing. Javascript plug-ins are increasingly popular and offer a way to build distributed SNS architectures that shift the integration layer to the client-side. REST and streaming APIs are more interesting from the point of view of social network federation. Through JSON is the preferred format, there are different kinds of resources and

representations which show a missing gap in this scenario that could be filled by Activity Streams.

The synthesis of distributed features shows three kinds of actors: native actors, who are registered in their own identity site, alien actors who are referenced from a remote site, and foreign actors, who are authenticated using a distributed authentication mechanism. It also shows a Social ID as a key feature for locating remote actors and preventing profile fragmentation. Actor's profiles and connections must be exported and imported through sites. Objects must also have an object ID, its representations must be exported and imported across sites. Visibility constraints should be maintained between sites. Actors' actions should be communicated between sites. This may affect the status of actor's connections or object's state.

- *Design and implement a framework for building distributed social network websites.*

  This work uses distributed social network features to describe the set of requirements a distributed SNF must fulfil.

  The architecture of the social network framework was extended to support distributed requirements. Additional modules include the authentication service, export representation of actor profiles and the social network and content objects, as well as update notifications on the activities and object incidences of the actor.

  The architecture was implemented in Social Stream and validated in two distributed scenarios. This final step sets Social Stream as a framework for building social network websites, which was the ultimate objective of this thesis.

11.2 Future work

The implementation of support for the *mention* action would allow Social Stream to be compliant with social web acid test (SWAT0), the current requirements defined in the federated social web.

An improvement on federation protocols is necessary in order to overcome the privacy constraints present in OStatus. With this setting, a full mesh of distributed social networks could be deployed, which would lead to a new stage in the paradigm of social networking.

Appendix A

Content Management in Ruby on Rails

This appendix reviews the implementation of content management features 2.3.2 in Ruby on Rails and its plug-ins. Tables A.1, A.2, A.3 and A.4 show each feature and where it can be implemented: Rails, Rails* for a minimal development needed or a external plug-in. Their popularity is measured relative to Rails (which is the most popular project). The most popular full-featured content management project built with Ruby on Rails (RadiantCMS [i]) is also included. Popularity is measured using The Ruby Toolbox [ii], a web site that collects projects from Github. Github [iii] is the web site where the Rails community lives in. The score of each project in The Ruby Toolbox is proportional to the number of developers that are watching it and its forks in Github.

Results show that almost all the content management features are available in the Ruby on Rails development framework. Most of them are integrated in the core framework, but some of them are available as plug-ins.

Table A.1 : Content collection features in Rails

Standard tools for content creation	Rails, Formstastic, WillPaginate
Multi-user support & authorship	Devise
Separation of content from presentation	Rails
Content syndication	Rails
Content preview	Rails*
Content versioning	VestalVersions
Relevant content types	Paperclip
Form support for catalogue type data	Rails
Localization	Rails
Shared database for content storage	Rails
Thin client	Rails
Real time access to CM functions	Rails

[i] http://radiantcms.org/

[ii] http://ruby-toolbox.com/

[iii] http://github.com/

Table A.2 : Workflow features in Rails

Flexible, multi-threaded	AASM
Workflow monitoring and control features	Rails
Workgroups	CanCan

Table A.3 : Content delivery features in Rails

Static content	Rails
Dynamic content	Rails
Automatic link checking	Rails
Data error checking	Rails
Separate environments	Rails
Content version rollback	Vestal versions
Multi-channel support	Rails, Prawn
Automatic site changes	Rails*
Content personalization	Devise

Web authentication means have evolved a lot in the last years. The most popular authentication plug-in (Devise) provides the methods explained in section 2.5, such as username/password, access token, OpenID and OAuth.

Other features such as confirm email address, recover passwords, track user information like IP address, timeout session or lock user account are supported as well.

Devise's authentication methods are configurable in the model, letting the content management developer decide which methods are appropriate for each case. It provides with custom controller and views for authentication mechanisms. Besides, it provides with helping methods so the developer can check if the user is authenticated and which is his identity.

Authorization is transverse to the MVC architecture.

- **Model**: the state of the data in the persistence layer (e.g. roles assignations and resources relationships) determines whether authorization is granted or denied.
- **Controller**: authorization mandate which actions can be performed in the business logic layer.
- **View**: the interface changes depending on authorization issues. For example, some links are displayed if the user has rights to perform the actions behind them.

CanCan, provides methods for controller and views. The authorization policies are decoupled from the MVC architecture. They are described in a separate `Ability` class, which it is

Table A.4 : Control and administration features in Rails

Role definition and user security	Devise, CanCan
Taxonomy	ActsAsTaggableOn
Audit trail	Rails*
Reporting functions	Rails*, Devise

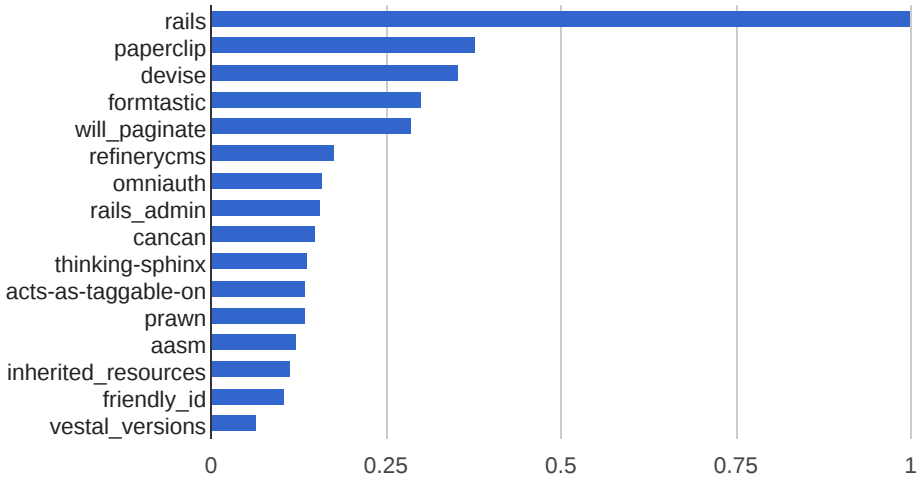

Figure A.1 : Popularity of Ruby on Rails content management plug-ins

instantiated for the user in every request.

Ruby on Rails follows resource oriented architecture (ROA) [Overdick 2007]. The framework provides resources management support at the three levels of the MVC architecture. Resources are tight related with the life cycle of content, collecting, managing and publishing [Boiko 2001]. At the model level, AASM state declarations for workflows, VestalVersions revisions and FriendlyID slug generation. At the controller, inherited resources provide the basic functionality for the life cycle of resources. Several plug-ins enhance the views. Formstatic powering forms generation, WillPaginate for index paginations and Prawn for PDF views. Finally, there are transversal plug-ins to the MVC architecture. These provide functionality at several levels of the MVC architecture. Examples are Paperclip for file management. RailsAdmin interface for resources management and ActsAsTaggableOn for taxonomies and folksonomies.

Appendix B

Analysis of OpenID identifiers

This appendix introduces a survey on the information attached to OpenID identifiers. It provides with an overview of current deployed web technologies in profile web pages. This approach tries to explore and learn from the technologies attached to identifiers and how they can contribute to distributed online social networks interoperability.

B.1 Method of the survey

The survey was made up of two phases: the harvesting of OpenID identifiers by a web crawler and its further analysis.

Many websites, e.g. Wordpress[i], Livejournal [ii] or Stackoverflow [iii], support using OpenID as an identification in user comments. Moreover, the comment's author is linked to its OpenID identifier, using HTML tags with specific, custom attributes.

A dozen of websites were harvested. They mostly consisted on blogs belonging to OpenID community leaders, but there were also technology oriented websites, such as Stackoverflow. The crawler went across the entire website without leaving it. OpenID comment patterns were searched using an XPath analiser, in order to extract the identifiers from each page's HTML. When a new identifier was found, OpenID discovery was performed. The identifier was discarded if no OpenID service was found. In the case of valid identifiers, the OpenID protocol discovery returns a claimed ID, which is the result of the specified normalization. The original website should have done discovery too, so the original ID must be the same as the one found. At this point, the new identifier was added to the database of OpenID identifiers.

In the second phase, each identifier was analised, attending to the characteristics of the own identifier; an analysis of the URL, protocol and distribution of domains. The second part of the analysis consisted on OpenID protocol versions and discovery technologies supported by the identifier. The last one was related to identifier dereferencing. The resulting HTML document of derefencing was analised as well. The goal was looking for more information attached to the identifier. The analysis of the HTML consisted on linked resources in HTML's head and semantic data embedded in the HTML's body.

---

[i] `http://wordpress.com/`

[ii] `http://www.livejournal.com/`

[iii] `http://stackoverflow.com/`

B.1.1 OpenID identifiers

Current specification of OpenID [Recordon 2006b] states that an OpenID identifier *is either a "http" or "https" URI [...] , or an XRI*. We classified only http and https identifiers, ignoring XRI. The reasons are that almost nobody uses XRI, there is not support in current browsers to resolve XRI and there is not a standard way to get a HTML document from them.

The distribution of the domains in the URLs, number of occurrences were also analised. There was also an analysis on how many unique domains exist in the URL sample.

Each OpenID identifier is attached with one or more OpenID providers. The OpenID provider is the web services supporting authoritative authentication for the identifier. The discovery service is used to link identifiers and providers.

B.1.2 OpenID protocol

OpenID specifications describe two ways of discovering the OpenID service given one identifier. The first, preferred way is using Yadis (section 2.2.3). OpenID authentication support is announced in XRDS files (section 2.2.2), discovered from the identifiers. There is one service supporting each version of the protocol. There are also other services available in the context of OpenID that will be explained later in the XRDS analysis, section B.1.4.

The other way described for OpenID service discovering from an URI is using HTML-based discovery. It uses a *link* tag in the *head* of the HTML document (section 2.2.2). OpenID defines several link relations in the specifications, but two of them are more relevant than others: *openid.server* for protocols *1.x* and *openid2.provider* for protocol *2.0*. Both announce that the URL referencing the HTML document, is an OpenID identifier and points to the provider of the document.

B.1.3 HTML

The OpenID identifiers collected bring an HTML document (section 2.2.2) when they are dereferenced.

Links in the HTML document to OpenID identifiers are interesting because they point to resources that have a relation with the identity. Besides OpenID protocol itself, HTML links are used for several standards, such as RSS (section 2.2.2) and Atom (section 2.2.2), the XML-based Web content and metadata syndication formats; or semantic web documents such as FOAF (section 2.2.2), the ontology describing persons, their relations to other persons and objects, and their behavior.

Other interesting technologies used in the HTML document are embedded inside the markup language itself. This is the case of Microformats (section 2.2.2). Microformats definitions include adaptations of vCard, iCalendar or geolocation formats, covering all aspects of data about identities.

Finally, the analysis includes another technology for embedding data in the HTML document, RDFa (section 2.2.2).

B.1.4 XRDS

We have also analised XRDS, the XML format for discovery of metadata about a resource used by OpenID. XRDS is part of OASIS XRI standard [Wachob 2008] and it is also used by other specifications beyond OpenID.

B.2 Implementation

Both, the crawler and the analiser were implemented in Ruby, using the following Ruby libraries (Rubygems): *ruby-openid* [i] (OpenID support), *nokogiri* [ii] (HTML parsing), *prism* [iii] (microformats parsing), *rdf-rdfa* [iv] (RDFa parsing). Data was stored in a MySQL database.

B.3 Results

A total of 1142 unique OpenID identifiers were collected.

B.3.1 OpenID identifiers

The analysis of collected OpenID identifiers showed that 97,3% of them are *http* URLs, while only 2,7% are *https* URLs (Figure B.1). Thus, the vast majority of OpenID users use their public, clear *http*. Very few of them seem to be concerned about security and use secure *http*.

Figure B.2 shows the distribution of domains. We found 563 unique domains in the 1142 URLs, which represents the 49% of the URLs. The distribution of identifiers corresponding to these domains is very irregular. There are some domains corresponding to big providers having a considerable percentage of the identifiers. The most popular is blogspot.com with a 13.57% of the market share, followed by myopenid.com with a 12.60%. On the other hand, there is a long tail of domains related with only one identifier. These are users that decide to set their own domain as OpenID identifier, typically people who maintain their personal page and their identifier is the URL of that page. 46% of the domains have only one identifier.

The results of the provider analysis are similar to the domains'. The distribution (Figure B.3) have a similar shape. The graph uses a different representation because OpenID supports that several providers give service to one particular identifier. As the domains distribution, the provider distribution has few important providers and a long tail of independent ones. Among the popular

[i] `http://rubygems.org/gems/ruby-openid`

[ii] `http://rubygems.org/gems/nokogiri`

[iii] `http://rubygems.org/gems/prism`

[iv] `http://rubygems.org/gems/rdf-rdfa`

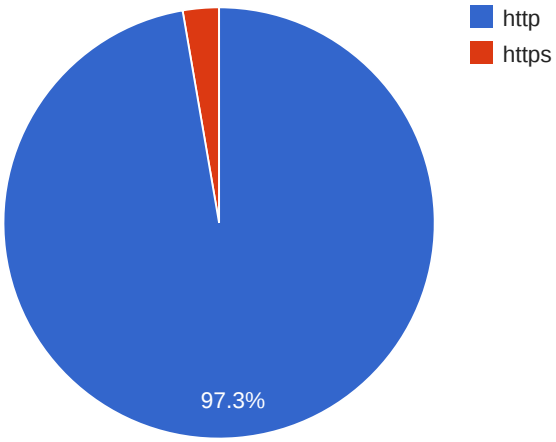

Figure B.1 : Distribution of OpenID identifier protocols

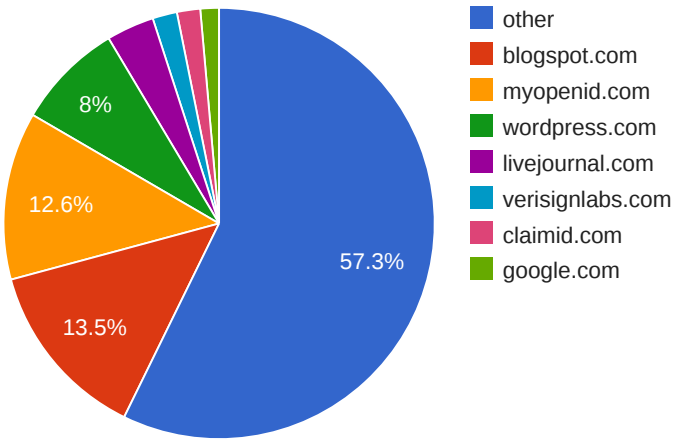

Figure B.2 : Distribution of OpenID identifier domains

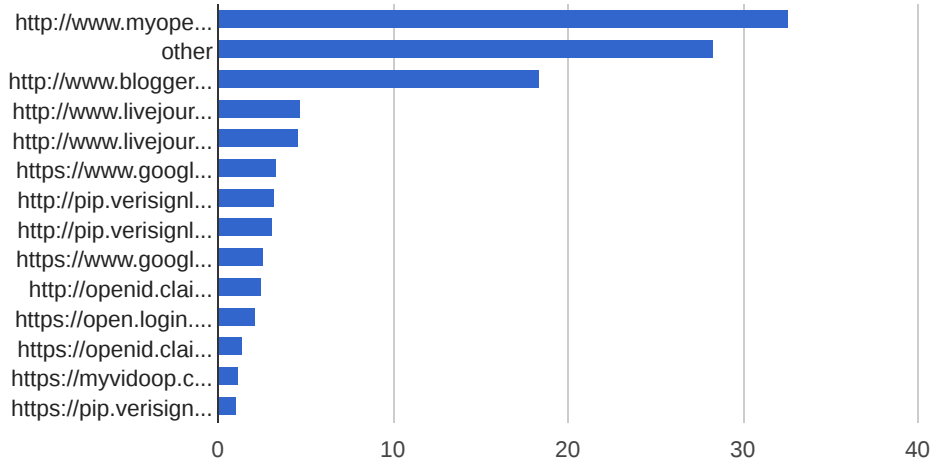

Figure B.3 : Distribution of OpenID identifier providers

ones are myOpenID (32.65%) and Blogger (18.39%). There are a lot of independent providers giving service to just one identifier. These constitute the 21%.

OpenID protocol

Our analysis of OpenID discovery shows that identifiers provide HTML discovery in 91.53% of the cases, while provide Yadis discovery in 41.32% (Figure B.4).

Regarding OpenID protocol versions, we have analised 1.x versions together. Thus they are distinguishable in Yadis, they are not in HTML discovery. The distribution of OpenID versions (Figure B.5) show a wide deployment of the first version of the protocol (97.32%). The second version of the protocol did not get so popularity (43.54%). 40.86% of the identifiers support both versions of the protocol.

B.3.2 HTML

Figure B.6 shows HTML head links distributions. The most popular is OpenID server link, followed by CSS stylesheets. Syndication formats are also popular (RSS more than Atom). There is an important group of links related with site edition (Really Simple Discovery, Windows Live Writer and Atom service post). 17.68% of the sites use XFN me attribute to link to profiles on other websites. There is also support for OpenSearch standard (13.31%). Finally, FOAF support

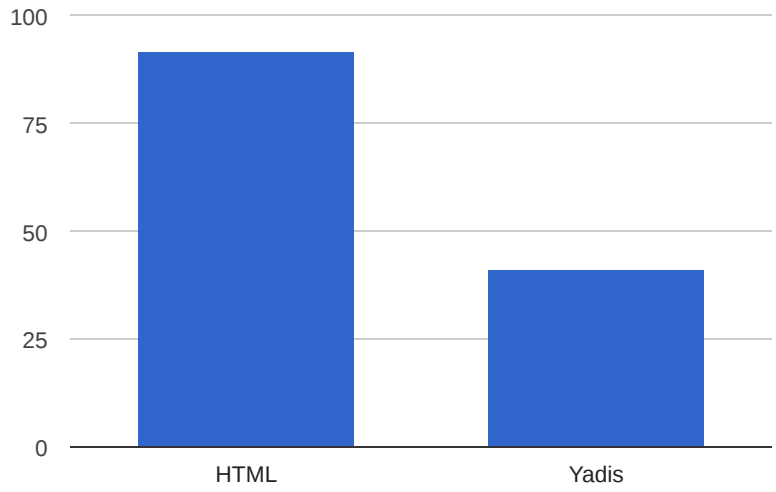

Figure B.4 : OpenID discovery protocols

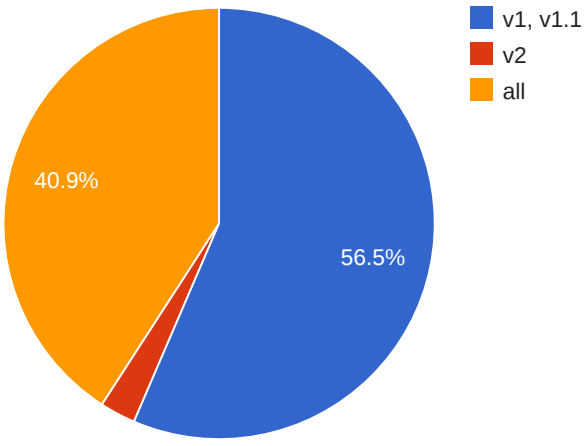

Figure B.5 : Distribution of OpenID versions

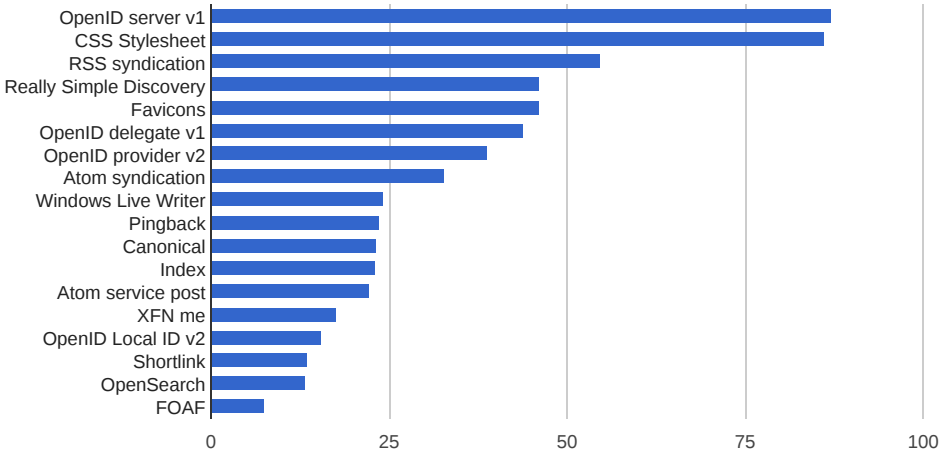

Figure B.6 : Recurrences of HTML head links

is very low (7.42%).

Microformats are widely used. They appear in 63.66% of the cases. The most popular is *RelTag*, a microformat for "tags" or "keywords" (42.56%). It is followed by *hCard*, an adaptation of vCard standard for personal information, which is used in a third of the cases (32.14%). *XFN* for network of relations and *XOXO* for personal outlines are also used (17.51% and 13.04%). *Adr* for address is used in a 10.06% of the cases. *Geolocalization* information is almost never provided (0.78%).

B.3.3 XRDS

XRDS service links are all related to the OpenID protocol, despite XRDS supports discovery of other types of services. Beyond authentication, OpenID Simple Registration Extension and OpenID Attribute Exchange are popular OpenID extensions supported by collected identifiers. They are followed by provider policy authentication extensions (PAPE), which assert policies applied by identity providers in order to assure end user authentication. The most popular is phishing-resistant policies. The rest of services are less common.

Finally, Figure B.9 and Figure B.10 show two comparatives. Most significant web technologies are compared in Figure B.9. RDFa is the most popular technology (88.01%). However, this is due to the integration of XHTML vocabulary, typically stylesheet and syndication head links. Leaving XHTML vocabulary (RDFa*) out lowers RDFa deployment

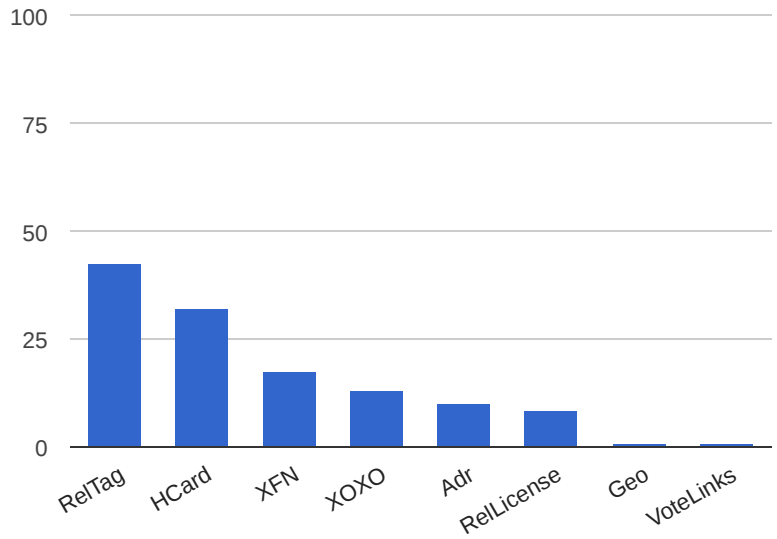

Figure B.7 : Microformats distribution

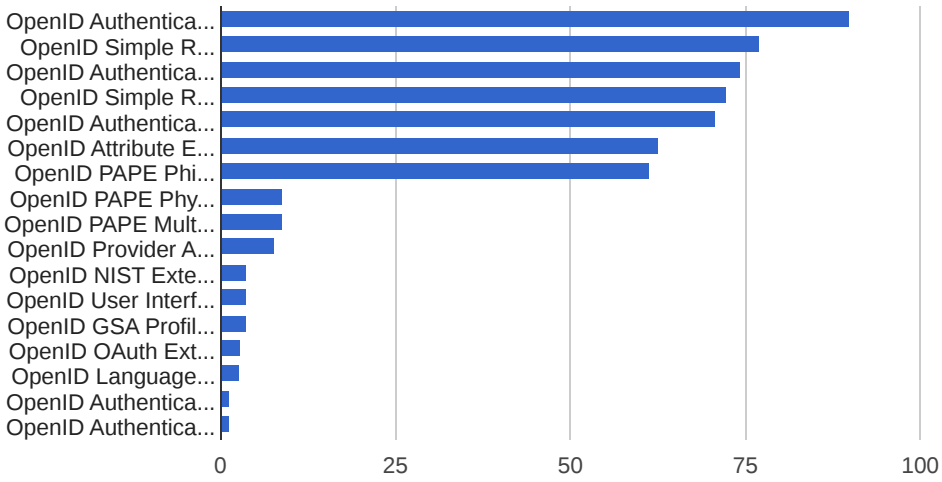

Figure B.8 : XRDS service links

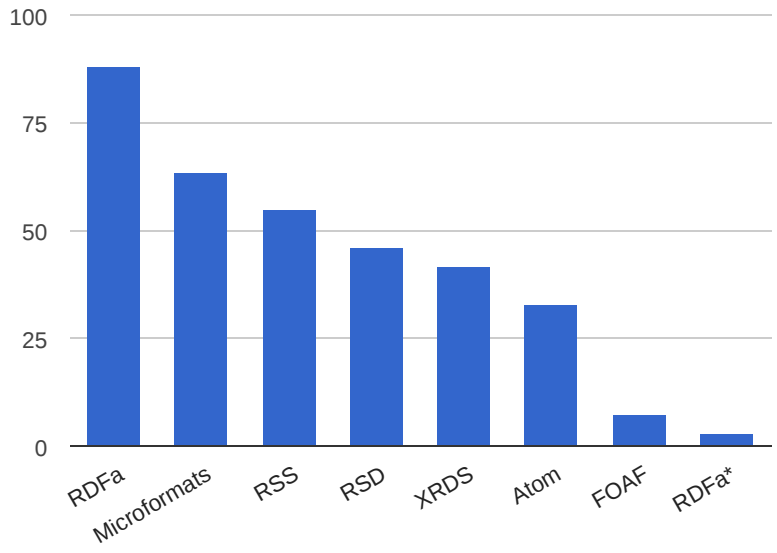

Figure B.9 : Comparative of web technologies

(3.22%) considerably. Microformats are very popular, followed by RSS and RSD. XRDS is also popular, but, as we saw above, only for OpenID-related purposes.

Figure B.10 shows two ways of expressing personal information, microformats (hCard) and Semantic Web (FOAF). Our analysis of OpenID identifiers show that microformat (32.14%) is preferred over semantic web (7.42%).

B.4 Conclusions

The harvesting of OpenID identifiers is focused to the spectrum of web pages to those of technologies adopters. However the spectrum of OpenID identifiers available is wider. Probably most of the available identifiers are never used by their owners. Considering all the identifiers available in the Web would probably dilute the percentage of technologies deployed.

Identifier security seems not to be a critical issue for OpenID users. Almost all of them use plain *http* identifiers. Since the user's client is redirected to the OpenID provider, using *http* identifiers suffers the risk from DNS poisoning attacks [Ope 2010].

Domain and provider distributions show that users are willing to deploy their own OpenID services. In addition, users delegate to their provider's service more often than they use their provider's identifier. While *myOpenID*, the most popular OpenID provider, has 13% of the market share of domains, it gives service to 33% of the identifiers. On the other hand, there are 46% of

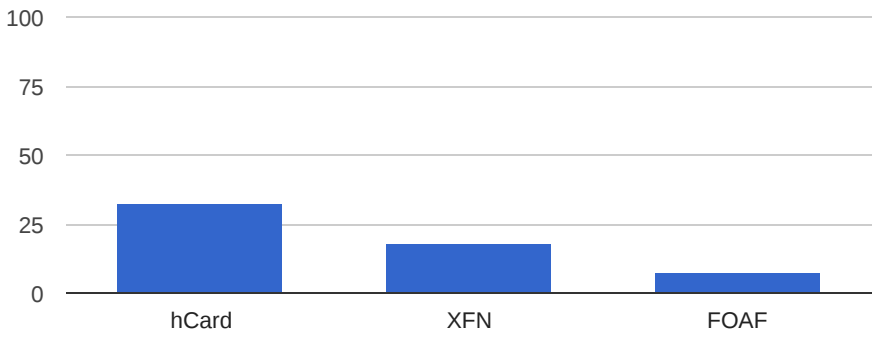

Figure B.10 : Comparative of personal information formats

identifiers with a unique domain, but only 21% of identifiers have a independent provider. This is consistent with OpenID origins. It is a technology pushed by web users and developers, and not deployed by a private or public organisation or consortium.

The deployment of OpenID 2.0 version is significantly lower than OpenID version 1. The enhancements introduced in current version did not seem attractive enough for practitioners to upgrade their systems.

Most popular HTML linked resources include activity syndication and edition. This suggests that OpenID identifiers are typically linked to blogs. Favicons are also popular, while semantic web profiles not so much. HTML embedded standards seem to be most convenient and easy means for web developers that are used to HTML markup. HTML OpenID discovery is more used than Yadis. This is despite of the specifications stating that *Yadis* is the preferred way to discover OpenID services. Microformats are another example of HTML integration success. Personal information markup is present in the third of cases, link relations in a forth, while geographical information is almost inexistent. Though RDFa results are promising, they are tied to the XHTML vocabulary. The use of other vocabularies is very scarce, which confirms the suitability of HTML integrated standards.

Appendix C

Analysis of SNS APIs

The APIs of seven popular social network platforms have been reviewed. First of all, there is an analysis of OAuth, the protocol used by all of them as sign in and authorization technology. The OAuth protocol has several versions, modes, permissions that every provider supports in a different degree. Besides there is an analysis of the available APIs featuring similarities and differences between them. Several patters have been identified, i.e. client oriented or Javascript, server oriented or REST and real-time/streaming oriented. The widget types used in Javascript APIs are enumerated. In the case of REST APIs, the formats and resource representations offered by the social network connect services are analysed, along with the streaming APIs.

C.1 Method

The APIs of seven popular social networking services have been reviewed; Facebook[i], the popular and well known contact-oriented social network platform; Twitter[ii], the popular microblogging platform; Google+[iii], the recent Google's bet in social networks; LinkedIn[iv], the social network platform for professional contacts; Github[v], the most popular platform for developers and social code sharing; Myspace[vi], the former popular champion in social networking and Foursquare[vii], the places check-in oriented social network.

These platforms offer extensive documentation on their APIs as well as online tools for testing them.

C.1.1 OAuth

OAuth is the web authorization protocol described in section 2.6.6. It is used by the analysed SNS to allow secure API authorization. Supported versions of OAuth, as well as grant types and authorization scopes are analysed.

---

[i] `http://www.facebook.com/`

[ii] `http://twitter.com/`

[iii] `http://plus.google.com/`

[iv] `http://www.linkedin.com/`

[v] `https://github.com/`

[vi] `http://www.myspace.com/`

[vii] `https://foursquare.com/`

C.1.2 API types

Social web platforms support several types of APIs; client side or Javascript, web server side or representational state transfer (REST) and streaming APIs.

Client side or Javascript API

The most straightforward way to integrate third-party web sites with social network platforms is through Javascript APIs. Social platforms provide Javascript libraries that can be included and used in any web site. It is an easy way to mashup local content with content from the social platform. There are a lot of applications that go from the popular "like" buttons or authentication to product recommendations based on social data. These APIs use OAuth's implicit grant type. All Javascript APIs use JSON format due to the fact that it does not need any heavy parsing step.

Web server side or REST API

Web-server-side APIs are oriented to server-based or desktop-based clients. They just provide data without any view-related content. They mostly follow the REST principles and verbs (section 2.2.1). REST APIs also support a variety of representations. Most common formats include Javascript's format JSON, XML, and their syndication extensions, RSS and ATOM, reviewed in section 2.2.2. The resource representations in REST APIs can also provide some commodities or extensions, such as pagination for large collections of data, or introspection that allows the examination of the resource's metadata.

Each resource has its own URL, so their representation can be retrieved and they can also be referenced from other resources, according to REST principles. Resources' addresses provided by REST APIs can be of different kinds. There can be a single resource URL scheme, or divided by resource type. In the case of a single URL scheme, the resource's representation provides a field declaring the kind of resource.

As is expected, every social network has a resource type to represent the user. Another metric we have used is comparing these fields to see if they follow any standard, which could be interesting to provide a uniform interface to the third-party developer that uses two or more social network providers.

Real time or Streaming API

The last kind of API provided by social network platforms is the streaming API. Unlike the former two APIs, in which the client request the platform for data, this goes the other way. The client subscribes to the platform, which pushes data to the client. This API is suitable for real time applications.

C.2 Results

This section presents the result of the analysis of the seven social network platforms mentioned above.

C.2.1 OAuth

While all the platforms support OAuth, the versions are currently distributed between 1.0a (Twitter, LinkedIn and Myspace), and 2.0 (Facebook, Google+, Github and Foursquare). Claims from users can be found in developer forums asking providers to support version 2.0, so it is probable that version 2.0 is implemented by all the platforms in the short term.

Between the platforms providing version 2.0, the support for authentication grants is disparate. All of them implement the web server/authentication code grant, which gives support to web applications. The next one supported is the implicit grant, primary used in the context of Javascript applications. This is implemented by three of four platforms. Even though LinkedIn claims to be using OAuth 2.0 in their Javascript API, their implementation is not standard, so it is not considered as that. Finally, client credentials grant is only supported by Facebook. They use it to provide third-party applications access to pending requests (`apprequests`) in the Facebook platform. They are the clear champions in OAuth 2.0 implementations.

Regarding OAuth scopes, their definition is very disparate and significant. Facebook defines more than 60 different scope types, such as `user_about_me`, `friends_about_me`, `user_activities`, `friends_activities`, `user_birthday`, `friends_birthday`, etc. Most of them are classified in information from a user and information from his friends. Many of the scopes are related to profile items, many others with collection items types like pictures, videos, etc. The special scope `manage_pages` provides an access token for impersonating pages. It is way of using the API impersonating the page. A complete reference on permissions can be found online[i]. On their side, Google+ has a full variety of OAuth scopes[ii]. They are providing a different namespace of permissions for each Google application (Analytics, Blogger, Calendar, Gmail, etc.), so their architecture is more modular than Facebook's. Github support only 4 types of scopes[iii] (`user`, `public_repo`, `repo`, `gist`), which are very focused on their application area: social coding. Finally, Foursquare do not use OAuth scopes.

Table C.1 gathers OAuth analysis results.

C.2.2 API types

Table C.2 summarizes support for the different API types.

[i] `http://developers.facebook.com/docs/reference/api/permissions`

[ii] `http://code.google.com/apis/gdata/faq.html#AuthScopes`

[iii] `http://developer.github.com/v3/oauth/#scopes`

Table C.1 : Analysis of OAuth support by popular SNSs

	Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
Version	2.0	1.0a	2.0	2.0*/1.0a	2.0	1.0a	2.0
Auth. code	✔	-	✔	-	✔	-	✔
Implicit	✔	-	✔	-	✘	-	✔
Res. owner	✘	-	✘	-	✘	-	✘
Client cred.	✔	-	✘	-	✘	-	✘
Scopes	✔	-	✔	-	✔	-	✘

Table C.2 : API types supported by popular SNS

	Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
Javascript	✔	✔	✔	✔	✔	✔	✘
REST	✔	✔	✔	✔	✔	✔	✔
Streaming	✔	✔	✘	✘	✘	✘	✔

Client side or Javascript API

Almost all the platforms provide a login button. This feature allows the third-party application delegating authentication and profile information to the social network. It is implemented by Facebook, Twitter, Google+, Github and LinkedIn. In the case of Google+, the login plug-in comes from Google accounts; this authentication service is not exclusive from Google+ and is used in every single Google product.

Another popular functionality is the "like" button, that allows users to post one site to their social network at the same time they are supporting it, since they are working towards a model where more likes means more quality. This is supported by Facebook (*like*) and Google+ (*+1*). There are other buttons available. Facebook has the bigger offer, including the send button to post to the Facebook wall, comments plug-in to comment about an activity, activity feeds and recommendations, as well as the login and register buttons. Twitter has the "*tweet this*", to publish a new tweet about the content of the third-party web site, or the tweet content that displays the number of tweets about the page. LinkedIn has the *Share this in LinkedIn* button. Github provides a "*fork me on github*" button that points to the open source repository.

Web server side or REST API

HTTP verbs Though the REST principles suggest using an HTTP per action, this issue is followed to different degree by current social network platforms. In one side it is Google+, that supports only read-only operations through `GET`. We expect that they support write operations soon. The next group of APIs include Twitter and Foursquare. These platforms only use `GET` and

`POST`. They both support updating and deleting operations, but they are performed using `POST`. Facebook follows, using the `DELETE` verb. Updating is rare in Facebook, but it is supported via `POST`. LinkedIn and Myspace perform these operations with a more appropriate HTTP verb: `PUT`, which was designed for operations updating resources. Finally, Github is the champion of this category, introducing yet another HTTP verb: `PATCH`. It is used for updating only selected fields from a resource, instead of `PUT` which is intended to update the whole resource. HTTP verbs support is gathered in table C.3.

Table C.3 : HTTP verbs supported by popular SNS

	Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
HTTP Verbs	GET POST DELETE	GET POST	GET	GET POST PUT DELETE	GET POST PUT PATCH DELETE	GET POST PUT DELETE	GET POST

Resource representations Table C.4 shows results on resource representations. JSON is the omnipresent representation format, supported by every analysed social network platform. It fits very well with Javascript APIs. Some of the platforms, such as Facebook, Google+, Github and Foursquare only support this format. Mostly all the platforms also support JSONP, allowing the API users to add a callback to JSON calls and avoid restrictions related to same origin policies [Oehlman 2011]. XML is supported as second format by LinkedIn and Myspace. Finally, Twitter is the champion in this category, supporting also XML derived formats such as RSS and ATOM.

Table C.4 : Support for resource representations in popular SNS APIs

	Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
JSON	✔	✔	✔	✔	✔	✔	✔
XML	✘	✔	✘	✔	✘	✔	✘
RSS	✘	✔	✘	✘	✘	✘	✘
ATOM	✔ Streaming	✔	✘	✘	✘	✘	✘

The resource types supported by each network vary from 3 by Google+ to more than 25 by Facebook. The are very different and depend on the focus of the social network. For instance, Github has *repositories*, *commits* and *forks*, while Foursquare support *venues* and *checkins*.

Nevertheless, all of the social network platforms support the user resource, which represent the users in the platform. We have analysed the JSON representation of these resources. The results

show that there is only one parameter in common: the `id`. All the other parameters are totally different in all networks. Table C.5 shows this issue. It gathers name-related parameters in each social networks. It is representative how there is not even two social network services using the same parameters for user names. The rest of resource fields are even more disparate.

Table C.5 : Id and name related fields in the JSON representation of users. While the id field is a common attribute, there are not two social network services using the same fields for names

Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
id	id	id	id	id	id	id
username name first_name middle_name last_name	displayName name (obj.) nickname	name screen_name	first_name last_name	login name	displayName nickname name	firstName lastName

Something similar happens with resource pagination (table C.6). Every social network uses different parameters for requests (e.g. `limit`, `offset`, `page`, `count`, `per_page`) and response (e.g. `previous`, `prev`)

Table C.6 : Extra features in resource presentation in popular SNS APIs

	Facebook	Twitter	Google+	LinkedIn	Github	MySpace	Foursquare
Paging	✔	✔	✔	✔	✔	✔	✔
Instrospection	✔	✘	✘	✘	✘	✘	✘

Resource addresses The common case in REST APIs is providing separate end-points for resources. Facebook is the only network that provides a single API address replaced their old-traditional REST API for the new Graph API[i]. It is a single interface to all the objects they manage. They also use the Opengraph protocol[ii] for adding metadata to the HTML of web pages, using the `meta` tag in the HTML's `head`. This way you can describe metadata like `title`, `image`, `url`, but it is redundant information, since even the Facebook's parser is able to obtain this information from other means (standard HTML tags). An interesting feature of the Opengraph is that Facebook uses this data to connect with Facebook applications and give admin permissions to users, through a couple of specific tags `fb:admins` and `fb:app_id`.

[i] `http://developers.facebook.com/docs/reference/api/`

[ii] `http://developers.facebook.com/docs/opengraph/`

Real time or Streaming API

Real time APIs allow third-party applications subscribing to updates from the social network platform. The most real-time of the APIs is Twitter's. They allow to get opened an HTTP connection and receiving a streaming of JSON objects, without closing the connection.

On the other hand, Facebook, Foursquare and Myspace support their platform performing HTTP requests to a third-party endpoint when a new event must be notified. Facebook uses an standard here, PubSubHubBub (section 2.2.3, while Foursquare requires the consumer to register on their web and using HTTPS. Posts are performed using JSON.

C.3 Conclusions

All the social network connect services analysed use OAuth at least in version 1.0a. The ones that use 2.0, they all implement web server support, and most of them client-side support. Authorization scopes are very particular to each API and there are not enough cases to obtain patterns. Nevertheless, two different models in architecture can be observed, Facebook's and Google's. While Facebook has a lot of different permissions, due to the centralized nature of their service, Google uses URI's in scopes, because of their decentralized architecture of services.

With general support for OAuth authorization code grant, SNS are providing their users with a way of granting access to their information to external SNS, leveraging distributed architectures. OAuth scopes provide a mechanism to control which parts of their profile are shared.

The switch from other authentication solutions, such as OpenID, to OAuth makes things harder to other identity services. OpenID uses URI identifiers, so all the providers have the same chances to be used. With OAuth, the third-party developer must put a sign in button for every identity provider he is willing to support favoring the more popular ones. Nevertheless, this issue was a tendency already appearing in OpenID implementations, where there were special buttons for most popular providers besides the uniform OpenID identifier field.

Regarding Javascript APIs, they are the quickest way of integrating a third-party applications with a social network services. We can see authentication widgets being very popular, with the ability to import authentication, profile information and even contacts to a third-party. Other plug-ins that are also very popular are the ones providing the ability to post to one's stream, both in the simpler form of likes as well as a more personal posts. Google uses the information of their +1 button to improve searches on their main product (searcher) that now will show the results that our friends recommend as featured entries. This is another example on how social experiences are integrated in web products, such as the most popular search engine of the web. Facebook is also the winner in this category with a wide range of plug-ins.

REST APIs are diverse and do not follow a clear standard, which makes things harder to the third-party developer that wants to integrate with several services, because he needs a specific library for all of them. The support for REST principles in HTTP verbs seems to be disparate, with some of them supporting them versus other of them overloading POST. The JSON format

is clearly the preferred in web services. Resource representations are very disparate and their are all focused in their social network field. Disparate resource representations are present even a common resource: the user, and also for the pagination extensions along all the social network services.

Bibliography

[Abdallah 2005] AliE. Abdallah and EtienneJ. Khayat. *A Formal Model for Parameterized Role-Based Access Control*. In Theo Dimitrakos and Fabio Martinelli, editeurs, Formal Aspects in Security and Trust, volume 173 of *IFIP International Federation for Information Processing*, pages 233–246. Springer US, 2005.

[Adams 2011] Paul Adams. Grouped: How small groups of friends are the key to influence on the social web. New Riders Press, 2011.

[Ali 2007] Bader Ali, Wilfred Villegas and Muthucumaru Maheswaran. *A trust based approach for protecting user data in social networks*. In Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, CASCON '07, pages 288–293, New York, NY, USA, 2007. ACM.

[Allsopp 2007] John Allsopp. Microformats: Empowering your markup for web 2.0. Apress, 2007.

[Anderson 2005] Terry Anderson. *Distance learning – Social software's killer ap?* 2005.

[Appelquist 2010] Daniel Appelquist, Dan Brickley, Melvin Carvahlo, Renato Iannella, Alexandre Passant, Christine Perey and Henry Story. *A Standards-based, Open and Privacy-aware Social Web*. W3C incubator group report, W3C, December 2010. http://www.w3.org/2005/Incubator/socialweb/XGR-socialweb/.

[Astels 2003] Dave Astels. Test driven development: A practical guide. Prentice Hall Professional Technical Reference, 2003.

[Bachle 2007] M. Bachle and P. Kirchberg. *Ruby on Rails*. Software, IEEE, vol. 24, no. 6, pages 105 –108, nov.-dec. 2007.

[Baecker 1995] Ronald M. Baecker and Others. Readings in human-computer interaction: toward the year 2000. Morgan Kaufmann Publishers, 1995.

[Barabási 2003] Albert-László Barabási. Linked: how everything is connected to everything else and what it means for business, science, and everyday life. Plume, 2003.

[Barnes 1954] IA. Barnes. *Class and committees in a Norwegian island parish*. Human Relations, no. 7, pages 39–58, 1954.

[Barra 2010] Enrique Barra, David Prieto Antonio Mendo and, Antonio Tapiador and Juan Quemada. *Resource Relationships in the design of collaborative Web applications*. In Proceedings of IADIS Multi Conference on Computer Science and Information Systems, 2010.

[Barra 2011] Enrique Barra, Antonio Mendo, Antonio Tapiador and David Prieto. *Integral solution for web conferencing event management*. In Proceedings of the IADIS International Conference e-Society 2011, pages 443 – 446, 2011. http://www.iadisportal.org/digital-library/integral-solution-for-web-conferencing-event-management.

[Bell 1998] D. Elliott Bell and Leonard J. LaPadula. *Secure Computer Systems: Mathematical Foundations,*, 1998.

[Benslimane 2008] D. Benslimane, S. Dustdar and A. Sheth. *Services Mashups: The New Generation of Web Applications*. Internet Computing, IEEE, vol. 12, no. 5, pages 13 –15, sept.-oct. 2008.

[Berners-Lee 1990] Tim Berners-Lee and Robert Cailliau. *WorldWideWeb: Proposal for a hypertexts Project*. Rapport technique, CERN, November 1990. http://www.w3.org/Proposal.html.

[Berners-Lee 1994] T. Berners-Lee, L. Masinter and M. McCahill. *Uniform Resource Locators (URL)*. RFC 1738 (Proposed Standard), December 1994. Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986, 6196, 6270.

[Berners-Lee 2000] Tim Berners-Lee and Mark Fischetti. Weaving the web: The original design and ultimate destiny of the world wide web by its inventor. HarperInformation, 2000.

[Berners-Lee 2005] T. Berners-Lee, R. Fielding and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (Standard), January 2005.

[Bieman 1992] James M. Bieman. *Deriving Measures of Software Reuse in Object Oriented Systems*. In Formal Aspects of Measurement. Proc. BCS-FACS Workshop on Formal Aspects of Measurement, pages 79–82. Springer-Verlag, 1992.

[Birbeck 2008] Mark Birbeck and Ben Adida. *RDFa Primer*. W3C note, W3C, October 2008. http://www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014/.

[Boiko 2001] Bob Boiko. *Understanding Content Management*. Bulletin of the American Society for Information Science and Technology, vol. 28, no. 1, pages 8–13, 2001.

[Boiko 2004] Bob Boiko. Content management bible. Wiley, 2004.

[Boulos 2006] Maged Boulos, Inocencio Maramba and Steve Wheeler. *Wikis, blogs and podcasts: a new generation of Web-based tools for virtual collaborative clinical practice and education*. BMC Medical Education, vol. 6, no. 1, page 41, 2006.

[boyd 2007] danah m. boyd and Nicole B. Ellison. *Social Network Sites: Definition, History, and Scholarship*. Journal of Computer-Mediated Communication, vol. 13, no. 1, pages 210–230, 2007.

[Bray 2008] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C recomendation, W3C, November 2008. http://www.w3.org/TR/REC-xml/.

[Carl Shapiro 1999] Hal R. Varian Carl Shapiro. Information rules: A strategic guide to the network economy. Harvard Business Press, 1999.

[Carminati 2008] Barbara Carminati and Elena Ferrari. *Privacy-Aware Collaborative Access Control in Web-Based Social Networks*. In Vijay Atluri, editeur, Data and Applications Security XXII, volume 5094 of *Lecture Notes in Computer Science*, pages 81–96. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-70567-3

[Carminati 2009] Barbara Carminati, Elena Ferrari and Andrea Perego. *Enforcing access control in Web-based social networks*. ACM Trans. Inf. Syst. Secur., vol. 13, pages 6:1–6:38, November 2009.

[Carrera 2012] Diego Carrera. Análisis, diseño e implementación de un núcleo para redes sociales integrado con videoconferencias. Master's thesis, Facultad de Informática, Universidad Politécnica de Madrid, Madrid, Spain, 2012.

[Casanova 2012] Eduardo Casanova. Diseño e implementación de un sistema de mensajería privada con notificaciones en una red social. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain, 2012.

[Corporation 1995] Lotus Development Corporation. Groupware: Communication, collaboration and coordination. Lotus Development Corporation, 1995.

[Cross 2005] Rob Cross, Jeanne Liedtka and Leigh Weiss. *A practical guide to social networks.* Harvard Business Review, vol. 83, no. 3, pages 124–132, 150, 2005.

[David Flanagan 2008] Yukihiro Matsumoto David Flanagan. Test driven development: A practical guide. O'Reilly, 2008.

[Dawson 1998a] F. Dawson and T. Howes. *vCard MIME Directory Profile*. RFC 2426 (Proposed Standard), September 1998. Obsoleted by RFC 6350.

[Dawson 1998b] F. Dawson and D. Stenerson. *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. RFC 2445 (Proposed Standard), November 1998. Obsoleted by RFC 5545.

[Dierks 2008] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[Downes 2005] Stephen Downes. *mIDm - Self-Identification the World Wide Web*, 2005.

[Duffy 2006] Peter D. Duffy and Axel Bruns. *The Use of Blogs, Wikis and RSS in Education: A Conversation of Possibilities.* In In Online Learning and Teaching Conference, 2006.

[Dusseault 2010] L. Dusseault and J. Snell. *PATCH Method for HTTP*. RFC 5789 (Proposed Standard), March 2010.

[Díez 2012a] Alicia Díez. Diseño de un modelo de interacción con el usuario para su aplicación en una red social. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain, 2012.

[Díez 2012b] Alicia Díez and Antonio Tapiador. *Social Cheesecake: An UX-driven designed interface for managing contacts*. In Proceedings of IADIS WWW/Internet 2012, 2012.

[E. Prodromou 2010] J. Walker E. Prodromou B. Vibber and Z. Copley. *OStatus 1.0 Draft 2*, 2010. http://ostatus.org/sites/default/files/ostatus-1.0-draft-2-specification.html.

[Economist 2006] T. Economist. *The new organization*. The Economist Surveys, 2006. http://www.economist.com/surveys/displaystory.cfm?story_id=5380483.

[Farmer 2005] J. Farmer. *Centred Communication: Weblogs and aggregation in the organisation*. In Proceedings of Blogtalk Downunder, 2005.

[Ferraiolo 2001] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli. *Proposed NIST standard for role-based access control*. ACM Trans. Inf. Syst. Secur., vol. 4, pages 224–274, August 2001.

[Fielding 1999] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[Fielding 2000] Roy Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. Doctoral dissertation, 2000.

[Fitzpatrick 2005] Brad Fitzpatrick. *Distributed Identity: Yadis*, 2005.

[FOA 2010] *FOAF Vocabulary Specification*, 2010. http://xmlns.com/foaf/spec/.

[Fong 2009] Philip Fong, Mohd Anwar and Zhen Zhao. *A Privacy Preservation Model for Facebook-Style Social Network Systems*. In Michael Backes and Peng Ning, editeurs, Computer Security – ESORICS 2009, volume 5789 of *Lecture Notes in Computer Science*, pages 303–320. Springer Berlin / Heidelberg, 2009.

[Frakes 1996] William Frakes and Carol Terry. *Software reuse: metrics and models*. ACM Comput. Surv., vol. 28, no. 2, pages 415–435, June 1996.

[Frakes 2005] W.B. Frakes and Kyo Kang. *Software reuse research: status and future*. Software Engineering, IEEE Transactions on, vol. 31, no. 7, pages 529 – 536, july 2005.

[Franks 1999] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen and L. Stewart. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617 (Draft Standard), June 1999.

[Fraternali 2010] Piero Fraternali, Gustavo Rossi and Fernando Sá andnchez Figueroa. *Rich Internet Applications*. Internet Computing, IEEE, vol. 14, no. 3, pages 9 –12, may-june 2010.

[García 2012] Carolina García. Diseño e implementación de un sistema de recomendación basado en localización sobre una red social distribuida. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain, 2012.

[Gates 2007] Carrie E. Gates. *Access Control Requirements for Web 2.0 Security and Privacy*. Web 20 Security and Privacy Workshop, vol. 05, 2007.

[Giunchiglia 2008] F. Giunchiglia, Rui Zhang and B. Crispo. *RelBAC: Relation Based Access Control*. In Semantics, Knowledge and Grid, 2008. SKG '08. Fourth International Conference on, pages 3 –11, dec. 2008.

[Gordillo 2012] Aldo Gordillo. Desarrollo de servicios de presencia y mensajería instantánea en redes sociales. Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain, 2012.

[Gradman 2004] Eric Gradman. *Distributed Social Software*. In LayerOne, Los Angeles, June 2004.

[Granovetter 1973] Mark S. Granovetter. *The Strength of Weak Ties*. American Journal of Sociology, vol. 78, no. 6, pages pp. 1360–1380, 1973.

[Gregorio 2007] J. Gregorio and B. de hOra. *The Atom Publishing Protocol*. RFC 5023 (Proposed Standard), October 2007.

[Gregorio 2012] J. Gregorio, R. Fielding, M. Hadley, M. Nottingham and D. Orchard. *URI Template*. RFC 6570 (Proposed Standard), March 2012.

[Grudin 1994] J. Grudin. *Computer-supported cooperative work: history and focus*. Computer, vol. 27, no. 5, pages 19 –26, may 1994.

[Guo 2010] Xiaofeng Guo, Jianjing Shen and Zuwei Yin. *On software development based on SOA and ROA*. In Control and Decision Conference (CCDC), 2010 Chinese, pages 1032 –1035, may 2010.

[Haas 2004] Hugo Haas and Allen Brown. *Web Services Glossary*. W3C note, W3C, February 2004. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.

[Hammer-Lahav 2010] E. Hammer-Lahav. *The OAuth 1.0 Protocol*. RFC 5849 (Informational), April 2010. Obsoleted by RFC 6749.

[Hammer-Lahav 2011] E. Hammer-Lahav and B. Cook. *Web Host Metadata*. RFC 6415 (Proposed Standard), October 2011.

[Harary 1955] Frank Harary. *The number of linear, directed, rooted, and connected graphs*. Transactions of the American Mathematical Society, vol. 78, pages 445–463, 1955.

[Hardt 2006] Dick Hardt. *Identity 2.0*, 2006.

[Hardt 2012] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749 (Proposed Standard), October 2012.

[Harrenstien 1982] K. Harrenstien and V. White. *NICNAME/WHOIS*. RFC 812, March 1982. Obsoleted by RFCs 954, 3912.

[Harrison 1976] Michael A. Harrison, Walter L. Ruzzo and Jeffrey D. Ullman. *Protection in operating systems*. Commun. ACM, vol. 19, no. 8, pages 461–471, August 1976.

[Hart 2007] Michael Hart, Rob Johnson and Amanda Stent. *More Content - Less Control: Access Control in the Web 2.0*. Web 2.0 Security and Privacy Workshop, vol. 05, 2007.

[Häsel 2011] Matthias Häsel. *Opensocial: an enabler for social applications on the web*. Commun. ACM, vol. 54, pages 139–144, January 2011.

[Heider 1946] F. Heider. *Attitudes and cognitive organization*. Journal of Psychology, vol. 21, no. 2, pages 107–112, 1946.

[Ignacio Fernández-Villamor 2008] José Ignacio Fernández-Villamor, Laura Díaz-Casillas and Carlos Á. Iglesias. *A comparison model for agile web frameworks*. In Proceedings of the 2008 Euro American Conference on Telematics and Information Systems, EATIS '08, pages 14:1–14:8, New York, NY, USA, 2008. ACM.

[Jacobs 1999a] Ian Jacobs, David Raggett and Arnaud Le Hors. *HTML 4.01 Specification*. W3C recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-html401-19991224.

[Jacobs 1999b] Ian Jacobs, David Raggett and Arnaud Le Hors. *HTML 4.01 Specification*. W3C recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-html401-19991224.

[Jasig 2004] Jasig. *Central Authentication Service (CAS)*, 2004.

[Johnson 1997] Ralph E. Johnson. *Components, frameworks, patterns*. SIGSOFT Softw. Eng. Notes, vol. 22, no. 3, pages 10–17, May 1997.

[Johnson 2005] R. Johnson. *J2EE development frameworks*. Computer, vol. 38, no. 1, pages 107 – 110, jan. 2005.

[Joshi 2001] James B. D. Joshi, Walid G. Aref, Arif Ghafoor and Eugene H. Spafford. *Security models for web-based applications*. Commun. ACM, vol. 44, pages 38–44, February 2001.

[Karunanithi 1993] S. Karunanithi and J.M. Bieman. *Candidate reuse metrics for object oriented and Ada software*. In Software Metrics Symposium, 1993. Proceedings., First International, pages 120 –128, may 1993.

[King 2009] I. King, Jiexing Li and Kam Tong Chan. *A brief survey of computational approaches in Social Computing*. In Neural Networks, 2009. IJCNN 2009. International Joint Conference on, pages 1625 –1632, june 2009.

[Ko 2010] Moo Nam Ko, G.P. Cheek, M. Shehab and R. Sandhu. *Social-Networks Connect Services*. Computer, vol. 43, no. 8, pages 37 –43, aug. 2010.

[Li 2009] Jianguo Li, Yong Tang, Chengjie Mao, Hanjiang Lai and Jun Zhu. *Role Based Access Control for social network sites*. In Pervasive Computing (JCPC), 2009 Joint Conferences on, pages 389 –394, dec. 2009.

[LID 2004] *Light-Weight Identity (LID)*, 2004. http://lid.netmesh.org/wiki/Main_Page.

[Liduo 2010] He Liduo and Chen Yan. *Design and implementation of Web Content Management System by J2EE-based three-tier architecture: Applying in maritime and shipping business*. In Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on, pages 513 –517, april 2010.

[Lihua 2010] Sheng Lihua and Xu Jiacheng. *Using social software to improve learning performance of deaf university learner*. In Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on, pages 703 –706, april 2010.

[Liu 2011] Yabing Liu, Krishna P. Gummadi, Balachander Krishnamurthy and Alan Mislove. *Analyzing facebook privacy settings: user expectations vs. reality*. In Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, IMC '11, pages 61–70, New York, NY, USA, 2011. ACM.

[Majumdar 2006] Swapan Kumar Majumdar. *A conceptual model for sustaining competitive advantage in digital economy*. In Proceedings of WEBIST 2006, 2nd International Conference on Web Information Systems and Technologies, 2006.

[man Au Yeung 2009] Ching man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne and Tim Berners-lee. *Decentralization: The future of online social networking*. In In W3C Workshop on the Future of Social Networking Position Papers, 2009.

[Mathes 2004] Adam Mathes. *Folksonomies - cooperative classification and communication through shared metadata*. 2004. http://adammathes.com/academic/computer-mediated-communication/folksonomies.pdf.

[Mcafee 2006] Andrew P. Mcafee. *Enterprise 2.0: The Dawn of Emergent Collaboration*. MIT Sloan Management Review, vol. 47, no. 3, pages 21–28, 2006.

[McKeever 2003] Susan McKeever. *Understanding Web content management systems: evolution, lifecycle and market*. Industrial Management and Data Systems, vol. 103, pages 686 – 692, 2003.

[Mislove 2007] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel and Bobby Bhattacharjee. *Measurement and analysis of online social networks*. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[Mooney 2008] Sean D. Mooney and Peter H. Baenziger. *Extensible open source content management systems and frameworks: a solution for many needs of a bioinformatics group*. Briefings in Bioinformatics, vol. 9, no. 1, pages 69–74, 2008.

[Nadel 1957] S.F Nadel. The theory of social structure. New York: Free Press, 1957.

[Nottingham 2005] M. Nottingham and R. Sayre. *The Atom Syndication Format*. RFC 4287 (Proposed Standard), December 2005. Updated by RFC 5988.

[Nottingham 2007] M. Nottingham. *Feed Paging and Archiving*. RFC 5005 (Proposed Standard), September 2007.

[Oehlman 2011] Damon Oehlman, Sébastien Blanc, Damon Oehlman and Sébastien Blanc. *Integrating with Social APIs*. In Pro Android Web Apps, pages 221–254. Apress, 2011.

[Oh 2008] Hyun-Kyung Oh and Seung-Hun Jin. *The Security Limitations of SSO in OpenID*. In Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on, volume 3, pages 1608 –1611, feb. 2008.

[Ope 2010] *OpenID Security Best Practices*, 2010.

[Ope 2011] *Open Web Foundation*, 2011.

[Overdick 2007] H. Overdick. *The Resource-Oriented Architecture*. In Services, 2007 IEEE Congress on, pages 340 –347, july 2007.

[Paul M. Duvall 2007] Andrew Glover Paul M. Duvall Steve Matyas. Continuous integration: Improving software quality and reducing risk. Addison-Wesley, 2007.

[Pierce 2002] Benjamin C. Pierce. Types and programming languages. MIT Press, 2002.

[RDF 2004] *Resource Description Framework (RDF)*, 2004. http://www.w3.org/RDF/.

[Recordon 2006a] David Recordon and Brad Fitzpatrick. *OpenID Authentication 1.1*, 2006.

[Recordon 2006b] David Recordon and Drummond Reed. *OpenID 2.0: a platform for user-centric identity management*. In Proceedings of the second ACM workshop on Digital identity management, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.

[Registry 2011] IANA Registry. *Link Relations*, 2011. http://www.iana.org/assignments/link-relations.xhtml.

[Richardson 2007] Leonard Richardson and Sam Ruby. Restful web services. O'Reilly, first édition, 2007.

[Robinson 2004] D. Robinson and K. Coar. *The Common Gateway Interface (CGI) Version 1.1*. RFC 3875 (Informational), October 2004.

[RSS 2009] *Really Simple Syndication (RSS)*, 2009. http://www.rssboard.org/rss-specification.

[Saint-Andre 2011] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120 (Proposed Standard), March 2011.

[Sandhu 1993] R.S. Sandhu. *Lattice-based access control models*. Computer, vol. 26, no. 11, pages 9 –19, nov. 1993.

[Schmidt 1992] Kjeld Schmidt and Liam Bannon. *Taking CSCW seriously*. Computer Supported Cooperative Work (CSCW), vol. 1, pages 7–40, 1992.

[Snell 2006] J. Snell. *Atom Threading Extensions*. RFC 4685 (Proposed Standard), September 2006.

[Snell 2007] J. Snell. *Atom License Extension*. RFC 4946 (Experimental), July 2007.

[Sparling 2011] E. Isaac Sparling and Shilad Sen. *Rating: how difficult is it?* In Proceedings of the fifth ACM conference on Recommender systems, RecSys '11, pages 149–156, New York, NY, USA, 2011. ACM.

[SPI 2011] *SPION D2.1 - State of the Art*, 2011.

[Sprague 2007] Robert Sprague. *Business Blogs and Commercial Speech: A New Analytical Framework for the 21st Century*. American Business Law Journal, vol. 44, no. 1, pages 127–159, 2007.

[Squicciarini 2009] Anna Squicciarini and Smitha Sundareswaran. *Web-Traveler Policies for Images on Social Networks*. World Wide Web, vol. 12, pages 461–484, 2009. 10.1007/s11280-009-0070-8.

[Squicciarini 2010] Anna Squicciarini, Federica Paci and Smitha Sundareswaran. *PriMa: an effective privacy protection mechanism for social networks*. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS '10, pages 320–323, New York, NY, USA, 2010. ACM.

[Stenmark 2002] Dick Stenmark. *Designing the new intranet*. PhD thesis, Göteborg University. School of Business, Economics and Law, 2002.

[Sánchez 2011] Víctor Sánchez and Antonio Tapiador. *Social network federation with Social Stream and Social2social*, 2011. http://d-cent.org/fsw2011/wp-content/uploads/fsw2011-Social-network-federation-with-Social-Stream-and-Social2social.pdf.

[Tapiador 2006] Antonio Tapiador, Antonio Fumero, Joaquin Salvachua and Sandra Aguirre. *A Web Collaboration Architecture*. In Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on, pages 1 –4, nov. 2006.

[Tapiador 2011a] Antonio Tapiador, Diego Carrera and Joaquín Salvachúa. *Tie-RBAC: An application of RBAC to Social Networks*. Web 2.0 Security and Privacy Workshop, 2011.

[Tapiador 2011b] Antonio Tapiador, Antonio Fumero and Joaquín Salvachúa. *Extended Identity for Social Networks*. In John Breslin, Thomas Burg, Hong-Gee Kim, Tom Raftery and Jan-Hinrik Schmidt, editeurs, Recent Trends and Developments in Social Software, volume 6045 of *Lecture Notes in Computer Science*, pages 162–168. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-16581-8_17.

[Tapiador 2011c] Antonio Tapiador and Antonio Mendo. *A survey on OpenID identifiers*. In Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on, pages 357 –362, oct. 2011.

[Tapiador 2011d] Antonio Tapiador and Joaquín Salvachúa. *Content Management in Ruby on Rails*. In Proceedings of the IADIS International Conference on Collaborative Technologies 2011, 2011.

[Tapiador 2012a] Antonio Tapiador and Diego Carrera. *A survey on social network sites' functional features*. In Proceedings of IADIS WWW/Internet 2012, 2012.

[Tapiador 2012b] Antonio Tapiador, Diego Carrera and Joaquín Salvachúa. *Social Stream, a social network framework*. In International Conference on Future Generation Communication Technology (FGCT 2012), 2012.

[Tapiador 2012c] Antonio Tapiador, Víctor Sánchez and Joaquín Salvachúa. *An analysis of social network connect services*. In Proceedings of WEBIST 2012, 2012.

[Thiruvathukal 2004] G.K. Thiruvathukal and K. Laufer. *Plone and content management*. Computing in Science Engineering, vol. 6, no. 4, pages 88 – 95, july-aug. 2004.

[Valdes 2005] Ray Valdes and David Mitchell Smith. *Web 2.0: Get Ready for the Next Old Thing*. Rapport technique, Gartner Inc., 2005.

[Wachob 2008] Gabe Wachob and Drummond Reed. *Extensible Resource Identifier (XRI) Resolution Version 2.0*, 2008. http://docs-oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html.

[Wasserman 1994] S. Wasserman and K. Faust. Social network analysis: methods and applications. Structural analysis in the social sciences. Cambridge University Press, 1994.

[Wild 2007] Fridolin Wild, Steinn E. Sigurðarson, Stefan Sobernig, Christina Stahl, Ahmet Soylu, Vahur Rebas, Dariusz Górka, Anna Danielewska-Tuecka and Antonio Tapiador. *An Interoperability Infrastructure for Distributed Feed Networks*. In Collaborative Open Environments for Project-Centered Learning, 2007. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-309/paper01.pdf.

[Wilson 1991] Paul Wilson. Computer supported cooperative work: An introduction. Kluwer Academic Publishers, 1991.

[XRD 2008] *eXtensible Resource Descriptor Sequence (XRDS)*, 2008. http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xri-resolution-V2.0-cd-03.html.

[Yad 2006] *Yadis*, 2006. http://yadis.org/.