



POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

A Virtuous Cycle of Semantics and Participation

Tesi di dottorato di:
Davide Eynard

Relatore:

Prof. Marco Colombetti

Tutore:

Prof. Andrea Bonarini

Coordinatore del programma di dottorato:

Prof. Patrizio Colaneri

XXI ciclo - 2009

POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32 I 20133 — Milano



POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

A Virtuous Cycle of Semantics and Participation

Doctoral Dissertation of:
Davide Eynard

Advisor:

Prof. Marco Colombetti

Tutor:

Prof. Andrea Bonarini

Supervisor of the Doctoral Program:

Prof. Patrizio Colaneri

XXI edition - 2009

POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32 I 20133 — Milano

Preface

On July, 3rd 1998, thanks to the recent birth of one of the very first free Web space providers, the dream of a read-write Web [56] finally became true for a (then) young computer engineering student, without him even knowing what a read-write Web was. A brand new website, containing just a collection of hacking and reverse-engineering related links was born.

On September, 7th 1999, the Perl programming language entered the life of that guy, and the website took advantage of the power of CGIs [61] to make its links *dynamic*: nobody knew what a folksonomy was, and metadata were just plain text reviews... But anyone could add new links to that site without being its webmaster, as it had been made read-write for them. Of course, almost nobody contributed.

On November, 2nd 2000, LAMP [159] (even if nobody knew what LAMP meant at that time) entered the life of that guy, and the website took advantage of the power of the PHP language and databases to evolve. Not just links anymore, but also a game: the site opened only for those few who were able to solve a riddle at its entrance; inside it there was a game, which allowed to advance level and get new resources whenever a riddle was solved. A community was born, even if the guy just used to call them *+friends*.

On April, 23rd 2001, a forum system was added to the website. Less than six hours later the first feedback message arrived. One week later, the first request: players wanted their rankings to be published. Incentives in the form of gratification entered the game.

In June, 2001, the first suggestions arrived through the forum. Users wanted to contribute in making the website better, giving their personal comments about the riddles, notifying bugs and errors, and requesting new features.

On September, 7th 2001, riddle number 5 appeared on the website: it was the first one created by a user [28].

The rest is history¹: <http://3564020356.org> has 28 levels, each one

¹Actually, Web history: everything can be found online on the Internet Archive's *Wayback Machine*, searching for the URLs <http://malattia.cjb.net> and <http://3564020356.org>.

with a riddle and a forum, and more than 20000 users. It has no advertisement and no spam, and most of its contents are not indexed by any search engine, as to enter it you still have to solve that first riddle. Nevertheless, the website still has around 2000 visits per month, even if it was officially frozen in 2005.

As one of the main rules of the game was “solve the riddles, no matter how you do it”, during its life the website has been attacked many times by its own players: every time a security hole was exploited, players used to send a brief explanation of how they cracked it so that the problem could be fixed and everyone else could learn something new out of it. In August, 2008, after years without new riddles, some users tried to exploit a bug in the website to gain full access on the server and... publish new riddles. Now, nothing can stop them from playing and learning.

356, as I like to call it, still means very much to me. It has always been there, looking at the world while it was changing. It survived the dot-com bubble, actually being a *dot-nothing* at that time². It changed place many times, starting with old tower PCs, moving to racks and finally retiring in a virtual machine. It saw generations of searchers and reverse-engineers playing with it, actually the most fair community I have ever seen. It was pure fun, participation, collaboration and self-moderation.

And I still haven’t understood why it worked so damn well.

Acknowledgements

My eternal gratitude and love go first of all to Elena and my family, for the patience they all had when I spent more time with the Semantic Web than with them. Then: my advisor, prof. Marco Colombetti, and my tutor, prof. Andrea Bonarini, for all the support they always gave and still give me. Dr. Sebastian Schaffert, the reviewer of this thesis, for his precious suggestions and the time he devoted to read the whole thing (with the hope he is not going to be the only one). My dear colleagues at DEI and HPL, as working with them has always been a great experience. And last but not least, all those friends who have shared in some way this experience with me, as your support and chats and whatever kind of more or less dangerous activity I have performed with you was precious to me. If you happen to belong to more than one of the aforementioned sets, please consider my acknowledgements as many times.

²The original url, which is also the reason of its weird domain name, was just <http://3564020356>.

Abstract

Participative systems are a particular class of social systems, in which people can interact, share information, or both. What characterizes them is not just a passive presence of users, but the fact that they actually “take part” into some activity, providing new value to the community thanks to their explicit or implicit contributes.

These systems have gained a great consensus both inside corporations and on the World wide web, and have become particularly interesting not only as a possibly successful business, but also as an object of academic research. In fact, as they deal at the same time with technologies and people, they present a plethora of interesting research problems that can be studied from the point of views of many different sciences.

The main objective of our research project is to study participative systems and the possibility to enhance them through semantics. We believe that the advantages of a contamination between these systems and Semantic Web technologies could be twofold: on the one hand, the huge quantity of information created by the participation of many users could be better managed and searched thanks to added semantics; on the other hand, Semantic Web community could exploit spontaneous participation to increase the amount of knowledge described through formal representations, making it available to many other applications. Following the double nature of this research (the social and the technological one) our research question is split in two: given a community, a task and a context, we want to understand (i) what is the most suitable tool to foster user participation and produce useful information, and (ii) how we could employ semantics to make this better.

Our effort in this project has been to find both the technical insights and the design paradigms to choose the best participative system for a particular community and its activity, or evaluate the health of an existing one, and address part of its limitations using Semantic Web technologies (after recognizing if it is possible in the first place). As a result, we provide a set of suggestions and patterns which can be used to bootstrap or fine tune a social system. To this knowledge we add the results of our experimentations on semantic participative systems. We describe how two of the main tools to manage semantics, that is

ontologies and reasoners, could be exploited to provide enhanced user interfaces for information visualization and consumption; we show how intrinsic limits in classification which depend on the ambiguity of English words could be addressed by semantic disambiguation techniques; finally, we show the advantages of sharing information as linked data in email servers and browser history, and how to use already available open information to make the browsing experience better and act as an incentive for user participation in collective activities.

Sommario

Quella dei *sistemi partecipativi* è una particolare classe di sistemi sociali all'interno dei quali è possibile interagire e condividere informazioni. A caratterizzarli non è una semplice presenza passiva degli utenti, ma il fatto che questi “prendono parte” a tutti gli effetti a una qualche attività, portando nuovo valore alla comunità grazie ai loro contributi espliciti o impliciti.

Questi sistemi hanno ottenuto un grande consenso sia in ambito aziendale che nel World Wide Web e risultano particolarmente interessanti non solo in quanto possibili business di successo ma anche come oggetti di studio accademico. Essi, infatti, basandosi allo stesso tempo sia su nuove tecnologie sia sulle persone che vi partecipano, presentano una grande quantità di problemi di ricerca interessanti che possono essere affrontati dai punti di vista di discipline differenti.

L'obiettivo principale di questo progetto di ricerca è lo studio dei sistemi partecipativi e la possibilità di migliorarli attraverso la semantica. Siamo infatti convinti che i vantaggi di una contaminazione fra questi sistemi e le tecnologie del Semantic Web possano essere duplici: da un lato, l'enorme quantità di informazioni generate dalla partecipazione di diversi utenti può essere gestita in modo più efficace grazie all'aggiunta di semantica; dall'altro, la comunità del Semantic Web potrebbe sfruttare la partecipazione spontanea per aumentare la quantità di conoscenza descritta tramite rappresentazioni formali e renderla disponibile a diverse applicazioni. Seguendo la duplice natura di questa ricerca (quella sociale e quella tecnologica), ci siamo ritrovati a rispondere a due domande: data una comunità, un compito da portare a termine e un contesto, desideriamo sapere (i) qual è il migliore strumento per sfruttare la partecipazione degli utenti in modo da produrre delle informazioni utili e (ii) come potremmo rendere questo strumento migliore attraverso l'uso della semantica.

Il nostro sforzo principale in questo progetto è stato diretto a trovare dei paradigmi tecnici e progettuali che ci permettessero, da un lato, di scegliere il miglior sistema partecipativo per una particolare comunità e la sua attività (o valutarne uno esistente), dall'altro di compensare parte dei suoi limiti grazie alle tecnologie del Semantic Web (dopo aver

riconosciuto, in primo luogo, se questo fosse possibile). Il risultato è una raccolta di suggerimenti e regole per la creazione o la correzione di un sistema sociale. A queste informazioni aggiungiamo i risultati dei nostri esperimenti su sistemi partecipativi semantici: mostriamo come due degli strumenti principali per la semantica, cioè le ontologie e i reasoner, possono essere utilizzati per fornire all'utente delle interfacce avanzate per la visualizzazione e la gestione delle informazioni; spieghiamo come i limiti intrinseci nella classificazione che dipendono dalle ambiguità lessicali possono essere aggirati tramite tecniche di disambiguazione semantica; infine, descriviamo i vantaggi ottenuti dalla condivisione delle informazioni come *Linked Data* in strumenti comuni come i server di posta e i browser, e mostriamo come sia possibile utilizzare informazioni già condivise su Internet per migliorare l'esperienza di navigazione e fornire incentivi alla partecipazione in attività collettive.

Contents

1	Introduction	1
1.1	Motivations	1
1.2	Objectives	4
1.3	Novel contributions	5
1.4	Structure of this document	7
2	Background	9
2.1	Social systems	9
2.1.1	Creating and publishing	11
2.1.2	Communicating	16
2.1.3	Sharing	19
2.1.4	Recommending	22
2.1.5	Coordinating	23
2.1.6	Networking	24
2.1.7	Playing	27
2.2	Taxonomy of participation	27
2.3	Basic concepts of social interactions	30
2.3.1	Wisdom of the crowds	30
2.3.2	Motivations and incentives	31
2.3.3	Communities of practice	33
2.3.4	Activity theory	36
2.4	Participation and semantics	38
2.4.1	Wiki systems	38
2.4.2	Folksonomies	43
2.5	Technologies	45
2.5.1	The Semantic Web	45
2.5.2	Tools	50
2.5.3	Datasets	53
3	Our approach	57
3.1	Designing a participative system	57
3.1.1	User-centered design	58
3.1.2	Dimensions of participative systems	60

3.1.3	Collective intelligence and LPP	65
3.2	Extending systems with semantics	68
3.3	Architectures	72
3.3.1	The client-server-server architecture	72
3.3.2	Wiki extensions	73
3.4	Experimental developments	74
4	Semantic Wikis	79
4.1	A Templating System for Resource Visualization in a Semantic Wiki	79
4.1.1	Project architecture and implementation	79
4.1.2	Conclusions	81
4.2	Semantic Management of Attachments Inside a Wiki Engine	82
4.2.1	Distillation	83
4.2.2	Visualization and Editing	84
4.2.3	Querying	84
4.2.4	Conclusions	84
4.3	Design of a Context Ontology Inside a Semantic Wiki . .	86
4.3.1	Conclusions	87
5	Folksologies	89
5.1	Using WordNet to Turn a Folksonomy Into a Hierarchy of Concepts	89
5.1.1	Project overview	90
5.1.2	Tag disambiguation	92
5.1.3	Building the tag semantic tree	93
5.1.4	User interface	94
5.1.5	System evaluation	96
5.1.6	Conclusions	98
5.2	Improving Search and Navigation by Combining Ontologies and Social Tags	99
5.2.1	Related work	100
5.2.2	Project overview	101
5.2.3	Matching and disambiguation	102
5.2.4	Project architecture	105
5.2.5	System evaluation	107
5.2.6	Conclusions	107
6	Linking Open Data	109
6.1	An IMAP Plugin for SquirrelRDF	109
6.1.1	Related work	110

6.1.2	Project overview	113
6.1.3	Project architecture and implementation	116
6.1.4	Software details	121
6.1.5	Tests and evaluations	125
6.1.6	Conclusions	128
6.2	Exploiting User Gratification for Collaborative Semantic Annotation	129
6.2.1	Related work	130
6.2.2	Project overview	131
6.2.3	Project architecture and implementation	134
6.2.4	System evaluation	138
6.2.5	Conclusions	140
6.3	Using semantics and user participation to customize per- sonalization	142
6.3.1	Prior work	145
6.3.2	Our Approach	148
6.3.3	Framework Architecture and Implementation	152
6.3.4	Example plugin: del.icio.us and Google's Motion- Chart	155
6.3.5	Example plugin: Freebase	158
6.3.6	Plugin Evaluation	160
6.3.7	Conclusions	165
7	Conclusions and Future Work	169
7.1	Future Work	171

1 Introduction

Participative systems are a particular class of social systems, in which people can interact, share information, or both. What characterizes them is not just a passive presence of users, but the fact that they actually “take part” into some activity, providing value to the system thanks to their explicit or implicit contributes.

Participative systems are widely used today, and are gaining a great consensus both inside corporations and on the World Wide Web. According to Tim O’Reilly [110] while explaining the meaning of “Web 2.0”, as applications are becoming more and more data-driven, spontaneous user participation adds value to a system because it helps in creating a new, unique and hard to recreate source of data. If this is evidently true on the Web (just think about different examples such as Wikipedia, Google Maps and Flickr¹), it is—maybe surprisingly—true even inside corporate intranets: in this case, as described in many success stories, participative systems are welcomed not only by companies, but also by the people who work for them, who consider these tools not only powerful, but also flexible enough to suit their own needs².

1.1 Motivations

The interest towards participation has grown in the last years to the point that almost every activity, whether it was offline or online, has seemed to become social. Just to give an example, we now have social news (such as in Digg or Reddit³), document writing (Google Docs⁴) and sharing (Scribd, Slideshare⁵), software development (SourceForge⁶), en-

¹<http://www.wikipedia.org>, <http://maps.google.com>, <http://www.flickr.com>.

²For instance, for an example of Wiki systems as successful participative tools inside corporations, see <http://twiki.org/cgi-bin/view/Main/TWikiSuccessStories>. For a definition of *Wiki*, see Section 2.1.

³<http://www.digg.com>, <http://www.reddit.com>.

⁴<http://docs.google.com>.

⁵<http://scribd.com>, <http://slideshare.net>.

⁶<http://www.sourceforge.net>.

1 Introduction

Figure 1. Hype Cycle for Social Software, 2008

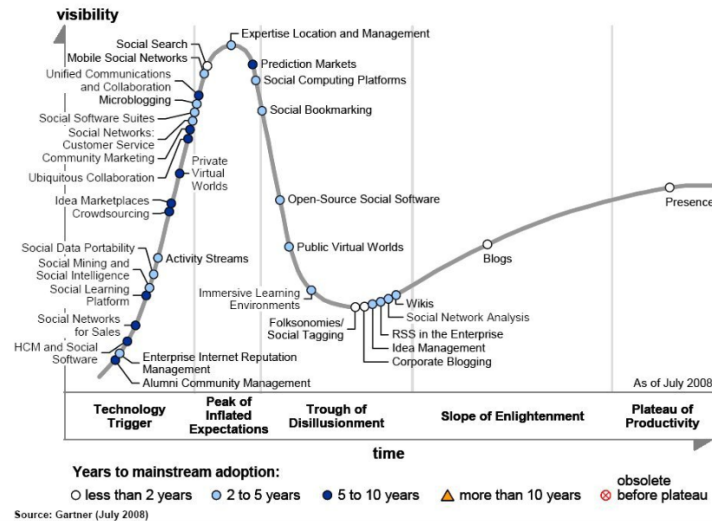


Figure 1.1: Gartner’s hype cycle for Social Software, July 2008.

cyclopedias (Wikipedia, Wikia⁷), and multimedia (YouTube, Last.FM⁸).

Another sign of where the current interest is going is that Gartner⁹, which every year provides a “Hype Cycle” about emerging technologies (a report with a graphic representation of the maturity, adoption and business application of specific technologies), in 2008 has published one exclusively for Social Software (see Figure 1.1).

Participative systems are particularly interesting not only as a possibly successful business, but also as an object of academic research. In fact, as they deal at the same time with technologies and people, they present a plethora of interesting research problems that can be studied from the point of views of many different sciences.

For instance, psychological and social sciences study the problem of the incentives in social systems¹⁰: users are more willing to contribute if their participation is incentivated in some way, so a consistent part of the designer’s work is to make the system gratifying or appealing for users.

⁷<http://www.wikia.com>.

⁸<http://www.youtube.com>, <http://www.last.fm>.

⁹<http://www.gartner.com>.

¹⁰The importance of incentives has been recognized even in the Semantic Web community, as shown by the presence of the first Workshop on Incentives for the Semantic Web at the 7th International Semantic Web Conference (see <http://km.aifb.uni-karlsruhe.de/ws/insemtive2008>).

From the point of view of human-computer interaction researchers, this translates in studying new interfaces and interaction paradigms to make systems more usable and intuitive. From a psychological point of view, instead, the “bootstrap problem” is a pretty common one: as few users are going to contribute to a project that still has not taken off, and as one of the main values they rely on to judge the success of a system is its contents, who is going to write the first critical mass of information, large enough to act as an incentive for users who contribute?

Another open problem is related to the precision of participative systems. There is always a tradeoff between users’ freedom of action and quality of the final data: the more open a system is, the more vulnerable to errors, no matter if they are entered by mistake or on purpose (as it might happen with spam). In some systems, users contribute with ratings or evaluations of particular resources (ie. news, links and so on). In these cases one might wonder how much these ratings can be trusted, given that not all the contributors are field experts. To describe this kind of dynamics James Surowiecki expresses his theory of the *Wisdom of Crowds* [142], writing that “under the right circumstances, groups are remarkably intelligent, and are often smarter than the smartest people in them”. Applied to the Web, this democratization has had some success, but not without drawbacks [125]: the main problem is that users’ evaluations are not unbiased, but often depend on previous ratings they trust unconditionally (this kind of “blind trust” is well described by Jaron Lanier in [88]); the consequence is that sometimes the decisions of few individuals, due to a cascade effect, have a disproportionately strong effect on the behavior of the group as a whole (like Malcolm Gladwell describes in [57]). Of course, these aspects have to be taken into account when designing a participative system.

Finally, even when the system works well and the user community is engaged and active, there is a problem with data. User contributions are often unstructured and produced just for human consumption, such as free text, images, or videos: they lack the structure and the metadata that would be useful for a machine to elaborate them. The result is that users often produce huge amounts of information which, unfortunately, is hard to find and reuse in other applications.

The Semantic Web [17] is “an extension of the current Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation”. Thanks to its standards and technologies, data produced by humans (no matter if in social systems or not) could be meaningful for machines too, so that they could help us

in searching, aggregating, and inferencing new information. It is quite natural, then, that the Semantic Web community is becoming interested into participation. Some consequences of this interest are the three workshops about Semantic Wikis held in the last years¹¹ and the birth of the W3C SWEO Linking Open Data community project, whose aim is to increase the amount of semantic data on the Web and link them together, also involving user participation.

1.2 Objectives

The main objective of our research project is to study participative systems and the possibility to enhance them through semantics. The advantages of a contamination between these systems and Semantic Web technologies are twofold (see Figure 1.2): on the one hand, the huge quantity of information created by the participation of many users can be better managed and searched thanks to added semantics; on the other hand, Semantic Web community can exploit spontaneous participation to increase the amount of knowledge described through formal representations, making it available to many other applications.

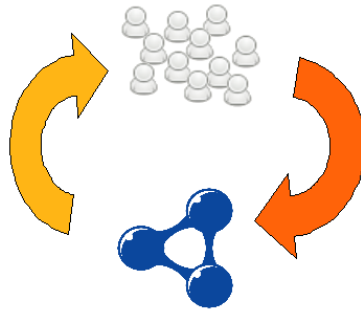
Following the double nature of this research (the social and the technological one) our research question is split in two: given a community, a task and a context (for instance, whether the community is in an intranet or on the Internet):

1. what is the best tool that can be used to foster user participation and exploit it to produce useful information? And
2. how can we make this tool better with semantics?

Actually, “making the tool better with semantics” still provides a very high-level description of our research question. Studying participative systems from different perspectives (i.e. from the point of view of Activity Theory and Communities of Practice - see Section 2), we came up to the conclusion that such a system might have success or not depending on many different factors: for instance, the tool might not be suited for the task, users might not be engaged, or the social dynamics of the underlying community might be incompatible with some restrictions imposed by the application. In these cases, even adding semantics to the system might not be enough to make it successful.

¹¹The first one has been held in June 2006 at the 3rd European Semantic Web Conference, the second in August 2006 at WikiSym, and the third one in June 2008 at the 5th ESWC.

Users, thanks to participation, create huge quantities of data. Our aim is to exploit spontaneous collaboration to increase the amount of formal knowledge.



Semantic Web technologies provide formalisms and tools for knowledge representation and management. Our aim is to use them to create more powerful and rewarding collaborative systems.

Figure 1.2: The virtuous cycle of semantics and participation.

On the other hand, there are other cases in which Semantic Web technologies could really add value to a system. However, when researchers or designers try to constrain information to a more formal structure, they often experiment a clash between the top-down approach, typical of ontologies and logical formalisms, and the bottom-up emergence of intelligence, which is typical of participative systems. Thus, the second point of our research question could be split in two different phases: first of all, we want to recognize if there actually is a place for semantics in the considered system; then, we want to find how to solve the problem of making the tool better using semantics without limiting, but rather fostering users' spontaneous participation.

1.3 Novel contributions

Literature shows many examples describing how semantics could improve knowledge management and how user participation could be used to produce huge amounts of information. However, research has only recently started to try to address both of the problems at the same time. Moreover, even in the cases in which this limit in scope is overcome, there is still often a limit in perspective: for instance, semantics could

be added to a participative system ignoring the interactions with users, thus making it less usable; or perfect solutions (from a technical point of view) could nevertheless lead to bad results into a system which does not fit its community's main activity.

Our effort in this project has been to find both the technical insights and the design paradigms to (i) choose the best participative system for a particular community and its activity, or evaluate the health of an existing one, and (ii) address part of its limitations using Semantic Web technologies (after recognizing if it is possible in the first place). As a result, we provide—in a very participative way—not a ready-made, top-down methodology, but a set of suggestions and patterns which can be used to bootstrap or fine tune a social system. Moreover, to these insights we add the results of our experimentation with Semantic Web technologies, and in particular:

- we show how two of the main tools to manage semantics, that is ontologies and reasoners, could be exploited to provide enhanced user interfaces for information visualization and consumption. In particular, working with *wiki* systems, we do not focus just on the semantics of their contents, but also on the “hidden layers” such as the context of use and the system itself, providing new models of interaction with the user (see Chapter 4);
- we show how intrinsic limits in classification which depend on the ambiguity of English words could be addressed by semantic disambiguation techniques. We apply these methods to the world of *folksonomies* (see Chapter 5), showing the advantages they carry both to information retrieval and to the users' browsing experience;
- we show the advantages of exposing and sharing information as *linked data* (see Chapter 6) in new fields: (i) exploiting data repositories such as Freebase to get the information to bootstrap new participative systems, (ii) exporting widely used formats such as IMAP to RDF to increase interoperability, and (iii) allowing the conversion of proprietary formats such as browser history to a standard shared representation that could be used more easily by browser-independent applications.

Finally, we bring in this document all the findings and the visions we collected after years of research on the topics of participation and Semantic Web. Even if we know that they might not be all novel for everyone, we hope that having all of them gathered in one place could be useful for at least some of the readers of this document.

1.4 Structure of this document

The document is organized as follows:

- Chapter 2 provides the background needed to understand the rest of the thesis: after a brief survey of the main social systems currently available and the introduction of a taxonomy for participation, we introduce some basic concepts about social interactions; we continue with a description of the current work on participation and semantics and conclude showing the main technologies and tools that we used inside our project;
- Chapter 3 shows our approach, which starts from the evaluation of the activity system as a whole (that is, considering not only the tool, but also its users as a community and their main activities) and continues with its extension thanks to semantics. We then describe our design approach, highlighting the architectures that are most common in our projects. Finally, we briefly introduce our experimental developments, which are then described with more detail in the following chapters;
- Chapter 4 is devoted to *semantic wikis*, that is semantic extensions to wiki systems, and the use of OWL ontologies and reasoning;
- Chapter 5 introduces the concept of *folksology*, that is the semantic extension of a folksonomy, and shows different applications of ontologies for the organization and visualization of unstructured, tag-based knowledge;
- Chapter 6 approaches the lowest level of semantics, that is linked data, and shows how “a little semantics goes a long way” [67] providing examples of data extraction, conversion and interoperability;
- finally, Chapter 7 shows our conclusions and suggestions for future work.

2 Background

This chapter describes the theories, techniques, and tools that we have used for our research project. As we showed previously, the success of participative systems is an interesting phenomenon that could be studied from many different perspectives: this is reflected in the different theories that are introduced in this chapter. However, albeit heterogeneous this collection might seem, for each approach we describe we also give our personal interpretation and motivations, thus providing a unifying view under our project's perspective.

The first section is a brief survey of existing social systems, categorized depending on their main features. The description of a *taxonomy of participation* follows, trying to shed some light on the different kinds of participation that characterize these systems. Then we introduce some basic concepts of social interactions, from the more informal ones such as the “wisdom of crowds” to the more formal ones, such as activity theory and communities of practice. The chapter continues with a description of the current work on participation and semantics (for what concerns our main fields of activity, semantic wikis and folksonomies) and ends showing the main technologies we used inside our projects.

2.1 Social systems

This section aims to be a survey on currently existing social systems, with the goal of giving a broad view of what is currently available and how it is mainly used.

One point which is very interesting from a social perspective is that many of these technologies are often used in practice for purposes that are different than the ones they had been built for. This might happen for different reasons, such as:

- availability: a particular system is the only one available for a community;
- locality: as social systems become *places* for people to gather around, a community might choose to let the tools provided by

a particular system constrain its activities just because it does not want to “move” elsewhere;

- **imitation:** as other similar communities had success using a particular system, new similar groups might tend to use the same one;
- **practice:** a community—not necessarily an online one—might already share some practices that are independent from the newly adopted system and try to shape it according to them.

Of course, not all of these systems work well in practice. However, what is most interesting is the fact that some of them actually do: for this reason, we decided to highlight these cases in the following descriptions.

The fact that many systems do not have a precise function, but rather the way they are actually used might change depending on their own communities, makes it difficult to categorize them as belonging to one category or another. So, we decided to group them by what we consider their main functionality, but always keeping in mind that they could be used for different purposes too. Moreover, we are sure that there are still many systems or tools that we have not shown or that could be described with more detail, but we decided to concentrate on the ones which seemed more related to our research. For those who want to delve deeper into this topic, a good starting point might be the “Collaborative Software” Wikipedia page [158].

The main categories we divided participative systems in are:

- **Creating and publishing**, with tools such as blogs, wikis, and realtime editors, which are used to collaboratively write texts;
- **Communicating**, with systems such as email, forums, and instant messaging tools;
- **Sharing**, which comprises all those systems that allow to share resources like videos, images, documents, and so on;
- **Recommending**, with all those systems where the fact of recommending something to others (either implicitly or explicitly) is central to the activity system;
- **Coordinating**, which comprises those systems that allow a group to coordinate or to manage a complex task together;
- **Networking**, with systems where linking to other users is the most prominent activity;

- **Playing**, with systems where playing together is the main activity.

2.1.1 Creating and publishing

Blogs

The word *blog* is the contraction of the term “Web log”, originally used to refer to Web sites organized as online diaries. Actually, modern blogs still have a structure which is very similar to the one of diaries: they are organized in articles and each of them is characterized by (at least) a date, a title, and a content. Usually articles are ordered by date, with the most recent one on top and the other ones following it; a fixed number of articles are shown into the main page, while older ones can be accessed through an archive, which is usually divided in months. Finally, each article can be given one or more categories, which are often specified by tags and allow to browse the archive according to another dimension.

Even if blogs started as a mainly textual medium, they can now be grouped into many categories which are specialized in different kinds of media: for instance *photoblogs* or *sketchblogs*, which are specialized on images, *vlogs*, devoted to videos, and *mp3 blogs*, which are used to publish audio files. Moreover, it is possible to use blogs both as personal websites (that is, with just one author publishing contents) or in a collaborative way, where the system aggregates in the same pages articles written by different authors. Posts can also be opened to comments, allowing for a two-way medium where the boundary between writers and readers becomes fuzzy.

All of these systems, however, share the very same structure and have a common representation of data, that has been standardized through the RSS format¹. The presence of a standard and the existence of many different applications using it to show, merge, or filter information, surely contributed to the success of blogs. According to Technorati², the blogs currently tracked are now more than 110 million as of August, 2008, and they grow at a pace higher than 120 thousands blogs per day (see Figure 2.1).

Wikis

Wiki is a term derived from the Hawaiian expression “wiki wiki” meaning “fast” or “quick.” Wikis are interlinked web sites, first introduced by Leuf

¹See <http://en.wikipedia.org/wiki/RSS>.

²See <http://technoratimedia.com/about>. For some less updated but more detailed statistics about blogs indexed by Technorati, check [134].

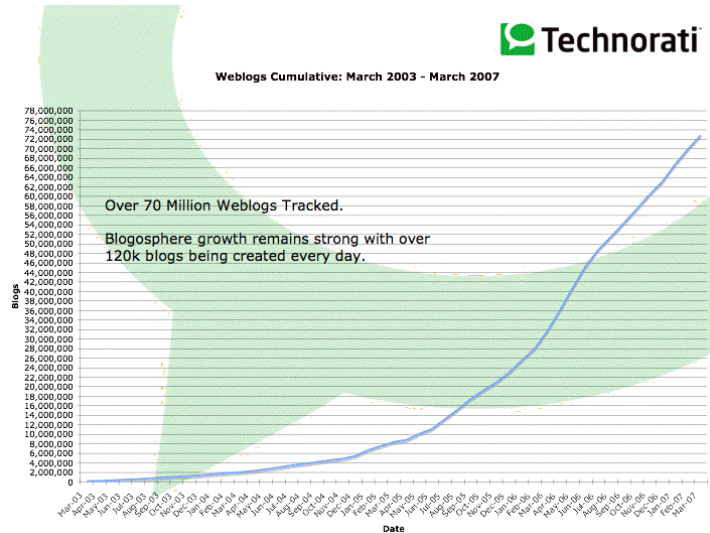


Figure 2.1: Growth of the number of tracked blogs from March, 2003 to March, 2007 according to Technorati.

and Cunningham [92], that can be collaboratively edited by anyone. On one hand, this means that wiki systems are so easy that even novice users can edit them: pages are written using a simple syntax, called *wikitext* or *wiki markup*, which can be learned very quickly. On the other hand, following the philosophy of making it easy to correct mistakes rather than making it difficult to make them, wikis usually put no restrictions on who can edit a page, so anyone can modify their contents. Of course this opens wikis to any kind of malicious contribution, such as spam or intentionally misleading information: for this reason, while some wikis still exist which have editing open to anyone, other more famous ones (such as Wikipedia³) decided to allow only registered users to modify part or all of their pages.

As described in [128], even if there is a wide range of wiki systems available, all of them share the same set of properties:

- **Editing via Browser:** pages can be edited directly from within the browser, without requiring any additional software to be installed. While this might not seem surprising in 2008, where blogs and many Web applications that allow to do the same thing are

³Wikipedia decided to close anonymous editing on a subset of its pages after the “Seigenthaler incident”, described in [160].

available, this was a great feature years ago, when the first wiki systems came out. One of its main advantages is that it adds immediateness to the system, as anyone who can see a wiki page has also at the same time all the tools that are needed to edit it;

- **Simplified Wiki Syntax:** the syntax used to describe how content has to appear in wiki pages (i.e. the text formatting features, the division of documents in sections, and so on) is much easier than HTML, which is the standard for generic Web pages. This makes it easier for users to write contents, as it does not require anyone to have any specific technical knowledge. Moreover, some wikis now also offer a WYSIWYG (“What You See Is What You Get”) interface, that makes them much more similar to any common word processor such as Microsoft Word;
- **Versioning:** wiki pages are *versioned*, that is there is a record for each change that is made to them. This is often described as the feature which balances their openness, as every change that contains errors could be eliminated by reverting to a previous version of the page. Whether this is actually feasible or not (this might change much depending on the type and the size of the system), it is still a precious tool as it allows users to check which are the latest changes or the differences between any two versions of the same page;
- **Strong Linking:** hyperlinking is one of the most important techniques for navigation inside wiki systems, as the only useful structure that can be given to a website that grows from the bottom up is the one that appears in its own pages. Thus, wikis usually allow users to create links in a very easy way (often just by writing words in *CamelCase*⁴) and there is not the concept of “dead link”, but rather any hyperlink that points to a page which does not exist shows an edit box that allow users to create that page.
- **Unrestricted Access:** as described previously, most of wiki systems allow anyone to create or modify pages without being a registered user. Some of them provide functions to restrict read or write access, making the wiki much more similar to a generic Content Management System tool.

⁴CamelCase is a way of writing words so that they start with and contain several uppercase letters. An example is the word “CamelCase” itself.

In the years many Wiki engines have been created, most of which are opensource⁵, and they are used for different purposes ranging from collaboratively edited encyclopedias (i.e. Wikia⁶) to software documentation (i.e. Jena⁷) and collaborative publishing⁸. Literature also shows experiments in wiki-based collaborative programming environments [62, 31]: source code is saved inside wiki pages and users can make it better or clone it into other pages to build their own applications. Acre is a more recent experiment⁹ which provides a full online development environment for Javascript-based, server-side applications: users cannot modify someone else's program, but can access others' source code and clone it into their workspace. Shared code allows users to build their work over someone else's efforts, avoiding to reinvent the wheel each time and, at the same time, providing useful feedback to the original programmers.

Collaborative editors

Collaborative editors are tools that allow to write documents collaboratively, helping to manage the problem of synchronization between many users. There are two main classes of systems, the synchronous and the asynchronous ones.

The class of tools that allow users to collaboratively edit documents in an asynchronous way contains the so-called *revision control systems*. These systems are usually centralized and enable developers to check out a copy of a document, edit it, and commit the changes back to the server. Some well known examples are CVS¹⁰ and Subversion¹¹.

One of the main advantages of these tools is that they do not only allow to keep a shared version of a document on a central server, but they also try to manage concurrent edits on the same documents. When this happens, if the edits have been made in different sections of the document they can be merged together. However, when concurrent edits are applied to the same part of the document, a conflict occurs: the

⁵See http://en.wikipedia.org/wiki/Comparison_of_wiki_software and <http://www.wikimatrix.org>.

⁶<http://www.wikia.com>.

⁷<http://jena.hpl.hp.com/wiki>.

⁸The project called "A Million Penguins" [96] describes a rather unsuccessful example of collaborative writing. A more succesful *writing community*, with a couple of published books, gathers at <http://ippolita.net/riso>.

⁹By Metaweb (<http://metaweb.com>), accessible at <http://dev.freebaseapps.com/wiki/index>.

¹⁰Concurrent Versions System, available at <http://www.nongnu.org/cvs>.

¹¹See <http://subversion.tigris.org>.

revision system is unable to merge both changes into each other and leaves the coordination task to its users.

As a consequence, this class of tools works very well on projects where many separate text files are involved (i.e. source code development): in this case, there are less chances that different users will edit the same document section at the same time. When, instead, the number of files is lower, chances for conflicts grow and the tool could not be able to automatically merge all the edits.

Finally, it is worth to note that these systems alone are not able to understand which edits could be conflicting when they do not belong to the same section, such as functions which are called in different parts of a program or formulas that are conditions for a theorem proof. The reason is that in these cases the system is not able to understand the *semantics* of what is written, thus requiring a coordination effort by the users.

Tools trying to overcome this limit by allowing users to coordinate in real-time, while they are writing a document, belong to the class of “collaborative real-time editors”. These systems show users a shared view of the same document, where everyone can see in real-time all the changes done by others. Some examples of these tools follow:

- Abiword (<http://www.abisource.com>) is an opensource word processor that provides a plugin, called AbiCollab [130], which allows to access remote documents on another user’s machine and edit them collaboratively;
- Gobby (<http://gobby.0x539.de>) is a collaborative realtime text editor which also provides a chat system, allowing users to communicate while they are writing (see Figure 2.2);
- Google Docs (<http://docs.google.com>) is a word processor that runs as a Web application. Some of its main advantages are that it allows users to share documents and edit them together in real-time; it has a discrete compatibility with the most common word processors, such as Microsoft Word and Openoffice; it works inside a browser, so no other program is needed; finally, it allows users to save documents on a personal space online, so that they can open them from any other computer which has an Internet connection.

2 Background

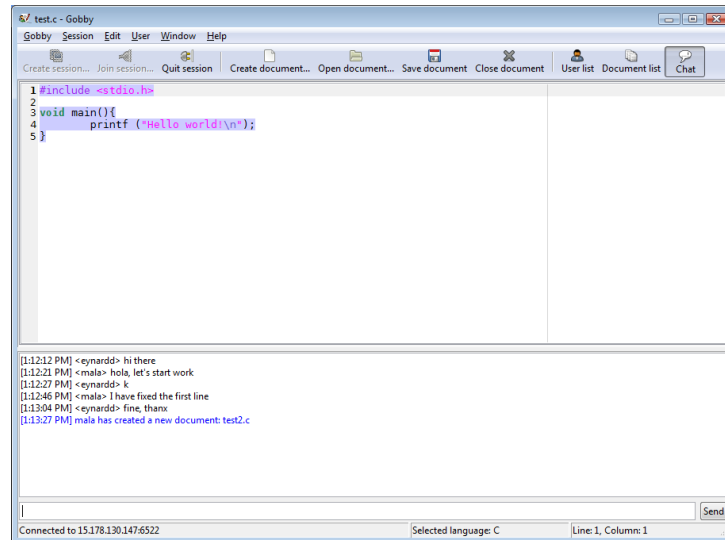


Figure 2.2: A screenshot from the collaborative realtime editor Gobby: users are editing a shared document and communicating via chat in the same window.

2.1.2 Communicating

E-mail

Being one of the first services set atop the Internet, email might seem outdated if compared with the most recent ones. However, it is still widely used and has a very important role into social systems thanks to its flexibility. With email, for instance, it is possible to:

- communicate with other users who have an email address;
- communicate with mobile phones, sending and receiving short messages¹²;
- browse the Web¹³;
- exchange any kind of file as an attachment;
- manage group communication with *mailing lists*;
- broadcast news with *newsletters*.

¹²See http://en.wikipedia.org/wiki/SMS_gateway.

¹³See, for instance, <http://www.www4mail.org>.

A feature which is typical of email is that, even if the system has been built to be “on demand” (that is, users have to connect to the email server in order to download their new messages), it actually works most of the times as a “push” medium. In fact, users often just leave their email clients open, automatically connecting to their mailboxes to see if there is something new. This is the reason why many social systems, which otherwise would require users to explicitly connect to them to have updates, keep them updated by email. For instance, MediaWiki (the same software that is used for Wikipedia) allows users to subscribe to some wiki pages and automatically be updated by email whenever one of them is modified. Facebook also uses this technique, providing users with a huge quantity of often totally uninteresting details by default (fortunately, it is possible to turn this feature off for most of the updates).

Forums

Forums are the Web descendants of the old BBSes (Bulletin Board Systems) and Usenet groups [161]. With both they share the possibility to publicly exchange messages with a group of users, and at the same time to leave these messages into an archive which is available for others to read. However, they are still different from Usenet as they do not have a network structure: actually, even if there are some experiments to create peer-to-peer forums, almost all of them are still services installed on single servers.

Forums are a very versatile tool, as they allow users not only to communicate but also to exchange any kind of resource. Some of them also take care of user privacy, allowing the exchange of private messages and allowing only registered users to write, and in some cases even to read their contents. This also offers some protection from Web crawlers (so forum contents will not be indexed by search engines) and from spammers.

Similarly to email, forums have also been used for many unforeseen purposes, such as file or link sharing: actually, they are often used as a high-quality, community-driven backbone for many (more or less legal) peer-to-peer systems. In the years they have not drastically changed, but they have slowly evolved into more complete services, providing more and more features to their users. As a signal that there is still interest around this kind of systems, earlier this year a new forum hosting service called Lefora¹⁴ has been opened, offering space and tools for anyone to open their customized forum.

¹⁴<http://www.lefora.com>.

Chat

Chat is another historical way of communicating over the Internet: IRC (Internet Relay Chat), for instance, is a communication protocol that was created in 1988 and there are still many networks now which rely on that. Chat systems have then evolved into Web-based ones and IM (Instant Messaging), offering users different interfaces and ways of interacting. However, their main characteristic remains always the same: they allow users to communicate synchronously.

IRC and many Web-based chats allow users to join *channels* (or *rooms*) devoted to different topics: after joining a channel, it is then possible to see the list of other participants and exchange messages with them, either privately or with the whole group. Instant Messaging systems offer one more feature: as everyone has to log into a central server, users can keep a list of their friends' accounts and see when they are online: this also makes the tool much more used for exchanging messages with single users rather than with groups.

IRC chats are characterized by the presence of *bots*, that is software agents that use the IRC protocol to connect as users and perform automatic tasks. For instance, some of them are used to provide authentication mechanisms inside a channel (giving users higher privileges if they identify themselves as registered), others to automatically moderate channels, and others to allow users to download files. This last category of bots transformed IRC in another file sharing network, with particular characteristics which make it unique: first of all, it is not a peer-to-peer network (see Section 2.1.3) but it is a client-server one (where servers can be users' machines too); then, it has no official, centralized list of files as anyone can be a file provider; similarly, as different softwares can be used for bots, it also has no standard way for searching and downloading files. As a consequence of this, file sharing over IRC is a practice which is not very common and is usually done just by few, more expert users.

Micro-blogging

Micro-blogging is a form of blogging that allows users to write brief text updates (usually 140 characters) and publish them, either to be viewed by anyone or by a restricted group which can be chosen by the user. Given the size of the messages and the different media these kind of systems can run on (messages can be exchanged i.e. by text messaging, IM, and email), micro-blogging could be considered more similar to a communication system rather than a way to publish contents online.

The most popular systems are Twitter¹⁵ and Jaiku¹⁶, but the total number of similar services is very high (more than one hundred in May 2007) and increasing. Moreover, many other social systems such as Facebook provide users with a way to update friends about their “status” which is very similar to micro-blogging.

2.1.3 Sharing

Systems belonging to this category allow users to share resources on the Internet. A resource is, using RDF terminology (see Section 2.5.1), “anything that can have a URI”. Thus, in these systems users can either share files of any kind (such as in *file sharing* applications) or URLs that point to them (such as in *social bookmarking* systems).

The classic client-server model

Before peer-to-peer, file sharing mostly followed the classic client-server model: few internet users, who had a chance to upload their data on some server, could make their files available to others via the HTTP or FTP protocol; it was left to their choice whether these files had to be accessible to anyone or just to a small group (usually authenticated with a password). The main limit of this system was that information was scattered around many different servers and there was no central authority providing a list of the available files. One solution was to use different specific search engines; the other was to rely on the many different communities that emerged at that time, which gathered around specific topics (i.e. security, videogames, music) and provided updated and commented links for different kind of resources.

Peer-to-peer

With peer-to-peer users can share files from their own computers without the need for third-party servers. Actually, in some p2p networks central servers are still present to provide a centralized index of all the available files; however, in many recent systems file requests are broadcasted over the whole network, so that no peer has a higher importance than others (which is a strategically important detail, as nobody can stop the system just by shutting down a single computer).

¹⁵<http://www.twitter.com>.

¹⁶<http://jaiku.com>.

2 Background

In peer-to-peer a radical change in user interactions occurs: to make the network more efficient, many p2p systems require users to share files while they are downloading them; thus every user is at the same time a provider (often without even knowing it) and a consumer.

The modern client-server model

In more recent times, new services became available which allow users to share files but, at the same time, take advantage of their centralized nature to provide a new value from the collection of everyone's shared material. Some examples of these systems are Flickr for images, Scribd for documents (i.e. plain text, PDF, Word, etc.), Slideshare for presentations, and YouTube for videos. The architecture of these tools is still client-server, but most of them are characterized by some additional features:

- differently from most of peer-to-peer networks, they mostly deal with user-generated material¹⁷;
- they often specialize in one or few specific file types and offer the chance to open them directly within the browser. This allows users to access any file they uploaded wherever they are, provided they have an Internet connection, without the need of any specific application;
- they often do not require users to share their files with everyone, but most of the times users *want* to do that anyway;
- users are not scattered among many different servers anymore, but fewer servers become central places for the activity of file sharing. This allows these systems to aggregate contributions from many users in different ways, thus providing incentives for those who contributed. For instance, users might be requested to rate an item and then they are provided with resources which could fit their tastes; files are clustered by similarity (either of the files themselves or of their metadata, i.e. tags) and "related resources" are automatically provided; search functions are present to help users in finding what they could be most interested in.

¹⁷Actually, there is a huge difference between *user-generated* and *original* or *non-copyrighted* material: this is still an open problem, as described in [4, 8] for the case of YouTube.

This model works fine with many different pieces of shared information, even if they do not come in self-contained files. For instance, Bibsonomy¹⁸ offers the chance to save and tag bibliography items inside a personal space. Once saved, they are made public to everyone and can be searched by tag, title or author, shared with other users, or exported to many different formats (i.e. BibTeX, EndNote, HTML, etc.).

Social bookmarking systems

Anything which is available on the Web, being uniquely identified by its URL, is a particular type of resource: thus, it can be added metadata and shared with others. Social bookmarking systems allow their users to save their favorite URLs inside an online repository; these bookmarks can be made public or private, but often they are shared with everyone; moreover, the user who saves them can categorize them with short, easy to remember text strings called *tags* (for a more in-depth description of tags, see Section 2.4.2).

The first advantage of using such a system is that the collection of bookmarks is stored online and can be accessed anywhere: this means that it cannot be lost easily, it can be shared between many computers, and is completely application independent as any browser can access it. This approach is very similar to the one used for other resources: basically, this kind of systems give users some online space where they can save their own data. However, in this case someone's "own" bookmarks refer to websites which are accessible (and possibly known) by anyone else. This overlapping between published resources automatically gives the possibility to infer some useful information: for instance, users who share many identical URLs or tags probably have similar interests; if the same set of tags is used for the same URL by many different users there probably is some kind of agreement about how to categorize it; searching for one particular set of tags a user might find new, possibly interesting URLs shared by someone else.

One of the most famous social bookmarking system is del.icio.us¹⁹, with about five million users as of August 2008²⁰. Many other similar websites, of course, are available on the internet (i.e. [Magnolia](http://ma.gnolia.com)²¹), trying to provide the same features plus some more; none of them, however,

¹⁸<http://www.bibsonomy.org>.

¹⁹<http://del.icio.us>, recently moved to <http://delicious.com>.

²⁰Information taken from <http://blog.delicious.com/blog/2008/07/oh-happy-day.html>.

²¹<http://ma.gnolia.com>.

has reached the same success that *del.icio.us* had. The main concept, that is saving URLs together with some attached metadata and getting value from aggregating this information, has been slightly changed and reused in other systems: for instance, *semantic annotations* work on very similar principles; file-specific bookmarking services can take advantage of the known filetype to provide online access to the saved resources (i.e. *Webjay.org*, now acquired and closed by Yahoo!, shared playlists built as collections of URLs of audio or video files); news services like *Digg* (see next section) or social systems like *Twine*²² allow to share and comment news-related URLs.

2.1.4 Recommending

One of the main characteristics of social systems is that it is often possible to infer new information from the aggregation of users' contributions. One of the first example of this inferred information is represented by *recommendations*²³.

Information used for recommendations can be gathered by the system or provided by users themselves. For instance, inside the two social news websites *Digg* and *Reddit*, users can vote their favorite news allowing them to advance to the front page. *Last.fm*, instead, offers personalized streaming music by allowing users to rate the songs they listen and suggesting related ones. While in the former case users explicitly specify their preferences, in the latter all users have to do is listen to music: if they listen to a song up to its end this is considered as a positive feedback; if they skip to the following song it is considered as a negative one; however, they can also give stronger feedbacks by clicking on the "love" or "ban" buttons on their players.

Another category of recommendation-based systems is represented by the so-called *social libraries*, which allow users to keep track of their interests online, saving a collection of their favorite books, movies, restaurants, or even beers²⁴. For each item, users can specify a rating which might help others to choose between resources and, at the same time, gives the system the information it needs to suggest users the related items they might like.

Many specific review systems are available which just collect reviews

²²<http://www.twine.com>.

²³According to Steven Johnson's interview about *emergence* [135], Amazon was already providing book recommendations back in 2002.

²⁴An example of each is available at <http://livingsocial.com>.

Planet Terror

Links

Homepage: <http://imdb.com/title/tt1077258/>
 See Also: http://en.wikipedia.org/wiki/Planet_Terror

Tags

[amputation](#) [body-modification](#) [bruce-willis](#) [cheese](#) [comedy](#) [exploitation](#) [film](#) [grindhouse](#) [homage](#) [horror](#) [movie](#)
[pastiche](#) [robert-rodriguez](#) [rose-mcgowan](#) [shoot-out](#) [texas](#) [zombie](#)

Reviews (1)

★★★★★ by martinp on 14 Jan 2008

Writer and director Robert Rodriguez has hit his very best with this horror comedy. It weaves together a zombie storyline with a Texan chef's search for the perfect barbecue sauce recipe. Relentless in its pace, it packs in a great many visual and dialogue gags and outrageous action while burning some very scary images into your retinas. Despite being a loving tribute to the exploitation movies of the grindhouse era (complete with film imperfections), it has a pretty coherent plot and - surprisingly - genuine character development. It packs in scenes of gore and disease that rival the sickest underbelly of the web: you have to applaud the director's visual imagination, then ask for his to be sectioned. Other zombie movies look po-faced by comparison. If you like this sort of thing, this is a great evening's entertainment.

```

<rdf:RDF xml:base="http://revyu.com/">
- <rdf:Description rdf:about="things/planet-terror-robert-rodriguez-bruce-willis-film-movie-ho">
  <rev:hasReview rdf:resource="reviews/ff320d73fee0e1d212a23dcd82b5b6ceac784d18"/>
  <tag:tag rdf:resource="taggings/ff320d73fee0e1d212a23dcd82b5b6ceac784d18"/>
  <rdf:Description>
- <owl:Thing rdf:about="things/planet-terror-robert-rodriguez-bruce-willis-film-movie-ho">
  <rdf:label>Planet Terror</rdf:label>
  <rdf:seeAlso rdf:resource="http://en.wikipedia.org/wiki/Planet_Terror"/>
  <foaf:homepage rdf:resource="http://imdb.com/title/tt1077258"/>
  </owl:Thing>
- <rdf:Description rdf:about="taggings/ff320d73fee0e1d212a23dcd82b5b6ceac784d18">
  <rdf:label>
    A bundle of Tags associated with this Thing, defining when they were added and by whom
  </rdf:label>
  <rdf:Description>
</rdf:RDF>

```

Figure 2.3: The review for a movie on Revyu and its RDF representation.

for particular classes of items: Yelp²⁵, for instance, is specialized in places, with 22 different categories which range from restaurants to religious organizations; Epinions²⁶, instead, is mainly specialized in products. A very nice experiment is the one made by Revyu, which merges reviews with Semantic Web technologies allowing users to review anything which has a URI and specify its category using tags. Information is then made available in RDF and can be queried through a SPARQL endpoint.

2.1.5 Coordinating

Between social systems there are some built to help users coordinate their collaborative efforts. Basically, they allow to share information about the group's common activities, such as data about the group itself (all users with their contacts and their roles), the object (i.e. a document or a program), the communications inside the group (i.e. bug alerts, call

²⁵<http://www.yelp.com>.

²⁶<http://www.epinions.com>.

for participation, or generic messages), and everything which is related to the management of time (project timeline, meetings, and so on).

Depending on the activity that takes place, different tools can be considered more or less useful by the group for its coordination. A description of the most common ones follows:

- *electronic calendars*, such as Outlook, iCal or Google Calendar, allow to organize meetings and share events with other users. The wide diffusion of the iCalendar standard [38] allowed many different systems and applications to support the same calendar information, making it not only sharable between different users but also portable everywhere, from Web servers to mobile devices;
- *project management systems*, such as SourceForge or Savane²⁷, do not only offer space for a project's files but also give all the tools that are needed to follow its development, such as a tracker (to manage bugs, feature requests, statistics etc.), forums and CVS/SVN;
- *online spreadsheets*, such as Google Docs' spreadsheets, allow not only to collaboratively and synchronously edit the same document in realtime (as already described in Section 2.1.1), but also to share structured data with others;
- *workflow management systems* allow for the collaborative management of tasks and documents within a knowledge-based business process;
- *knowledge management systems* allow to collect, share, and manage information in different forms.

2.1.6 Networking

Following the definition in [24], we consider social networks those “Web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system.” Note that while generic connections between users have already been shown within other systems (such as “similar users” for recommendation systems or

²⁷<https://gna.org/projects/savane>.

tag-based ones), social networks are different as they allow any user to explicitly specify her own relations, such as friends or business contacts.

Social networking sites are widely known (especially because of the great success of services like MySpace²⁸ or Facebook) and have recently become very common (see Figure 2.4), reaching the point that a need for a semantic “portable user profile” has been envisioned [23]. Different networks often have different purposes: for instance, MySpace is more music oriented, while LinkedIn is devoted to business contacts and Facebook to real-life friends.

From the very first social networks, which just provided the means to interconnect users and had a rather short life, a more modern approach evolved that could be defined as *object centered sociality* (as defined by Jyri Engeström [43], borrowing the term from Karin Knorr Cetina), where social networks consist of people who are connected by a shared object. Engeström uses as successful examples Flickr (where the objects are photos), del.icio.us (with bookmarks) and Upcoming.org (with events). An extension of this model is represented by Facebook applications, which can be programmed by anyone and provide more or less useful/entertaining mediating objects for users to build connections through.

Another example is Naymz²⁹, which has a network similar to the one provided by LinkedIn (actually, it offers the possibility of automatically importing contacts from LinkedIn, together with the main email clients), but centers on references and reputation: the mechanism of the RepScore (a score from 1 to 10 which depends on many parameters, such as the completeness of the personal profile, the number of contacts and the references provided by them) works as an incentive to make people build and maintain a social network.

Finally, Twine³⁰ and Plaxo³¹ build their own social network on an aggregation of objects and other already existing networks. They allow users to specify the details of their other accounts and automatically import and aggregate pieces of information that are published elsewhere. The main advantage of these services is not only the fact that they allow to build a social network out of common interests, but also that they work as community-based filters for new, incoming information: in fact, only updates generated by connected users or groups are usually shown.

²⁸<http://www.myspace.com>.

²⁹<http://www.naymz.com>.

³⁰<http://www.twine.com>.

³¹<http://www.plaxo.com>.

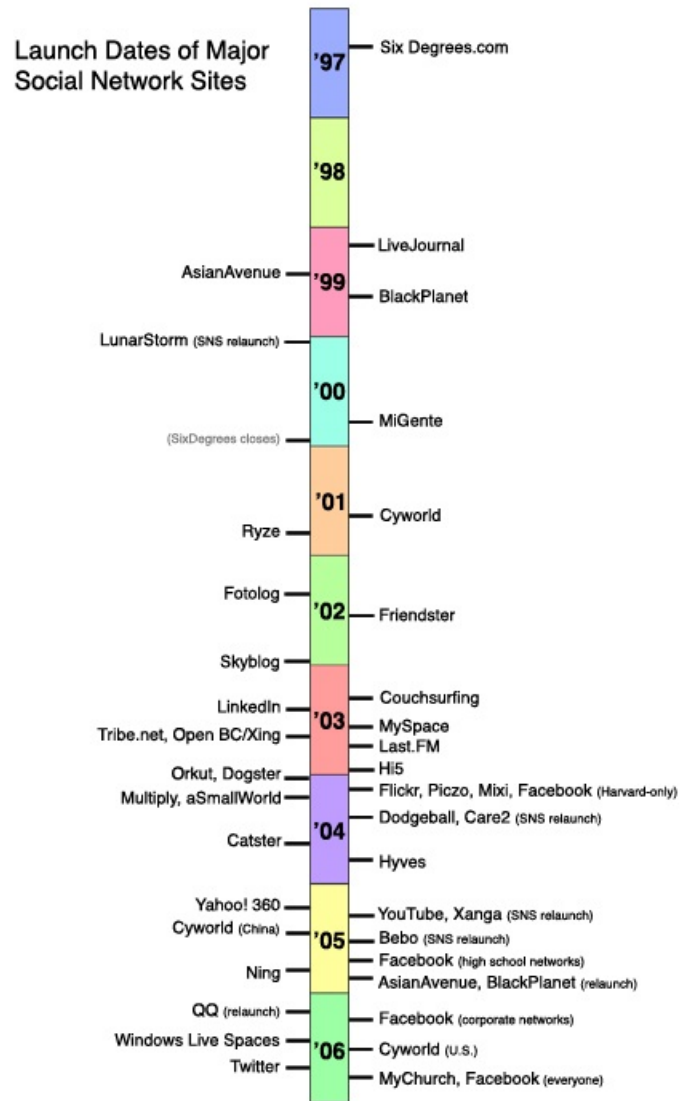


Figure 2.4: A timeline of social networking services, courtesy of D. M. Boyd & N. B. Ellison.

2.1.7 Playing

If social networks work well when they are centered on a common object, making the activity of playing as central has always been a strong incentive for users to gather. Starting from the first MUD³² systems and arriving to the latest MMORPGs³³ such as World Of Warcraft, these systems allow users to assume the role of a fictional character (often in a fantasy world) and interact with other users or with computer-operated characters. Differently from single-player games, the virtual world in which the game takes place is persistent, that is it evolves even when the player is away from the game.

Another example of virtual world is Second Life, where players do not have a particular quest to solve but they can do anything, from building their own house to running an online business. The main characteristic of Second Life is that it introduced money in a virtual world, attracting not only the interest of gamers but also the one of many commercial entities.

Finally, PMOG³⁴ is a game which highly exploits user participation and at the same time mixes it with Web browsing and metadata. Using a browser extensions, users can attach different types of metadata to any website (hidden information, traps, presents, and so on) and gain points and rewards just by browsing the Web. This idea is particularly interesting for us as it uses architectures and technologies which are very similar to the ones we adopted in some of our experimental developments.

2.2 Taxonomy of participation

The reason why this section is called “taxonomy of participation” and not “taxonomy of participative systems” is the following: while it has been not so hard to give an interpretation to the basic concepts related to participation and then order them according to what they meant to us, it would be much harder to actually classify a social system in a unique way. This happens because almost every existing system (or system family) is characterized by different levels of participation, so the same one could be classified in different ways according to how it is used. Nevertheless we found this taxonomy useful, as it provides a common

³²Multi User Dungeon, see <http://en.wikipedia.org/wiki/MUD>.

³³Massively Multiplayer Online Role Playing Game, see <http://en.wikipedia.org/wiki/MMORPG>.

³⁴Passively Multiplayer Online Game, see <http://pmog.com>.



Figure 2.5: A possible taxonomy for participation.

and more precise way to describe the different forms of participation we were able to find in social systems.

We decided to use the term *social* to describe all those multi-user systems in which people can interact, share information, or both. Most of these systems can either be found on the Internet or inside enterprise intranets. One of their main characteristics is that, differently from previous multi-user systems, they exploit not just the presence of users inside the system, but also the interactions and the relations that occur between them. These interactions and relations can be either explicit or implicit, in the sense that they can be performed or specified either by the user or by the system itself: some examples follow in Table 2.1.

Our definition of social systems is broad enough to account for a lot of different tools and technologies, so we decided to refine it by suggesting

	Interaction	Relation
Explicit	Email communication	User adds friends to his Facebook ³⁵ network
Implicit	Feed subscribers get new articles from a blog	Users are automatically linked by a clustering algorithm according to their behavior

Table 2.1: Examples of explicit and implicit interactions and relations.

some subsets which could better describe the specific behavior of particular families of systems. The higher-level subset we defined is the one of *participative* systems, in which users are not just present, but actually “take part” into some general activity. This definition mainly focuses on the fact that users are active in the system, no matter if they actually contribute to it. As an example, think about users who listen to music on last.fm³⁶: even if this is their only action, they participate to one of the system activities providing useful information, even if they do this in a totally implicit way. In this case, the tool takes care of aggregating data from user behavior and elaborates them to provide better suggestions. Systems like this can be described from two different point of views, the one of the user and the one of the tool, and they are often also defined as *collective* [154].

Within participative systems there are some that we define as *contributive*. The word “contribute” derives from the Latin *contribuere*, which is built on the two words *cum* (“together”) and *tribuere* (“to give”). As the word implies, in this case users do not just take part into an activity, but they also explicitly provide something to the system. In one of the most basic definitions we found, every contribution could be classified as a creative action, a suggestion or a resource. Trying to give a more modern, internet-related interpretation, an action could be the creation and publishing of some Web contents; a suggestion could be a feedback, like a comment inside a blog, a rating, or a classification of some resource; a resource could be represented by a multimedia file, such as a video or an image, or more generally by a URI (Uniform Resource Identifier), which uniquely identifies a resource on the Internet. All of these different contributes can be provided by users who are working alone or together with others. Systems which allow for this last kind of activities, where users work together with a common goal, are called *collaborative* (from the latin *cum+laborare*, “work together”).

As we previously stated, it is almost impossible to univocally classify a system according to our taxonomy. For instance, Wikipedia articles have been often used as an example of a collaborative effort, as many users often converge on the same article and work together to build or enrich it; however, in many cases users participate with single, atomic contributions, such as small stubs or error corrections, for which no coordination with other users is required. Conversely, a tag-based system like del.icio.us is not considered collaborative, as usually everyone tags his own bookmarks for personal purposes; however, people can use it

³⁶<http://www.last.fm>.

collaboratively by agreeing on using the same tag set to build a personal collection of links, or by suggesting links to other users by using special “for:username” tags.

To describe why classification is such a difficult task for participative systems we use the “county fair example”. In a county fair, a lot of different people participate: the organizers collaborate to prepare the event; donors contribute with offers, builders prepare the stage, and everyone is going to provide suggestions about how to organize everything; food sellers are there for their own business, while normal people just go there to have fun. However, the success of the fair depends on each one of them. Similarly, any non-trivial social system opens to (and depends on) different kinds of participation, allowing anyone to be more or less involved in it.

This example describes participation only from the system’s point of view, temporarily ignoring the presence of users. However, we know that for a participative system to work we need not just the system itself, but also user contributes. For this reason, in the next section we describe, together with the basic concepts of social interactions, the different levels of participation from the user’s point of view (see Legitimate Peripheral Participation), showing that they are strictly connected with the ones that are provided by the system.

2.3 Basic concepts of social interactions

2.3.1 Wisdom of the crowds

One of the concepts which are most quoted when trying to describe why collective intelligence works is the “Wisdom of crowds”, from the book written by journalist James Surowiecki [142]. Quoting an experiment made by the scientist Francis Galton in 1906, when a heterogeneous group of 787 people was able to guess (on average) the weight of an ox, he concludes that “under the right circumstances, groups are remarkably intelligent and often smarter than the smartest people in them, especially if individual guesses are aggregated and averaged”.

Surowiecki then shows that for the conclusion to be valid some requirements have to be met:

- *diversity of ideas*, as it adds perspectives that would otherwise be absent. The lack of difference would lead to what is defined as “collective cognitive convergence” [113], which facilitates mutual understanding and coordination, but if left unchecked can lead the

group to collapse cognitively, becoming blind to viewpoints other than their own. On the contrary, the presence of differences would make the group more versatile and better at solving problems;

- *independent thought*, meant not as isolation but rather as relative freedom from the influence of others. When this does not happen the crowd does not behave as wise anymore, as described in [125, 88];
- *decentralization*: this means that the power of the group does not fully reside in one central location, and important decisions are made by individuals based on their own knowledge.

2.3.2 Motivations and incentives

Motivation is (see Wikipedia) “the reason or reasons for engaging in a particular behavior”. An incentive, instead is “any factor that provides a motive for a particular course of action, or counts as a reason for preferring one choice to the alternatives”. As described in [79], some motivations for contributing could be anticipated reciprocity, reputation, sense of efficacy, need, and attachment or commitment to a community. Examples of incentives, instead, could be features like ongoing interactions, identity persistence, knowledge of previous interactions, visible contributions and recognitions, and the presence of well defined group boundaries.

Even if for clarity of presentation motivation has sometimes been referred as *internal* (or intrinsic) and *external* (or extrinsic), it actually is something that spawns and grows only within an individual and cannot be created or given by someone else. So, to drive people’s actions through motivation (i.e. in our case, making a person participate to a collective activity) it is necessary to give them the incentives which could provide the right motivations.

Actually, this is a simplification and both incentives and motivations work in a much more complex way. Different sciences approach the problem of defining or classifying them from different perspectives, and often yield to different solutions too. For instance, a classification of incentives that is very common in economics [74] is the following one: *remunerative* incentives exist if it has been made known to a person that behaving in a particular way will result in some form of material reward she will not otherwise receive; *moral* incentives are present if a person has been taught to believe that behaving in a particular way is the “right” or “proper” or “admirable” thing to do; *coercitive* incentives are said to

exist where a person can expect that the failure to act in a particular way will result in physical force being used against her.

These classes, however, do not comprise all those *personal* incentives that might motivate an individual. These are described much more in detail, instead, by psychology and sociology, which take into account concepts such as tastes, desires, and sense of duty and so on. For instance, among the incentives to join a group or social network there is the fact that groups can provide encouragement and support; establish identity with others and fulfill the need to feel included (see also [103]); or, simply, they might help to establish friends, relationships, and the opportunity to interact with others.

What we witnessed inside participative systems is that there are complex dynamics that could only in part be described by what we found in literature: a deeper study about these topics is needed to better understand them and we decided to take it into consideration as a possible future extension of our research. With the tools we currently have at hand, however, we were able to approach the concept of *gratification* as one of the possible user incentives and its application in participative systems.

In many successful systems we studied (just to name a few, del.icio.us, digg, and almost all peer-to-peer) we found there always was something to encourage users' participation: it is a *gratification* they get when they contribute to the systems. We were able to spot three different types of gratification:

- *instant* gratification is something that users get immediately after contributing to the system. For instance, inside a knowledge management system (whether social or not) the reward could be represented by automatically generated links between the newly entered piece of information and the related one inside the knowledge base (such as in SemperWiki [111]);
- *long term* gratification: in a healthy system the gratification due to a contribution cannot just be instant. For the system to be really useful there should be some long-term incentive: a typical one is that contributed information is stored online and accessible anytime and anywhere (i.e. del.icio.us bookmarks, Google Docs, Flickr, etc.);
- *expected* gratification is the driver for people who have not contributed yet to actually participate. The lack of such an incentive makes very hard for a system to bootstrap, so ways to provide it

should be found: for instance displaying past contributions, acknowledging those users who made them, and showing how they impacted the whole system increasing its overall value.

All these three types of gratification are important, however being able to solve the bootstrap problem (especially for a new system) is fundamental. One way to address it is to provide the initial contents for a participative system, nurturing the community by building a base of readers and then gradually turning those readers into contributors (the different levels of participation are described more in detail in the next section). As Matthew Burton³⁷ said³⁸: “People want to be a part of something that is already good; they don’t want to be part of a founding effort that MIGHT make it. You can’t rely on people to build your online presence. You have to build it yourself and make it a home for them. The content IS the marketing effort.”

In some case providing contents is made automatically, even without the users knowing it. For instance, inside a peer-to-peer file sharing network the main gratification is the ability to find and download any file in a relatively short time. To make this possible, many users have to share the same file, so that its availability is high enough to offer high transfer rates. Some p2p networks solve this problem by making contribution automatic: while someone is downloading a file, he is sharing all the pieces he has already transferred on his computer, thus helping to increase the file availability [27, 143].

Other more explicit dynamics take into account transfer ratios. Some BitTorrent³⁹ communities gather around forum-like systems, providing high quality links to online resources. To increase the quantity of information available for the community, a ratio system is applied so users are incentivated to “seed” (that is, leave available for upload) the files they downloaded for a longer time.

2.3.3 Communities of practice

Communities of practice are “groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly” [156]. According to Wenger, this concept has found a number of practical applications in business [157], government, education, and of course the Web. It has been adopted most readily by

³⁷Creator of readablelaws.org, more info on <http://matthewburton.org>.

³⁸Quoted from http://www.newassignment.net/blog/david_cohn/nov2007/12/newcorrespondent, last accessed on 14-Oct-2008.

³⁹Bittorrent is a peer-to-peer file sharing protocol.

people in business, because of the recognition that knowledge is a critical asset that needs to be managed strategically. Compared to the first *groupwares* (knowledge management information systems which mostly resulted in huge failures), systems which rely on the concept of community of practice could provide a new approach, focused on people and on the social structures that enable them to learn with and from each other.

Communities of practice can be spread over vast distances and communicate via e-mail, or they can be local and meet face-to-face regularly. They can be made up of people from the same discipline, or they can combine people with different backgrounds. Some examples⁴⁰ are “a tribe learning to survive, a band of artists seeking new forms of expression, a group of engineers working on similar problems, a clique of pupils defining their identity in the school, a network of surgeons exploring novel techniques, a gathering of first-time managers helping each other cope.”

Although the purposes of different communities of practice can vary, they all share a basic structure that combines the following three elements:

- a *domain of knowledge*: this creates common ground and defines the group’s identity. Membership implies a commitment to the domain and a shared competence that distinguishes members from other people;
- a *community*: members of a community of practice do not necessarily work together on a daily basis, however engaging in joint activities and discussions allows them to help each other and share useful information. The relationships they build enable them to interact and learn from each other;
- a *practice*, that is a set of frameworks, ideas, tools, information (i.e. ways of addressing recurring problems), and documents that community members share.

Legitimate Peripheral Participation

Legitimate peripheral participation (LPP) is a theoretical description of how newcomers become members of a community of practice. According to LPP, newcomers initially participate in peripheral yet productive tasks that contribute to the overall goal of the community (and, at the same time, whose possible failure would not impact the system as a

⁴⁰See http://ewenger.com/theory/communities_of_practice_intro.htm.

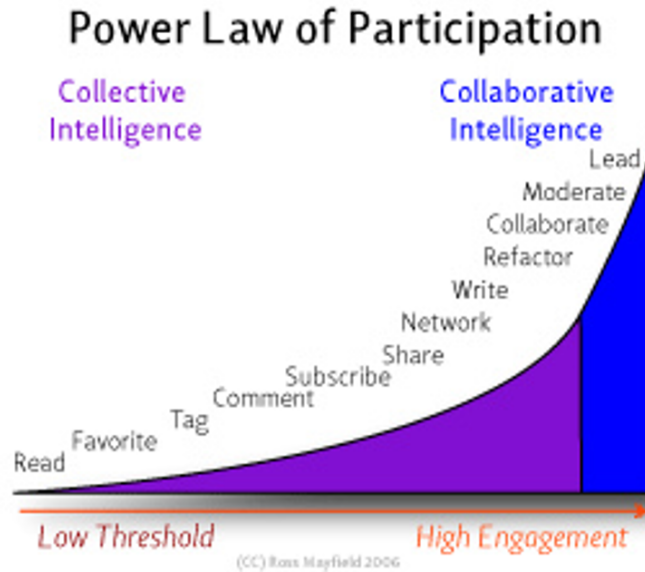


Figure 2.6: The power law of participation, courtesy of Ross Mayfield.

whole). To show the role of LPP in different communities of practice, Lave and Wenger [90] examined five apprenticeship scenarios: Yucatec midwives, Vai and Gola tailors, naval quartermasters, meat cutters, and nondrinking alcoholics involved in AA.

As an example, tailor apprentices practice sewing by working on easy tasks such as informal children's clothing, and as soon as they gain more experience they can move to more complex activities. Gradually, as apprentices become more acquainted with the practices of the community, their engagement increases and their participation becomes more and more central. This scenario has been also cited in [28] to show that the behavior of users inside a Wiki system such as Wikipedia can be very similar: users usually start contributing with small corrections to the articles and then they become more and more involved, learning how to better use the tool and at the same time collaborating with other users not just in writing articles, but also in improving the community itself. Similarly, [69] divides user population in *creators*, *synthesizers*, and *consumers* and [98] suggests its *power law of participation* (see Figure 2.6).

Some very interesting lessons—that could be useful both in evaluating an existing participative system or in designing a new one—can be learned by analyzing existing communities of practice. In the sce-

narios described by Wenger, as in a successful participative system like Wikipedia, the key to newcomers' success included, for instance:

- access to all that community membership entails;
- involvement of newcomers in productive activity;
- the possibility to “talk about and talk within a practice”, thus learning the discourses of the community;
- willingness of the community to capitalize on the inexperience of newcomers. At any level, everyone “can to some degree be considered a newcomer to the future of a changing community”.

To conclude, if we compare these considerations with the ones from Section 2.2, we can see that any participative system can be described from two different points of view: the user or the system itself. Both of them show different levels of participation, and it is not a case if they match in some way: thus, if users feels more involved in his community, they can easily move to another kind of participation whenever the system provides the chance to do it.

2.3.4 Activity theory

Activity theory originated in the former Soviet Union as part of the cultural-historical school of psychology founded by Vygotsky, Leontjev, and Lurija. For our study, we mainly focused on the framework provided by Yrjö Engeström [44] as an extension of Lev Vygotsky's work [153] on artifact-mediated and object-oriented action.

According to Vygotsky, the fundamental unit of analysis is the *human activity*, which is directed towards a material or ideal *object* satisfying a need, which constitutes the overall motive of the activity. Vygotsky describes human actions with a tripartite structure, where the relationship between human agent and objects of the environment is mediated by artifacts. This mediation has been defined [84], at the same time, as “enabling and limiting”: on one hand, artifacts are able to collect all the experience related to that particular activity and thus they (ideally) allow agents to perform it in the best known way; on the other, however, they constrain agents in performing a restricted range of actions, leaving some potential features hidden to them.

Engeström expands this tripartite structure into the model of a whole collective activity system, as depicted in Figure 2.7. The activity is a

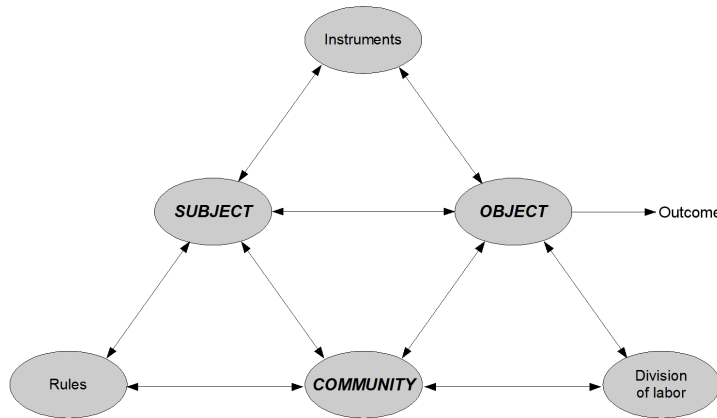


Figure 2.7: Engeström's structure of a human activity system.

form of doing which is directed towards a specific object, aiming at the transformation of the object into an outcome. Activities can be distinguished from each other according to their objects, and every individual or sub-group (subject) can and usually does participate in several activities at the same time. The *community* is a set of individuals and/or sub-groups who share the same general object. The three relationships subject-object, subject-community and community-object are mediated by three different artifacts:

- the relationship subject-object is mediated by *instruments*, that is anything which is used in the transformation process, including material tools and signs;
- the relationship subject-community is mediated by *rules*, which cover norms, conventions and social relations within the community;
- the relationship community-object is mediated by *division of labor*, which describes the implicit and explicit organization of the community (for instance the horizontal distribution of tasks or the vertical division of roles into a hierarchy).

Bryant, Forte and Bruckman [28] have already used activity theory as a theoretical framework to describe their results in the field of participative systems. An interesting conclusion is that, even if the object of the common activity always remains the same (i.e., to build and share

knowledge in an encyclopedic form), as subjects evolve the same happens for the way they use the tool, some rules are perceived differently and users start accepting new roles inside the community.

In our work activity theory plays an important role, as it allows us to give a very compact—but at the same time expressive—interpretation of some dynamics that take place inside participative systems. Especially while dealing with semantics, for instance, it is important to make the tool enable user activities (as it has to be easy to use and powerful), but at the same time limit them (i.e. constraining them to the contribution of structured content, but hiding the inner semantic details). Also, it is very interesting to note how in current participative systems the tool encodes the activities as a whole, dealing with user actions, rules (such as organization of contents) and division of labor (such as managing user access privileges).

2.4 Participation and semantics

Currently a huge amount of different participative systems exists and most of them could be made better thanks to the application of semantic technologies. Among them, we chose to focus on wiki systems and folksonomies for different reasons. First of all, they are very used and there are examples of huge communities of users for both of them (such as in Wikipedia and del.icio.us). Then, they both provide unstructured data by default: in the former case Web pages (mostly free text, with the exception of some semi-structured data coming from templates), in the latter tags in the form of text strings. They can both be made more powerful just by working on content semantics. Finally, they have some characteristics that make them very particular and different from each other: wiki systems are notable from the point of view of activity theory, as the tool (the system itself) can be easily upgraded by users themselves; folksonomies, thanks to their very simple structure, allow for the study of their community as a social network and can be easily extended with system semantics.

Literature provides many examples of interactions between semantic technologies and wiki systems or folksonomies, as we describe in the next two sections.

2.4.1 Wiki systems

The types of interactions between wiki systems and semantic technologies can roughly be divided in two classes:

1. wiki systems used to manage semantics (also defined as “W4ST”, that is “Wikis for Semantic Technologies”⁴¹). In this case, wikis are seen as tools that can be used to publish information which is already formalized and structured;
2. semantics used to manage wiki contents (also called “ST4W”, that is “Semantic Technologies for Wikis”). In this case semantics are used to make wikis better, providing new ways to query or browse information and to allow interoperability with other applications.

Wikis for semantic technologies

Examples of the W4ST approach could again be split in two separate classes: attempts to *produce* formal knowledge (i.e. using wikis as ontology engineering tools) and attempts to *extract* structured information from wikis and give it a formal interpretation.

MyOntology [138] is a wiki-based project of the first type, focused on the development and maintenance of lightweight ontologies and designed to be easy and horizontal. The main advantage of such a system is that the creation of the ontology itself is left to the community, which is then able not only to formalize knowledge, but also the very same model that is used to describe it. Moreover, an editing paradigm like this would allow both ontology engineers and knowledge workers to collaborate in the same system, addressing the well-known problem that ontology experts are usually not also expert about the domains they have to describe.

However, this field still leaves some non-trivial open challenges. For instance, it would be very useful to give different perspectives of the same content to different users: ontology-related information would thus be available to ontology engineers, while knowledge workers would be able to access the system with another, more domain-related view. Another problem which still cannot be solved easily is the strong distance between the open, horizontal, bottom-up wiki editing paradigm and hierarchical, top-down ontology creation. On one hand, making users agree on concept definitions and interpretations is not easy even for small ontologies, especially if they are planned to be used in some specific applications and not just as a plain description of the reality: being able to capture different point of views but, at the same time, to maintain a coherent and usable ontology is the first problem. On the other hand, concurrent

⁴¹See the material published for the first session of the “Semantic Wiki Mini Series” event, available at http://ontolog.cim3.net/cgi-bin/wiki.pl?ConferenceCall_2008_10_23.

edits in different parts of an ontology could result in clashes that have to be resolved: ways for visualizing (or even previewing) the consequences of any change should be implemented, and again a system to allow the ontology to remain consistent during changes should be devised.

AceWiki [83] is another wiki-based project that tries to address some of these problems by using controlled natural language and a smart user interface. Its aim is to provide an easier user interface and, at the same time, a way to support expressive ontology languages in a general way. The controlled natural language that is used for AceWiki is called ACE⁴² and allows to express even complex axioms in a natural way. ACE sentences can be beyond the expressivity of OWL, so the system automatically tags them with different colors whether they are inside or outside OWL. Similarly, in order to ensure that the ontology is always consistent, AceWiki checks every new sentence immediately after its creation: if it is not consistent, the sentence is highlighted and it is not included inside the ontology.

For what concerns the second class of examples, many applications have been built which work as collection engines, either to add new contents to wiki systems or to extract information from them. These softwares greatly vary in scope and functionalities: for this reason, rather than trying to be exhaustive, we describe some notable examples offering different kind of functionalities.

In Wikipedia bots are widely used to automatically create pages, using knowledge from external sources; for example Rambot⁴³ has been used to create most of the US city and county articles and keep them updated. On the other hand, some tools have been proposed to extract structured information from Wikipedia. YAGO [141], for instance, is a large ontology built combining facts extracted from Wikipedia (and in particular from category and redirection pages) with WordNet. DBpedia [9] is a project aimed at extracting structured information from Wikipedia (exploiting infobox templates, categorization information, images, geo-coordinates, and links to external Web pages) and represent it in RDF triples. The DBpedia dataset describes almost 2 million entities and more than 100 million RDF triples, and is interlinked with other Open Data datasets on the Web⁴⁴. [68] shows how Wikipedia entries can be used as reliable identifiers for ontology concepts and proposes the use of the Wiki paradigm to build ontologies.

⁴²Attempto Controlled English, see [55] and <http://attempto.ifi.uzh.ch>.

⁴³<http://en.wikipedia.org/wiki/User:Rambot>.

⁴⁴<http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>.

The use of data mining and natural language processing techniques to improve information representation in wikis has been largely suggested in literature: [2] uses clustering algorithms to discover missing links among Wikipedia pages; [162] proposes a tool to automatically generate and update structured information in Wikipedia infoboxes, processing natural language in the pages; [123] proposes the use of natural language processing to semi-automatically annotate wiki content with semantic relationships.

Magnus' Map Sources/GeoHack⁴⁵ is a Web application that gets geo coordinates as input and returns a collection of links to mapping and GIS-related services. The tool first converts the coordinates in many different formats (fitting the requirements of the mapping services), then it builds the output page from a template which is scraped from Wikipedia (GeoTemplate) and completes it with the coordinates.

Semantic technologies for wikis

The ST4W class includes all those systems that have been defined as *semantic wikis*. Basically, semantic wikis are a way to embed a knowledge model inside a wiki: this, of course, can be done in practice in many ways and for different purposes and applications (see, for instance, [81]).

The first semantic wiki dates back to 2004. It was called Platypus Wiki [144] and offered semantics in terms of RDF/OWL metadata that could be entered into a separate edit box. It was followed shortly by another wiki called Wiksar [11], which provided the first example of inline metadata (structured information directly embedded into the wikitext) and queries.

SweetWiki is a semantic wiki which focuses on hiding semantics to users, providing them instead with what they could perceive as original features. It is, however, semantic from the ground up: in fact, it is based on an OWL schema which describes concepts such as wiki words, wiki page, forward and backward link, and so on. To keep the interface friendly, the system implements a WYSIWYG editor and the concept of social tagging, allowing users to tag pages and pictures and manage the system's folksonomy.

Semantic MediaWiki [151, 82] is an extension of the MediaWiki software (the same used by Wikipedia) which allows users to add semantic annotations based on a OWL-DL model. Annotations can be inserted with special tags inside the wiki text; each wiki page matches one element of the substanding ontology and every annotation in the article

⁴⁵<http://tools.wikimedia.de/~magnus/geo>.

makes statements about this single element. The system provides a semantic browsing interface and a language for semantic queries that can be inserted inline in the wiki.

IkeWiki [127] is a standalone wiki implementation that supports reasoning and different levels of formal expressiveness, from RDF to OWL. It has originally been developed as a prototype tool for the collaborative formalization of knowledge, however the project has evolved and the tool has become useful for different classes of users: in fact, its interface has been studied for various level of expertise. IkeWiki allows users to annotate pages and links between them with semantic annotations. This information can then be used for different purposes: advanced queries, context-specific presentation of pages, and reasoning (either to verify the consistency of the KB or to infer new knowledge). Compared to SMW, IkeWiki appears as more restrictive as there is no possibility for simple users to annotate a link with a predicate that is not defined in a pre-loaded background ontology. Another difference with Semantic MediaWiki is that IkeWiki stores its semantic metadata separately from the page content. The main advantage of this approach is that the maintenance of the KB is easier, however its main drawback is that it does not allow versioning of the metadata.

At least a couple of significative projects have spawned from Ikewiki. The first one is SWiM [85], a wiki for collaboratively building, editing and browsing a mathematical knowledge base which relies on the OMDoc ontology⁴⁶ for its page format. The second is KiWi⁴⁷, a project funded by the European Commission which extends IkeWiki with an improved rule-based reasoning support, information extraction, personalization, and advanced visualization and editors.

BOWiki [14] is a MediaWiki extension built to allow biologists to develop a collaboratively maintained knowledge base that automatically verifies its ontological adequacy. To accomplish this task, BOWiki uses the Pellet OWL Reasoner, the top-level ontology GFO and the biological core ontology GFO-Bio⁴⁸.

OntoWiki [10] is a wiki-like tool for knowledge engineering whose goal is to decrease the entrance barrier for projects and domain experts to collaborate using semantic technologies. To accomplish this, it provides users with a very easy and intuitive user interface, presenting the knowledge base as an information map and providing inline editing for RDF

⁴⁶<http://www.omdoc.org>.

⁴⁷Knowledge in a Wiki, see <http://wiki.kiwi-project.eu>.

⁴⁸See <http://www.onto-med.de/ontologies/gfo> and <http://onto.eva.mpg.de/gfo-bio.html>.

content. Collaboration is promoted by allowing users to keep track of changes, write comments about any part of the knowledge base and rate the popularity of content.

SWOOKI [120] is a peer-to-peer semantic wiki, built on top of the Wooki software [155]. Its aim is to address the problems of scalability, performance, fault-tolerance and load balancing by replicating wiki contents over a p2p network and allowing for distributed semantic queries. Moreover, relying on a distributed and redundant architecture it does not suffer of the censorship problem.

2.4.2 Folksonomies

Folksonomies are a technology that is used inside many online or offline⁴⁹ systems, rather than a participative system on its own. The term “folksonomy” is generally attributed to Thomas Vander Wal, and refers to a (usually internet based) information retrieval methodology consisting of collaboratively generated labels (also called *tags*) that can be used to categorize different kind of resources. According to Vander Wal [149], “folksonomy is the result of personal free tagging of information and objects (anything with a URL) for one’s own retrieval. The tagging is done in a social environment (shared and open to others). The act of tagging is done by the person consuming the information.” Even if folksonomies have been sometimes described as derived from “folk taxonomies”, they are actually like the antithesis of taxonomies: categorization is not pre-defined and exclusive, but is rather inclusive and redefined each time a user tags a resource with a new label.

Long time before folksonomies caught the attention of the scientific community, different individuals started to write about them in blogs, describing their characteristics and the dynamics of their usage [102, 101, 80]. At the same time, the success of the first tag based collaborative systems encouraged the creation of many other ones [65, 94]. From these first publications we learn basic concepts about folksonomy structure, such as the distinction between *broad* and *narrow* folksonomies [148] (eventually shrinking to *personomies*), their advantages and drawbacks [97, 118].

Due to their wide acceptance, collaborative tagging systems have caught the attention of many different research groups. Social sciences study the “folks” part of folksonomies and try to explain how unconstrained systems like them can work so well. Cognitive sciences are interested in

⁴⁹Like in the case of tag-based filesystems [22].

another explanation of tagging popularity, either in terms of cognitive processes [136] or behavior [137]. In the latter case, the concept of *reinforcement* is quite important: according to it, users continue to tag because they get an instant gratification when they do it, having access to more information than the one they have tagged and seeing themselves interlinked within a community. Finally, information architects such as Peter Merholz [104] make an analogy between the usage of tags inside a folksonomy and the creation of “desire lines” inside a landscape: both of them describe how users choose to move *before* a paved path has been created.

Working with folksonomies and semantics, many different research approaches have one assumption in common: as one of the main characteristics of tags is their immediateness (users just write a short string and that’s all), the interaction model has to be kept the same even when trying to add or extract semantics from them. This is particularly interesting for two main reasons: first of all, adding new features without asking users to modify their interactions is not a trivial task (for instance, this could not be done with semantic wikis); then, this technique has the intrinsic advantage of working as a “plug in” with different kind of tag-based systems. The main approaches currently present in literature are:

- extracting structure or semantics from tag-based systems: the goal in this case is learning more about folksonomies to understand if it is possible to extract semantics from them. A lot of work has been done in this field: [58] analyzes the structure of collaborative tagging systems, discovering regularities in the tagging activity and suggesting a partition of tags in different families, according to their purpose; [95] suggests a taxonomy of tagging systems, showing how their behavior might change depending on their main characteristics; a *tripartite graph model* for folksonomies has been proposed by [105], which allows to better understand and study the relationships between users, tags and resources; [73] uses measurements and statistics to better describe the structure of folksonomies (and the frequent occurrence of power law distributions), detect trends and propose a ranking for their elements; [132] describes the emergence of power laws from tag distributions and presents an approach for ontology extraction from tagging.
- mapping folksonomies with ontologies: this approach tries to find matches between tags and concept names inside an ontology. The

main advantage is that easy to use free text strings, provided by the community with a bottom-up approach but lacking of structure, are matched with formal concepts, built in a top-down way by few experts, connected to each other through relations, and organized inside a hierarchical structure. This technique can be used with static, already existing ontologies, or as a way to create or update an ontology by promoting tags to concepts. [86] is an example of the first approach: tags from del.icio.us are mapped with concepts inside WordNet to solve ambiguities and provide users with a hierarchical structure to navigate related tags. [25], instead, propose an ontology maturing approach which relies, in its most basic steps, on the emergence of ideas through folksonomies and on the creation of concepts from tags.

- describing folksonomies with ontologies: in this case, ontologies are used to formally describe the folksonomy system as a whole. [60, 59] and [99] introduce the concept of folksology, showing its main characteristics and advantages (ie. sharing, interoperability, and disambiguation).

2.5 Technologies

2.5.1 The Semantic Web

The Semantic Web [17] is “an extension of the current Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation”. According to Jim Hendler in a recent position paper [67], Semantic Web activity is going in two main directions. One follows the “Web” and focuses on the development of mostly Web-based applications that use very little semantics but provide a powerful mechanism for linking data entities together using the URIs that are the basis of the Web; the standards it relies on are the Resource Description Framework (RDF) to describe knowledge and the SPARQL language to query it [117]. The other direction follows the “Semantics” and looks at providing models that can be used to represent knowledge in an expressive way; this approach is mostly based around the Web Ontology Language OWL and, thanks to the use of reasoners, it can also offer the chance to infer new knowledge from the one that has already been explicitly asserted.

The Semantic Web Layer Cake

The technologies at the basis of the Semantic Web are often presented as a set of layers, stacked one above each other as in a “layer cake” (see Figure 2.8). The cake has evolved in time and will probably continue to do that in the future, however—even if the names of some technologies changed or were introduced from scratch—the general meaning has remained the same. The layers above ontologies and rules still contain technologies that are not yet standardized or suggestions about what should be implemented to realize the Semantic Web, so they will not be described in detail here.

At the bottom of the cake, URIs and Unicode are the two main standards every other layer is built on: the first one provides a way to build identifiers for any kind of resource, while the second is used as the standard for character sets. XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents. RDF is a data model for objects (called *resources*) and relations between them; it provides a very simple semantics based on the concept of *triples* (see below) and can be represented using the XML syntax even if other, more readable notations exist, such as N3. RDF Schema is a vocabulary for describing properties and classes of RDF-based resources: for example, using RDFS it is possible to create hierarchies of classes and properties. OWL extends RDFS by adding more advanced constructs to describe the semantics of RDF statements: for instance, it can describe cardinality (i.e. “everyone has exactly one mother”), relations between classes such as disjointness (i.e. between male and female gender), or characteristics of properties such as transitivity (i.e. “the ancestors of my ancestors are my ancestors too”). SPARQL is a query language that can be used to query any RDF-based data, including of course both RDFS and OWL, as it can be represented using the RDFS syntax. Finally, RIF and SWRL are languages that extend the capabilities of OWL, adding the support of rules into reasoning.

Ontologies

The concept of *ontology* occurs very often when speaking about Semantic Web: RDF(S) and OWL are defined as *ontology languages*, knowledge is often (even if not always) formalized using ontologies, and many of the prototypes we developed use them. An ontology, according to T. Gruber [93], defines a set of representational primitives which can model a domain of knowledge or discourse. We can formally define an ontology

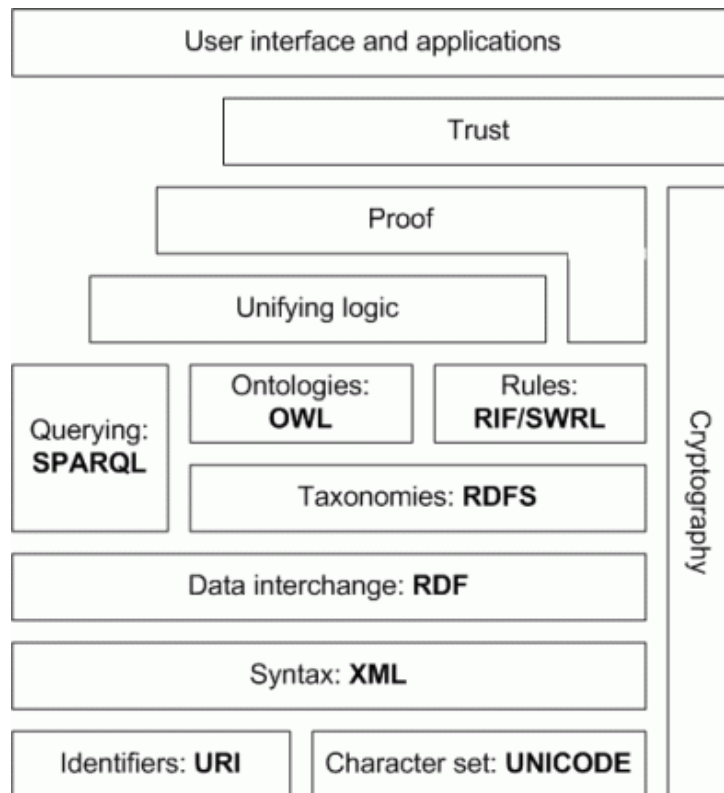


Figure 2.8: The Semantic Web “Layer Cake”.

as a 4-tuple $O = \langle \mathcal{C}, \mathcal{R}, \mathcal{I}, \mathcal{A} \rangle$ where \mathcal{C} is a set of *concepts* (or classes) which are subsets of a common domain Δ , \mathcal{R} is a set of relations including both binary relations between classes, called *roles*, and binary relations between concepts and datatypes, called *attributes*, \mathcal{I} is a set of individuals (or ground symbols) which belong to Δ , and \mathcal{A} is a set of axioms (or assertions) in a logical form which are valid in the domain and restrict the number of possible interpretations of the ontological model.

As an example, if we look at Tim Berners-Lee's FOAF⁵⁰ file we can see that he is an individual of the class **foaf:Person** (that is, of the class "Person" as defined by the FOAF ontology), that his given name is an attribute whose value is Timothy, and that he knows `<http://danbri.org/foaf#danbri>` (that is, the URI representing Dan Brickley).

There are different reasons for using ontologies to describe knowledge. First of all, this description is done formally and can be interpreted by a machine: this allows information to be used and shared more easily by many applications. Another reason is the possibility of enabling computers to provide different kinds of reasoning services about the described knowledge. For this reason, ontology languages such as OWL are based on well known logics, which trade some of their expressivity with the possibility of running decidable inferencing algorithms in acceptable times.

RDF

(Resource Description Framework⁵¹) is a standard model for data interchange on the Web. RDF is built on the following definitions:

- a *Resource* is anything that can have a URI⁵². This includes, for instance, all the world's Web pages which are identified by a URL;
- a *PropertyType* is a resource that has a name and can be used as a property. For instance, speaking about a bookmark, Name and Date could be a couple of related properties. The fact that a property type is a resource too allows it to have its own properties: for instance, we can say that the range for the Date property we

⁵⁰Friend Of A Friend (FOAF) is a very common ontology that is mainly used to describe people and the relations between them (i.e. "knows"). Tim Berners-Lee's FOAF file is available at <http://www.w3.org/People/Berners-Lee/card>.

⁵¹See <http://www.w3.org/RDF>.

⁵²Uniform Resource Identifier. URIs, which are a subset of URIs, are probably the most common URIs used to refer to a resource. For more information, see http://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

defined for bookmarks is a string that conforms to the XML Date-Time datatype (so we know how to parse it), has a human-readable description that says it is the date the bookmark was saved, and has a label that reads like “Date” in English and “Data” in Italian.

Using these building blocks, RDF allows to express assertions in the form of *triples* (subject, predicate, object), where the both the subject and the predicate are resources while the object can be either a resource or a constant string, called *literal*. As an example, if we wanted to say that the page at URL `http://www.example.org/index.html` has the title “My Page” and its author is John Smith, identified by the URL `http://www.example.org/people/123456`, we could write:

```
<http://www.example.org/index.html>          (Subject)
<http://purl.org/dc/elements/1.1/creator>    (Predicate)
<http://www.example.org/people/123456> .     (Object)

<http://www.example.org/index.html>          (Subject)
<http://www.example.org/terms/page-title>    (Predicate)
"My Page" .                                  (Object)
```

Typed properties and the triple structure facilitate data merging even if the underlying schemas differ and support the evolution of schemas over time without requiring all data consumers to be changed. This is one of the main reasons why we chose to use RDF for many of our projects: in fact, this format allows us to easily use the same information across different applications, being able at the same time to extend it whenever we need it.

OWL

OWL (Web Ontology Language) [70, 72, 71] is not just a language, but a whole family of more or less expressive knowledge representation languages with well defined semantics. [100] describes in detail the features of OWL, such as relations between classes, cardinality, equality, richer typing of properties, characteristics of properties (i.e. symmetry and transitivity), and enumerated classes.

Different versions of OWL are currently available. Version 1 provides three increasingly expressive sublanguages called OWL Full, OWL DL, and OWL Lite. OWL Full is not actually a sublanguage, as it contains all the OWL language constructs and provides free, unconstrained use of RDF constructs. The “DL” in OWL DL means that this language

is based on a particular family of *Description Logics*⁵³, which is called $\mathcal{SHOIN}(\mathcal{D})$ (the meaning of each letter in the acronym is explained in table 2.2); its restrictions allow the maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure can exist for an OWL reasoner. OWL Lite uses all the constraints set by OWL DL plus some more, with the aim to provide the minimal useful subset of language features that are relatively straightforward for tool developers to support. It matches the DL \mathcal{SHIF} and allows for reasoning with a lower complexity.

Letter	Capabilities
\mathcal{S}	The following terms can be used: $\top, \perp, C \cap D, C \cup D, \neg C, \forall R.C, \exists R.C$. Plus, there is support for <i>transitive roles</i>
\mathcal{H}	<i>Role Hierarchy</i> , that is the possibility of creating relations of inclusion between roles: $R \subseteq S$
\mathcal{O}	<i>Enumerations</i> : $\{a_1, \dots, a_n\}$
\mathcal{I}	<i>Inverse roles</i> : R^-
\mathcal{F}	<i>Functional roles</i> : $\geq 1RC, \leq 1RC, = 1RC$
\mathcal{N}	<i>Number restrictions</i> : $\geq nR, \leq nR, = nR$
\mathcal{Q}	<i>Qualified number restrictions</i> : $\geq nRC, \leq nRC, = nRC$
\mathcal{D}_n	<i>Datatype properties</i> (as distinct from roles)

Table 2.2: Description Logics families.

OWL 1.1 is an evolution of OWL which matches the DL $\mathcal{SROIQ}(\mathcal{D})$. It provides many extensions which are almost all related to roles: to role hierarchy—usually expressed by the letter \mathcal{H} —they add *disjoint roles*, *reflexive and irreflexive roles*, *negated role assertions*, *complex role inclusion axioms*, the *universal role*, and finally the construct $\exists R.Self$, which is used to express “local reflexivity” of a role.

OWL 2 introduces the concept of *profiles*, which are defined by placing restrictions on the OWL 2 syntax. All OWL Lite ontologies are OWL 2 ontologies and so OWL Lite can be viewed as a profile of OWL 2. OWL 1 DL can also be viewed as a profile of OWL 2. The currently available profiles for OWL 2 are called *OWL 2 EL*, *OWL 2 QL*, and *OWL 2 RL* and they are described in detail in [106].

⁵³Description logics [12, 13, 30, 63] are a family of logic-based formalisms used for knowledge representation which are particularly used for automatic reasoning.

2.5.2 Tools

For our tests and developments we mostly programmed using the Perl or Java programming languages, trying both to use and to develop mostly free (as in speech) software. Thus, as most of our work is built, not just in research but also during development, “over the shoulders of giants”, we decided to briefly describe the main tools we used and at the same time acknowledge their creators.

Reasoner

The main reasoner we used is Pellet⁵⁴. Pellet [139] does not only provide standard reasoning services, but it also incorporates various optimization techniques described in the DL literature and contains several novel optimizations for nominals, conjunctive query answering, and incremental reasoning. Based on the tableaux algorithms for expressive Description Logics, Pellet supports the full expressivity of OWL DL, including reasoning about nominals (enumerated classes). As of version 1.4, it supports all the features proposed in OWL 1.1, with the exception of n-ary datatypes.

Programming Frameworks/Libraries

Jena⁵⁵ is a Java framework for building Semantic Web applications. Of the different components provided by Jena we mainly used:

- **SDB**⁵⁶, which provides scalable storage and query of RDF datasets using conventional SQL database as a backend. It can be used in standalone applications and is designed specifically to support the SPARQL query language;
- **Joseki**⁵⁷ is an HTTP engine that supports the SPARQL Protocol and the SPARQL query language. It can be configured to work with SDB as a backend and provides a SPARQL endpoint to the RDF store. We decided to use Joseki as it provides a layer of abstraction on the ontologies which allows programmers to access it just by using SPARQL; moreover, there are already many other

⁵⁴<http://pellet.owldl.com>.

⁵⁵<http://jena.sourceforge.net>.

⁵⁶<http://jena.sourceforge.net/SDB>.

⁵⁷<http://www.joseki.org>.

applications that work with SPARQL endpoints, so this choice allows us to automatically make our data useful for other projects too;

- **SquirrelRDF**⁵⁸ is a tool which allows non-RDF data stores to be queried using SPARQL. It currently includes support for relational databases (via JDBC) and LDAP servers (via JNDI). It provides an API for Java access, a command line tool, and a servlet for SPARQL http access.

Ontology editors

The two main tools that we used to develop our ontologies were Protégé⁵⁹ and Morla⁶⁰. Protégé is a very powerful application, implementing a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. We mainly used it for the development of OWL ontologies and took advantage of its inline reasoning features to test our work. Morla is much simpler but also lightweight and easy to use, and we mainly used it for the development of small RDF ontologies (when they were not so small that they could be managed with just a text editor).

Databases

MySQL⁶¹ is a very well-known relational database management system, written in C and C++. Our biggest dataset (the one gathered from del.icio.us) is stored in a 2GB MySQL database, and the 3564020356.org website (see Preface) uses the same DB technology.

HSQldb⁶² is a relational database engine completely written in Java. It is multi-platform, small, and fast and supports both in memory and disk based tables. We chose this engine for some of our projects as it can easily be run in a standalone mode as part of an application program, in its same Java Virtual Machine.

⁵⁸<http://jena.sourceforge.net/SquirrelRDF>.

⁵⁹<http://protege.stanford.edu>.

⁶⁰<http://www.morlardf.net>.

⁶¹<http://mysql.com>.

⁶²<http://hsqldb.org>.

Wiki systems

JSPWiki⁶³ is a Wiki system built around the standard J2EE components (Java, servlets, JSP). We used this software as the basis for our semantic wiki prototypes for the following reasons:

- it has already been used by other semantic wiki projects, such as Makna ([39]);
- it is developed in Java, which allowed us to build a semantic extension using the Jena framework;
- it provided many features such as attachments, ACLs, and plugins;
- it had a live community of developers.

Of course, we also experimented with MediaWiki⁶⁴ and its semantic extension⁶⁵: our research group's wiki⁶⁶ is built over these technologies.

Browser

Firefox⁶⁷ is a well-known, opensource browser, which with about 650 million downloads⁶⁸ is getting more and more used not only by computer-savvy people, but also by many common internet users. Being open-source, it has also become a basis for other projects such as Flock⁶⁹ (a “social Web browser”, specialized in interfacing with many social applications) and Google Chrome⁷⁰. Naming a browser as one of the main tools for the development of a PhD thesis might sound weird, but Firefox has actually been precious for us as we built many of our prototypes as its extensions.

2.5.3 Datasets

del.icio.us

Del.icio.us represents for us both an object of study and a useful dataset to better understand the dynamics of tags and tag-based systems. To

⁶³<http://www.jspwiki.org>.

⁶⁴<http://www.mediawiki.org>.

⁶⁵Semantic MediaWiki, see <http://semantic-mediawiki.org>.

⁶⁶<http://airwiki.elet.polimi.it>.

⁶⁷<http://www.mozilla.com/firefox>.

⁶⁸Data gathered from <http://www.spreadfirefox.com> on September, 8th 2008.

⁶⁹<http://www.flock.com>.

⁷⁰<http://www.google.com/chrome>.

gather information from the website we built a scraper⁷¹ that first harvested user names from the system and then downloaded their full collection of tagged links. The scraper has been built with adequate sleep times, not to hammer del.icio.us website with too frequent requests, and error recognition to allow graceful stops whenever the service was not available. As it could be paused and resumed in any moment, it has been run not continuously for few weeks. The final result is a dataset containing information related to about 31000 users, 482000 tags, and 3.7 million resources. The total number of *taggings*, that is $\langle user, resource, tag \rangle$ triples, is more than 21 millions.

Albeit tiny if compared to the actual size of del.icio.us (which is about two order of magnitude bigger), the dataset has been particularly useful for us to understand how folksonomies work and to experiment with some real-world user data. Some examples of our experimental developments that make use of this information are available in Section 5.

WordNet

WordNet⁷² is a large lexical database of the English language. It groups English words into sets of synonyms called *synsets*, each expressing a different concept. Every synset has a short definition and relations (which can be semantic or lexical) with other synsets. As an example, the word “Web” belongs to seven different synsets as a noun, where one is “entanglement (an intricate trap that entangles or ensnares its victim)” and, of course, there also is “World Wide Web (computer network consisting of a collection of internet sites that offer text and graphics and sound and animation resources through the hypertext transfer protocol)”.

WordNet can be accessed online or downloaded as a desktop application. It has libraries that allow programmers to use it from many different programming languages and many user-built extensions. We used WordNet for three main reasons: first of all, it served as a reference to verify how many tags in del.icio.us were common English words (see Section 5.1.1); then, it provided a hierarchical structure to map tags on and provided a non-flat tag space for users to browse; finally, it has been the backbone for the algorithms we used to disambiguate tags (see Section 5.1.2).

⁷¹Which is available at <http://davide.eynard.it/?p=28>.

⁷²<http://wordnet.princeton.edu>.

```

[
  {
    "directed_by" : "George Lucas",
    "name" : null,
    "starring" : [
      {
        "actor" : "Harrison Ford"
      }
    ],
    "type" : "/film/film"
  }
]

```

```

| [
|   {
|     "directed_by" : "George Lucas",
|     "name" : "Star Wars Episode IV: A New Hope",
|     "starring" : [
|       {
|         "actor" : "Harrison Ford"
|       }
|     ],
|     "type" : "/film/film"
|   },
|   ...

```

Figure 2.9: An MQL query asking for all the films by George Lucas starring Harrison Ford (left) and the first of its results (right).

Freebase

Freebase⁷³ has been defined as “an open database of the world’s information”. It has been built drawing information from large open data sources, such as Wikipedia and MusicBrainz⁷⁴, and reconciling it so that a particular topic appears only once in Freebase and contains all the information related to it. The result is a collection of structured data about many popular topics such as movies, music, people and locations, which can be easily edited online within a wiki-like interface or queried by anyone with the provided API. The underlying structure of Freebase allows users to run complex queries using a query language called MQL (Metaweb Query Language), which uses JSON (JavaScript Object Notation) syntax, making it ideal for JavaScript and Python based clients. The typical query is made *by example*, that is providing a template of the structure of the desired data, where all the information we do not know are simply left blank; the query response will simply fill the template with the missing information. An example, asking for George Lucas films starring Harrison Ford, is shown in Figure 2.9.

We decided to use Freebase as a backend for some of our prototypes for different reasons: first of all, being based on Wikipedia knowledge, it contains data which is not limited to a specific domain; then, it provides some structure and semantics that we decided to exploit for our needs; finally, it is a free source of information which will continuously grow thanks to the direct contributions of its own users or the indirect ones coming from the users of other open data communities.

⁷³<http://www.freebase.com>.

⁷⁴<http://musicbrainz.org>.

3 Our approach

Our research approach takes into account two different aspects of participative systems. The first one is related to the activity system and includes users—considered both as individuals and as members of a community—, their objectives, and their tools. The second one, instead, relates to the possibility of enhancing a system with semantics. In the current chapter we describe these two aspects, showing our methodology and providing some hints for designing a new participative system (or evaluating an existing one) and adding semantics to it. Then we show why evaluation is not trivial in our case, as it has to deal both with technologies and with people using them, and we explain how we approached this problem. We continue describing the main architectures that we relied on and conclude introducing our prototypes and explaining the organization of the following chapters.

3.1 Designing a participative system

Designing a participative system is not a trivial task: from a technologist's point of view the most important thing might seem to design a good tool, flexible and scalable enough to enable the community's activity, and at the same time stable and secure. However, CSCW¹ has a long history of apparently perfect systems that did not make it, apparently without any specific reason (see “I love the system—I just don't use it!” [15]). Actually, one of the main problems of unsuccessful participative systems lies in focusing on the tool only, ignoring all the other elements inside an activity system. The result is that the tool could be technologically very good, but wrong for that particular activity. Borrowing some terms from the activity theory diagram (see Section 2.3.4), a tool could be wrong with respect to

- the subject: as the tool both empowers and limits the subject, a wrong tool is one which either cannot enable users' activities or limits them too much. Some typical errors in tool engineering are

¹Computer Supported Collaborative Work.

3 Our approach

bad interface, missing incentives, or the requirement for a knowledge which is too specific. A possible solution to this problem could be a design which is more centered on users;

- the object: the tool should be built to fit the activity, not chosen with the hope that activities might change to fit it. And if it is true that many participative systems had success using one particular tool, this does not mean that the same tool will be enabling for other kind of activities. As an example, a wiki might be a good tool to allow the Internet community to build an encyclopedia with no time constraints, but it is not as good to allow a group of ten coworkers prepare one single-document relation in one day. To address this problem, we tried to spot the main characteristics of participative systems (that we called *dimensions*, as they could be used to split them in classes) and derive some design patterns from them;
- the community: even if in the diagram of activity theory the community appears as not directly related to the tool, in practice it is strongly bound to it as far as its rules and division of labor are coded inside the tool itself. However, different communities might be more or less suited to use a specific tool: the reasons for this might include the context, privacy issues, or license issues. Moreover, every community has to be nurtured into participation: for instance, given two similar activity systems, one might work while the other might not just because the first started with more contents available to users. Finally, while incentives intuitively represent a good key to participation (at least with the wide, difficult to classify user base of the Internet), they are just one of the many dynamics that might push a particular community into participation: a more accurate approach would take into consideration also social dynamics, cognitive factors, information architecture, and user interaction. In our work, we decided to look at participative systems from Wenger's perspective, assuming they are used by communities of practice and trying to exploit their legitimate peripheral participation.

3.1.1 User-centered design

The reason why we chose to define the design of the system as “user-centered” is, of course, related to the literature about human-computer interaction, but at the same time it aims at recalling some of the “Web2.0

design patterns” described by Tim O’Reilly in [110]: users have become a precious resource as they provide information which is unique and hard to replicate, so systems have to be designed around them and user participation has to be encouraged. This means, at the same time, that tools have to be easy and intuitive to use and that they have to provide the right incentives to make users contribute again.

For what concerns user interaction, we used the “seven principles of user-centered design” by Donald A. Norman [109] as a source for inspiration. We then came up with our own principles, which take into account interfaces, activities, and semantic technologies:

- keep semantics hidden, as far as it is possible: of course, a collaborative ontology authoring tool would need its users to know what they are doing, but an image sharing system does not. In this case we could assume that most of its users neither know about semantics nor they are interested in learning it. So, one of the main tasks would be to make the system work without requiring its users to know its inner details;
- reuse and standardize: when possible, it is better to use already well known interfaces, metaphors and interaction paradigms. For instance, we chose to build many of our prototypes as browser extensions to allow users access them within a familiar environment;
- exploit the power of constraints: sometimes limiting users in what they can do with the tool prevents them from committing many errors; moreover, in a collective environment it might suggest a direction where contributions could converge. Of course, the choice of the constraints is a trade-off with the flexibility of the tool: in this case *suggestions* (such as giving a template for a particular wiki category or suggesting the most used tags in a social bookmarking system) might help;
- do not alter the main activity of the group: if a participative system has to be used by a group which already has some common practices, these practices should be studied so that they can be still enabled by the tool. New features (especially when semantics is involved) are sometimes not only considered unuseful, but often not even welcomed if they require users to change the way they were accustomed to use the system;
- in every case, keep it simple: the system does not need to become harder than it is. Designers could help users, for instance, by

3 Our approach

always making the possible actions clearly visible and by always showing them at which step of a process they are.

While working on incentives we decided to mostly focus on two aspects: instant gratification and system bootstrap. In the first case, we decided to devote our attention to all those systems that allow collective intelligence to emerge from user participation, by aggregating their contributions, providing links to related information, suggesting resources that might be interesting, and so on². Starting from our findings we developed the model for collective intelligence that is described below.

Trying to deal with the problem of system bootstrap, we found that currently one of the most implemented solutions (apart from the theoretically trivial one, that is writing actual contents) is starting the system as an invite only “beta version”: exploiting a community of more motivated users, filtered thanks to the barrier at the entrance, the system is tested and at the same time filled with contents, making it ready to be launched for the wide public³. A possible solution we tested and we can suggest is the reuse of already existing knowledge to provide, at the same time, working incentives and the feeling of an already live system. Freebase, for instance, is using a very similar approach, importing and converting information from many other data sources; with all these contents already available, it is nurturing a community of “power users” that are also developers and are helping in showing how useful (and already used) the system is.

3.1.2 Dimensions of participative systems

To help us in finding many different dimensions that characterize participative systems we used the method of the *Six Ws*⁴. This list does not want to be exhaustive and an extension of it is surely going to be part of our future work, yet we think it provided some good insights in understanding and designing a participative system.

²[131] is a manual that provides a very practical, however interesting, description of this field.

³A very recent example is Twine, that opened on Oct, 22nd 2008 after one year of beta, with about 20000 *twines* created by its users and about one million items added to the system.

⁴Also called “Five Ws and one H”, see http://en.wikipedia.org/wiki/5_Ws.

Who

These are the dimensions that are mostly related with users and communities.

What kind of user? Knowing what type of users are going to use the system is fundamental to design it correctly. For instance, we might be interested in gathering a community of generic internet users with only a very basic knowledge about computers, a group of professionals with very specific interests, or some power users that can use their programming skills to immediately contribute to the system software. Moreover, we might want to decide if we want to allow anonymous users or not, as this can change many different dynamics: for instance, being anonymous could be perceived as an enhancement in privacy, but could make the public recognition of someone's contribution totally meaningless. Cultivating *identity* in a community could be a way to address this problem, without necessarily asking users to provide their personal details (see [28]).

What kind of community? Knowing the type of community that has to deal with the system is also very useful. For instance, designing a system for a generic Internet community or for a corporate intranet is very different: inside a corporate environment anonymous access could be disabled, as every user already has to authenticate whenever he logs into his system; designing for the Internet, instead, the problem of privacy might be more important, but at the same time the existence of a category of “bad users” (such as spammers, hackers, or generally malicious users) should always be taken into account. Another dimension that characterizes participative systems with respect to the community is whether they are open or closed: they could be accessible by anyone, only to registered users, or only to few people whose registration request has been approved by an administrator. This is similar to, but more general than the previous case. Then, any kind of barrier at the entrance could be provided before registration approval: this of course changes the dynamics of the group, as it is going to be built on more or less motivated users⁵.

Who is the contents producer? One user or the whole community? Systems like blogs work thanks to their whole community, but often articles are produced by only one person: this is called a “one-to-many” architecture. In a “many to many”, instead, different users provide con-

⁵A barrier at the entrance—no matter how simple it was—and a mechanism of incentives have provided a good filter inside the `3564020356.org` community, which is basically self-moderated.

3 Our approach

tents: some examples are Wikipedia, file sharing systems like YouTube, and peer-to-peer systems. In both cases the presence of a community of readers and commenters can act as an incentive, but in the first one the system is not scalable: there is an intrinsic limit on the quantity of information that can be produced by one single person, and it does not matter how popular the system is, in some cases the producer becomes the bottleneck for the whole system. In this case alternative interactions for readers should be found, such as allowing them to discuss in forums or giving them a space to publish their own contents.

Who is the addressee? Reading or, more generally, accessing collaboratively published information is considered as the first level of participation. This can be given two meanings: from the users' point of view it is the first activity they can perform, and interest in the system's contents is the first incentive they might have; from the system's point of view, instead, this means that just the presence of readers might be a strong incentive for publishers. Thus, knowing and letting active users know about the group of readers is very important and that could be done in different ways: the most simple operation is to add a counter on every accessed resource; then adding comments or a mechanism for "favorites" gives a more detailed feedback; finally, developing users' identities through user accounts and profiles might provide even better incentives.

What

These are the dimensions that are mostly related with the object of the activity (that can be very concrete, such as the final product the community is building, or very abstract, as the common interest it is gathered around) and some characteristics of the tool.

What kind of tool is used? If the work has to be done on an already existing tool, this is the first question that should be asked. As previously described, every tool has been designed with a main purpose in mind (see Section 2.1): checking if this purpose fits the group activity provides the very first evaluation about the system.

Is there a central object? Some communities gather around a common type of shared object, such as documents or videos. Others have a central object in the form of a "final product", such as the collaborative encyclopedia built with Wikipedia. Finally, others have a common activity, such as listening to music or communicating with friends. The presence of a common object such as in Wikipedia definitely has to be taken into account as it definitely impacts group dynamics: in this case,

in fact, users have to coordinate their work, to communicate, and to find agreement over different matters. Wikipedia is a wonderful example of coordination that is spawned out apparently from nowhere, as the tool itself does not automatically provide all those constraints that have been developed by its community only in a second time. Also, it has a very particular object which is, at the same time, common but not atomic: even if the final product (the encyclopedia) can be considered as a whole, many users can contribute at the same time on different articles without even interacting with each other. The scalability of the project is surely one of the reason of its success. Conversely, experiments with wikis used as publishing tools for atomic products such as fiction books went really bad [96]: the main problem was that it was impossible for the multitude of users to agree and coordinate even on small parts of the work, and the final result was incoherent and not homogeneous.

Do users leave persistent traces of their contributions? This is a characteristic of the tool that might become part of the final object. In fact, while for some systems (such as wikis) it could be very important to have a “history” for contents as it provides versioning and rollback, for many others the interaction itself becomes part of the final product. For instance, comments to a post in a blog or video replies on YouTube are persistent traces of user contributions that become an important part of the system’s contents.

How, When, Where

These are the dimensions that are mostly related with the activity.

How do users participate? The type of participation, according to the taxonomy we defined in Section 2.2, also defines the types of interactions with the system: for instance, if users contribute with files or comments they should be given the possibility to publish them; if they have to collaborate on the creation of some contents they should be also given a way to coordinate and communicate.

Is participation explicit or not? Contributions from users might be explicit (i.e. a comment on a blog, a video shared on YouTube) or implicit (i.e. the traces of a visit left in a website, the preference for one resource between many, etc.). Defining whether the participation has to be explicit or not also helps into defining what kind of incentives have to be found for users. Moreover, allowing collective intelligence to emerge from implicit participation is a problem *per se*: even if many aggregation algorithms are already known and widely used, every system is different and might require an in-depth study to find specific solutions.

3 Our approach

Does the activity need a high coupling or not? Coupling is considered as the degree to which individuals rely on each other to accomplish a task. For instance, a live concert requires a high coupling between members of the band to play their songs; conversely, the phenomenon of “virtual bands” has shown that now it is possible to have a group of people playing together the same song without even knowing each other⁶. While there are participative systems which work well even if loosely coupled (such as most of the sharing systems, where everyone can act independently from each other), there are many—in particular collaborative ones—where coupling is essential. In this case it is necessary to provide a support for user *presence*, *coordination*, and *communication* when needed. For instance, a multi-user real-time text editing tool like the one provided by Google Docs could be useful to collaboratively author a document, however a system like Gobby—which at the same time provides a list of the participants and allows them to communicate through a chat—might be even more efficient.

When: is participation synchronous or asynchronous? Depending on the type of object and participation different interaction models should be studied: for instance, if the object is atomic (i.e. a shared document) and many people have to collaborate in a synchronous way, they also need a synchronous communication system; otherwise, if users can participate independently and in different moments they can communicate asynchronously (i.e. Wikipedia and its discussion pages).

Where does the activity take place? This question is actually a collection of many more specific ones. For instance, we might be interested into knowing what is the context of use (i.e. work, home, university, etc.); if the system is going to be accessible from mobile devices; if the activity requires users to be connected to the internet or if it can proceed offline too. Every context might require a different approach on data: information has to be *tailored* for mobile users or depending on the different context [35]; applications (i.e. offline readers or Desktop clients—such as Picasa for photos) and technologies (i.e. Google Gears⁷) have to be developed to allow offline participation.

Is the system centralized or distributed? A distributed system might be more reliable on the long term, as it does not have a single point of failure. However, it also needs a longer time to bootstrap and, as available resources are directly proportional to the participants, it might

⁶As an example, search “virtual band” on YouTube or check <http://www.youtube.com/watch?v=KkGJ0gM7RDI>.

⁷<http://gears.google.com>.

require targeted incentives to attract the very first “critical mass” of users.

Why

These are the dimensions that are mostly bound to engagement and incentives. While we already spoke about incentives (see previous section), one of the very first questions we should ask is: *do we actually need to engage users or do they already have their motivations to participate?* In fact, depending on the community users might already be engaged into an activity: in this case, the purpose of the system should be to take advantage of this to make them participate more easily.

3.1.3 Collective intelligence and LPP

User contributions are usually considered as the main value inside participative systems, as they are unique and hard to recreate. However, we think that in all the systems characterized by a collective intelligence the tool that aggregates information, while not as valuable as the information itself, can nevertheless be very important. The reason is that the tool can add its own “intelligence” to the final result, making it much more valuable than a simple sum of the single contributions (see Figure 3.1). Actually, this is quite an evident fact, as intelligence in the tool can be easily spotted in many different cases. For instance:

- in the model and the algorithm that are used to provide related information: as an example, the “related resources” functionality which appears in many social websites (like the related videos on YouTube or related tags on delicio.us) can be provided using different, more or less intelligent algorithms;
- in the possibility to aggregate information coming from different sources: for instance, tag clouds allow users to see the tags that are most used inside a folksonomy and, at the same time, access all the resources which match one particular tag, no matter who added them to the system;
- in the possibility to infer new knowledge from the one which has been collected, for instance using a reasoner.

Due to the many different specializations the tool can have, it is quite difficult to give it a name which fits without being too generic. As the

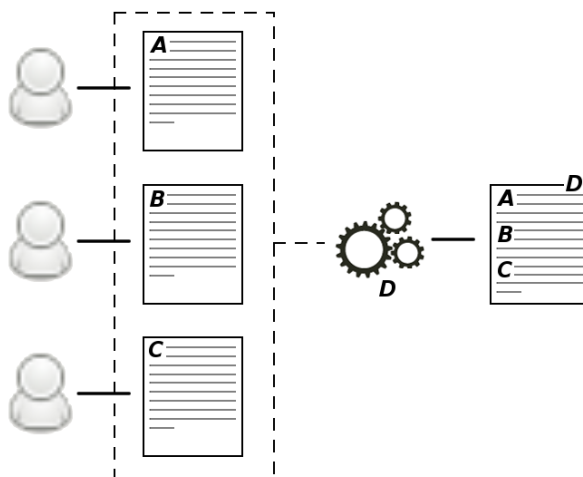


Figure 3.1: Intelligence as a product of the data sources and the collection engine.

common point between all the specializations is the collection of data, we refer to the tool as a collection engine: however, it is important to note that collection is not the only operation performed but, as shown in the previous examples, it can be followed by aggregation, inference, or more generally an elaboration of data.

To design a collection engine, there are some points that we need to keep in mind. First of all, as in any computer-aided discovery system, we should consider the machine (in our case the tool itself) as working on just a model of the system. This means that the collection engine can only see a part of it, that is the one which is described by the model; also, access to the information provided by the model is regulated by the model itself. This might have different consequences, depending on whether we want to create a new system from scratch or just add new features to an existing one. In the first case, we might want to expose (either to some internal collection engine or through an external API) all the information that we consider potentially interesting: this could include user and resource information, relations between users and resources, statistical information gathered from the interactions of the community with the system and so on. When the work is an extension of an existing system, instead, the main task is providing the highest possible value just by taking advantage of what is available. For instance,

adding related information by linking the existing one with other data sources has become a very common task recently: the main reasons for the success of this approach could be found in the high amount of information which is already available for free and in the fact that this partly solves the problem of bootstrapping the system with new contents.

According to activity theory, the tool is built both to enable users in their activities and to limit them. But activities might change in time together with user involvement in the community (see Legitimate Peripheral Participation in Section 2.3.3), so the system has to be flexible enough to allow tasks which vary in complexity and final effects on the system: for this kind of interactions we use a model which describes the collection engine and the different ways users interact with it. Looking at the system from the perspective of the machine that runs it (i.e. the physical or abstract machine, or the interpreter), there are three possible types of inputs: the *program*, that is the actual code executed by the machine; the *parameters*, which—given the same input data—allow the program to run in many ways (i.e. perform different operations or generate different output) without the need to modify the code; finally *input data*, which are the information that the program elaborates.

To better describe this concept, we take as an example Magnus' Map Sources/GeoHack tool, a Web application that gets geo coordinates as input and returns a collection of links to mapping and GIS-related services. The tool first converts the coordinates in many different formats (fitting the requirements of the mapping services), then builds the output page from a template scraped from Wikipedia (GeoTemplate) and completes it with the coordinates. The tool is a collection of php scripts that run on a Web server, so it is a program; it uses a wiki page as a template for its outputs, so users can customize the way results are presented just by editing that page; finally, the input data it accepts is represented by a pair of geographical coordinates which are usually specified inside Wikipedia pages using another template.

Writing these three inputs requires different levels of knowledge. The program can be created or modified only by someone who knows (at least part of) the wiki system, a programming language and the algorithms and data structures needed to perform the desired actions. Parameters (in this case MediaWiki templates) usually do not require such an in-depth knowledge, however they might still contain information which is related the program and they might have to conform to some specific formats. Input data, instead, has the lowest requirements, that is, a basic knowledge about what is being written (e.g., the content of a generic wiki

page).

These three levels of knowledge match three different user classes: the programmer, that is the person who creates a wiki, its extensions, or a collection engine; the expert user, who is able not only to edit pages, but also to create and use templates and customize the way applications work; and the beginner user, who can create and modify wiki pages, but is not able to do much more. On the other hand, changes made at different levels reflect a different impact on the system. A change in the input data might have a small scope: in a wiki, it can result in just one or few pages. A change in the parameters might have more evident consequences (for instance, changing a template modifies all the pages which use it), however they are constrained by the way the program uses them. A change in the programs, instead, might modify the behavior of the application as a whole.

Note that this model does not take into account another very important class of users, namely the readers, which most of the times are much more than the writers: this is due to the fact that we are only considering users who actively contribute to the system. A possible extension to our model might take into account another input class, that is metadata, which can be generated by any kind of interaction with the system: this would also allow us to describe all those features (like page history and recent changes in wikis, but also access statistics, ratings and recommendations in many other systems) that allow intelligence to emerge from very basic user behavior.

3.2 Extending systems with semantics

One of the main advantages of participative systems is that users contribute huge amounts of data. However, most of it is in an unstructured form, good for human consumption but not interpretable by a machine: thus, albeit useful, this information cannot easily be processed and reused inside other applications. On the other side, Semantic Web technologies usually provide structure and formalisms which allow data to be easily understood by machines; however, they lack of data and participation, mainly because of their complexity which acts as an entrance barrier even for expert users. Thus, a contamination between these two worlds would provide great advantages to both of them: on one hand, communities could be exploited to produce large amounts of structured information; on the other, semantically enabled data could be better linked, searched, and interpreted by machines.

No wonder, then, why in the last years the Semantic Web community has grown such an interest in participative systems. Also, semantics can be added to any one of them with different purposes and varying degrees, fitting the needs of each particular community. Basically, we can spot (at least) the following three levels of “semantification,” according to the main techniques that are applied to some system’s data:

- link data: the main purpose of semantics here is using well-known standards and vocabularies to describe data. Currently, the most used standard is RDF and there are also some very common vocabularies (just to name a few: DC⁸, FOAF, and SIOC [26]). Once information is described in these formats, it can be made available to any application which supports them; also, it is possible to create new value just by linking different data items from different sources. Many efforts and advances have been done recently in this field: there is now a whole community collaborating on the W3C SWEO Linking Open Data project, with the goal of sharing various open datasets as RDF on the Web and connecting different data sources with links between their data items. Also, other projects such as Freebase are gathering information from different sources, adding semantics to it and sharing it through APIs that can be easily accessed from any application;
- better describe knowledge: every data-intensive system has to describe its own contents in some way: semantics allow to do this more formally, making information easier to search and manage. The tool which is used to formalize knowledge is the ontology (see Section 2.5.1): the use of its classes and relationships allows for a very expressive description of the domain of knowledge and provides new means to organize, browse and search information;
- infer new knowledge with reasoning: inference is a rational activity that can be performed by a machine, and consists in finding new implicit pieces of information as logical consequences of what has already been explicitly asserted. Reasoners are programs which, given an ontology as their input, perform different operations on it, such as verifying its consistency or expanding the set of axioms with inferred statements (some examples of these programs are Pellet, FaCT++ and Racer). However, very basic inference processes are also present in the two previous levels and can be

⁸<http://dublincore.org>.

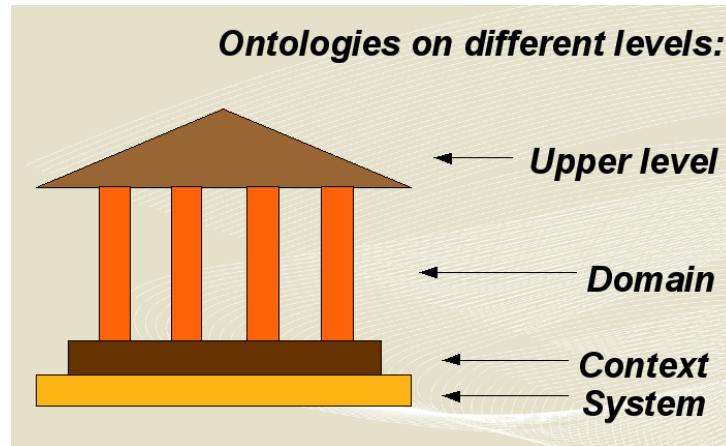


Figure 3.2: The “temple” structure of semantic layers.

performed without using a reasoner. For linked data, an example is the simple concatenation of relations: if one data source says that A works with B and another one contains the email address of B, after the two sources are linked together it is possible to ask for all the email addresses of A’s co-workers. When ontologies are used, a basic example of inference is retrieving information about classes which are more or less general than a specified one inside the hierarchy: for instance, if a Web page speaks about chihuahuas and chihuahua is defined as a subclass of dog, then it is possible to find the page just by asking for all the pages that speak about dogs.

These techniques can be used, either alone or together with others, to improve a participative system, but of course there is no strict rule about which one to apply. On one hand, it might seem that the real new value in this technology is to “let the machine reason for us,” but there are still many constraints to deal with: just to name a few, the problem of scalability with reasoners, the need for absolute precision (one wrong axiom and all the inferred information might be false), and the unintuitive formalisms that are so far from our common way of expressing knowledge. On the other hand, Jim Hendler’s motto “A little semantics goes a long way” perfectly summarizes how much can be obtained just by linking information coming from different sources. This also seems like one of the current trends on the Web, and it is surely producing a lot of new useful systems and data.

When ontologies are involved, semantics can be provided on different layers with respect to the system: according to our classification (see Figure 3.2), each layer has its own characteristics and can be used with different purposes. The lowest level (that is, the one which is more near to the tool) is represented by *system ontologies*. As the name implies, these are the ontologies that are used to describe the system itself. Usually they are very useful for interoperability purposes, as they can provide details about the different components and give users a way to easily link information from one system to another. For instance, if different wikis adopted the same ontology to describe their pages, with all their components and their contents, it would be much easier to extract information from one system and migrate it to another one; if two folksonomies adopt the same folksonology (as described earlier) it is possible to study the usage of tags in a way which is independent from the system they come from. At the same time, system ontologies provide a good, formal and extensible interface which can be used both by users and applications to speak about other programs. For instance, many applications in the Jena framework use RDF as the main language for their configuration files: as a result these files are very expressive and easily usable across different applications of the same framework.

Context ontologies allow to model everything which is not exactly the tool, but which is related to it due to some particular context of use. There are different ways these ontologies can be applied: for instance, a generic ontology about relations between documents could be applied to the pages of a wiki (as described below for one of our projects); an ontology about e-zine publishing processes could give shape to an open editing system to constrain it according to community rules and division of labor, and so on.

At the following layer we can find *domain ontologies*, which are very common in semantically enabled participative systems. These ontologies allow to describe the contents of the systems, that is the data that is published inside them, with a formal description of a particular, usually narrow, domain of knowledge. The main advantage of these ontologies is that they can be very detailed, providing many useful information and links between different concepts. However, their specificity is also their main drawback: while they fit well for very specialized systems (e.g., a wiki about programming languages), they cannot describe all the knowledge that is found inside a more general one (e.g., *del.icio.us* folksonomy, or Wikipedia). In this case using a lexicon like WordNet is much better, as it at least provides a taxonomy of all the common terms

inside the English dictionary.

At the topmost layer we find *upper level ontologies*, which describe concepts so general that can fit in many different systems. For instance, ontologies speaking about dates, geo coordinates, unique identifiers (such as ISBN codes or IMDB ids) can be considered upper level ones. Their main advantage is that they can act as a common vocabulary to enhance interoperability between different application and the linking of different datasets.

3.3 Architectures

3.3.1 The client-server-server architecture

Client-server-server is a friendly name we used for an architecture which is not novel and is very similar to the one used by Web annotations⁹, borrowing from design patterns such as the proxy pattern¹⁰, intermediary architecture [145], and layered systems (as described in [53], Section 3.4.2).

One of the main reasons why we used this architecture is that it allowed us to build our prototypes using a common browser. We believe that this tool is a very powerful client to access information on the Web; moreover, people are getting more and more used to it and exploiting an already well-known tool makes our work on user interfaces and interactions much lighter; finally, last-generation browsers can be easily extended with plugins and there are already many examples available on the Web. For all these reasons we extensively used this architecture, choosing Firefox as the browser and developing extensions for it, contacting Web services developed by us or which were already available on the Internet.

Our architecture is shown graphically in Figure 3.3 and operates as follows:

- (a) the browser connects to a chosen website and the user performs some operations inside it;
- (b) an extension working within the browser gets information related to the user's request. It then sends this data to a metadata server, which reacts accordingly, saving information and/or returning a reply;

⁹See <http://c2.com/cgi/wiki?WebAnnotation>.

¹⁰See http://en.wikipedia.org/wiki/Proxy_pattern.

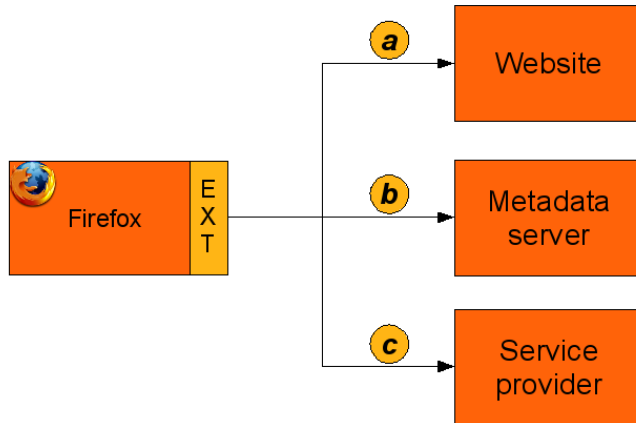


Figure 3.3: The client-server-server architecture.

- (c) when the client receives the metadata server response, it can elaborate this information (possibly accessing other service providers) and then show it to the user.

As an example, the extension might read the current URL and contact the metadata server to search for annotations about it; the returned metadata could be the annotations themselves, that are then shown by the extension overlaid on the currently shown Web page. In another use case the system could allow users to select some text inside a page, check for information about it and enrich the page with related contents. Some examples of our projects using the client-server-server architecture are an extension which provides an alternative navigation pane for del.icio.us tags (Section 5.1), a semantic annotation tool (Section 6.2), and a sidebar which uses Freebase and different Web services to provide site-related information (Section 6.3).

3.3.2 Wiki extensions

Another very common architecture in our projects is the one that characterizes all of our wiki prototypes (see Chapter 4) and makes use of JSPWiki.

JSPWiki is a wiki engine written in Java that we extended either through plugins¹¹ or by defining new classes inside the project itself. On

¹¹See <http://www.jspwiki.org/wiki/JSPWikiPlugins>.

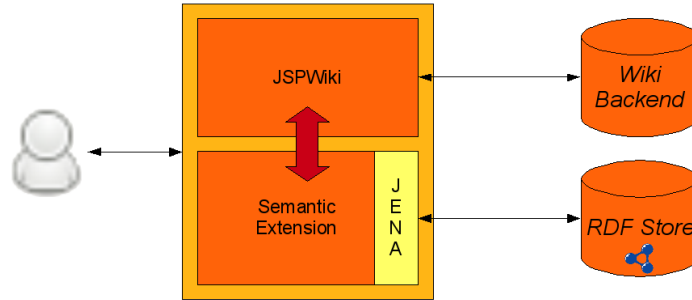


Figure 3.4: The generic architecture for our wiki extension prototypes.

top of the original backend we added an RDF store, used to manage all the formal knowledge inside the wiki. We decided to keep metadata separate from page contents as it allowed to manage knowledge more easily, even knowing that in this way we would have lost the “built-in” metadata versioning. Depending on the project, RDF has been saved into a simple OWL file (especially when the ontology was small and did not need to be updated) or into a persistent store when we needed extensibility and scalability. Finally, we used our custom code as a bridge between the original wiki, which still provides the unstructured page contents, and Jena, which is used to access the RDF store and provide reasoning capabilities to the system.

The architecture is shown graphically in Figure 3.4. Every time a wiki page is requested, JSPWiki code retrieves it from the backend which can vary depending on the *provider* used by the system: for instance, pages could be files on the filesystem, like in the default provider, or could be stored inside a MySQL database. Then, JSPWiki leaves control to our extension which uses Jena to get metadata from the RDF store, elaborates them, and finally shows the enriched output to the user.

3.4 Experimental developments

In this section we introduce some of our past projects, as examples of our approach. For each of them we provide a short description and a name which refers to a row inside Table 3.1. This provides a compact view of the projects’ characteristics: *Tool* is the tool that has been created or extended within the context of the project; *Expressiveness* specifies what kind of ontology has been used (basically thesaurus, RDF, or OWL); *Use*

of semantics shows how or with which purpose semantics has been used; *Architecture* specifies which architecture we chose while designing the system.

These projects will be described in detail in the following chapters, in the same order we used to introduce them here: the first three appear in Chapter 4, the next two in Chapter 5, and the last three in Chapter 6. In this way, they are not ordered only by the type of system we used, but also by decreasing semantics (from the first one containing light reasoning to the last ones, only aiming at linking data) and, at the same time, with lower and lower requirements for user interactions (that is, the systems tend to be more and more automatic).

[SWTemplate] *A templating system for resource visualization in a semantic wiki*: one of our semantic wiki prototypes, provides users with templates (similar to Wikipedia infoboxes) which are generated automatically from the underlying ontologies. The information generated with templates is saved as RDF and can be accessed both offline, through an RDF endpoint, and inline, allowing users to embed query results in the text of any wiki page.

[SWAttach] *Semantic management of attachments inside a wiki engine*: another wiki prototype, automatically extracts metadata from file attachments and saves them inside an ontology. These metadata can then be queried offering users the chance to easily find/filter information which is attached to wiki pages.

[SWContext] *Design of a context ontology inside a semantic wiki*: another wiki prototype, uses a context ontology which describes relations between documents such as “complementary” or “propaedeutic”. Then, when users browse the wiki, they are offered an interface to access related documents (note that some relations are determined by inference too).

[WordFolks] *Using WordNet to turn a folksonomy into a hierarchy of concepts*: the project uses WordNet to disambiguate the related tags (that is, tags co-occurring with a given one) inside del.icio.us and shows them in a hierarchy. This provides, at the same time, an alternative navigation interface for users and a way to (partially) solve the problems of homonymy, synonymy, and spam inside folksonomies. This work has also been published in [86, 87].

[Tagonto] *Improving search and navigation by combining ontologies and social tags*: the goal of this project is to combine ontologies and folksonomies by automatically mapping (social) tags to more structured domain ontologies. The automatic discovery of associations is based on

3 Our approach

a set of matching algorithms computing similarities. Disambiguation heuristics are then used in order to debug multiple associations between tags and concepts in the ontology. This work has also been published in [18].

[SquirrelIMAP] *An IMAP plugin for SquirrelRDF*: an adapter tool for the IMAP protocol, developed as a plugin of SquirrelRDF, which allows users to query IMAP mailboxes using SPARQL. The information returned looks like RDF, is always current, and can be reused and integrated inside other applications. This work has also been published in [50].

[SpeakinAbout] *Exploiting user gratification for collaborative semantic annotation*: presents a new collaborative semi-automatic annotation approach for Web pages, which requires almost no knowledge about semantics on the user side. Using domain ontologies or structured sources like Freebase as a suggestion tool for concepts, the tool links annotated data to a set of online services and resources specific to their related concepts, thus providing an instant reward for users in the form of additional available information. This work has also been published in [49].

[RDFMonkey] *Using semantics and user participation to customize personalization*: the goal of this project is to allow users access their own personal information (in specific, their browser's history) and build custom applications that exploit it. To do this, we exported Firefox history and bookmarks in RDF and made them accessible both through a SPARQL endpoint and from within a Firefox extension. As an example, we developed two plugins: one for the visualization of user history, using Google's MotionChart, and one for enhancing browsing with related information which is downloaded in realtime from Freebase. This work has also been published in [48].

Codename	Tool	Expressiveness	Use of semantics	Architecture
SWTemplate	Wiki	OWL	Manage Knowledge	Wiki plugin
SWAttach	Wiki	OWL	Light Reasoning	Wiki plugin
SWContext	Wiki	OWL	Light Reasoning	Wiki plugin
WordFolks	Folksonomy	Thesaurus	Manage Knowledge	Client-Server-Server
Tagonto	Folksonomy	OWL	Manage Knowledge	Other
SquirrelIMAP	Other (Mail)	RDF	Link Open Data	Other
SpeakinAbout	Other (Annotation)	RDF/OWL	Link Open Data	Client-Server-Server
RDFMonkey	Other (Browser)	RDF	Link Open Data	Client-Server-Server

Table 3.1: Classification of our experimental developments according to different dimensions.

4 Semantic Wikis

4.1 A Templating System for Resource Visualization in a Semantic Wiki

Templates are a common tool inside wiki systems which allows users to show data in a consistent way. Wikipedia makes an extensive use of templates: for instance, articles about cities use them to provide information such as Country, State, Area, Population, and so on. Even if templates were not built with the main purpose of structuring information, it is still possible to extract part of their contents and give them a semantic interpretation: DBPedia project does exactly this, extracting structured information from different Wikipedia components (templates but also categorization of information, images, geo coordinates, and links to external pages) and collecting them inside a knowledge base which is then made available on the Web.

Our main purpose in this project was to provide wiki templates built not only for presentation purposes, but also for the creation and management of structured information. The tool is an extension of JSPWiki and allows users to import different ontologies into the system. When a wiki page is created or modified, it is then possible to match it with a particular class: this automatically gives users access to a template showing the possible properties for that class. Information is then shown by default as an infobox inside the wiki page, but it can also be accessed in different ways: the box can be customized during editing and visualization, so that only the fields that the user considers interesting are shown; information is available as RDF data and is accessible through a SPARQL endpoint; finally, a JSPWiki plugin allows users to insert SPARQL queries inside the wikitext, updating page contents with information extracted on the fly from the knowledge base.

4.1.1 Project architecture and implementation

This project follows the architecture described in Section 3.3.2: the `WikiContext` class inside JSPWiki has been used to call our semantic wiki engine instead of the default one; the class `SemWikiEngine`,

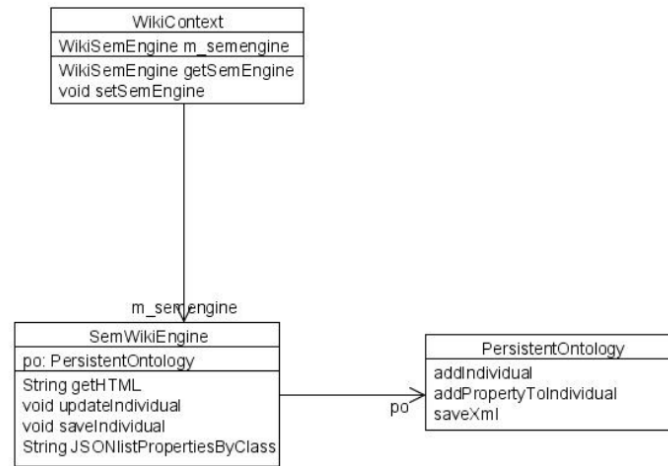


Figure 4.1: Class interaction diagram for the semantic template project.

then, manages semantic templates and the knowledge base through the **PersistentOntology** class (see Figure 4.1).

Pages in this system are considered as textual descriptions of resources, while templates provide an interface to show and modify their formal properties. The system, of course, is parametric with respect to domain ontologies. We use them as formal definitions for our templates: when a new page is created, we ask users to choose what class it belongs to (between the ontologies that are available), then the template fields are automatically loaded. Pages can also be matched with non-local URIs, automatically importing information inside their templates from a remote source.

When a user loads a page, the semantic engine checks if there is a related resource and shows its properties inside the template: literals are shown as plain text, while all the resources have a link to their related pages. In any moment, user can click over a field and modify or add information to the template.

Our template design followed a very basic, but often ignored, assumption: while ontologies provide a very formal approach to knowledge, as any other system which is designed in a top-down way they tend to describe reality from a single perspective which might not be the one every user agrees on. Now, even if we decided not to allow users to modify the ontology, we at least wanted them to be able to customize their own interface to access it: as a result, users can show or hide template fields depending on their interests. This is particularly useful when a person

The figure shows two versions of a web form for a semantic wiki, illustrating a customization where the 'knows' field can be active or not.

Left Form (Active 'knows' field):

- URL: http://localhost:8080/SemJSPWiki/RDFFiles/Leo_Ortolani
- type: [Person](#)
- birthday: "14 gennaio 1967"
- firstName: "Leonardo"
- gender: "Male"
- knows: [Lorenzo Ortolani](#)
- knows: [Andrea Plazzi](#)
- surname: "Ortolani"

Right Form (Inactive 'knows' field, Active 'surname' field):

- URL: http://localhost:8080/SemJSPWiki/RDFFiles/Leo_Ortolani
- type: [Person](#)
- birthday: "14 gennaio 1967"
- firstName: "Leonardo"
- gender: "Male"
- surname: "Ortolani"

Figure 4.2: An example of a customized template, with the “knows” field active or not.

has to fill the same fields for many individuals and does not want to read across huge forms, or when a student has to compare similar information between different wiki articles. An example of template customization is shown in Figure 4.2.

Once information is saved into templates, it can then be retrieved from within the wiki page. Thanks to a wiki plugin, users can export template fields into the wikitext (with a SPARQL query in the prototype, but a much more intuitive interface is currently being developed): the result is that information is always current and updates made just in one place (the template) are automatically reflected in the text referring to it.

To enable reuse of the information published within our wiki and make it more easily interoperable with other applications we decided to make template information immediately available through an RDF endpoint. Every wiki page contains a link that points to its RDF representation: this information can then be imported into other applications or shown with a compatible browser, as depicted in Figure 4.3.

4.1.2 Conclusions

In this project we used semantics to allow users to easily save information in a structured format, using content ontologies and a paradigm (the wiki template) that was already well known to users. As a use case, we chose to use the system to document software projects using the DOAP ontology: this also makes the generated information easily linkable with other

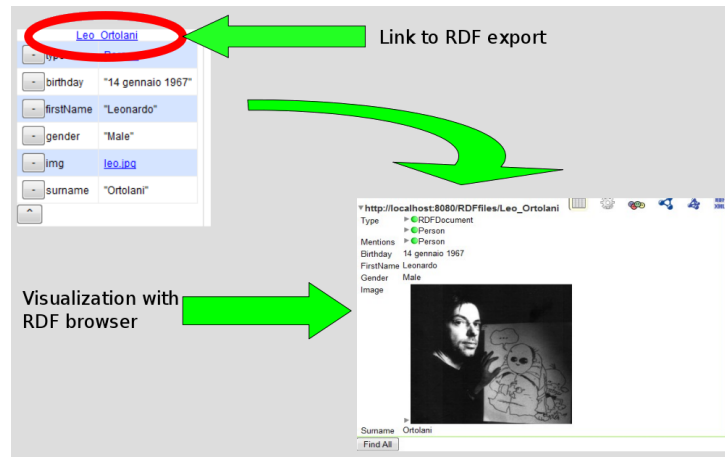


Figure 4.3: RDF information can be easily exported from any wiki page and then shown with an RDF browser.

existing data sources. We changed the user interface so that data could be added with a WYSIWYG interface rather than with wikitext, and we allowed users to customize the appearance of templates (i.e., which fields are shown). One huge difference from normal wiki systems is that there is less freedom in changing template fields as they are directly loaded from the ontologies. Actually, we believe that leaving the development of part of the ontologies to users would be a very interesting example, however we still lack tools to make it easy for everyone.

4.2 Semantic Management of Attachments Inside a Wiki Engine

Some wiki systems let users upload files as attachments to a page. This feature gives users new ways to use a wiki: for instance, pages could be used as collectors of metadata related to the attached file, or as a “gathering place” around which users can share different objects. As an example, a page devoted to a particular document could contain all the revisions of that document, with user comments and already available in various formats for consumption from different platforms.

The goal of our project was to provide a semi-automatic tool for the management of attachments’ metadata inside a wiki system. We developed a prototype extension for JSPWiki which is able to detect the file type, extract its metadata and save it inside an ontology for later

retrieval. The main advantages of this tool is that the user just has to upload the file and then the system takes care of everything else: if he likes, however, he can modify existing data or add new ones. The system uses ontologies to describe the hierarchy of multimedia files (to allow queries like “return all the pages that have audio files attached to them”) and their metadata (for queries like “return all the pages that contain files whose author is Davide Eynard”). Finally, it exports its information through a SPARQL endpoint and in RSS format: this last option allows the system to behave like a rich, wiki based podcasting system powered by semantics.

We divided the problem of designing such an application in four main tasks: *distillation*, that is the extraction of metadata from multimedia files and their storage, *visualization*, *editing* of metadata from within the wiki system, and *querying* of the saved information.

4.2.1 Distillation

To extract information from attached files we used JHOVE¹, an extensible framework which provides functions to perform format-specific identification, validation, and characterization of digital objects. JHOVE is implemented as a Java application with a very modular structure. Every module takes care of one or more specific file formats: for instance, there is one module to manage the GIF file type (in the two versions GIF87a and GIF89a), one for PDF files, one for WAVE and so on. A JHOVE module for a certain file type is in charge of:

- managing the identification pattern by which the file is identified;
- validating the file following some schema;
- extracting meaningful information (metadata), if they are encoded inside the file.

File types are organized in an ontology that contains a classification of different formats. Actually information is saved in a taxonomy, where “File” is the root and file types are the leafs. For instance, the path to “Mp3” is File->Data->Multimedia->Audio->Mp3. An ontology is used to store all the metadata that are extracted from a file (i.e. a video file might have an author, a date, a bitrate, and so on). The process of extraction and saving of metadata inside the RDF store is shown in Figure 4.4.

¹JSTOR/Harvard Object Validation Environment. See <http://hul.harvard.edu/jhove>.

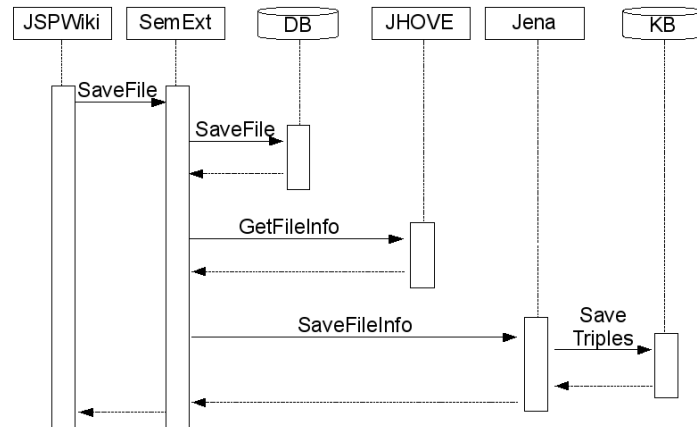


Figure 4.4: Sequence diagram of metadata extraction and saving for a file attachment.

4.2.2 Visualization and Editing

Once uploaded, files are shown at the bottom of the wiki page where they are attached. The wiki code has been modified to allow the display of file metadata: depending on the file type, different information are shown. Every piece of information can be modified by users in a very intuitive way, just by clicking on the field, entering text, and hitting the Submit button. An example of this is shown in Figure 4.5.


4.2.3 Querying

In our prototype, two methods for accessing attachments have been implemented. The first one is a form that allows users to ask SPARQL queries and see the results inside an HTML table (see Figure 4.6). The second is a simpler interface that offers the possibility to specify some constraints on the author, type, and name of the file, and generates an RSS feed with the attachments that satisfy the query: this allows users to subscribe to a podcast of their favorite attachments and be automatically updated whenever a file they might be interested in is uploaded.


4.2.4 Conclusions

Differently from other experiments, this project could be useful as a support tool for a new kind of activity: instead of using a wiki for publishing

Attachments

[Eric Johnson - Cliffs of Dover.mp3](#)  3990836 bytes

Track n°: 2
 Title: Cliffs of Dover
 Length:
 Author: Eric Johnson
 Album: Ah Via Musicom
 Year: 1990
 Genre: 17
 Bitrate (kbps): 128
 File Size (byte): 3990836 bytes
 Upload Date: 2008-01-29

[test.jpg](#)  389202 bytes

Title: Splashin' Lemon
 Author: Unknown
 Shot Date: 2007-12-30
 Shot Hour: 08:28:21
 File Size: 389202 bytes

Figure 4.5: Metadata automatically extracted from the attachments are shown inside a wiki page and can be edited by users.

```
SELECT ?file ?author
WHERE {?file ONTURI:haAutore ?author}
```

AnswerQuery

Your trail: [Main](#), [Query](#)

Domain URI is: <http://www.owl-ontologies.com/Ontology1197022770.owl>

file	author
Main/Eric Johnson - Cliffs of Dover.mp3	Eric Johnson

Figure 4.6: The result of a SPARQL query over the wiki attachment metadata.

textual contents, the same tool could be used with the purpose of sharing files. With our extension, adding a new file to the library would be rather easy and, at the same time, thanks to the simple wiki interface it would also be possible to write a description of the file, add related information (i.e. links to other related websites), or comments about its contents.

We designed this extension with the intention of adding the advantages of semantics without requiring users to know anything about them. The form for SPARQL queries might seem in contrast with our objective but it was implemented only for debugging purposes and advanced queries. Apart from that, interaction with the system would remain basically the same: users could just write wiki pages and upload their files, waiting for the system to automatically extract metadata; advanced users could decide to manually add other information. In both cases, metadata are saved in the RDF store, where they can be queried or aggregated—thanks to simplified interfaces—in common formats like RSS.

4.3 Design of a Context Ontology Inside a Semantic Wiki

Instead of using domain ontologies to describe the contents of a participative system (i.e. an ontology about food in a recipe website, the same kind we used in the semantic template project) or system related ones to describe its components (i.e. OMDoc inside SWiM, to describe mathematical documents, or the ontology we used to specify file-related information in the semantic attachments project), it is possible to use an intermediate one, that we called context ontology, to describe those parts of the system that are independent both from the contents and the system itself. As an example we developed a simple ontology that describes different relations between documents, such as the fact that two documents are part of a bigger one (meronymy), that a document provides an in-depth examination on some topics found in another one, or that a document is propaedeutical to another one (see Figure 4.7).

An ontology like the one we developed could be used in many different systems, as these relations hold for any kind of textual document and as different but similar relations could be created for other file types (i.e. package dependances inside a linux distribution or libraries inside a program). To test our work we used another wiki extension: a new navigation interface has been provided, allowing users to select related documents or to specify relations between them; reasoning has also been used

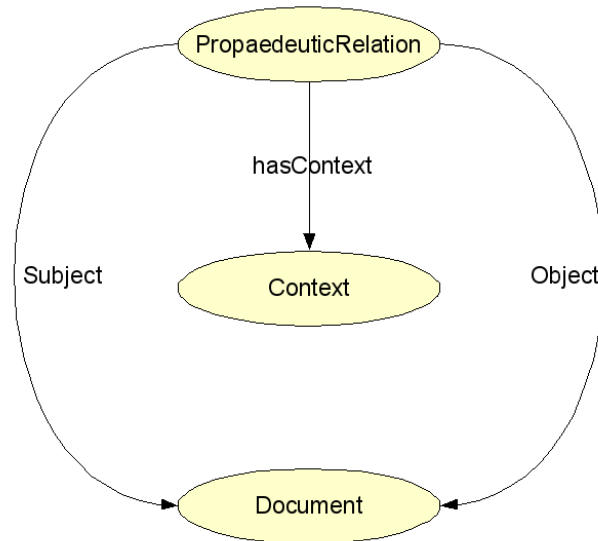


Figure 4.7: Graphical representation of the reified Propaedeutic relation.

to infer implicit knowledge (e.g., the transitive closure of the “propaedeutic” relation).

4.3.1 Conclusions

This project provides semantics on a more advanced level, that is one which also implies reasoning. The examples here were very simple, but nevertheless very useful for users browsing the website. Actually the system requires users to interact in a completely new way (that is, specifying relations between pages), but this kind of contribution is not compulsory and we thought it as a more advanced action that could be performed by expert users, more interested into keeping the overall system quality high rather than just contributing with page contents. The final result provided by the collection engine is a completely new navigational structure that is generated on the fly thanks to reasoning.

5 Folksologies

5.1 Using WordNet to Turn a Folksonomy Into a Hierarchy of Concepts

As the amount of information available in the Web grows every day faster, the task of classification is getting harder, the traditional top down approach is getting inadequate [133], and the new collaborative approach of *folksonomies* is emerging [118].

In folksonomies users can associate freely chosen tags to resources and in this way they produce knowledge for the entire community. Beside their dynamism and low cost, folksonomies present many disadvantages: in particular, their lack of hierarchy limits the possibility of searching and browsing related information.

Joshua Schachter, founder of *delicio.us*, defined it as “*a way to remember in public*”; in folksonomies each user can generally explore two spaces, the one of his bookmarks and the one of everyone’s bookmarks; tags can be used to filter items.

As the work of categorization is performed by users, folksonomies are democratic, scalable, current, inclusive and have a very low cost. On the other hand, the absence of an authority and of a unique coherent point of view on the domain bears several limitations: the lack of hierarchy, the absence of synonym control, the lack of both precision and recall, the possibility of *gaming* [80] [132]. While the traditional classification schemes, based on taxonomies, favor searching and browsing, folksonomies encourage another paradigm of navigation, based on *finding* and *serendipity* [97].

Despite their strong limitations, folksonomies are rapidly gaining momentum: according to Clay Shirky¹, “*The mass amateurization of publishing means the mass amateurization of cataloging is a forced move.*”

As tags are just text strings, with no explicit semantics associated, it is not trivial to organize them for presentation to the user. The most common way to show a set of tags are *tag clouds*, visual representations

¹See http://many.corante.com/archives/2005/01/22/folksonomies_are_a_forced_move_a_response_to_liz.php.

where each tag is displayed with a font size which is proportional to its popularity. Tag clouds, however, do not keep into account relationships among tags or their meaning.

To allow the discovery of interesting and related items many applications have introduced links to *related tags*, where relatedness is generally measured with metrics based on co-occurrence data. For example in del.icio.us, when a user visits the page containing all the bookmarks tagged with a certain tag, a list of tags related to that one is shown inside a sidebar.

Flickr, a popular folksonomy for photo sharing, introduced *clustering* as an interesting feature to help navigation in the space of a tag. The system is able to find clusters of related keywords, so items corresponding to different contexts for that tag are grouped together.

These features are very useful but often insufficient, for different reasons. First of all, they leave the lack of hierarchy problem unsolved: they build flat spaces of tags, so there is no criterion to organize them and only a small set of items can be displayed. Furthermore, there is no explicit connection with the meaning of keywords or semantic relationships among them, that might help users to orient themselves in the tag space.

An interesting study to integrate a *top down* classification paradigm with folksonomies is presented in [119]. Some investigations about the challenge to derive ontologies from folksonomies are presented in [129] and [37].

5.1.1 Project overview

The goal of our work is to investigate the possibility of integrating an ontology in the navigation interface of a folksonomy, filtering tags through a predefined semantic hierarchy to improve the possibilities of searching and browsing. In particular we chose to improve the *related tags* panel in del.icio.us; filtering a set of related tags through WordNet noun hierarchy it is possible to display a much higher number of them, organized according to a semantic criterion. As WordNet is a semantic lexicon of English, developed to reflect the semantics of natural language and the way in which humans classify objects, the relations and categories that it contains are likely to be immediately understood by most people [51].

The first problem when trying to map tags to WordNet is the one of tags that are not recognizable as words in the lexicon, even after a stemming process, and therefore cannot be mapped. To evaluate the relevance of the excluded data we have collected a large dataset, relative

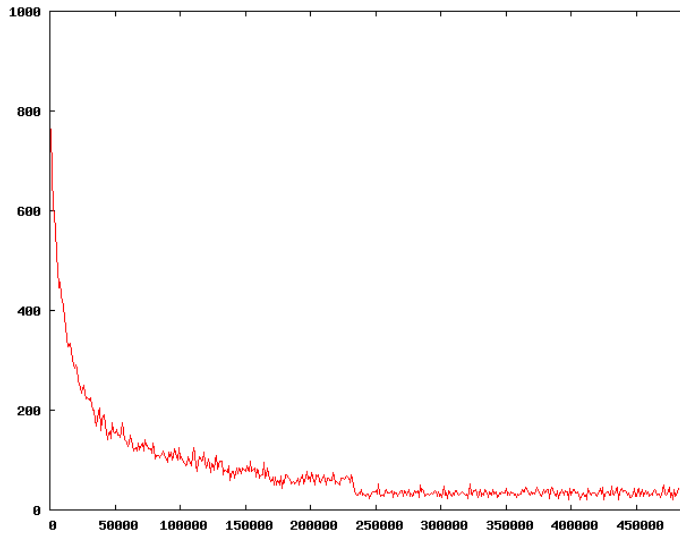


Figure 5.1: The image shows the probability that a tag belongs to WordNet, in (inverse) function of its popularity. Along the X axis are represented tags from our dataset, grouped by 1000 and ordered by decreasing popularity; the Y axis shows the number of tags belonging to WordNet for each group of tags. The most popular tags are much more likely to belong to WordNet, following a power law distribution.

to about 30,000 del.icio.us users and containing about 480,000 different tags. Studying these data we found that only about 8% of the different tags used are contained in the lexicon, but we also observed that the most popular tags have a much higher probability of belonging to WordNet. This distribution in particular follows a power-law curve, very common in the field of collaborative systems, as showed in Figure 5.1. Of the 20 million total tagging relations present in our dataset, about 68.1% involve words contained in WordNet. We think this data might be much increased by using local wordnets in other languages and domain ontologies to cover more specific terms.

There is then the problem of words that are recognized as belonging to the lexicon, but not as nouns: these tags also cannot be mapped, as the hierarchy of WordNet is only defined on nouns. According to the distinction formulated in [58] among *factual*, *subjective* and *personal* tags, we can argue that factual tags tend to correspond to nouns, as nouns fit better to describe factual knowledge, while adjectives tend to

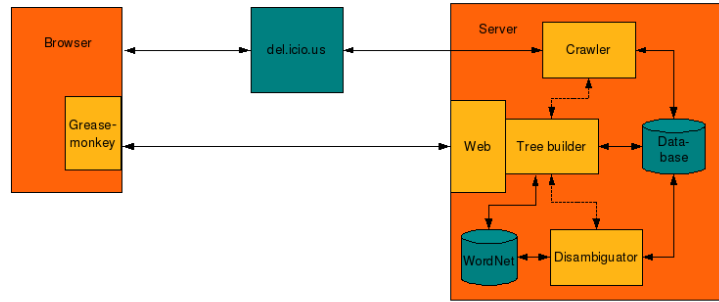


Figure 5.2: The system architecture.

correspond to subjective tags. Further studies about this issue can be found in [3]. From a quantitative point of view, our dataset confirms the intuition that most of the tags, and especially most of the most popular tags, are nouns. Indeed 85% of the different tags recognized by WordNet are nouns, while of the over 20 million total tagging relations, about 64.9% involve WordNet nouns, and just about 3% involve words belonging to the lexicon without being nouns; in other words this data tells that, in our dataset, about 95% of the times that a tag belonging to WordNet is used it has almost one meaning as a noun: the power law distribution is accentuated for nouns.

The application we have developed is based on a client-server paradigm, where all the tasks relative to the processing and storing of information are left to the server and the client has only to manage the visualization of results. The system architecture is shown in Figure 5.2.

The server is composed of a *scraper*, that extracts the data from delicious.us HTML pages and stores them on a database, a module for *tag disambiguation* and a core module that builds the *semantic tree* of tags related to a given one, based on the hierarchy of concepts of WordNet. On the client side, according to the principle of *active navigation*, a JavaScript script executed inside the browser dynamically modifies the pages visualized by the user, integrating the additional information provided by the server.

5.1.2 Tag disambiguation

One problem when trying to map tags on an ontology is polisemy: as no explicit semantics is associated to tags by the users, the same tag can

have different meanings according to different acceptance of the word, and consequently different positions in the ontology. For instance the word “turkey” may refer to the country or to the animal, and in the second meaning you could want to distinguish between biological and gastronomic meaning, according to the context. In WordNet semantic relationships are not defined among words, but among *synsets*, groups of synonyms that represent units of meaning; each word can belong to different synsets according to its different acceptations. The word “turkey”, for example, belongs to five synsets, where the first one is “turkey, *Meleagris gallopavo*” and the second is “Turkey, Republic of Turkey” .

To properly map a tag to the corresponding position in the ontology you need first to disambiguate it, in relation with the context in which it has been used. A fair solution naturally offered by a folksonomy is to use the other tags associated by some users to the same resource as the context for disambiguation.

Our algorithm for tag disambiguation acts for each tagged resource in the following way: the C most used tags for the resource are compared among them, and for each of them the meaning that is more strictly related to the other tags is selected; semantic relatedness among tags is calculated according to a choice of metrics based on WordNet [116] (adapted Iesk, Hirst and St. Onge) and disambiguation is performed using the Perl library SenseRelate [115]. In the same way the remaining tags are disambiguated using the first C as a context. This solution is effective, as it reduces the sensitivity to less used tags, and efficient, as it avoids the exponential growth of the algorithm complexity with the number of different tags associated with a resource.

5.1.3 Building the tag semantic tree

The core module, for the construction of the tree of related tags, acts in four steps: *tree building*, *compression*, *branch sorting* and result output. All the algorithms developed have linear complexity with the number of input tags.

The set of tags to be considered is selected by collecting, for each of the latest N sites associated with the given tag, the M most frequent tags for that site; M and N are parameters that can be specified in the HTTP request. The construction of the tree is performed by an iterative algorithm; for each different tag present in the set of interest in a particular acceptance, the chain of the hypernyms is created as a path till the unique root of the noun hierarchy of WordNet and then merged with the existing tree. At the end of this process the tree is a subpart

of WordNet noun hierarchy, chosen to contain all the tags of the set of interest.

As WordNet is very fine-grained, it can take more than 10 steps to descend from the root to a word; the tree has to be compressed to be useful for navigation, eliminating the useless nodes. The compression algorithm performs a breadth-first visit of the tree, in which all nodes considered unnecessary are deleted and replaced by their children. On one hand, all the nodes corresponding to high level categories in WordNet, contained in a black list, are deleted; the information content of these nodes is generally too low to be useful for navigation. On the other hand all the nodes that do not correspond to any tag and have a branching factor lower than K or have no siblings are replaced by their children. The default value for K is 2; in this way the structure of the hierarchy is preserved and at the same time the most specific terms can ascend in the tree.

The branches are ordered by weight, where the weight of a node is calculated as the number of resources in the set of interest that have been tagged with the corresponding word in that acceptance. This guarantees that the branches of the hierarchy that are most strictly related to the given tag are shown first to the user. As a last step, the tree is output by the server in HTML or XML format.

5.1.4 User interface

The system rests on Firefox Browser and Greasemonkey extension to execute some JavaScript code inside the browser. When the user is visiting the del.icio.us page for a certain tag, the script connects to our server to get the semantic tree of related keywords for that tag; as soon as the information is ready, a new sidebar is dynamically integrated in the page, showing an expandable tree. For each node of the hierarchy there are two links, one directed to the del.icio.us page for that tag and one to the page of the resources tagged both with that tag and with the given one; the size of each tag's intersection with the current keyword is shown in parenthesis and represents an indicative measure of relatedness for the users. Tooltips guide users showing WordNet definitions of the concepts corresponding to each node and indicating the destinations of links.

Figure 5.3 shows the result obtained for tag “pasta”, where all the tags associated to the latest 300 sites tagged with “pasta” are displayed; in the picture you can see the first branches (i.e. the most related ones, in this case those about “food”), that have been expanded.

5.1 Using WordNet to Turn a Folksonomy Into a Hierarchy of Concepts

The screenshot shows the del.icio.us interface for the 'pasta' tag. The main content area on the left lists several recipe links, such as 'Sweet Potato Gnocchi with Brown Butter and Sage' and 'Pasta with cauliflower - The Boston Globe'. The right sidebar contains two sections: 'tags semantic tree' and 'related tags'. The 'tags semantic tree' is an expandable hierarchy starting with 'food (102)', which expands to 'pasta (283)'. Under 'pasta', it lists various types like 'macaroni (9)', 'noodle (9)', 'spaghetti (6)', 'ravioli (3)', 'gnocchi (2)', 'penne (2)', 'vermicelli (1)', and 'pasta in short tubes with diagonally cut ends'. The 'related tags' section lists terms like 'cooking', 'recipes', 'vegetarian', 'italian', 'healthy', 'main', 'parmesan', 'food', and 'sauce'.

del.icio.us / tag / **pasta** popular | recent
[login](#) | [register](#) | [help](#)

All items tagged **pasta** → view **popular**

« earlier | later »

[Sweet Potato Gnocchi with Brown Butter and Sage](#) [save this](#)
by questhoya to recipes pasta gnocchi sweet-potato ricotta ... [saved by 6 other people](#) ... 1 hour ago

[Pasta with cauliflower - The Boston Globe](#) [save this](#)
by dcrowley to recipes pasta ... 1 hour ago

[Weight Watchers Spaghetti Carbonara Recipe-Kitchen Crafts 'n' More](#) [save this](#)
by plm5087 to pasta ... 6 hours ago

[Aglio e olio - en utflykt i det italienska köket: PASTA MED MOROT, SPENAT OCH BALSAMVINÄGER eller PASTA CON CAROTE, SPINACI E ACETO BALSAMICO](#) [save this](#)
by horrorhead to mat recept pasta huvudrätt lunch ... 8 hours ago

[The Kitchen Pantry: Noodles, again and again and again...](#) [save this](#)
by skippytpe to recipes pasta asian ... [saved by 2 other people](#) ... 14 hours ago

[Fish on Fridays: Linguine with Clam Sauce](#) [save this](#)
by lofiEDDIE to recipes clam pasta ... 15 hours ago

[Chicken or Turkey Tetrazzini - Allrecipes](#) [save this](#)
by hulmekka to chicken turkey tetrazzini dinner recipes pasta comfort ... [saved by 1 other person](#) ... 15 hours ago

[cooking: Spaghetti Sauce?](#) [save this](#)

▼ **tags semantic tree**

- [-] food (102)
 - [-] pasta (283)
 - macaroni (9)
 - noodle (9)
 - spaghetti (6)
 - ravioli (3)
 - gnocchi (2)
 - penne (2)
 - vermicelli (1)
 - pasta in short tubes with diagonally cut ends**
 - lasagna (1)
 - rigatoni (1)
 - lasagne (1)
 - fettuccine (1)
 - [-] produce
 - [+] vegetable (12)
 - [-] edible_fruit
 - [-] veggie (2)
 - [-] meat (5)
 - [-] poultry
 - sausage (9)
 - [-] beef (7)
 - [-] cut
 - pork (5)
 - veal (1)
 - lamb (1)
 - [-] seafood (16)
 - [-] fish (6)
 - [-] baked_goods
 - chocolate (1)
 - [-] takeout (1)
- [-] communication
 - [-] message
 - [-] written_communication

▼ **related tags**

- cooking
- recipes
- vegetarian
- italian
- healthy
- main
- parmesan
- food
- sauce
- gnocchi

Figure 5.3: A screenshot from the del.icio.us page for tag “pasta”, where the inner sidebar shows an expandable hierarchy of related tags, provided by our application.

5.1.5 System evaluation

We tested the system with different kinds of tags, according to different dimensions. The first dimension is the specificity of the tag from which the exploration starts; it is very different to display the space of a keyword situated in a specific domain or in a generic one. In the first case the resulting tree tends to be compact and to allow easier navigation, while in the second case it tends to have a high branching factor and a high number of first level nodes; anyway, as the branches are always ordered by weight, the most interesting concepts in relation to the given one are reachable exploring the first branches, also in case of very general keywords. The second dimension is given by the popularity of a tag, while the third one is given by the semantic field; each semantic field has its specificity and some of them rest on more conventional and ordered sets of words, such as the *food* context, visible in Figure 5.3, while some others are more prone to slang and neologisms, such as the one of *software*.

Figure 5.4 shows the result obtained for tag “blog”; as “blog” often refers to a kind of site more than to the content it can be considered a particular case, and a very general tag as there are blogs almost about everything. “Blog” is also one of the most popular tags in del.icio.us, so it is an extreme case also according to the second dimension. We obtained this result considering the latest 2000 del.icio.us bookmarks tagged “blog”, and only the 15 more used tags for each of them, to cut the *long tail* of less used tags. In the picture you can see the hierarchy of scientific disciplines expanded.

According to this and other tests, the main problem for scalability seems to be the high number of nodes in the first level of the tree; some improvements could be obtained by making the tree compression algorithm more dynamic.

Comparing the related tags suggested by del.icio.us with the results we obtained, we observed that they are always somewhere in the first branches in the new sidebar. An exception must obviously be done for the words that do not belong to WordNet, that are absent in the new sidebar. Experimenting, for example, with the “Greasemonkey” tag (the experiment is possible even though the word itself is not contained in the lexicon) we found that many important related tags, like “JavaScript”, are not recognized, while other important words, such as “extension”, are interpreted in a wrong way as WordNet does not contain the acceptance related to software; all the tags for which there is in WordNet an acceptance related to software have instead been correctly interpreted

5.1 Using WordNet to Turn a Folksonomy Into a Hierarchy of Concepts

« earlier | later »

Holistic Learning
save this
by fournier48 to blog ...
just posted

トレンダーズ社長 経 沢香保子の「人生を 味わい尽くす」プロ グ save this
by tomojp to blog ...
saved by 1 other
person ... just posted

Seth Godin article - Be a Better Liar.
save this
by ffmike to blog
business article ...
saved by 20 other
people ... just posted

My RSS 管理人 ブログ (工事予定) : RSS 記事のクリック率 7% のほとんどは 「ポット」 save this
by kenkity to rss Blog ...
saved by 12 other
people ... just posted

L8dybug's Journal
save this
by INP to blog ... 1 min
ago

Radium Software Development-hype について save this
私自身が hype を燃料とし て動くエンジンのようなもの だったということに気がつい てしまったのです。。なる ほど！
by kenkity to Blog ...
saved by 5 other

tags semantic tree

- [+]written_communication
- [+]instrumentality
- [+]person (9)
- [+]activity
- [+]message (1)
- [+]content (6)
 - [+]knowledge_domain
 - [+]discipline
 - [+]science (113)
 - politics (184)
 - [+]economics (47)
 - finance (6)
 - psychology (34)
 - [+]physics
 - electronics (22)
 - optics (1)
 - astronomy (1)
 - [+]biology (10)
 - government (21)
 - math (15)
 - [+]geography (13)
 - [+]mathematics (10)
 - [+]anthropology (7)
 - maths (6)
 - ip (6)
 - medicine (5)
 - [+]linguistics (2)
 - sociology (3)
 - taxonomy (3)
 - [+]chemistry (1)
 - geology (1)
 - nlp (1)
 - technology (414)
 - [+]engineering
 - [+]architecture (61)
 - arts (20)
 - [+]theology (7)
 - [+]literary_study
 - communications (6)
 - [+]philosophy (1)
 - trivium (4)
 - english (2)
 - history (1)
 - symbology (1)
 - study (11)
 - [+]idea (147)
 - [+]belief (1)

related tags

- design
- web2.0
- news
- blogs
- art
- technology
- sex
- webdesign
- inspiration
- music
- web

active users

- BazookaDance
- birdphone
- unfoldingrose
- jrhyen
- pensar_custa
- miguelyn
- jameskoole
- IzuXIII
- sacrifice94
- rightmindx
- luochao1987
- yark
- chengshan
- lokidesign
- futher
- eckartwalther
- joetomczak

Figure 5.4: A screenshot from the del.icio.us page for tag “blog”, where the inner sidebar shows an expandable hierarchy of related tags, provided by our application.

by the system. These limitations could be addressed by resting on some domain ontologies to integrate WordNet and on Wikipedia for reconstructing slang forms to more conventional ones (for example, Wikipedia recognizes “nyc” as an alternative form for “New York City”, while WordNet does not).

In many cases synonyms or just different ways of spelling a word happen to be close to each other and easily recognizable in the tree provided by the new sidebar: the semantic hierarchy helps to face the problem of the synonym control to which a folksonomy is naturally prone.

As a last consideration we want to mention the problem of gaming. It is not unusual in *del.icio.us* to see the related tags sidebar entirely mucked up by spam, as we found in some of our examples. Gamers can trick *del.icio.us* to gain a good position for the tags they want to show and, as there are just a dozen tags suggested, the whole sidebar can easily be compromised. In the new sidebar the problem is embanked: as a much higher number of tags is shown, the presence of some spam tags does not make the whole suggestion system unuseful; however, the order of branches could be gamed.

5.1.6 Conclusions

We have proposed a new approach to integrate the navigation interface of a folksonomy adding explicit semantics provided by an ontology; we have developed a tool that uses WordNet to build a semantic hierarchy that helps users navigate and find related resources in *del.icio.us*.

We have shown that in this way it is possible to combine some advantages of the traditional top down approach to classification with the ones of the collaborative paradigm that is emerging on the Web, providing richer possibilities of searching and browsing, and dealing with some of the limitations to which folksonomies are prone, such as lack of recall, synonym control and gaming.

Our application is actually just a prototype and can be improved in several directions. The algorithm for the tree compression is one of the most delicate issues and could be improved by making it dynamic also for higher levels of the hierarchy, instead of just eliminating words contained in a black list.

Many improvements might be reached in tag recognition by using local wordnets in different languages and domain ontologies for specific terms.

As future work, it would be also interesting to use the results of tag disambiguation, performed by our application, to filter resources and not only tags; in this way it might be possible, for example, to show, among

the *del.icio.us* bookmarks tagged as “turkey”, only the ones that have been individuated as related to the geographical acceptance.

5.2 Improving Search and Navigation by Combining Ontologies and Social Tags

The Semantic Web is the “high road” toward a better exploitation of the vast amount of heterogeneous data available on the web. The overall goal is to mediate the access to existing sources, by means of formalized, shared, and explicit representation of the data semantics through ontologies, and to deliver value added interactions. This “high road”, appreciated in the academic environments, requires high switching costs and a wide distributed and coordinated effort, which is hard to achieve in practice. On the other hand, the recent phenomenon of the Social Web and in particular of tag-based systems represents a more practical and viable “low road” toward a better fruition of the web. The goal of the *TagOnto* system is to bridge the two roads, by automatically mapping tag-based systems with the more structured world of ontologies. The main contribution of our approach is to enhance the user experience by providing features typical of the “high road” while requiring only limited commitment, typical of the “low road”, from users and content providers. The system exploits a rich set of heuristics, ranging from simple string-distance measures to web-based tag disambiguation techniques, to discover correspondences between tags and concepts of domain ontologies. Therefore, the unstructured and uncontrolled nature of the *folksonomies*—as often the social tagging systems are named—is balanced by the formal rigor of the ontology-based component of our system. *TagOnto* enriches the user browsing experience by enhancing navigation and tag-based search with ontology-based search capability, which allows to disambiguate tags and to focus the user attention. The system platform is available for download and testable as an on-line demo². Both in the demo and here we use the simple and well-known Wine ontology³ as a running example. To show system extensibility we integrate in this example not only the standard tag engines such as *del.icio.us*, but also the wine community *Vinorati*.

²The on-line demo can be reached from: <http://kid.dei.polimi.it/tagonto>.

³Available at: <http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine>.

5.2.1 Related work

Our work is strongly based on the use of ontologies: on the one hand, it relies on domain ontologies to enhance the expressive power and, more generally, the usefulness of tags by mapping them with ontology concepts; on the other hand, it relies on an ontology to formally describe these mappings. For this reason, works which are related to ours can either involve domain ontologies, used to describe the contents (ie the data present inside the system) or sytem ontologies, used to describe the tag systems themselves.

SOBOLEO (SOcial BOokmarking and Lightweight Engineering of Ontologies, [165]) is a tool which allows for tagging resources in the Web using ontology concepts and interacting with the ontology, modifying concept labels and relations. It has an approach which could be defined dual to ours, as it uses ontology concepts as tags while we start from tags to find their matching concepts. [37] suggests an integrated approach to build ontologies from folksonomies, combining different resources and techniques like statistical analysis, online lexical resources, Semantic Web resources, ontology mapping, and functionalities to help communities achieve and maintain consensus [5] present an approach to enrich the tag space with semantic relations by “harvesting the Semantic Web”. This approach makes use of a tool [124] that allows for ontology mapping by dynamically locating and using relevant background knowledge. [36] addresses the problem of translating a folksonomy into a lightweight ontology in a corporate environment: a 6-step approach is described, including techniques such as the Levenshtein metric, co-occurrence, conditional probability, transitive reduction and visualization. [114] uses the SIOC ontology in order to represent connections between tags and concepts from a domain ontology. [87] maps tags from del.icio.us with concepts from WordNet, and uses this mapping to provide an alternative interface for browsing tags: instead of a flat tag-space or a tag cloud, users have access to a hierarchy of tags that allow them to better find tags that are related to a chosen one.

Gruber [60, 59] models the act of tagging as a quadruple (resource, tag, user, source/context) or a quintuple with a polarity argument, allowing to bind tagging data according to one particular system. Thus, tags from different systems can coexist in this model and it is possible to specify relations between them, allowing for a much higher interoperability between tag-based systems. [107] defines a tag ontology with three main concepts such as Tagger, Tagging and Tag that are used to describe the tagging activity. This ontology also provides relations such as related-

Tag or equivalentTag to define relationships between tags. [77] presents SCOT, an ontology for sharing and reusing tag data and representing social relations among individuals. The ontology is linked to SIOC, FOAF and SKOS to link information respectively to resources, people and tags. The Tag class can be used not only to represent the concept of a tag, but also to describe its statistical and linguistic properties. [41] proposes a method to model folksonomies using ontologies. The model consists of an OWL ontology, able to define not only the main participants in the tagging activity (like, for instance, User, Tag and Resource), but also more complex relations that describe tag variations (like hasAltLabel or hasHiddenLabel).

5.2.2 Project overview

TagOnto is a *folksonomy aggregator* that offers services to relate, navigate and combine results of different tag-based systems. The key features of the system are: *a tag-based search engine*, mashing up several folksonomies to retrieve resources (bookmarks, images and videos); *an ontology-based query refinement*, exploiting a domain ontology, co-occurrence of tags and disambiguation techniques to filter prior results; and *an ontology-based navigation interface*, allowing the user to retrieve further results by graphical navigation of the ontology concepts. The above features provide two orthogonal and complementary ways, typical respectively of social and semantic web, to navigate the search results: associated-tag and ontology-navigation. The ontology is used as a com-

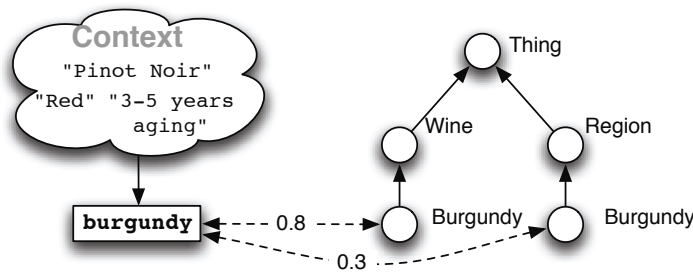


Figure 5.5: An example of tag to ontology matching.

mon vocabulary and bridges the various folksonomies integrated in the system as a global schema of a federated database; the system provides facilities to efficiently load the desired ontology before starting a web search. The typical user interaction is the following: (i) the user searches

for a tag, e.g., **burgundy** (see Figure 5.5), (ii) navigates the concept of the associated ontology to refine the query, e.g., by selecting *burgundy* as a *wine* instead of as a *region*, or (iii) makes the query more general by navigating on more abstract concepts in the ontology. These actions are intuitively supported by the AJAX interface discussed in Section 5.2.4.

The associations between tags and ontology concepts are automatically discovered by the system, but also added, improved and maintained collaboratively. The automatic discovery of associations between folksonomies and domain ontology, represented as dashed lines in Figure 5.5, is based on a set of matching algorithms computing similarities. Disambiguation heuristics are then used in order to debug multiple associations between tags and concepts in the ontology.

Folksonomies are accessed by using dedicated wrappers exploiting three main methods to retrieve the needed resources: (i) Web2.0 APIs, (ii) RSS feeds, and (iii) Page scraping. The first approach, by relying on existing APIs offered by Web2.0-enabled websites, is our preferred one. In the second approach, the source of information is an RSS feed parsed and processed by a dedicated wrapper. The last technique is used when no other solutions are available; *TagOnto* uses page scraping to retrieve the needed information by making extensive use of regular expressions over the webpages to obtain tags and resources associated with them.

5.2.3 Matching and disambiguation

As sketched previously, one of the main problems in *TagOnto* is how to match a tag to a concept in the ontology. Given a tag and a reference domain ontology, the *matching process* (i) searches the ontology for *named concepts* whose name matches the tag, and (ii) looks for *related terms* which may refine the query for a better search. Moreover, (iii) a disambiguation process is often needed to reduce the noise produced by the collaborative tagging. Once the association has been created, the matched concepts are associated to each resource tagged by the corresponding tags. More precisely, given the set \mathcal{T} of all the available tags and the set \mathcal{C} of all the named concepts defined in a specific ontology, the matching is defined as a relation $M \subseteq \mathcal{T} \times \mathcal{C}$. The relation M allows multiple associations between tags and concepts. Figure 5.5 shows an example of such ambiguity: the term *Burgundy* might be referred either to the wine with that specific appellation or the region of France where that particular wine is produced. To distinguish the two different word acceptations, *TagOnto* associates to each matching a similarity degree by introducing the function $s : \mathcal{T} \times \mathcal{C} \rightarrow [0, 1]$.

To establish the matchings and to compute the similarity degree, *TagOnto* relies on the set of matching algorithms shown in Table 5.1. The matching algorithms can be classified on the basis of their effect on the set of matchings, in particular we distinguish between *generators* which generate new matchings starting from a tag and previous matchings and *filters* which choose the best candidates from a set of matchings. Another classification considers the metrics used to compute the matching degrees; we can distinguish between *language-based matching* which uses only morphological and lexical information such as string-distance metrics to compute the similarity and *semantic matching* which uses semantic and background knowledge to create new matchings. Notice that the matching problem has been extensively studied for ontologies [46] and many different classifications are present in the literature. In our context, the main difference is the absence of structure in folksonomies which does not allow an exploitation of structural similarities between the terms in the folksonomy and those in the ontology. Language-based generators use well known string-distance metrics, such as Jaccard similarity and Levenshtein distance. On the contrary, an example of language-based filter is the *Google Noise* algorithm, which suggests possible corrections for misspelled keyword by using the “did you mean” feature of Google. In a similar way, a semantic generator is the *WordNet Similarity* algorithm which computes the Leacock-Chodorow [91] distance metric in WordNet between the term used in the tag and the concepts of the ontology. In *TagOnto* we use the implementation of the algorithm which is used in X-SOM (eXtensible Smart Ontology Mapper) [34] since it offers some extensions to handle compound words, acronyms and language conventions which are quite common in both folksonomies and ontologies. Since *TagOnto* is supposed to work online and with a fast response time, the class of syntactic filters includes some rather simple algorithms to select

	Language-based	Semantic
Generators	Levenshtein Distance Jaccard Similarity Google Noise Correction Concept Instances Similarity	Wordnet Similarity
Filters	Max Threshold	Graph Connectivity Neighbors Google Search

Table 5.1: Some matching heuristics.

the best candidate matchings for a given tag, some examples are the *threshold filter*, which selects only matchings having a similarity degree greater than a specified threshold, and the *max filter* which selects the k matchings with the highest similarity degree. On the contrary, semantic filters are extremely useful in the disambiguation process since they alter the similarity degree of a matching by analyzing the concepts correlated to a tag using the structural information of the ontology. The disambiguation process is composed of two steps: (i) given a tag, the most frequent co-occurring tags are retrieved in order to specify its meaning (i.e., its *context*), and (ii) the ontology is analyzed in order to identify the concept which the closest meaning to the tag in that particular context.

The first process is carried out by the *Google filter* algorithm which retrieves the co-occurrent tags by issuing a query into Google and analyzing the first result. The second step, called *Neighbors filtering* leverages a common functionality of tag-based systems: the *tag-clouds*, which associate to each tag another set of tags whose meaning is correlated to the original one. After this information has been retrieved, *TagOnto* updates the similarity degrees of the matchings. As an example (see Figure 5.6) suppose we have the tag **Burgundy** with multiple matching concepts in the ontology (called *root concepts*); in first place *TagOnto* matches the co-occurrent tags obtained from tag clouds with the concepts of the ontology. The second step leverages the structure of the ontology by counting, for each matching, the number of links which connect matched concepts with each root concept, producing a vector of connectivity degrees v . The last step modifies the matching degrees of the root concepts according to the connectivity degrees computed in the previous step. For each matching i , *TagOnto* computes an offset measure $\varepsilon_i = \frac{D[i]}{MAX(v)}$ which is compared with the average connectivity $AVG(v)$; if $\varepsilon_i < AVG(v)$ then the new matching degree is decreased by a factor $\alpha \cdot \varepsilon_i$ where $\alpha \in [0, 1]$ is a configurable discount factor (currently set to 0.2 after the test phase); in the same way, the matching degree is increased if $\varepsilon_i > AVG(v)$. If the updated matching degree exceeds the values in $[0, 1]$ the value is truncated to fit the range.

How these heuristics are combined depends on the selected matching strategy. We provide two different strategies: a *greedy strategy* which first invokes the syntactic and semantic generators and then applies the syntactic filters, and the *standard strategy* which invokes the greedy strategy and then disambiguates the results by invoking semantic filters. Semantic disambiguation is not always necessary, especially when tagging is done within small communities of practice which share a common vo-

cabulary without many ambiguities. In this cases the greedy strategy can provide results comparable with the standard one in a shorter time. Whenever, instead, the user base is large (such as in many Web-based services) and tags are not restricted to one specific domain, the higher mapping quality of the standard strategy compensates the higher processing time.

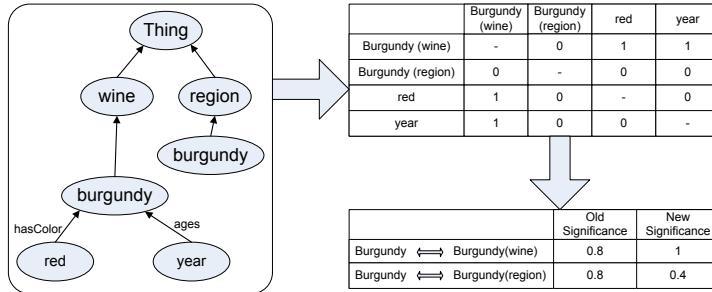


Figure 5.6: An example of the disambiguation process.

5.2.4 Project architecture

The overall architecture of *TagOnto* is logically divided into three different components: a tag-based search engine extensible with plugins, a heuristic matching discovery engine and a web-based user interface.

TagontoNET: TagontoNET provides core search engine functionalities and takes care of the integration of the results coming from folksonomies. The plugin-based architecture decouples the interaction between tag providers and *TagOnto*'s business logic. The system currently implements seven plugins to interact with some of the most popular tag-enabled websites such as Flickr, YouTube, del.icio.us, and Zvents. TagontoNET offers two main functionalities: tag-based resource retrieval and neighboring tag computation (needed by TagontoLib as discussed in the following). The results are delivered through a RESTful [52] web service, implemented in PHP, to further decouple this functionality, which might be used independently with the ontology-based portion of *TagOnto*.

TagontoLib: a Java library implementing the core matching functionalities of the system. The matching engine developed in Java implements the matching heuristics and strategies described in Section 5.2.3.

To overcome performance limitations an effective caching technique has been built, maintaining recent matching tags and ontological concepts. As for the previous component, much attention has been devoted to the modularization of the tool. The communications between this library and the interface has been, in fact, based on a REST-like communication paradigm [52].

TagOnto Web Interface: one of the distinguishing features of *TagOnto* is its web Interface which offers to the user the support of the Ontology within a comprehensive view of the results collected from a number of different tag engines. Users can import new ontologies into the system just by entering their URIs into a special page. The interface is then divided into two horizontal portions: the upper one reports the search results, the lower one is dedicated to the ontology representation and navigation. Each user query triggers searches in both the ontology and the tag-engines. The results from these two sources are respectively shown in the upper and in the lower part of the page. This provides a unified view of the ontological meaning of a tag and the available resources (tagged with that keyword). It is possible to exploit the support of the ontology to improve the search by navigating the ontology and thus triggering a query refinement procedure that will retrieve more specific resources based on the associated tags.

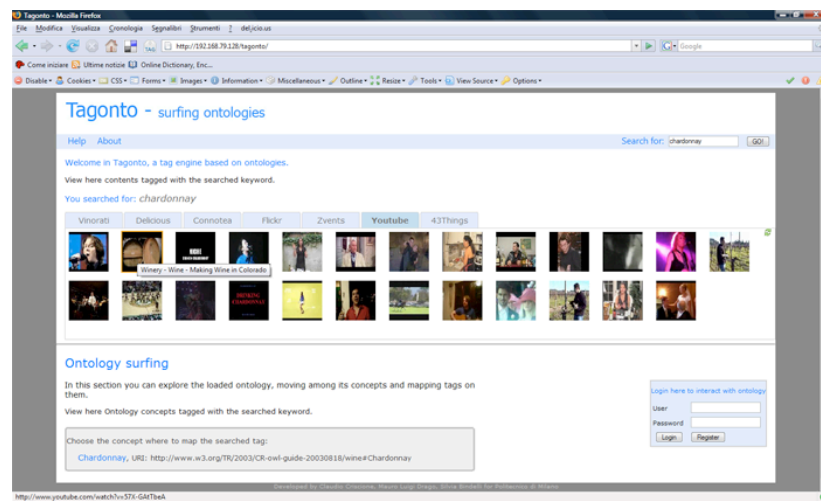


Figure 5.7: The basic *TagOnto* web interface.

The interface provides several tabs reporting the results obtained by

searching each folksonomy. Textual results are presented in a “Google-like” way, while for picture results (e.g., Flickr resources) a thumbnail of the matching image is shown. The lower part of the page is dedicated to the presentation of the ontological concepts associated to the search. When a keyword is typed in the search field, a so-called “disambiguation box” appears in this area, to let the user choose among the concepts *TagOnto* computes as best matches. Once a concept has been chosen, previously mapped tags and resources are shown. The system also provides a box-based representation of other concepts related to the selected one, allowing an ontology-based navigation. During this navigation process the co-occurrence of tags is used to provide feedback to the user and to suggest further directions for the exploration.

5.2.5 System evaluation

We measure system performance in terms of efficiency of the analysis and matching process, while an extensive usability study is part of our research agenda. To measure system efficiency, we stress test *TagOnto* when performing the two most expensive tasks occurring at run-time: (i) the time needed by *TagOnto* to analyze a new ontology to be deployed, and (ii) the time needed to automatically generate matchings. Figure 5.8 shows outcomes of our analysis. The time needed to perform an ontology analysis depends mostly on the number of concepts and properties declared in the ontology, with polynomial complexity as shown in Figure 5.8(a) while, with fixed concepts and properties (i.e., fixed schema), the number of instances declared in the ontology influences the execution time linearly as shown in Figure 5.8(b). Figure 5.8(c) shows the distribution of response time obtained by issuing 344 tag-queries (i.e., queries composed by a single term) taken from a set of terms referring to the wine domain.

5.2.6 Conclusions

In this section we presented *TagOnto*, a *folksonomy aggregator*, combining the collaborative nature of Web2.0 with the semantic features provided by ontologies, to improve the user experience in searching and browsing the web. The design of the system has been such that very limited overhead is imposed to users and content providers to enable these new features. *TagOnto* key components are a multi-folksonomy, tag-based search engine, and an ontology-based query refinement facility, which exploits a domain ontology to filter results and to focus

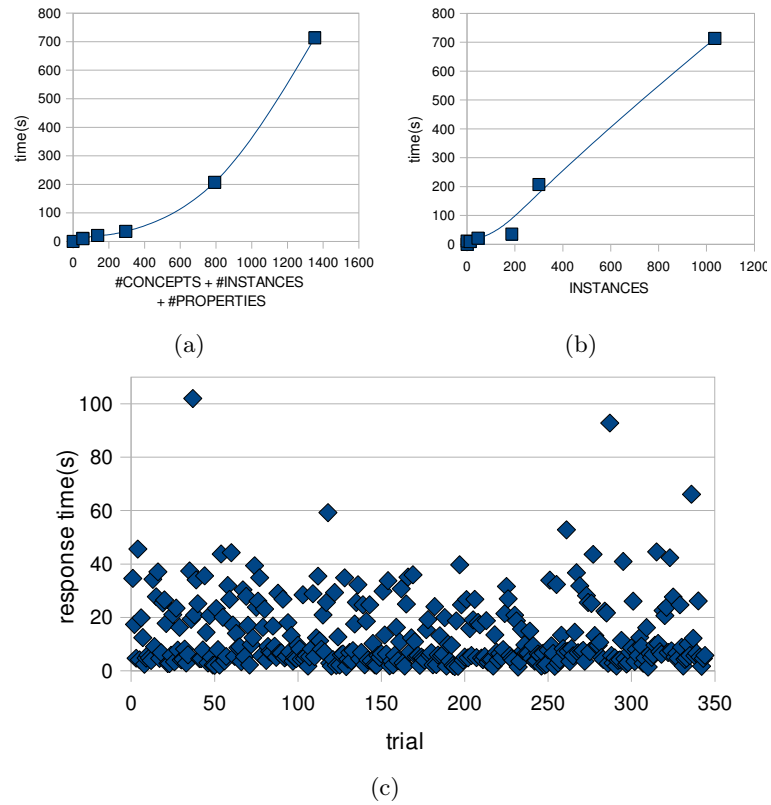


Figure 5.8: Tagonto performance.

users' attention. In the best Web2.0 tradition, these features are delivered through an intuitive and reactive AJAX interface. The system is released and demonstrated online, and has been successfully tested on several domains. Nonetheless, we consider *TagOnto* a starting point for further developments and we plan to devote more work on three key aspects: usability, performance and extensibility.

6 Linking Open Data

6.1 An IMAP Plugin for SquirrelRDF

The Semantic Web is a new Web paradigm that aims to make information accessible to both humans and machines [17], using standard formats for data and making information available in a formal and structured way. This means that to make it work inside the current Web it is necessary, on the one hand, to publish new information so it is *meaningful* for machines, and on the other hand to convert old data so they are available in new, more standard and structured formats.

While both of these approaches are currently being studied by the Semantic Web community, the latter is probably the one which seems more challenging from a technical point of view. And while the task might be difficult for free, unconstrained text, it becomes much easier for information already published in a structured way. This is, fortunately, the case of many standard file formats and protocols.

This kind of conversion can be usually done in two ways: the first one is a *batch* conversion, and is run once (or periodically) on the whole data source, while the second one is an *online* conversion, which is run on the fly on the single pieces of information which need to be accessed. To be more precise, tools of this last kind allow users to query the knowledge base as if it already was described in the destination format, and translate only the results of the query while they are given as an answer to the user.

Each of the two approaches has its pros and cons: batch conversion is better suited if the data source is not going to be updated and whenever there is the need to work offline (that is, disconnected from the original source of data); online conversion requires a live connection to the data source, however the information it returns is always current and consistent with the data source.

In this section we describe a conversion tool which allows IMAP mailboxes to be queried with SPARQL, as if they originally contained information in RDF format. The tool has been developed as a plugin of SquirrelRDF, an application which is part of the Jena Semantic Web Framework. As an IMAP mailbox is a source of data that is often up-

dated, we perform an online conversion, providing users results which are always current.

6.1.1 Related work

Since the advent of RDF[89] there have been many efforts to extract and convert existing information to this format. The World Wide Web consortium has set up wiki pages¹ to keep track of them through a list of links. The MIT SIMILE project has developed many offline conversion tools, calling them RDFizers², which range from e-mail to BibTex, JPEG metadata, and XML [29]. They also provide a Web service, called Babel³, which allows for the conversion between different formats.

The SIOC project [26] aims at interconnecting different online communities (such as the ones which gather around forums and weblogs) through a common ontology and a collection of tools (called *exporters*) that convert published information into a common format. These tools can work not only on Web-based sources like RSS feeds, blogs and forums, but also on email based ones like mailing lists (see for instance the SWAML⁴ research project [54]).

The D2R project [19, 21] uses a declarative language to describe mappings between relational database schemata and OWL/RDFS ontologies. The mappings can then be used to export data from a relational database to RDF (as a batch conversion tool) or to access the content of non-RDF databases as an online tool, using Semantic Web query languages like SPARQL⁵.

The Gnowsis Email project⁶ provides an adapter to extract RDF information from emails. Thanks to this adapter it is possible to transform any IMAP email object (such as a store, a folder or a message) into a standard RDF model, or extract attachments from an imap message. In a paper [126] about the Gnowsis Adapter Framework (on which the Email project is built) the authors provide an interesting classification of *adapter* tools. According to the paper, adapters are software tools that can, on request, extract data from existing structured data sources and represent them as RDF. They can follow three basic approaches:

¹<http://esw.w3.org/topic/ConverterToRdf>

http://www.w3.org/2005/Incubator/mmsem/wiki/Tools_and_Resources.

²<http://simile.mit.edu/wiki/RDFizers>.

³<http://simile.mit.edu/babel>.

⁴<http://swaml.berlios.de>.

⁵<http://sites.wiwiiss.fu-berlin.de/suhl/bizer/d2rmap/D2Rmap.htm>

<http://sites.wiwiiss.fu-berlin.de/suhl/bizer/D2RQ>.

⁶http://www.gnowsis.org/Projects/gnowsis_email.

- *Graph and query adapters*, which implement the interface of an RDF graph or a query language like RDQL, SPARQL or TRIPLE
- *Concise Bounded Description* adapters, that can return a small subgraph that describes exactly one resource in detail
- *File extractors*, that read files, parse them and return some meta-data that was extracted from the data stream

According to this classification, SquirrelRDF [140] is a Graph and Query adapter, as it allows non-RDF data stores to be queried using SPARQL. It currently includes support for relational databases (via JDBC) and LDAP servers (via JNDI). It provides an ARQ QueryEngine (for Java access), a command line tool, and a servlet for SPARQL http access. For instance, running the command line tool with the following query (directed to HP LDAP server):

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix hp: <http://jena.hpl.hp.com/schemas/hpcorp#>
SELECT ?mbox ?manager_name
WHERE
{
    ?person foaf:name "Davide Eynard" .
    ?person foaf:mbox ?mbox .
    ?person hp:manager ?manager .
    ?manager foaf:name ?manager_name .
}
```

returns the following result:

```
-----
| mbox                                | manager_name    |
=====
| <mailto:davide.eynard@hp.com> | "Craig Sayers" |
-----
```

The servlet tool, instead, generates an XML file containing SPARQL query results and then uses an XSLT script to format them into XHTML. The output is shown in Figure 6.1, while the XML code looks like this:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="xsl/result2-to-html.xsl"?>
```

SPARQL Query Results to XHTML (XSLT)**Variable Bindings Result**

Ordered: false

Distinct: false

mbox	manager_name
URI mailto:davide.eynard@hp.com	Craig Sayers

Figure 6.1: The HTML output of a SPARQL query, as returned by SquirrelRDF servlet.

```

<sparql
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://www.w3.org/2005/sparql-results#" >
<head>
  <variable name="mbox"/>
  <variable name="manager_name"/>
</head>
<results ordered="false" distinct="false">
  <result>
    <binding name="mbox">
      <uri>mailto:davide.eynard@hp.com</uri>
    </binding>
    <binding name="manager_name">
      <literal>Craig Sayers</literal>
    </binding>
  </result>
</results>
</sparql>

```

The advantages of such a tool are quite clear: information looks like RDF and query results can be easily integrated inside other applications. As an example, in [164] SquirrelRDF is used to solve the real-life problem of automatically extracting complex information from an LDAP directory. In [163] the technical details about this project have been described, together with a tutorial about setting the system up and expanding it to provide support to different types of SPARQL queries and output formats.

6.1.2 Project overview

Of the many different formats which were already available on the Internet, we decided to focus on e-mails because they provide useful information not only about their actual content, but also about the people who wrote them, the relations between these people (ie. who wrote to whom) and their dynamics (ie. who replied about what).

Current adapters for emails can be roughly divided in two categories: those which work on information saved on the client side (for instance on mailboxes in *mbx* format, or some application-dependant tools), and those which work by downloading data from servers (connecting to IMAP or POP3 servers).

We decided to work on information stored on the server and not on the client side, because we did not want to stick with some specific application or format. Also, as we wanted to access current information, we chose to build a query adapter for IMAP servers. Gnowsis Email project looks very similar to the one we had in mind, but it apparently does not allow to directly query the IMAP server with SPARQL. With this premises, SquirrelRDF instead seemed to satisfy all our prerequisites, so we chose it as a starting point to develop our adapter.

The Data Model

To write our IMAP plugin we chose to first study how the existing ones (RDB and LDAP) worked. In particular, the mapping used by the LDAP plugin is very compact and efficient and allows to simultaneously specify search constraints and extract the right attributes from the results returned by the LDAP server.

Unfortunately, this kind of mapping is not usable for the IMAP plugin. In fact, there is a huge asymmetry between the way messages are searched and how fields are extracted from search results, that requires us to describe the two operations in different ways. Moreover, we wanted to develop something flexible enough to allow programmers to create new plugins in an easy way, given similar kinds of problems. So we thought about a different mapping model (albeit inspired by the LDAP one), which maps properties specified in the SPARQL query with *methods* that are called by our application.

The details of our mapping model can be found in Figure 6.2. Every `imap:Map` can have one or more *server profiles* and one or more *property mappings*. The first ones are used to describe the properties of the different servers the application can connect to: each server has an alias

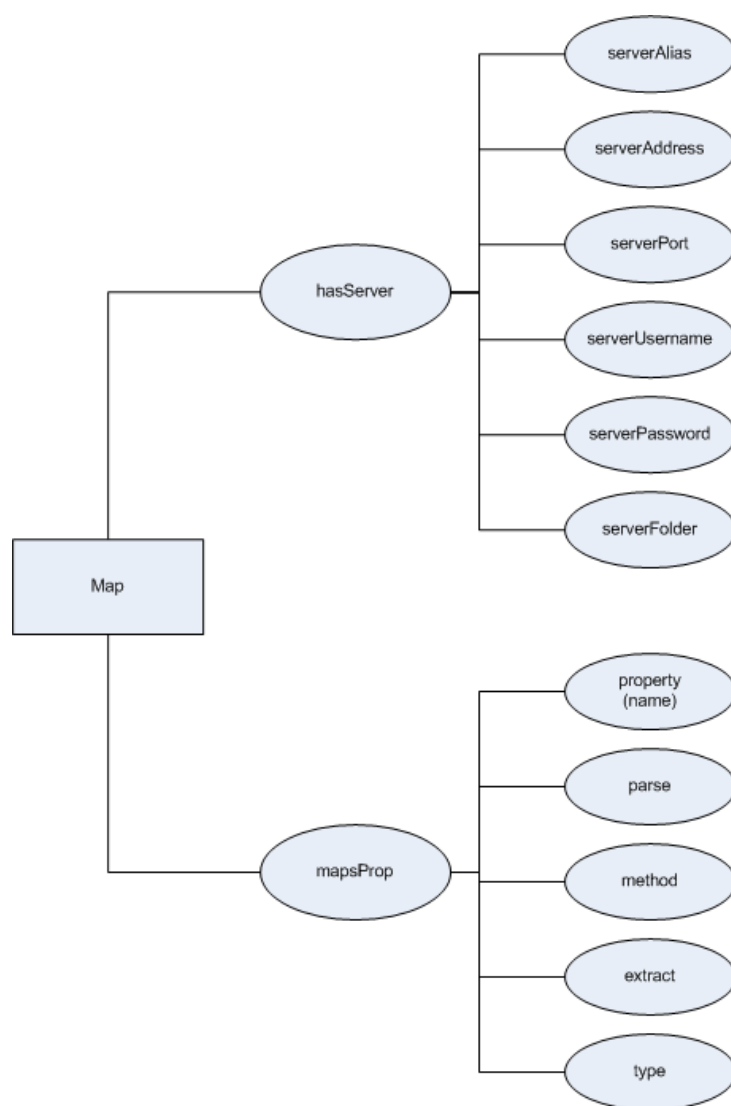


Figure 6.2: A description of the IMAP Map and its properties.

Field name	Predicate name	SearchTerm (JavaMail)	Extract Method (JavaMail)
Body	email:body	BodyTerm	getContent, getBodyPart
Subject	email:subject	SubjectTerm	getSubject
From	email:from	FromStringTerm	getFrom
To	email:to	RecipientStringTerm	getRecipients
Cc	email:cc	RecipientStringTerm	getRecipients
Bcc	email:bcc	RecipientStringTerm	getRecipients
MessageID	email:messageID	MessageIDTerm	getMessageNumber
headers	email:header	HeaderTerm	getAllHeaderLines
flags	email:flag*	FlagTerm	getFlags
Date	email:date	ReceivedDateTerm	getReceivedDate
Sent	email:sdate	SentDateTerm	getSentDate

Figure 6.3: Mapping between email field names, predicates from our email ontology, JavaMail SearchTerm subclasses and JavaMail extract methods.

(which is used inside SPARQL queries to refer to it), an address and a port to connect to, a username, a password, and a default folder to connect to if no folder is specified inside the query.

Property mappings are used to match properties with methods: every mapping contains the name of the property which has to be mapped; a property type can be specified, so it is possible to run specific actions over it; also, every property can be matched with three methods (through `imap:method`, `imap:extract`, and `imap:parse`) which are called inside the application (more details on it in Section 6.1.4).

The Email Ontology

As described above, our mapping describes connections between predicates used inside SPARQL queries and methods. These predicates come from an ontology which describes every single piece of information we can query: in our specific case, the ontology describes emails and their different fields.

In the first two columns of Table 6.3, the field names for a generic email and the predicate names we chose for our ontology are shown. In the third column **SearchTerm** classes are specified: these are the classes used by JavaMail (see Section 6.1.4, “The JavaMail API”) to specify the search terms inside an IMAP query. In the fourth column, extract methods are shown: these are all methods provided by the JavaMail **Message** object, which allow to extract information from the messages returned by the IMAP server.

Looking at the table, a direct mapping between the predicates and the classes/methods from JavaMail might seem the most intuitive and the easiest one. However, we decided to put one more layer of abstraction in the process, creating mappings between predicates and methods which use these objects to search and extract information. One of the main

advantages of this approach is the possibility to create many different methods which get the same pieces of information, but then convert them in different ways.

For instance, the `getFrom` method is able to extract the sender from an email message. However, the `From` field is built up of a name and an email address: in some cases we might be interested in both, while in others we might want to extract only one of them. Using our approach it is possible to create three different methods (ie. `extractFrom`, `extractFromName`, and `extractFromAddress`) and match them with three different properties; also, one might just keep one property and update the mappings depending on what kind of information a particular query has to return.

Actually, mappings can be changed quite easily: they are saved in RDF format inside a configuration file and changing them is just a matter of editing this text file. Even creating a new mapping from scratch is rather easy. Once a predicate name is chosen, its mapping can be described like in the following example:

```
imap:mapsProp [ imap:property email:body ;
                 imap:method  "searchBody" ;
                 imap:extract "extractBody" ;
                 a  imap:ExactStringProperty ;
               ] ;
```

In this case, for instance, we chose to map the `email:body` property with two different methods: `searchBody` to search the IMAP server for a particular body, and `extractBody` to extract body information from the results. Also, this property requires the specified search term to be matched not as a substring, but as an exact string.

As it appears in the table, the email ontology is very simple and flat and it does not reuse already existing ontologies. However, as it is described only through mappings inside one text file, it is very easy to change it and make it more complex and compatible with other ontologies. For instance, it is possible to map the subject with a `dc:subject`, the sender with a `dc:creator`, and the date with a `dcterms:dateSubmitted`.

6.1.3 Project architecture and implementation

The plugin has a structure which is very similar to the other Squirrel-RDF components: an RDF configuration file describes the mappings and all the parameters needed to connect to the server(s); an RDF schema

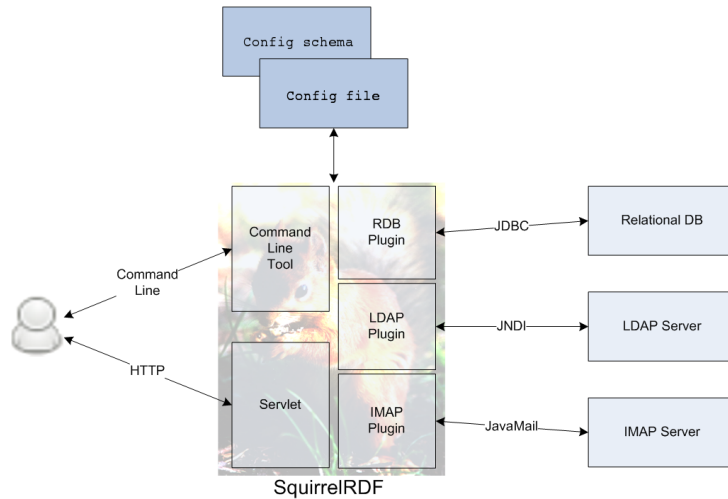


Figure 6.4: SquirrelRDF structure.

describes how the configuration file can be built; different classes provide different ways to access the data (ARQ QueryEngine, command line, servlet). In this section we describe the main choices we made during the design phase and how we implemented them, highlighting the differences between our plugin and the already existing ones.

The structure of SquirrelRDF is shown in Figure 6.4. From a user perspective, there are two ways to use the application: the command line tool and the servlet. Both of them access a configuration file whose name can be specified as a parameter during startup: this file contains the mappings and the parameters needed for the plugin to run and its vocabulary is defined inside another RDF file containing its schema .

As the configuration defines the type of the mapping, once it is loaded SquirrelRDF can choose which plugin has to be used. Then it parses the SPARQL queries according to the mappings, retrieving information from the servers when needed and binding them with the variables used inside the queries. Depending on the plugin, SquirrelRDF uses different protocols and APIs to connect to the servers; also, the methods used to extract information from search results change from one plugin to another. However, all these differences are invisible to the user, who can just query the data sources as if they were RDF graphs.

Multiserver queries and preparsing

One of the main purposes of describing email messages with RDF is the possibility of integrating their information with different and heterogeneous data sources. However, of course, we also wanted to provide ways to integrate the same information with other *homogeneous* sources, that is other IMAP accounts, folders, and servers.

For example, one user might have different email accounts from different providers, and want to find a particular message without remembering where it is saved. In another use case, one might want to be able to see at a glance all his new emails, without having to care about where they are stored. A researcher might need to search in different mailing lists (folders) if people are writing about the same subjects, or if the same writers appear in different communities, and so on.

Allowing users to specify servers and folders inside a SPARQL query means specifying new properties for messages: given a message as a subject, two new predicates `imap:server` and `imap:folder` have been created. Two main problems arise from this feature extension: the first one is related to where and how connection data (server IP, port, login and password) have to be specified; the second is related to how we want to manage predicates like these inside a SPARQL query.

We decided to associate connection information with a server alias, so that users just have to specify servers and folders as string literals inside their own queries. The alias is saved inside the configuration or, in the case of the servlet application, can be specified at runtime.

The management of “special” predicates or triples is done with a `preParseQuery` method. This method is called before the actual parsing inside the `ImapSubQuery` class: it checks all the triples in the subquery and if one of them has a property whose type is `ParseProperty` then it calls (using reflection) the matching method from the `ParseMethod` class. If the method returns true then the triple has to be deleted from the query, otherwise it can be kept.

For instance, suppose we have the following lines in the configuration file:

```
imap:hasServer [ a imap:Server ;
                  imap:serverAlias      "server1" ;
                  imap:serverAddress    "localhost" ;
                  imap:serverUsername   "user01" ;
                  imap:serverPassword   "pass01" ;
                  imap:serverFolder     "INBOX" ; ] ;
```

```
imap:mapsProp  [ imap:property email:server ;
                  imap:parse "setServerAlias" ;
                  a imap:ParseProperty ; ] ;
```

and suppose the user enters the following query:

```
select *
where{
    ?x email:subject      ?subj1 .
    ?x email:server       "server1" .
}
```

When the second triple is evaluated this is what happens: given that `email:server` is a `ParseProperty`, the `preParseQuery` method will call its matching method (that is `setServerAlias`, from the `ParseMethod` class). This method reads the server alias from the current triple (“server1”) and whenever the program has to connect to an IMAP server it will do it using the parameters specified in the configuration file (that is, it will connect to localhost using “user01” as a login and “pass01” as a password). Finally, as the method returns the boolean value `true`, the triple will be deleted from the query.

SPO queries

An SPO query is a single triple pattern, with optional subject (parameter “s”), predicate (parameter “p”), and object (parameter “o”). This is usually inserted in SPARQL queries as a test pattern, or to get all the triples for the available resources. As the other two SquirrelRDF plugins did not support these kinds of queries, we wanted to work on this to see if we could provide some useful contribution to the project we borrowed so much from.

The default behavior of SquirrelRDF (at least for its LDAP plugin) is to build a mapping between the variables that appear inside the SPARQL query and the values that are extracted from the data source.

The mapping is done inside the `ImapSubQuery` class using a list of `HashMap` objects. Their keys are `String` objects containing the variable names; their values are `Jena Nodes` (`com.hp.hpl.jena.graph.Node`) containing the pieces of information extracted from email messages. According to this model, the results can be seen as many rows inside a database table, where each binding specifies in which column a particular value has to be saved (Figure 6.5).

x	subject	from	date
nodeID b0	email without attachment	eynardd <eynardd@localhost>	2007-08-28T22:25:25Z
nodeID b1	email with attachment	eynardd <eynardd@localhost>	2007-08-28T22:24:27Z
nodeID b2	another test	eynardd <eynardd@localhost>	2007-08-25T22:21:31Z
nodeID b3	mail with empty body	eynardd <eynardd@localhost>	2007-08-25T22:17:36Z

Figure 6.5: A result from a normal SPARQL query.

s	p	o
nodeID b0	http://davide.eynard.it/rdf/email#flagNew	false
nodeID b0	http://davide.eynard.it/rdf/email#flagRecent	false
nodeID b0	http://davide.eynard.it/rdf/email#subject	Comment on Wikipedia page (Semantic Wikis)
nodeID b0	http://davide.eynard.it/rdf/email#to	eynardd@localhost
nodeID b0	http://davide.eynard.it/rdf/email#date	2007-06-21T23:22:44Z
nodeID b0	http://davide.eynard.it/rdf/email#from	eynardd <eynardd@localhost>
nodeID b0	http://davide.eynard.it/rdf/email#flagDeleted	false
nodeID b0	http://davide.eynard.it/rdf/email#sdate	2007-06-21T23:22:44Z
nodeID b0	http://davide.eynard.it/rdf/email#flagSeen	true
nodeID b0	http://davide.eynard.it/rdf/email#flagDraft	false
nodeID b0	http://davide.eynard.it/rdf/email#flagAnswered	false
nodeID b0	http://davide.eynard.it/rdf/email#flagFlagged	false
nodeID b0	http://davide.eynard.it/rdf/email#messageID	1

Figure 6.6: A result from an SPO query.

When an SPO query is issued, the results cannot fit into rows anymore, but rather in blocks: in our particular case, every email message has many rows describing all its properties one by one and their values. So we decided to translate this with a multiple bindings data structure (Figure 6.6), which can hold all the results returned by every single message (and which, in the simplest case, can just be a block made of one row).

To manage this kind of queries we took advantage of the prepararsing feature: the variable triple is detected in advance, deleted from the query and a flag is set to warn that a special query has been issued. Then, for each message that satisfies the search constraints (potentially all messages), a set of fields is extracted and transformed into bindings. It is possible to change how this set is built just by toggling the `CheckProperty` type inside the configuration file: this allows to limit data transfer and optimize performances.

As an example, suppose we have the following lines in the configuration file:

```
imap:mapsProp [ imap:property email:subject;
                ...
                a imap:CheckProperty ; ];

imap:mapsProp [ imap:property email:from;
                ...
                a imap:CheckProperty ; ];

imap:mapsProp [ imap:property email:date;
                ...
                a imap:CheckProperty ; ];
```

and that all the other properties are not of this type. Then the results of the query will include only these three predicates, when present.

According to this model, the main limit in the implementation of SPO queries is the server providing the information. In case of unconstrained queries, it should return all the elements it contains: while this is feasible with IMAP, it could be impossible with other servers.

6.1.4 Software details

The JavaMail API

As described in [126], an adapter tool might depend on the data source (that is, on its format and the ways to access it), so a preliminary analysis

is needed. Fortunately, email is a mature standard and IMAP protocol is well described by its RFC [33]. Moreover, the JavaMail API⁷ provides an easy way to access an IMAP server which is compliant enough with the protocol, so we chose it as a gateway between our tool and IMAP servers.

The main JavaMail API classes we used are `IMAPStore` and `IMAPFolder`: the first one is used to manage connection (plain or SSL, with `IMAPSSLStore`) and authentication, and to specify which is the current folder; the second one offers all the methods needed to manage messages saved inside a particular folder. Between these, the `search` method proved to be particularly useful for our purpose of building a SPARQL-to-IMAP translation. In fact, even if filtering can be done client-side by the SPARQL interpreter, we decided to take advantage of advanced IMAP search features too: this allows users to pre-filter messages on the server side, lowering the number of bytes they have to download.

The search method requires a `SearchTerm` object as input and returns an array of `Message` objects as an output. `SearchTerms` are objects which follow quite straightly the RFC specifics: there is one for each type of field one might want to search and it is possible to join many of them through `AND` terms. `Messages` are objects which provide all the methods needed to extract the different components of an email message. The third and fourth columns of table 6.3 show the mappings between the different email fields, `SearchTerms` and `Message` methods.

Class structure

The IMAP plugin for SquirrelRDF is composed of eight classes (plus two “main” ones which implement the command line and the servlet tools). A brief description of them follows:

- `ImapMap` class defines the schema of the configuration file. Actually, the RDF schema is defined inside an RDF file, then the class is created automatically by the `schemagen` tool. Once built, it provides all the vocabulary definitions needed to describe the RDF mapping model inside the Java application.
- `ImapQueryEngine` and `ImapSparqlMap` are mostly refactorizations of the original classes used for the LDAP plugin. The first one extends the `ARQ QueryEngine` object, providing access to the query plan elements; the second one analyses them, dividing the queries

⁷<http://java.sun.com/products/javamail>.

in blocks of triples who share the same subject, and calling the `ImapSubQuery` class (see below) to work on these blocks. This grouping by subject comes very useful to us, as we are mostly dealing with email messages and it is much more efficient to get all the information about one message at once.

- `ImapSubQuery` is the heart of the IMAP adapter. It gets “subquery” elements (that is, the groups of triples which share the same subject), builds the IMAP queries, extracts the required fields from downloaded messages, and finally creates and manages the bindings between SPARQL variables and their values.
- `MatchMethod`, `ExtractMethod` and `ParseMethod` are the classes which contain the methods used in the mappings. They are all instantiated inside the `ImapSubQuery` class, and their usage is better described in the following section.
- `CfgManager` is the class used to manage the pieces of information which are common between all the other different classes. Currently, it contains the RDF Model object which describes the mapping and all the methods needed to access it.

Reflection

In our particular case, reflection is used to manage the dynamic calling of methods from some particular classes, allowing users to specify the name of these methods in the configuration file that is loaded at runtime.

The three main uses of these methods are search, extraction and preparsing. The `MatchMethod` class contains all the methods that build up the `SearchTerms` used to search email messages on the IMAP server. The `ExtractMethod` class contains the methods used to extract pieces of information from an email message (such as the subject, the body and so on). The `ParseMethod` class contains methods that are called in a subquery preparsing phase, for instance the ones which set the server or the folder for the current query.

The first step we accomplish is instantiating the class with the right parameters: all of them require at least a `CfgManager` object and the current triple; `ExtractMethod` also requires the current message to extract information from, and `ParseMethod` needs the current `ImapSubQuery` to change some of its parameters. Once an instance of the class is created, it is then possible to call its only public method, called `run`.

The `run` method first finds the name of the right method to call, extracting the predicate from the triple and then matching it with the configuration. Then it gets the method from its name using `getClass().getDeclaredMethod` and invokes it. Every single method has then access to the configuration model, the triple, and the additional parameters, as they had been saved into private attributes when the class was instantiated.

As an example, suppose we have the following lines inside the configuration file:

```
imap:mapsProp [ imap:property email:subject;
                  imap:method "searchSubject" ;
                  imap:extract "extractSubject" ;
                  a imap:ExactStringProperty ; ];

imap:mapsProp [ imap:property email:body ;
                  imap:method "searchBody" ;
                  imap:extract "extractBody" ;
                  a imap:ExactStringProperty ;
] ;
```

and suppose the user enters the following query:

```
select *
where{
    ?x email:subject      "test" .
    ?x email:body         ?body .
}
```

When the query is parsed, predicates are extracted from the triples and the matching search methods are called if the object is not variable or if the variable is bound to some value: in this case, only the subject has been specified so only the `searchSubject` method (from the `MatchMethod` class) is called. Then, when the IMAP server returns the results of the search (in this case, all the messages whose subject *contains* the string “test”), the extract methods matching the specified predicates (`extractSubject` and `extractBody` from the `ExtractMethod` class) are called. Moreover, as both the predicates are `ExactStringProperties` and the subject has been specified, the application will also check that, of all the messages returned by the IMAP search, only the ones whose subject actually *is* “test” will be returned.

The main advantage of this technique is that it is very flexible and allows users to change drastically the behavior of the application with

few and simple changes. For instance, alternative “searchFrom” methods can be built to extract the whole From field, only the address or only the name of the sender; then the user can change the method that has to be called just by updating the matching string in the configuration file. Moreover, methods can be used not only to return a value formatted in a particular way, but also to perform particular actions inside the system (such as modifying the configuration or retrieving information outside of the current triple).

This solution also has some drawbacks: in fact, to avoid managing too many particular cases, during the design phase it is better to define some standards and constraints in method calls, losing part of their flexibility. For instance, we decided to keep the type of the returned values fixed for all the methods in a single class (`SearchTerm` for `MatchMethod`, `String` for `ExtractMethod`, `Boolean` for `ParseMethod`) and we used roughly the same parameters for all of them. Anyway this is not a huge limitation in our case, and we think it might be relaxed in more complex cases, at the cost of more type checks inside the application code.

6.1.5 Tests and evaluations

To test the application, we chose to feed it different families of queries:

- basic ones, which just ask for properties of email messages;
- queries using `OPTIONAL` clauses;
- queries using `FILTER` clauses;
- SPO queries;
- multiserver queries, mixing information between different servers and
- folders or merging it with `UNION` clauses.

The application returns the expected results; however, during the tests we had to face some IMAP limits and found some workarounds for them.

Exact String Search

The search function within IMAP does not match exact strings, but just searches for substrings within email fields. This is particularly limiting because, for instance, when you write the triple

```
?x email:subject "This is a test" .
```

you expect only messages whose subject *is* (and not *contains*) the specified string. A workaround for this problem consists of checking the result messages before extracting information from them: if a particular field matches with a property defined as `ExactStringProperty` inside the configuration file, then it is compared with the result and if the strings do not match exactly the message is dropped before any other evaluation.

Date Search

The date search function provided by IMAP is not precise, that is it does not take into account time but just returns all the messages received or sent on one particular day. Of course, a fix similar to the previous one could be implemented for date fields. However, we found a workaround to this problem : specifying the date constraint both in the query triple and in the FILTER allows to obtain the expected result.

Server lag

Usually, filtering results with just a FILTER clause is not very efficient: this is due to the fact that in this case all the messages are downloaded, then the filter is applied on them. What we would like to have, instead, is a preliminary filter on the IMAP side which would allow us to download only the messages we are interested in. To accomplish this task on message bodies (which are usually the heaviest part of emails to download), we created an ad-hoc predicate that we called `imap:bodyfilter`. This predicate maps to the very same method used for `imap:body`, but it does not require the string match to be exact. In this way, messages are first filtered with a normal substring search on the IMAP server, then they are FILTERed with a regular expression. Of course, this solution cannot be easily applied with very complex regular expressions, but is useful in most common cases.

OPTIONAL subqueries

Queries which contain an OPTIONAL clause are usually managed in a more complex way than in the basic case: the non-optional part is first evaluated, then for all the results the OPTIONAL ones are, resulting in our case in $N \times M$ searches on the IMAP server (where N is the number of results, and M the number of optional clauses). This raises a big performance problem, which can become more serious depending on

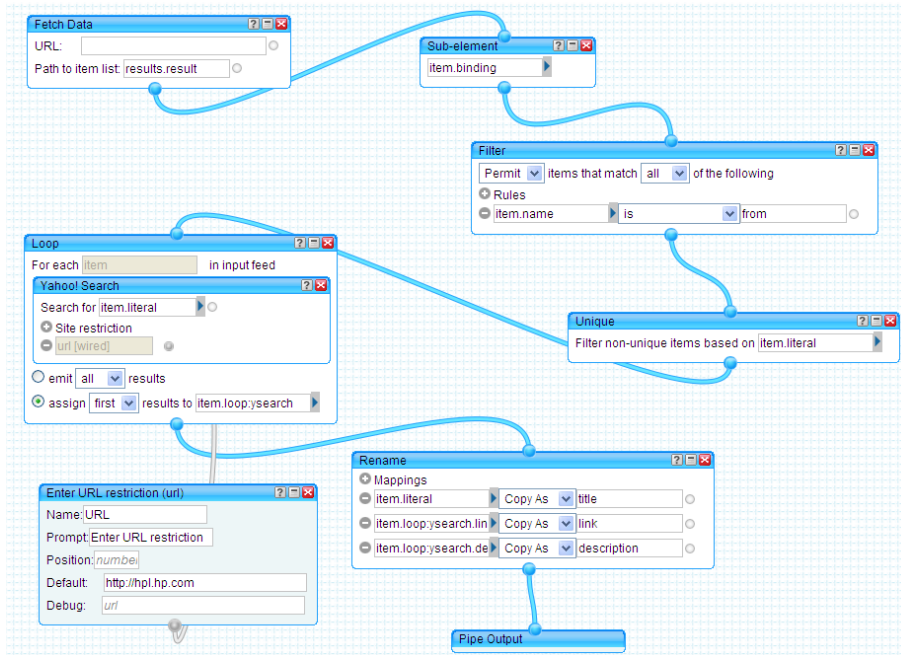


Figure 6.7: A *Pipe* extracting email authors from a mailbox and searching for them on Yahoo.

the connection speed between the client on which SquirrelRDF runs and the IMAP server. Our solutions worked primarily on disabling optionals when possible and trying to return empty results instead of *null* ones, but we think that further study is needed on this topic.

A mashup test

One of the main advantage of SquirrelRDF is that the information extracted from the IMAP server is provided in very standard and common formats such as RDF and XML, so it can be easily reused within other applications. As an example, we developed some very simple mashups with Yahoo Pipes⁸ which get email data from the servlet, augment them with Internet searches, and then publish everything in standard formats such as RSS.

These tests gave us some very interesting results: first of all, as SquirrelRDF allows to automatically convert structured information into RDF, querying the IMAP server and connecting the results to a mashup tool

⁸<http://pipes.yahoo.com>.

like Yahoo Pipes is really easy and fast. In fact, the *Fetch Data* widget allows users to load any XML page and extract information from it: we used this feature to download SquirrelRDF servlet results page and automatically extract the list of bindings. Then, information can be parsed, filtered, and reused inside other plugins.

This chance of redirecting information from one service to another immediately proved to be quite useful: for instance, we used it to automatically extract all the authors from the emails, search them on the internet and provide the results of these searches as additional information about them. The final product is a document which contains the list of authors, followed by a link to a page which is related to them, and a brief summary of that page.

Finally, as these new data can be published in very common formats such as RSS, they can be read, shown or furtherly processed by many different applications. As an example, we imported the results of the previously described pipe as Firefox Live Bookmarks, so the additional information about email authors is always available within the browser.

6.1.6 Conclusions

The plugin we developed gives the expected results and integrates well with the SquirrelRDF application. Performances are good on the local IMAP server, and they are expected to be more than acceptable on a remote one. Information returned by the application is current and provided in a standard (RDF/XML) format, so it can be easily shared and reused inside other applications.

As a planned future work, as there are still some limitations in the model (in this very moment it only works with flat email ontologies), we have begun to implement a new version which will enable more complex ones. By *flat*, we mean that we still cannot deal with second-level relationships: for instance, while we can answer a query like

```
select ?from where
{
    ?x email:from ?from .
}
```

where *?from* is a variable containing an email address, we cannot answer another like the following:

```
select ?realname ?address where
{
```



```

    ?x email:from      ?from .
    ?from email:realname ?realname .
    ?from email:address ?address .
}

```

where `?from` is an anonymous node linking to a real name and an address value.

Our test queries showed that connections to the IMAP server are the bottleneck in performances of SquirrelRDF: for this reason we are planning to optimize them with caching (both of connections themselves and of search results). Also, we have already started to work on a new plugin which would use the same kind of predicate-to-method mapping: this plugin will connect via HTTP on a Twiki website and allow users to access wiki contents with SPARQL queries. Finally, we are planning to experiment more with mashup services and build some custom ones to enrich and merge all the information we can extract thanks to SquirrelRDF.

6.2 Exploiting User Gratification for Collaborative Semantic Annotation

One of the biggest challenges for the Semantic Web community is trying to add semantics to information which has already been published in the form of unstructured text. Many approaches have been tried to add semantics to unstructured pages, and the idea of annotating Web contents seems a good one, allowing for a real “read-write Web” where any user or machine can add metadata to any piece of information.

While both automatic and manual annotation systems still present some open issues (just to name few, word disambiguation [121] for the former ones and lack of precision [45] for the latter ones), the semi-automatic approach currently seems the most feasible. However, at the present time even this kind of systems still lack that wide acceptance that would make a semantic annotation system really useful on the World Wide Web.

Our project starts with the assumption that one of the possible reasons of this is that these systems do not provide enough advantages to motivate the user efforts to make them work. Trying to overcome this problem, we decided to increase user motivation building an easy and rewarding annotation tool.

To do this, we envisioned a collaborative semi-automatic annotation

approach for Web pages, which allows to connect pieces of unstructured text with standard concepts without requiring specific knowledge about semantics; as a result, annotated data become automatically linked to a whole set of services and resources specific to their related concepts, thus providing an instant reward for users in the form of additional available information.

6.2.1 Related work

Several tools and approaches exist to create annotations of both Web resources and abstract concepts: [122] provides a survey of the main semi-automatic annotation platforms, while [112] presents a unified formal model, able to describe and integrate annotations inside traditional documents, semantic wikis, semantic blogs and collaborative tagging systems.

Our project shares one basic principle with collaborative tagging: users annotate for themselves, but the system automatically shares personal annotations between users so that everyone contributes to the overall value of the system. At the same time, the differences between collaborative tagging systems and ours are rather strong: first of all, the granularity of our semantic annotations is much higher, as it involves single words inside a Web page instead of more complex kind of resources; moreover, users in our system cannot annotate using unconstrained strings, but they have to choose one concept from a list of suggested ones. This last point is particularly important, as it defines a completely different annotation paradigm: while tagging is bottom-up and flat, our semantic annotation is top-down and hierarchical, with all its advantages and limitations [133].

Annotea [75] is a semantic annotation tool which enhances collaboration via shared metadata based Web annotations, bookmarks, and their combinations. It uses an RDF based annotation schema for describing annotations as metadata and XPointer⁹ for locating the annotations in the annotated document. Different client softwares for Annotea have been built: between these Annozilla¹⁰, created as an extension of the Mozilla browser.

KIM [78] is a software platform for automatic annotation, indexing and retrieval of information. Its approach is based on the assumption that *named entities* [32] have to be handled in a special way, as they denote particulars (individuals or instances) while other words denote

⁹<http://www.w3.org/XML/Linking>.

¹⁰<http://annozilla.mozdev.org>.

universals (concepts, classes, relations, and attributes). It then uses NLP to recognize and identify them inside a text, with respect to a predefined ontology.

MnM [150] provides both automated and semi-automated ontology driven support for annotating Web pages with semantic contents. The annotations are written as markup inside a document. Magpie [40] uses an ontology infrastructure to semantically markup Web documents on-the-fly. Both of these tools work as browser extensions and provide new pieces of information related to the annotated text, but both seem to get these information just from the ontology, and not from external data sources.

Revyu [66] is not a generic annotation tool, but rather a reviewing and rating Web site. We consider it related to our work for its attention towards Linked Data principles and best practices [16, 20]. This system does not only work as a service usable by humans, but also provides information in a reusable format that can be easily integrated with other data.

Gnosis¹¹ is probably the project which is most similar to ours. It works as a Firefox extension and when a Web page is loaded inside the browser it immediately locates key information such as people, organizations, companies, products and geographies hidden within the text. It then highlights these concepts in the page and provides links to specific search engines which change depending on the resource type. One of the main drawbacks of this tool is that it relies on NLP and on a read-only knowledge base, both for what concerns named entities and for the search engine list. While this is surely convenient for kickstarting the application, active user participation could make the tool more precise and powerful; moreover, it could help to disambiguate strings that match many different concepts.

6.2.2 Project overview

We started our project with the assumption that one of the possible reasons why many semantic annotation systems still do not have a wide acceptance between Internet users is that they do not offer them much, or at least not much enough to motivate their efforts for using them. The question we try to answer is: how can we exploit the concept of gratification to make users semantically annotate unstructured text?

To the best of our knowledge, available semantic annotation tools cur-

¹¹<http://gnosis.clearforest.com>.



Figure 6.8: Some possible links for “Harry Potter”, when considered as a book.

rently provide some kind of reward to the user in the form of additional, concept-specific information that is accessible when a piece of text is annotated as being an instance of one particular concept. However, this information is usually taken from the ontology used by the annotation system, and as a result it is very schematic and constrained by the ontology itself.

As an example, think about a user who is browsing a Web page containing a review about one of the Harry Potter books. The user might be interested in knowing something more about these books, so why should she annotate “Harry Potter” as being a book? No ontology is currently able to provide the same quantity of information that can be found on the Web about this topic, and probably if one was built to do so it would not be able to keep the pace of the ever growing Web. However, it is still possible to link this concept, once we know it is a book, to a huge number of services, data sources and search engines which will provide huge quantities of related information: as an example, Figure 6.8 shows results from some book-related services, ranging from textual descriptions to RDF data, from user ratings to the complete books in PDF format.

The purpose of our project is to build an annotation system able to

provide this kind of experience to the user: the additional context-related information should be accessible through a template page like the Geo-Hack page¹² in Wikipedia, and could provide links to specific services, search engines and query strings¹³. As a result, we will be able to use user communities and Semantic Web technologies in a virtuous cycle: on the one hand we will exploit user spontaneous collaboration to increase the amount of semantic metadata available; on the other hand, we will use these new data to make the system more rewarding, thus encouraging user participation.

Users

The main goal is to make users spontaneously contribute to the system with semantic annotations. This translates in two main requirements: motivate users and make their contributions useful by allowing them to be visible and reusable in the “Web of data”.

One way to increase motivation in users is by giving them some kind of reward for their participation [111, 27]. This is the reason why our system is designed to provide it in a double way:

- an instant gratification, by automatically linking the annotated data with additional sources of information,
- and a long term one, by sharing annotations in a standard and structured way so they can be searched, accessed and linked with other data.

Keeping the average Internet user in mind, we also have to face the problem of identifying what kind of related information might be considered interesting enough for a particular community of practice [156]: for instance, an adult does not have the same needs as a teenager, and people from different parts of the world might want to access different local services instead of more general ones which are available worldwide. For this reason, these links should be collected in pages which are very easy to create, edit and share such as inside a wiki system: anyone has a page by default for a particular concept, but he can customize it or directly choose another one.

Finally, the system has to be easy to use and intuitive: users do not have to know specific details about semantics, but they just have to

¹²Visible, for instance, clicking on the geo coordinates at <http://it.wikipedia.org/wiki/Milano>.

¹³Like the “search webbits” at <http://www.searchlores.org/rabbits.htm>.

choose a concept from a list of suggested ones. Also, the only part of the system they have to use is a browser extension, which adds a new option to the contextual menu that appears when the right mouse button is clicked (following the affordance of this button).

Data

Making semantic metadata reusable by other applications basically means publishing them in RDF and making them available through a SPARQL endpoint. A particular attention has been devoted to using common ontologies for interoperability with other systems, and providing a mapping system to automatically translate internal terms with terms from well known RDF vocabularies.

Whenever a term is tagged as being instance of a particular concept, additional information can be automatically harvested from the Internet to make the model richer, and some automatic annotations (such as the ISBN for a book, such as in [66], or an IMDB id) could be added to the knowledge base.

6.2.3 Project architecture and implementation

The architecture of our system is shown in Figure 6.9: its main modules are the client extension, the knowledge management tool and the annotation server.

The client

The client application is nothing more than a normal browser with a plugin. The user just has to select some text and click the right mouse button, choosing the contextual menu option “Speakin’ about”: a popup window then appears, allowing her to choose the name of the concept related to the selected string.

Actually, as soon as the menu option is chosen the browser extension communicates with the knowledge management tool (KMT), sending the selected string and asking for possible concept suggestions. The KMT replies with its suggestions (see next paragraph for details) and when the user chooses one it collects all the information needed for the annotation. Once the annotation is saved, the selected text becomes a link to a special page which contains a collection of concept-related links that can be followed to find new pieces of related information.

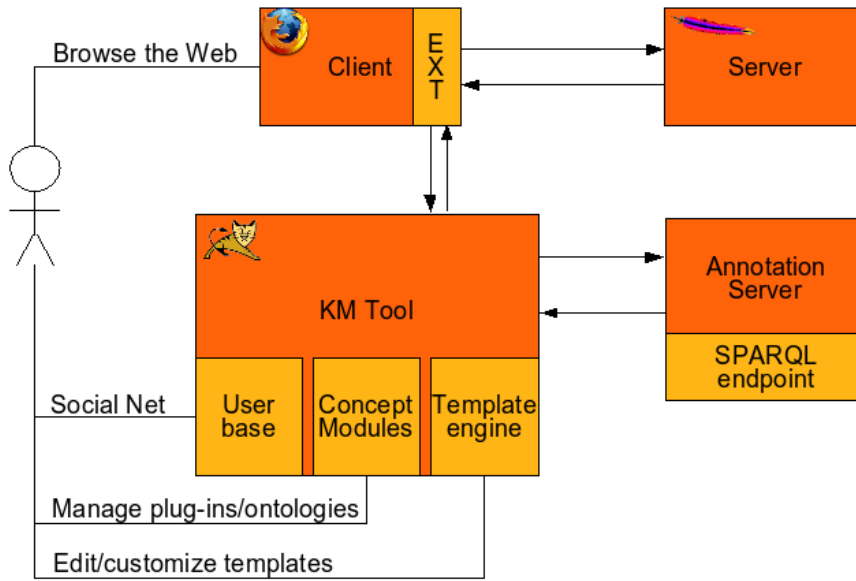


Figure 6.9: The system architecture.

The Knowledge Management Tool

The KMT manages the communication with the annotation server, saving the metadata when it receives them from the user. Also, it provides additional functionalities thanks to its three submodules: the user base, the concept modules and the template engine;

- **User Base:** As everyone can write any kind of annotation, at least an authentication mechanism is needed to bind annotations with users. This way, users can either see everyone else's annotations on one page or override them with their own ones. Additional features which might be useful are a social network to connect users and a trust system to allow users to allow (or deny) by default someone else's annotations.
- **Concept Modules:** As users submit their strings, they should be shown some concept suggestions: the concept modules take care of this, searching for the string inside ontologies, Web sites, and online services to find a matching concept to suggest. The structure is modular as many different approaches can be taken: for instance,

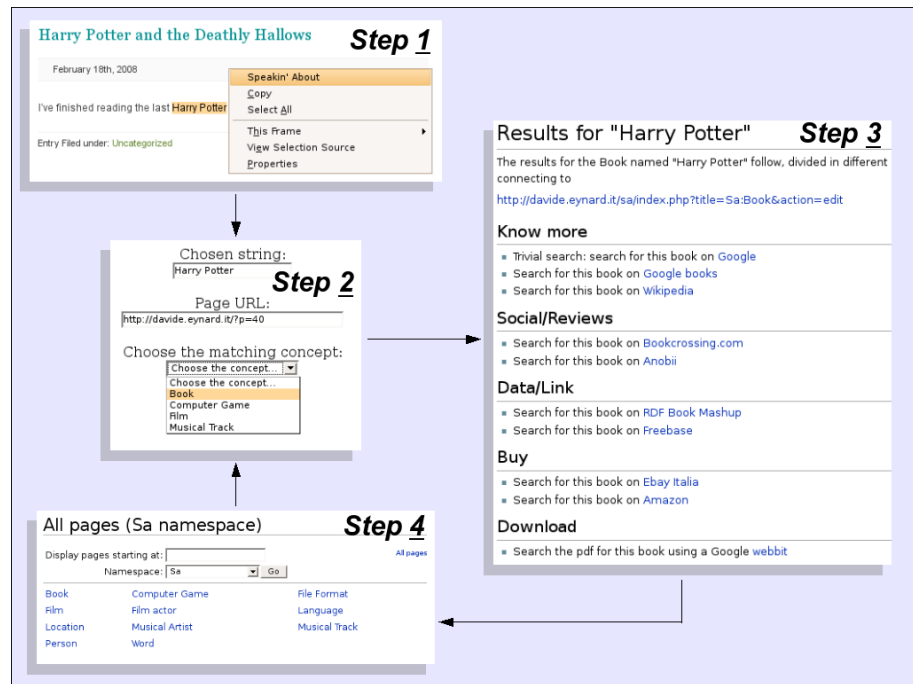


Figure 6.10: A usage example of the tool, from the context menu to the search page.

a module we developed in our prototype searches for matching concepts or instances inside some domain ontologies; another one searches inside Freebase; another one could use Wikipedia, or a search engine for ontologies like Swoogle. As any module might suggest concepts with different names, this component also comes with a mapping service which allows to map these names with some fixed concepts inside one main vocabulary (which, basically, is the ontology of the concepts for which a special page already exists).

- **Template Engine:** Special pages with related links exist as templates, that is pages that receive the selected string as a parameter and use it to build all their links (as an example, check the wiki code of the GeoHack page described above). The template engine is the system which allows to manage templates and matches them with concepts. It could be very simple (a database which contains the matches, plus some HTML files) or more complex: in our case, we chose to manage the templates with a wiki-like system, allowing users to easily create, edit and share them. In this way, the possibilities offered by the application are not constrained by anyone and can grow thanks to user contributions.

The annotation server

The information contained in an annotation made with our tool basically connects (at least) the annotated URL, the string that represents the name of the concept instance, the matching concept and the name of the user who has saved the annotation. The annotation server is in charge of storing this information and making it available in a standard format.

Designing our system we decided to consider the annotation server as a separate module: the reason was that we wanted to build our system on already available and consolidated technologies, rather than programming everything from scratch. For instance, Figure 6.11 shows how we could save our metadata in an Annotea server, following its ontology specification [75].

Implementation

Currently we have a very simple prototype, developed as follows: the browser extension is a Firefox extension programmed in Javascript; the KMT has been written in Java and runs as a servlet on a Tomcat server; the annotation server is a light application written in Java, which saves its information inside a SQL database and exports it in RDF; thanks

Annotea property	Value in Speakin' about
rdf:type	Annotation (as defined in Annotea)
annotates	The URI of the annotated resource
context	XPointer of the annotated piece of text
body	Could be left empty, or used to correct/disambiguate annotated text
dc:creator	Username of annotation creator
created	Date of creation
dc:date	Date of last update
related	URI of the related concept

Figure 6.11: Matching between Annotea properties and annotation values used by Speakin' about.

to Joseki, it can then be queried as a SPARQL endpoint; the browser extension communicates with the KMT via HTTP, and the KMT communicates with the annotation server via RMI.

6.2.4 System evaluation

The tool has been designed with ease of use in mind: for this reason, the whole process of semantic annotation reduces to just few steps. At the first step (see Step 1 on Figure 6.10), the user selects some text from the current Web page, clicks the right mouse button and chooses the “Speakin' about” option. Then (Step 2) she is shown the chosen string, the originating page and a list of suggested concepts. When the user chooses one of the concepts and submits the information, the annotation is done and the Web page is updated with a new link on the annotated text. If the user clicks on that link, she is shown the search page (Step 3), containing the list of search engines related to the particular concept chosen by the user. Of course, once the user understands how wiki templating works she can change existing search pages or create new ones, allowing the concept selection tool to provide more choices (Step 4).

As the system is inherently dependent on user participation and, at the same time, it aims at linking information from different sources, it opens to two different evaluation approaches: on the one hand an evaluation of usability and user satisfaction, and on the other hand a test on the exported data and how well it integrates with heterogeneous sources. Being still in the prototypal phase it was not possible to test the system with a realistic user base, however we were able to perform

Class	Gnosis	Speakin' about
City	6	5
Company	2	2
Continent	1	1
Country	18	17
Industry Term	3	0
Organization	6	4
Person	15	10
Product	1	1
Province or State	3	3

Figure 6.12: Comparison between the number of concepts automatically found inside the main English Wikipedia page by Gnosis and the ones also recognized by Speakin' about.

user-independent tests on it. For instance, we verified that the application was actually able to provide users with the promised additional information and links: to do this, we decided to compare our results with the ones provided by Gnosis. We performed the test on the main English Wikipedia page, checking how many concepts were automatically detected by Gnosis. Then we passed the matching strings to our Knowledge Management Tool to see if it was able to suggest the same concepts. For this task we used the Freebase module, which relies on the vast amount of knowledge harvested from Wikipedia and enriched by its user community.

The results are summarized inside Figure 6.12: Speakin' about is able to recognize almost all of the concepts detected by Gnosis, however it is much more sensible to typos (which is the reason why it could not detect one City and some names); moreover, it is not able to recognize industry terms like “bank” or “environmental law” as they are not named entities and they do not appear in Freebase. Conversely, Speakin' about offers a larger taxonomy, which is the one provided by Freebase: in the same page, it was able to detect football teams, book titles, actors, and so on. Also, similar concepts are suggested based on partial matches of the selected strings; more concepts are provided at the same time, so that there are more chances for users to disambiguate the term; finally, the related search engines provided by Speakin' about are usually much more than the ones provided by Gnosis, and their number can grow thanks to user contributions.

For what concerns data evaluation, we have put our effort in following Linked Data recommendations, with the purpose of making our meta-data public and easily available to other applications. Our prototype system is already able to export its information in RDF and provides a SPARQL endpoint to allow queries over the knowledge base. It is thus possible to ask for all the pages which “speak about” some concept (or some instance), and link this information with other data to answer more complex queries: for instance, importing an ontology about cinema, a user can ask for all the pages that speak about movies in which a particular actor has starred.

Other collateral advantages spawn from annotating information with *Speakin’ about*: first of all, the system allows not only to add semantic metadata about URIs (that is, saying that a page is about some particular concept), but also to map strings with specific concepts, disambiguating them and offering access to a wealth of concept-specific services (see an example of disambiguation in Figure 6.13, where “Verdi”, which in Italy is a color, a composer and a political party, is identified as a composer). In particular, users instantly have access to new pieces of linked information, which grow thanks to collaboratively edited templates, harnessing the power of available services and specialized search engines. Some template pages (ie. books and movies), containing links to specific services and search strings, are already available as a proof of concept, and thanks to a wiki system users can easily edit them and create new ones.

6.2.5 Conclusions

In this section we described a collaborative semi-automatic annotation approach for Web pages, which allows users to connect pieces of unstructured text with related concepts. As an immediate result of these annotations, users are provided with related information about annotated concepts, in the form of pages containing links to concept-specific services and search string. These pages are built up from templates which can be created, modified and shared by the community inside a wiki-like system.

The novelty in our approach is represented by the following aspects: first of all, the additional information about annotated concepts is not provided by the system itself but it is harvested from the Internet, taking advantage of all the systems which freely share their data; then, all the saved metadata are made public and easily available to other applications, as they are saved in RDF and exposed through a SPARQL

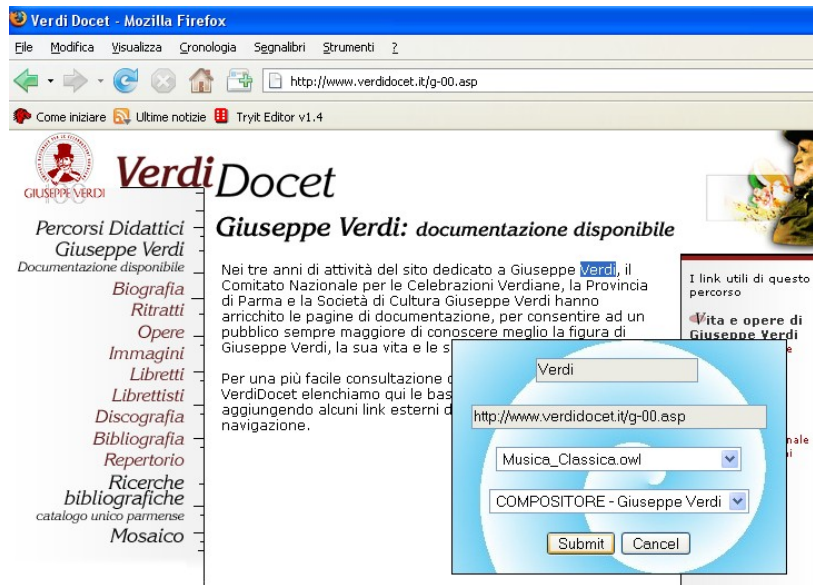


Figure 6.13: Our prototype in action: thanks to the ontology plugin, the ambiguous string *Verdi* is disambiguated thanks to its association with the *composer* concept.

endpoint; finally, the whole system relies on user participation and uses the new linked information as a reward for the users.

Our application is currently in a prototypal form. As a future work, we plan to build a system complete in all its modules (in particular, the user management and the connection with an Annotea server) and make it available on the Internet: this will allow us to complete our evaluation with a real user base, in terms of participation and user feedbacks.

6.3 Using semantics and user participation to customize personalization

Personalization, under a general interpretation, means tailoring a product or a medium to a user, according to some personal details they provide. More specifically, on the Web it is the process of providing relevant content based on individual user preferences. These preferences can be provided to the system either implicitly or explicitly. Examples of explicit data collection include user ratings (i.e. stars on YouTube or diggs on Digg), rankings, or wish lists (such as Amazon's wish list¹⁴). Examples of implicit data collection include observing which Web pages are viewed by users and the time they spend on them, keeping a record of the items that a user purchases online, reading playlists of multimedia contents, and analyzing users' social networks.

Once user preferences have been collected into what, from now on, we will call a *user profile*, personalization engines can access them to provide a tailored Web experience with personalized contents or interfaces. Of course, there always needs to be someone who provides material which can be tailored on each person's profile.

Customization takes place when users directly affect the behavior of the engine which provides the contents, by specifying (usually explicitly) different parameters inside a *configuration*. A schema of how both personalization and customization work is shown in Figure 6.14. Looking at the figure, it appears quite evident how much the two are similar, and it also explains why they are often confused with each other: basically, for the tailoring engine (which takes care of personalization, customization, or both) the profile and the configuration are nothing but input parameters that tell how it has to operate; once it reads them, it can tailor contents to the user's desire. However, from the user perspective personalization and customization work in very different ways: in the former

¹⁴<http://www.amazon.com>.

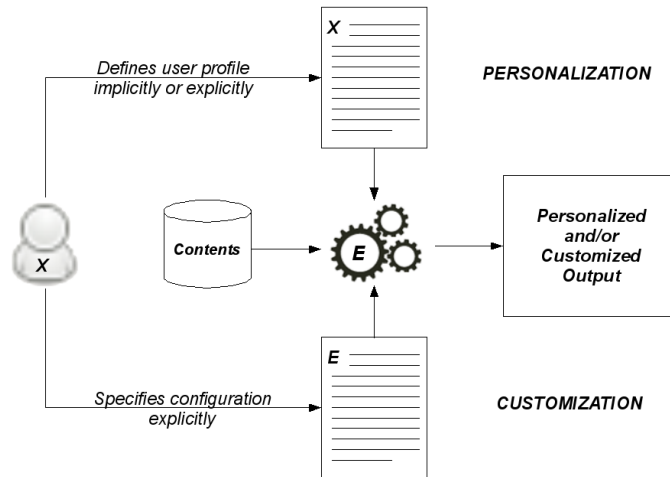


Figure 6.14: Personalization and Customization.

case, the only knowledge users have to provide is about themselves (i.e. who they are, what they like, and so on); in the latter, as users have to provide information about how the engine has to operate they need to know something about the system itself.

The fact that customization is so dependent on the knowledge of the system makes it much more complex for beginners: this is the reason why many applications often have some settings enabled by default, so that unexperienced users will basically have to know nothing about the program to be able to run it. However, the more users get experienced the more they appreciate the possibility of customizing the behavior of their applications so that it better reflects their preferences. The result is that customization allows users to know what to expect from the programs they have configured: as an example, a media player can be configured not only to play the songs a user likes, but it can also play them following a particular order which is specified inside a playlist.

Even from this example it might seem that customization is preferable to personalization, as it allows users to have full control of their systems, this is also useful to understand why sometimes personalization could be much better than customization. After all, who does not ever get bored of always listening to the very same sequence of songs? Systems like Last.fm or Pandora¹⁵ have a great success lately as they provide personalization

¹⁵<http://www.pandora.com>.

on radio streams: users can choose the bands or the genres they like most and, thanks to a system that learns their preferences, they listen not only to the songs they already know, but also to other unknown ones that they might like given their profile. In an interconnected world where the songs we know will always be much less than the ones we do not (and, by analogy, where all the Web pages we can read will be just a tiny part of the ones that are published), personalization has the advantage of providing us with new, interesting information which suits our taste.

A huge limit of personalization, however, is that it highly relies on the user profile: on the one hand this affects the system precision, as profiles often have to be built from scratch and it takes time for the system to learn users' preferences; on the other hand, giving away their personal information is often a problem for users, who want to protect their privacy and security as much as they can.

Summarizing, both customization and personalization have their pros and cons:

- *customization* allows to have a finer grained control on the system, allowing users to know what to expect; however, already knowing what to expect is sometimes boring and users might need to know too many technical details, which could make actual customization unfeasible for beginners. Also, as configurations usually show only the details of the system which are exposed to users, they might give them the false feeling of having control on something they can only see in part.
- *personalization* is less predictable and shows potentially new and interesting information, requiring virtually no technical knowledge on the user's side. However, it takes time to train and it works on a model of the user preferences (the profile) which can be wrong, limited, or incomplete. Also, the way the profile is used to generate the final results, and often the profile itself, are usually hidden to users, so they can only take advantage of what they are given, hoping it is going to provide something useful for them. The impossibility of choosing how their own profiles are used, together with concerns about their privacy, often make users choose for the easiest choice, that is eliminate personalization¹⁶.

¹⁶A very interesting real-world example of "customization to disable personalization" is shown at <http://googlesystem.blogspot.com/2007/04/how-to-disable-google-personalized.html>.

Starting from these assumptions, we decided to give users the chance of *customizing personalization*, allowing them to access their own profiles with different layers of abstraction and providing them with an easy way to develop, share, and run applications which use their profiles to do something useful. Even if the number of users which are also programmers is small, both literature [90] and real-world examples (see Section 6.3.1) show that it is possible to exploit the different levels of user engagement to have different qualities of participation.

For our experiments we chose a very specific context: user history and bookmarks within the Firefox browser. We then developed an ontology to describe this knowledge and a set of tools that provide the following functionalities:

- offline conversion of Firefox 3.0 history and bookmarks to RDF;
- a framework for the development of personalization plugins, in the form of a Firefox extension which updates the ontology in realtime (i.e. while the user is browsing the Web) and uses information taken from it to customize the Web experience.

Finally, we developed some example personalization plugin which allow for information visualization, augmented browsing, and website customization.

The report is organized as follows: in Section 6.3.1 we describe the technologies and the software we used and the related work; in Section 6.3.2 we describe our approach and motivate our choices; in Section 6.3.3 we describe the architecture of our project and show the most important details of our software; in Section 6.3.6 we evaluate the project and in Section 6.3.7 we conclude and suggest future developments.

6.3.1 Prior work

Semantics and personalization have already been studied from different point of views. As an example, [1] provides some papers collected for the Semantic Web Personalization Workshop that took place at the 3rd European Semantic Web Conference in Budva, Montenegro.

Semantics can be used to provide better recommendations. [42] uses an ontology to describe a domain and maps keywords extracted from Web pages to concepts inside the ontology. The ontology terms are used to annotate the Web content and users' navigational patterns, and at the same time to do word sense disambiguation. The personalization

framework built onto this ontology is then used to enhance the recommendation process with content semantics.

[146, 64] use ontologies as a base to calculate similarity between items on a website (respectively, Web pages or documents in a bibliographic peer-to-peer system). In the first case, a domain ontology is used and the similarity depends on the relations that connect concepts inside the ontology. In the second, the SWRC (Semantic Web for Research Communities) ontology¹⁷ and the ACM topic hierarchy¹⁸ are used as a base for different similarity functions.

Semantics can be used to improve search by filtering search results. [108] shows how information gathered from social bookmarking services can be used to provide better search results. A similarity metric has been suggested that uses the tags to describe the web results, and has been used to rerank search results.

Semantics can be used to describe and make the user profile portable. [23] uses Semantic Web technologies and the two ontologies SIOC¹⁹ and FOAF²⁰ to model user information and user-generated contents in a machine-readable way. The advantage is the possibility of reusing user data and having her network information saved in a standard and well known format.

Semantics can be used to personalize the tool. [6] describes the use of FOAF and a modification to the HTTP protocol to create personalized Web pages on the server side. The same authors in [7] show a similar approach, however suggesting the very interesting alternative of using personal information for personalization on the client side. [76] also shows how Web history can be used to customize the user interface on the client side, filtering or re-ranking the Web applications suggested by many social websites depending on which ones have been accessed more in the past. Finally, all the recent browsers (Firefox 3, Google Chrome, and Flock) provide some way to show users their most accessed pages: Firefox shows them as “Smart Bookmarks” in the bookmark bar, while both Chrome and Flock provide a personalized start page with the favorite pages. Flock currently even goes a step further, with a start page which has content gathered from different social Web applications and the possibility for the user to customize it.

¹⁷<http://ontoware.org/projects/swrc>.

¹⁸<http://www.acm.org/class/1988>.

¹⁹<http://sioc-project.org>.

²⁰<http://www.foaf-project.org>.

Client-side Customization

The idea of customizing Web pages on the client is not new, but began to have a wide success only lately. The concept at the base is pretty simple: all the information entering a machine can be customized by the machine itself before they are shown to users: this allows them to have control over what they see, rather than using the computer as a TV[47].

Some of the first programs which allowed users to do client-side customization on Web contents were Scott R. Lemmon's Proxomitron²¹ and Philtron²². They worked as local proxy servers and were able to filter incoming HTML pages before they were loaded by the browser for rendering; most of the customization was done exclusively on the page contents.

While client-side customization tools did not seem very useful at their origins (and surely their complexity was much higher than the actual advantage of using them, at least for the large majority of users), they are now much more useful as they can customize not only page contents, but also user interfaces. This is partly due to the fact the Web is currently used not only as a repository of information, but also as a provider of online applications, which share with Web pages the same client-side modifiable representation. Moreover, latest browsers like Firefox are built to be customized: with its *extensions*, which are basically Javascript applications that are embedded into the browser and can access its APIs, it is possible both to operate on the interface (i.e. adding functionalities) and on the page contents (i.e. filtering advertisements).

An example of Firefox extension, which acted as an inspiration to this project, is Greasemonkey²³. It provides a set of functions, which allows to easily customize the content of Web pages, and a script manager, which allows users to write and share their scripts easily. As a proof of the interest of users towards this kind of products, this extension is one of the most popular ones on the Firefox Add-ons website with about 9.5 million downloads, and provides more than 20,000 scripts on its community website²⁴.

²¹<http://www.proxomitron.info>.

²²<http://philtrn.sourceforge.net>.

²³<http://www.greasespot.net>.

²⁴Data gathered from <https://addons.mozilla.org/it/firefox/addon/748> and <http://userscripts.org/scripts> on September, 10th 2008.

6.3.2 Our Approach

Our project is based on the assumption that personalization could provide much more useful results and be better accepted by users if they could have access to their own profiles. As a specific example, we decided to develop a tool which works on the browser history and bookmarks to provide personalized experiences to users.

There are different reasons why we chose this particular application. First of all, as both computer applications and multimedia entertainment are becoming more and more web-driven, there are going to be more and more chances to have a quite detailed personal profile just by browsing our history (that is, what websites we have visited) and our bookmarks (that is, what websites we want to remember). Then, as the quantity of information available on the Web increases, the classic hierarchical bookmarking paradigm is going to become old and very difficult to manage (supposing it is not already). Social bookmarking (i.e. *del.icio.us*), Web search (i.e. Google's search history), and starred items (such as in Firefox 3) are some possible solutions to this problem, but we believe that allowing users to easily access their own history could provide a bigger advantage to them. Finally, browser bookmarks and history are already available on every computer and, even if there are some tools which actually use them²⁵, they are not shared with a standard format.

The immediate advantage of our approach is that personal information such as history and bookmarks are available to the user in a standard and simple format; moreover, data is updated in realtime and can be accessed in any moment by any application, either internal or external to the browser. This makes a big difference, as Firefox 3 locks its database while it is running and the only chance to access it is from within the browser (i.e. programming an extension) or working offline when it is closed.

The Data Model

To build our ontology, we decided to study the schema of an existing history profile: we chose Firefox 3 `places.sqlite`, a SQLite 3 database file which contains both history visits and bookmarks. The file is usually saved together with Firefox user profile information: for instance, under Windows Vista it can be found in

```
<user home>\AppData\Roaming\Mozilla\Firefox\Profiles\<profile>\places.sqlite
```

²⁵See, for instance, the default start page in Google Chrome, which shows users their most visited websites.

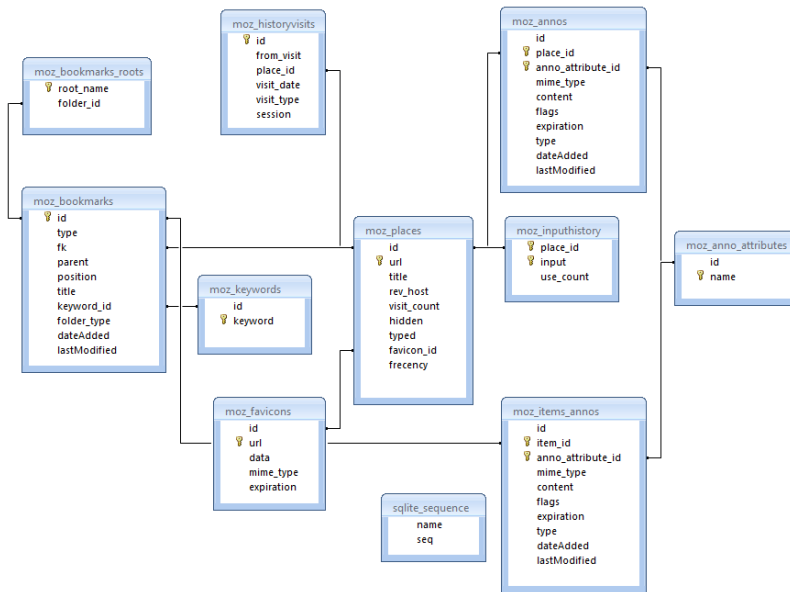


Figure 6.15: The schema for Firefox 3 places.sqlite file.

Under Linux, instead, it is available in

`~/.mozilla/firefox/<profile>/places.sqlite`

The database schema is shown in Figure 6.15. The tables we are currently gathering data from are:

- **moz_places**, which provides information about URLs. For instance, together with an id and the actual URL, it also has a field containing the page title, one with the number of time it has been visited, and one linking to the favorite icon;
- **moz_historyvisits**, which contains the history. It links to places from **moz_places** and provides information about the referrer, the visit date, the type of visit (i.e. a direct connection, a page redirection, etc.), and the browsing session;
- **moz_bookmarks**, which contains the bookmarks collection. For every bookmark different information are provided, such as the type (bookmark, folder, or separator), the name, the parent folder, and the dates of creation and modification.

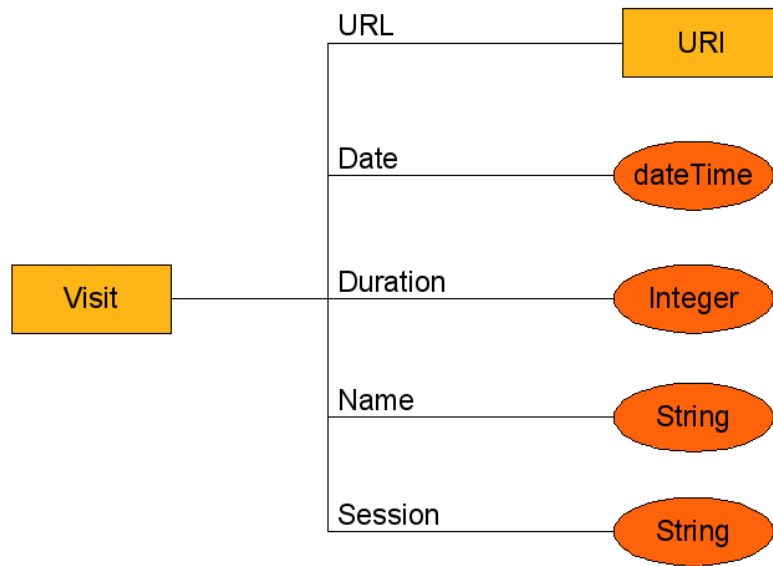


Figure 6.16: Data model for visits.

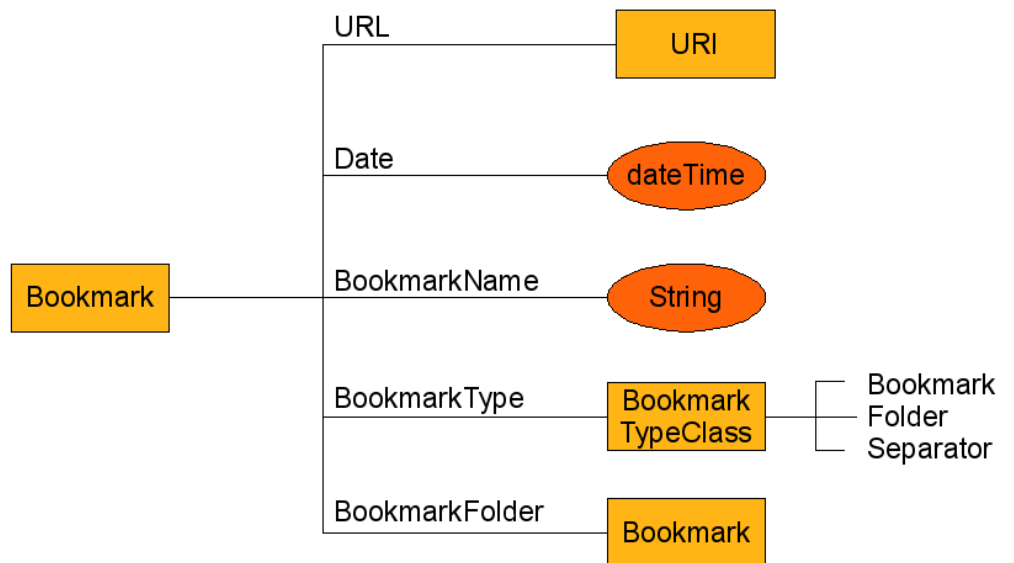


Figure 6.17: Data model for bookmarks.

6.3 Using semantics and user participation to customize personalization

```
PREFIX HPLBrowseData: <http://hpl.hp.com/schemas/FFontology#>
PREFIX mozh:          <http://hpl.hp.com/schemas/history#>
PREFIX xsd:           <http://www.w3.org/2001/XMLSchema#>

mozh:826 rdfs:type                HPLBrowseData:Visit ;
        HPLBrowseData:Session    "45524179520808800"
        HPLBrowseData:Referrer   mozh:0
        HPLBrowseData:URL        <http://delicious.com/>
        HPLBrowseData:Name       "Delicious"
        HPLBrowseData:Date       "2008-08-11T23:04:11Z"^^xsd:dateTime
        HPLBrowseData:DeliTopTag "delicious"
        HPLBrowseData:DeliPosts  741
```

Figure 6.18: A history item in RDF, with related delicious information attached to it.

The ontology we built has a simpler structure, taking into account only two main classes:

- **Visit** (see Figure 6.16) describes browser history and its properties are *URL*, *Date*, *Name*, *Session*, and *Duration*. Currently only URL points to a resource, while all the other properties point to literals (datatypes are xsd:DateTime for Date, xsd:String for Name and Session, and xsd:Integer for Duration). The last property describes the quantity of time that a user spent on a specific page and is not provided by Firefox history;
- **Bookmark** (see Figure 6.17) describes bookmarks and its properties are *URL*, *Date*, *BookmarkName*, *BookmarkType*, and *BookmarkFolder*. URL and the last two properties point to resources, while Date and BookmarkName point to literals whose datatypes are, respectively, xsd:dateTime and xsd:String.

One of the main advantages of having these data saved in RDF format is that the knowledge base can be expanded very easily, just by asserting new triples about the existing resources. For instance, the one shown in Figure 6.18 is an RDF description of an element from the history: the last two lines represent the top tag used for that particular URL on delicious.us and the total number of posts at that particular time. These two pieces of information have been added by another application (explained more in detail in Section 6.3.4) and are used together with the main data for visualization purposes.

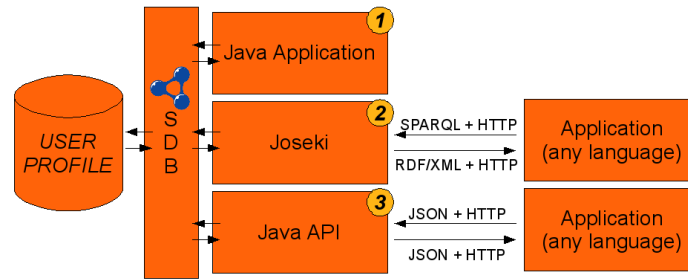


Figure 6.19: Three possible architectures to access the RDF store.

6.3.3 Framework Architecture and Implementation

A very modular architecture has been decided for the project, allowing users to access the profile with different levels of abstraction and complexity (see Figure 6.19). Basically, user browsing profile (history and bookmarks) is saved in RDF inside an HSQLDB database; SDB provides the layer of abstraction that allows us to have a database-backed RDF store and offers access to data in (at least) the three following ways:

1. a generic Java application can directly access the RDF store using Jena and the SDB library;
2. Joseki can be configured to access the RDF store using SDB, then any external application can manipulate the store via HTTP requests, using the SPARQL endpoint;
3. information could be exposed through a (Java+Jena+SDB) API, then any external application can manipulate the store exchanging JSON data over HTTP.

Note that the second and the third approach, using Java applications that use Jena and SDB, are particular cases of the first one. However, each one provides a different interface to the profile and has its pros and cons:

- the first solution gives full control over the ontology through the Jena API and provides the best performances. However, it is so low-level that it also leaves programmers the burden of dealing with database connection details (i.e. the authentication parameters)

and requires an in-depth knowledge about the ontology structure, RDF, and SPARQL. Moreover, it constrains programmers to use Java, Jena, and SDB.

- Joseki, allowing programmers to work at a higher level of abstraction, removes the constraints on programming language and does not require users to know any detail about the database connection: basically, any application able to send SPARQL requests over HTTP can thus access the ontology. The advantages of this solution are clear: it relies on well known standards, it is very fast to deploy, and it works out of the box with other applications that already support the used protocols. However, it still requires programmers to know SPARQL and the structure of the underlying ontology.
- the API solution uses JSON, which is another well known standard and is already used in many large-scale projects (such as in Freebase or del.icio.us APIs). Its main limit is that it offers less control over the ontology, but its main advantage is that it does not even require programmers to know that an ontology exists one underneath the interface. Applications can be built with any programming language and do not require any knowledge about SPARQL.

Developed tool: Places2RDF

is an example of type 1 application: it is a command-line tool that loads the `places.sqlite` file from a Firefox profile and converts it to RDF. Output is shown as text or can be directly saved inside an RDF store using SDB. We have bundled Places2RDF with a local, self contained, platform independent SPARQL endpoint built with the previously described technologies (Joseki, Jena, SDB, and HSQLDB), so that information could be immediately available to SPARQL-enabled applications.

The program has been built to be flexible enough to support different databases and ontologies. All the details about the data structure are saved inside a configuration file, which is built as an ontology: a sketch of its structure is shown in Figure 6.20.

The application relies, in a way which is very similar to the Squirrel-RDF program (more details in [50]), on the concept of *mapping* between fields inside a table and properties. In the configuration file users can specify:

- the path of the database file;

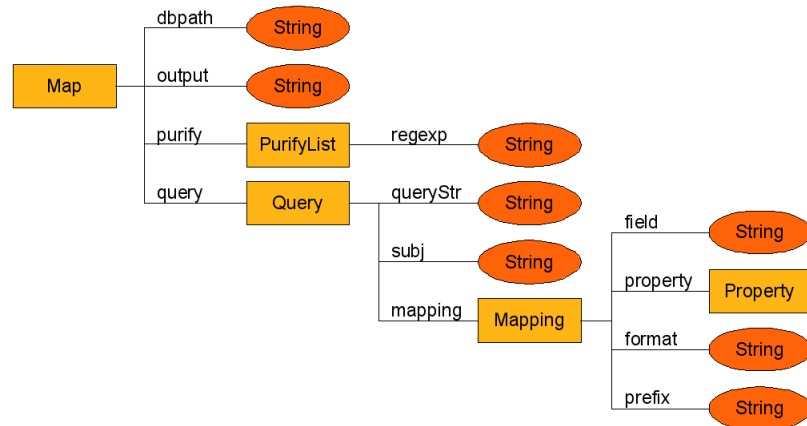


Figure 6.20: Data model for Places2RDF configuration file.

- the output format for the ontology (i.e. "N3");
- a list of regular expressions for urls that have to be filtered away (for instance, secure http connections or selected domains);
- a list of *queries*: every query contains the actual SQL query, the default subject for the triples that will be created, and a list of *mappings*. Every mapping matches a field with a particular property in the output ontology, specifies the format of the data and allows to specify a default prefix that is currently used to convert table values to custom IDs.

The final results are similar to the ones shown in Figure 6.18. One of the main advantages of this tool is that it does not depend on the particular Firefox data file, so it could be adapted to become a general conversion tool for different formats (just to name one, Google Chrome's history and bookmarks file) and different ontologies.

Developed tool: RDFMonkey Customization Extension

is an example of type 2 application. It is a Firefox extension that accomplishes the following tasks: first of all, it checks in realtime which websites the user is browsing or saving as bookmarks and gathers related data; then, it passes this information to its plugins, which can use it for different purposes.

The default action is to save data as RDF inside the ontology: this is done sending SPARQL UPDATE queries to the Joseki server. An example of such a query is the following, which updates the ontology stating that a bookmark has just been saved:

```
PREFIX HPLBrowseData: <http://hpl.hp.com/schemas/FFontology#>
PREFIX mozb: <http://hpl.hp.com/schemas/bookmarks#>
PREFIX mozh: <http://hpl.hp.com/schemas/history#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT {
  "{23592b76-e4ea-4d93-81e3-bea0d2ed9cd1}0" a HPLBrowseData:Bookmark;
    HPLBrowseData:URL <http://www.google.com>;
    HPLBrowseData:BookmarkName "Google";
    HPLBrowseData:BookmarkType <HPLBrowseData:BookmarkClass>;
    HPLBrowseData:Folder "{a6358e91-aa9d-4c36-8626-8a4bc61ebaa1}1";
    HPLBrowseData:Date "2008-09-17T17:36:54Z"^^xsd:dateTime;
}
```

To test what the advantages of having user's browsing information exposed and accessible, we developed two plugins that exploit the data saved by RDFMonkey: the former takes advantage of the open and extensible nature of RDF to enrich it with information taken from del.icio.us and provides a visualization of user's browsing history; the latter augments Web browsing, using Freebase to find information related to the current page and providing links to other potentially interesting related data sources.

6.3.4 Example plugin: del.icio.us and Google's MotionChart

Our first plugin takes advantage of the open structure of RDF to merge history information with data downloaded on the fly from del.icio.us. Its architecture is shown in Figure 6.21. For every visited page (a), it accesses del.icio.us API²⁶ and (b) gathers information about it: if the URL has been saved in the social bookmarking website, then its popularity (i.e. how many people have saved it) and its most popular tag

²⁶More info available at <http://delicious.com/help/api>.

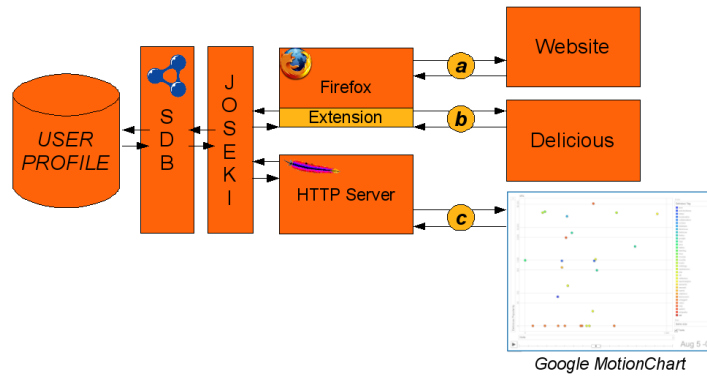


Figure 6.21: Architecture of the RDFMonkey visualization plugin.

are retrieved. This information is then saved together with the history item inside the RDF store. In this way, the downloaded data is bound to the particular date the visit took place: it is thus possible to keep track of the evolution of this information during time.

To visualize (c) the information we saved, we use Google's MotionChart²⁷. This tool allows to manage different kind of information at the same time by drawing bubbles on a plane: data can affect their position, their size, their color, and can also change in time.

MotionChart access information saved in a particular format, that is the same one used by Google Spreadsheet to save its data. To make our history (augmented with del.icio.us information) available to MotionChart, we set up a service that accesses the RDF store and automatically converts the available information in the desired format. The conversion is made easy thanks to the fact that RDF properties also describe their own datatypes, so it is possible to set the right format for the different fields automatically.

The final result inside MotionChart is shown in Figure 6.22 and 6.23. Every bubble is a visited URL and its color matches the color of the most common tag that has been used for it. A list of all the tags appears on the right and whenever the mouse pointer hovers on either a URL or a tag, their matching elements blink. For each axis it is possible to choose one of the following data sets: del.icio.us popularity, del.icio.us number of posts per date, and total number of visits by the user. The same data can be used to change the bubbles' size. A scroll bar below the graph

²⁷See <http://www.gapminder.org/graphs.html>.

6.3 Using semantics and user participation to customize personalization

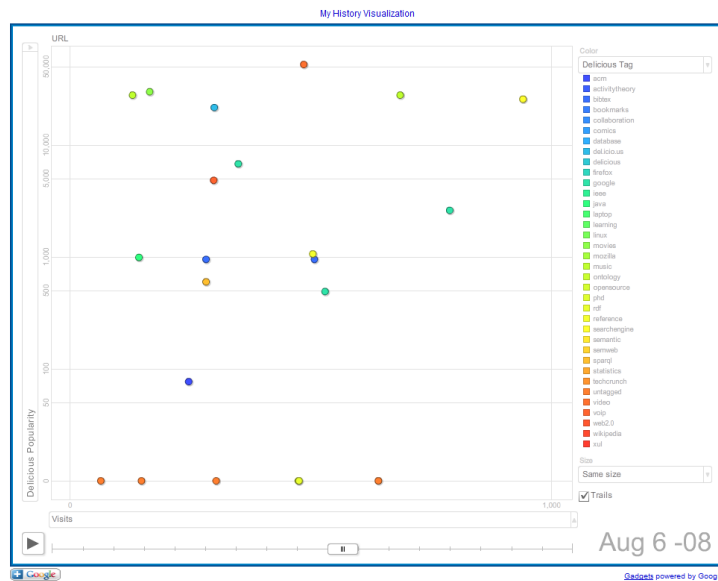


Figure 6.22: History visualization through Google's MotionChart.

allows to change the data across the timespan in which information has been gathered. A “play” button allows to automatically see the evolution of someone's history.

Having a chance to see all this information at a glance allows users to give many different interpretations to what they have visited in a period of time, depending on what's more interesting for them. For instance, Figure 6.22 shows the static information for a particular day, with number of visits on the x axis and del.icio.us popularity on the y axis: from this, a user might know whether she has visited more or less websites than in the other days; she could easily understand which websites she has visited most (as they appear on the right) and if they are also popular (as they appear on the top) or not; watching at the evolution of her history in time, she could see whether the main topics she is interested in are changing or not, and whether she is a trend follower or not.

In Figure 6.23, instead, the x axis still matches the total number of visits at a particular day but on the y axis del.icio.us posts per day are mapped, while the size of the bubbles increases with del.icio.us popularity. From this view a user could see if trends change independently from the actual popularity of a website: in Figure 6.23, for instance,

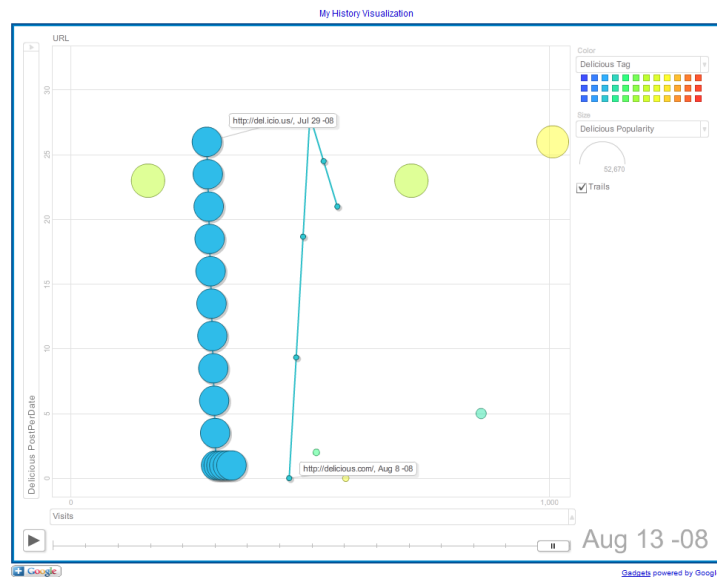


Figure 6.23: Delicious.com versus del.icio.us on MotionChart.

it is possible to see how delicious.com, the new URL for del.icio.us, is becoming more trendy than the old address, despite the fact that its overall popularity (i.e. the number of people who actually saved it) is still smaller.

6.3.5 Example plugin: Freebase

Our second plugin tries to enhance users' browsing experience by adding some links and information which are related to a webpage while the user is accessing it. The starting point for this small project is the following: a lot of topics in Freebase contain links that point to external websites which could provide additional information for the topics themselves. For instance, the topic about the band "Metallica" has links that point out to Wikipedia, Musicmoz, the New York Times "Times Topics" devoted to the band, and, of course, their official homepage.

Thus, whenever users visit one of these particular webpages, it is possible to follow the links backwards and see what topic contains them. From the topic, then, it is possible to get the matching Freebase types and some of their properties. Finally, it is possible to use these values to run other Web services that provide related information, in a way which is very similar to the one described in [49]. The architecture for this

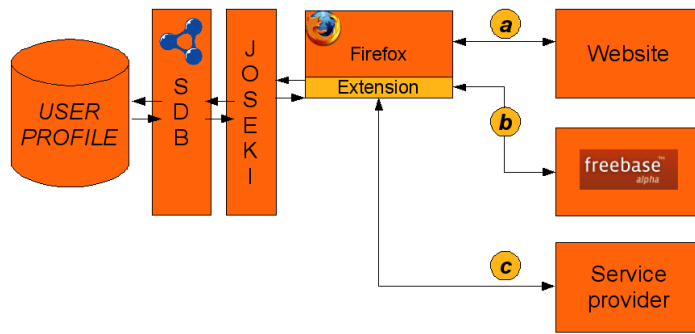


Figure 6.24: Architecture of the RDFMonkey sidebar plugin.

plugin is shown in Figure 6.24.

The tool shows the additional information inside a sidebar in the Firefox browser. The way information is presented depends on XUL²⁸ templates, which are stored in JSON format and can be easily modified by users themselves: thus, anyone can choose what kind of related information is shown by the browser and how it is presented to the user. As an example, the following code provides the information to show when a page related to a “/music/artist” is shown:

```

"/music/artist" : [
  {
    "href" : "http://www.last.fm/music/{{NAME}}",
    "image" : "chrome://rdfmonkey/skin/lastfm.png",
    "text" : "Listen to this artist on Last.fm",
    "xul" : "link",
    "NAME" : "eval(obj.source.name)"
  }
]

```

The previous code is used to load a XUL template containing an image, a link to Last.fm and a simple text. Thus, whenever a musical artist page is found, users are automatically shown a link that points to that artist’s radio on Last.fm. This is just a very simple example, which takes information already available (the name of the Freebase topic which is linking to the current page) and just links to another website. However, with more advanced templates and additional queries on freebase

²⁸XUL User Interface Language. It is a language made by Mozilla to allow programmers build rich cross-platform applications which are very easy to customize.

it is possible to create something more useful and appealing for users: for instance, Figure 6.25 shows (a) how it is possible to directly embed the Last.fm radio into the sidebar, (b) a link to Amazon that appears whenever a book-related page is shown, and (c) a map that appears for anything that is described as a location, obtained by first querying Freebase for the object's coordinates and then passing them to the Google's Maps widget which is embedded in the sidebar.

6.3.6 Plugin Evaluation

For our evaluation we mainly focused on the information we could gather from Freebase. For the tool to be actually useful, we need it to work with as many pages as possible and, at the same time, to provide information which could be considered useful by the users.

Our first task was to extract URLs appearing in Freebase as outgoing links (the ones that from now on we are going to simply call *Freebase URLs*, even if they are not Freebase's): our dataset has been built downloading the July 2008 “/common/webpage” data dump (Updated July 9, 2008) and searching for all the HTTP URLs present in the file. The total number of URLs extracted from Freebase was 310278.

Of course this number is ridiculously small when compared to the size of the whole Internet, but it has a completely different value when it is compared to the subset of the webpages that are actually visited by users. To prove this, we decided to calculate the incidence of Freebase URLs inside a dataset provided by Nielsen²⁹. These data describe the anonymized browsing history of hundreds of thousands of users, together with a classification of the visited websites and some additional information about user census. For our work we chose to use only a subset of this dataset, related to the month of January 2008: this amounts to a total of more than 200 million page hits, done by a set of more than 61000 users.

We ordered the set of visited URLs depending on how many times they had been accessed by users. We then kept the top million URLs, with the first one having 2.6 million hits and the last one only six. Inside that set, the total incidence of Freebase URLs is 0.77%: this means that out of the top million web pages only 7722 were also appearing in Freebase. This value, however, becomes much higher if we weight the URLs by the times they have been actually visited: out of about 59 millions visits, about 5.7 millions were directed to web pages that also appear in Freebase, with

²⁹<http://www.nielsen.com>.

6.3 Using semantics and user participation to customize personalization

Wikipedia is sustained by people like you. Please donate today.

Metallica
From Wikipedia, the free encyclopedia

For other uses, see *Metallica* (disambiguation).

Metallica is an American heavy metal band that formed in 1981 in Los Angeles, California. Founded when drummer Lars Ulrich posted an advertisement in a Los Angeles newspaper, Metallica's original line-up consisted of Ulrich, rhythm guitarist and vocalist James Hetfield, lead guitarist Dave Mustaine, and bassist Ron McGovney. These last two were later replaced from the band, in favor of Kirk Hammett and Cliff Burton, respectively. In September 1986, Metallica's tour bus skidded out of control and flipped, which resulted in Burton being crushed under the bus and killed. Jason Newsted replaced him less than two months later. Newsted left the band in 2001 and was replaced by Robert Trujillo in 2003.

Metallica's early releases included fast tempos, instrumentals, and aggressive musicianship that placed them as one of the "Big Four" of the thrash metal subgenre alongside Slayer, Megadeth and Anthrax. The band earned a growing fan base in the underground music community, and some critics say the 1986 release *Master of Puppets* is one of the most influential and "heavy" thrash metal albums. The band achieved substantial commercial success with its self-titled 1991 album, which debuted at number one on the *Billboard* 200. Some critics and fans believed the band changed its musical direction to appeal to the mainstream audience. With the release of *Load* in 1996, Metallica distanced itself from earlier releases in what has been described as "an almost alternative 'rock approach'", and the band faced accusations of "selling out".

In 2000, Metallica was among several artists who filed a lawsuit against Napster for sharing the band's copyright-protected material for free without the band members' consent. A settlement was reached, and Napster became a pay-to-use service. Despite reaching number one on the *Billboard* 200, the release of *St. Anger* in 2003 disappointed some critics and fans with the exclusion of guitar solos, and the "steel-sounding" snare drum. A film titled *Some Kind of Monster* documented the recording process of *St. Anger*.

Navigation:

- Main page
- Contents
- Featured content
- Current events
- Random article

Search:

Interaction:

- About Wikipedia
- Community portal
- Recent changes
- Contact Wikipedia
- Donate to Wikipedia
- Help

Toolbox:

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Cite this page

languages:

- العربية
- Aragnón
- Български
- Бразилски
- Brezhoneg

Freebase page: Metallica
Link type: **Article**
Topic: **Film actor**

IMDb
Topic: **Film music contributor**
Topic: **Musical Artist**

lost.fm

Metallica's Top Tracks

Track	Time
Metallica - Nothing Else Matters	05:27
Metallica - Enter Sandman	05:30
Metallica - One	07:54
Metallica - Master of Puppets	09:34
Metallica - The Unforgiven	06:25

Music like Metallica
Including: Megadeth, Slayer, Pantera, Iron Maiden.

Metallica's Top Tracks

Topic: Topic
Topic: **Record Producer**
Topic: **Musical Group**
Topic: **Musical Group Member**
Topic: **Breakfast Artist**
Topic: **Interesting topic**

(a)

Freebase page: Cryptonomicon
Link type: **Web Link(s)**
Topic: **Topic**
Topic: **Book**

amazon.com

Topic: Written Work
Topic: **Award-Winning Work**

(b)

Freebase page: Palo Alto
Link type: **Web Link(s)**
Topic: **Topic**
Topic: **Location**

Topic: City/Town
Topic: **Top Architectural City**
Topic: **Statistical region**
Topic: **Dated location**

(c)

Figure 6.25: Related content provided by the RDFMonkey Sidebar plugin.

an incidence of 9.75%.

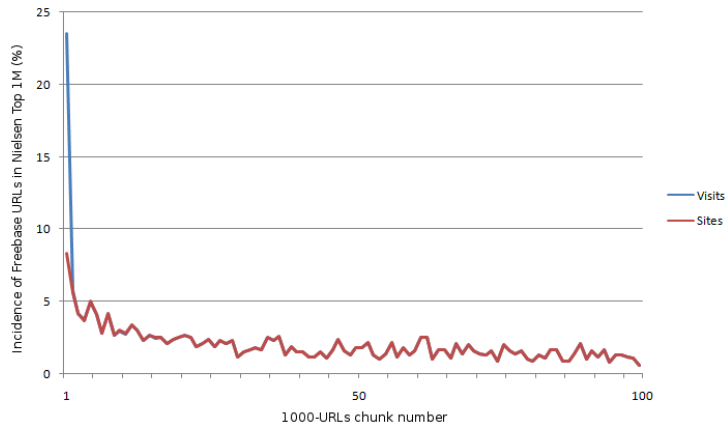
Given how much the number of visits can influence the final results, we also checked how the average incidence changed moving from the most popular URLs to the less visited ones. Thus, we divided the dataset in chunks of 1000 URLs each and we calculate incidence (both on sites and on visits) for each one of them. Then we plotted the resulting values both for each single chunk and as a global average that changes in time. The first plot is shown in Figure 6.26(a), where the top 100 chunks appear with their respective incidences: only the very first ones have significative values, then they stabilize around a very low threshold. After the first few chunks, the values for sites and visits also appear to be very similar as the weights tend to be distributed in a much more homogeneous way (i.e. the difference between the first URL and the 1000th is more than 2.6 million hits, the one between the 1000th and the 2000th is only 1800 hits). The second plot is shown in Figure 6.26(b) and shows how the total incidence changes while more chunks are taken into account. As expected, it decreases quickly at the very beginning and then almost stabilizes around the final value.

Checking the top URLs in Nielsen database we noticed that particular categories of links (such as search engines and online email providers) were more common than others. As we are interested in just particular classes of pages (i.e. we are not going to find our personal email messages linked into Freebase), we decided to take advantage of Nielsen’s website categorization to run the same kind of statistics for each available class. The result is shown in Figure 6.27 and presents the incidence of Freebase URLs in each category, weighted by the number of visits. Out of the 15 categories, the one named “Search Engines/Portals & Communities” has the highest percentage (more than 23% of the visits are directed towards websites that are also linked inside Freebase); it is then followed by “News & Information” with more than 11% and then by “Finance/Insurance/Investment” with about 7%; all the other categories have an incidence which is below 5%.

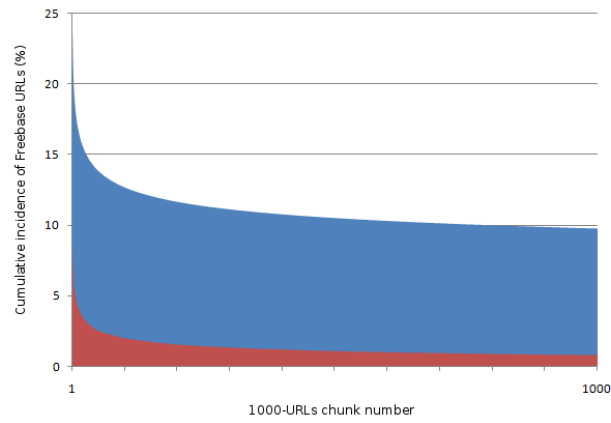
Knowing which classes of URLs have a higher incidence of freebase URLs help us give an interpretation to the previous results: for instance, despite having high ranked URLs in Nielsen data, the category “Telecom/Internet Services” does not have a very high incidence, while News—which could provide more interesting related information—has a higher value.

Finally, it is worth noting that there are some websites which almost do not appear in the list of Freebase URLs, but that could be easily

6.3 Using semantics and user participation to customize personalization



(a)



(b)

Figure 6.26: Incidence of freebase URLs inside the Top 1 million visited urls: values for single 1000-URLs chunks (a) and behavior of the overall value as the set grows bigger (b).

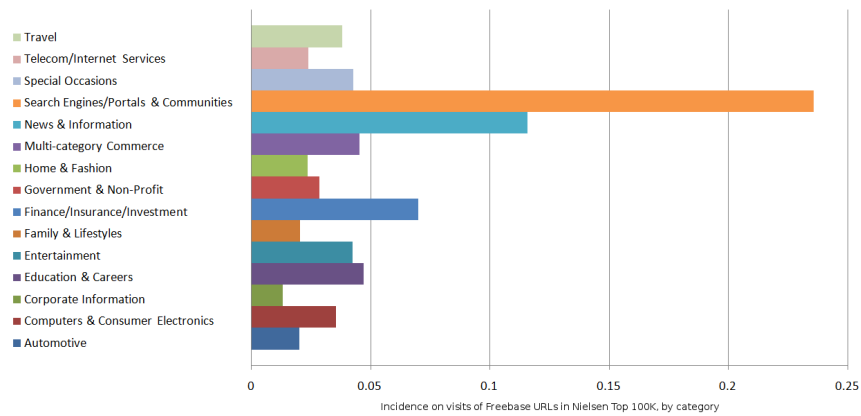


Figure 6.27: Incidence of Freebase URLs inside visits, divided into Nielsen categories.

linked automatically. These are all those websites that are used as *authorities*, that is the ones that provide resources with a unique id such as Wikipedia, IMDB, Netflix and so on. As these IDs are also imported into Freebase, it is possible to link to a related Freebase topic starting from any page in these systems. This means that our tool could be particularly useful for specific classes of users, such as Wikipedia contributors or movie aficionados.

After checking the incidence of URLs, we decided to focus on Freebase types. The reason is that we are not just interested into having a particular page linked into Freebase, but also into having interesting related information about it. Basically we assumed that the more the types and the properties the topic has, the highest amount of potentially interesting related information it can provide.

Thus, we first ranked Freebase types according to their usage. Freebase has more than 4000 types and their distribution is shown in Figure 6.28. In the top 15 types (see Table 6.1) there are of course “common topics”, that is one of the most generic (and from our point of view unuseful) Freebase types, but also musical tracks, artists and albums, people, and locations. In the top 100 a lot of other potentially useful types are present, such as cities, films and actors, books and book authors, restaurants, companies, and videogames.

Thus, if it is true that there are a lot of topics whose type is “/common/topic”, it is also true that there are many which share other types. Moreover, even in the worst case, they can provide some additional in-

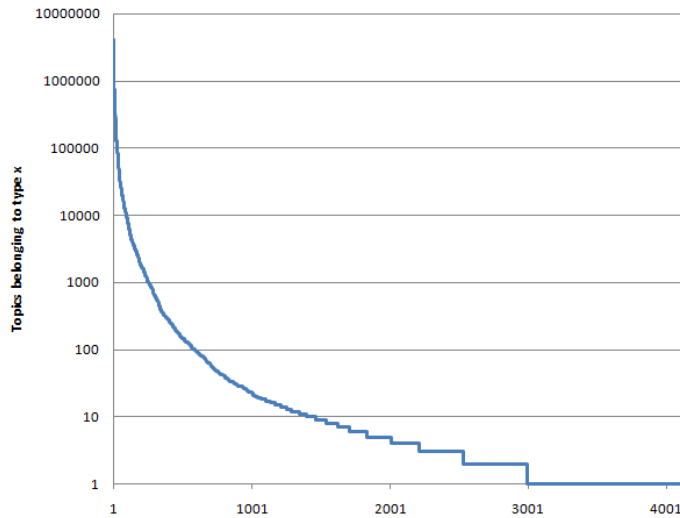


Figure 6.28: Distribution of Freebase types: the most used one (/music/track) is associated to 4.1 million topics.

formation through their own links, for instance to Wikipedia: in fact, out of all the topics which share the common type, only 21810 do not have links to Wikipedia, while the remaining 1.1 million link at least to their matching Wikipedia article.

6.3.7 Conclusions

Both personalization and customization are techniques which can bring value to a system, making the user feel like it is more “personal”. Currently, however, while more and more systems allow users themselves to apply their own customizations (such as UI skins, security settings, or personal preferences), there are still few which allow users to directly access and use their personal information.

Looking at the huge success many customization tools had, not just on a particular set of expert users but also on the wide Internet public, we decided to make personalization customizable, that is to allow users access their own personal profile and build custom applications that exploit it.

To do this, we chose a specific use case: the information that users leave when they use their browsers, that is the history of the visited websites and the collection of their bookmarks. We then converted this

Type	Topics
/music/track	4110740
/common/topic	3911182
/common/document	2607367
/people/person	775876
/type/content_import	773153
/type/content	747299
/common/image	705154
/media_common/creative_work	511159
/music/release	505623
/location/location	500534
/music/album	464239
/music/artist	365760
/common/webpage	331361
/location/geocode	262923
/business/employer	235032

Table 6.1: The top 15 types in Freebase, ordered by number of topics.

information from the proprietary format used by Firefox 3 in an open, RDF-based format. We built tools to do the conversion offline or in realtime and a browser extension that uses these data to provide new, potentially interesting, information. The whole set of tools is built to be extensible, so it makes heavy use of configuration files that can be easily modified by users to provide additional features.

As one of our prototypes aims at enhancing the browsing experience by providing related information gathered from Freebase, our evaluation focused on Freebase links and types. Using the Nielsen database, we drew statistics about the incidence of Freebase URLs inside the top million visited websites and we found promising results: while the percentage of top Nielsen links that also appear in Freebase is still pretty low (0.77%), the percentage of *visits* that take to a link belonging to freebase is rather high (9.75%). We repeated the same calculations over 15 different groups of URLs, belonging to the different website categories provided by Nielsen, and were able to detect how the incidence is distributed across the groups. Finally, trying to detect among the Freebase topics which ones were more useful from our point of view, we ordered Freebase types by usage and analyzed the results: even if the most generic types appear in the most used ones, there are also many potentially interesting types in the top 100 that could be exploited to

have useful related information.

One thing that is missing in this project, due to its time constraints, is an analysis of the potential security problems of such a system. We are aware of the fact that merging our personal data with external services is a potential risk, as whenever we get related information we are also providing ours to all the services we query. We are currently leaving the decision of sharing the profile to users themselves, but the problem is definitely worth our attention and will be studied with more detail.

7 Conclusions and Future Work

Participative systems are a topic which is object of great interest, not only as a possibly successful business, but also as an object of academic research, presenting challenges that can be approached from the point of views of many different sciences.

At the same time, however, even successful systems are characterized by a strong limitation: user contributions are often unstructured and published just for human consumption, lacking the structure that would be useful for a machine to elaborate them automatically. As a result, users often produce huge amounts of information which is hard to find and reuse. Our research tried to address this problem, searching an answer to the following question: given a community, a task, and a context, what is the best tool that could be used to exploit user participation to produce useful information? And how can we make this tool better with semantics?

To answer the first part of the question, we studied participative systems without limiting ourselves to a single point of view, but trying to consider both the technical and the social aspects of this subject. Analyzing both successful and unsuccessful examples of such systems on the Internet, we concluded that their popularity often depends on many parameters which are related not only with the tools themselves, but also with the communities that use them, their activities and the context in which they take place. For the same reason, it is difficult to classify these systems depending on the activity alone (as the community might decide to force the tool on a task it was not planned for) or on the way users participate.

To better understand the different kinds of activities users might perform, we provided a description of the main social systems and we suggested a taxonomy which describes the different levels of participation inside one of these systems. We then introduced basic concepts about social interactions and used them to describe what happens when a group of people shares a common practice. Starting from these concepts we developed a design approach that takes into account users, the system as a whole, and the dynamics inside the community that gathers around it. As a result, we provide a set of design patterns and suggestions that

can be used to bootstrap or fine tune a social system.

To answer the second part of our research question, we focused on some classes of participative systems and studied the different ways semantics were used to improve them. We defined a methodology that uses semantics on three different levels, respectively with the purpose of linking data, better describing and managing knowledge, and inferring new information with reasoning. When ontologies are involved, semantics can be provided on different layers with respect to the system: for this we devised a “temple model” which includes system, context, domain and upper level ontologies. We then used these models to develop our semantic applications.

Our experimental developments have been divided into three categories: semantic wikis, folksonomies (that is, interactions between folksonomies and ontologies), and linking open data. These categories do not only match different tools, but also different levels of semantics (from high to low) and user interaction (from less to more automatic). Albeit heterogeneous these experiments might seem, they are all based on the same theoretical bases and show how versatile our approach is. Moreover, they also carry some specific novel results:

- with our semantic wiki prototypes, we have explored new ways to extend a wiki with semantics: ontologies could be used to describe generic relations between articles (considered as an element of the wiki and not as the topic they describe), to manage attachment metadata, or to automatically create customized templates to associate structured information with a wiki article (see Chapter 4);
- our experiments on folksonomies show how tags could be automatically mapped with concepts inside an ontology. We also show how semantic disambiguation techniques could be used to solve (part of) the limits which are characteristic of folksonomies, such as homonymy, synonymy, and basic level variations. Finally, we suggest new interfaces to browse and search tags which exploit the mapping between folksonomies and ontologies (see Chapter 5);
- trying to link information available in different systems, we showed how to exploit already existing data repositories such as Freebase to create new types of user incentives. At the same time, we created new potential sources of linked data by converting widely used formats such as IMAP and browser history into a standard, shared, RDF-based representation (see Chapter 6);

During the process of evaluation of our systems we collected and analyzed huge quantities of data such as tag collections, knowledge bases, and server logs. When possible, we made the collection tools available to the public so that these datasets could be replicated. At the same time, we showed the results of the elaboration of these data, providing some interesting findings:

- we have shown that the distribution of tags inside a popular folksonomy is such that the a high percentage of the most used tags are also nouns in the English language;
- we have found that different classes of tags have very different statistical behaviors. For instance, “task organizing” tags such as *toread* can be very popular but there is no collective agreement on what has to be tagged with them, so they do not converge on the same resource;
- we have studied the distribution of external URLs in Freebase, showing that a good percentage of them also belongs to the most visited websites. Moreover, we have described the distribution of Freebase types showing that there is a huge quantity of topics that could provide potentially interesting related information.

7.1 Future Work

As it often happens in research, our future work todo list is much longer than the list of what we have already done: this is partly due to the limits we spotted in our own research, and partly to the speed this topic is growing, carrying with itself new emerging trends and interesting problems to solve. What we describe here is just the top of our prioritized list.

Even if our work has provided some good results, it could still be made more complete. Both the description of social systems and the list of dimensions are dynamic and can change whenever a new system comes out: rather than just updating them each time, our plan is to share this information with a community which could be interested in the topic and maintain it with the help of other motivated people.

Another future goal is to increase our knowledge of incentives, not just in theory but also developing and releasing an experimental system that would allow us to have a real user base: this would also help us in evaluating our systems from the users’ point of view. Speaking

about evaluations, even if we agree on the position expressed in [147] we think that a *participative* semantic web application should have one more dimension which is related to its social component: thus, we are planning to define an evaluation methodology which takes into account user interfaces, quality of data, algorithms, and social interactions.

Finally, we believe that our future direction is going to be strongly influenced by the developments in the field of linked data. We think this is a very interesting field from which participative systems can benefit most and to which they can provide new value, in a virtuous cycle of participation and semantics. Currently there are enormous quantities of already structured data and it is easy to get excited when we see them linked together, however we do not have to forget that this information is finite and it has been provided by people in the first place. Finding ways to allow people participate not only to the Web, but also to the Web of Data is important now and is going to be even more important in our future.

Bibliography

- [1] *International Workshop on Semantic Web Personalization, Budva, Montenegro*, 06 2006.
- [2] S. F. Adafre and M. de Rijke. Discovering missing links in wikipedia. 2005.
- [3] H. S. Al-Khalifa and H. C. Davis. Towards better understanding of folksonomic patterns. In *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia*, pages 163–166, New York, NY, USA, 2007. ACM Press.
- [4] J. Ali. Licensed vs. user-generated video, July 2006. <http://www.imediaconnection.com/content/10357.asp> [accessed 06-Oct-2008].
- [5] S. Angeletou, M. Sabou, L. Specia, and E. Motta. Bridging the gap between folksonomies and the semantic web: An experience report. In *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*, pages 30–43, 2007.
- [6] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandečić. The two cultures: mashing up web 2.0 and the semantic web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.
- [7] A. Ankolekar and D. Vrandečić. Kalpana - enabling client-side web personalization. In P. Brusilovsky and H. C. Davis, editors, *Hypertext*, pages 21–26. ACM, 2008.
- [8] J. Atwood. Youtube: The big copyright lie, October 2007. <http://www.codinghorror.com/blog/archives/000972.html> [accessed 06-Oct-2008].
- [9] S. Auer, C. Bizer, J. Lehmann, G. Kobilarov, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon,

- J. Golbeck, P. Mika, D. Maynard, G. Schreiber, and P. Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007)*, Busan, South Korea, volume 4825 of *LNCS*, pages 715–728, Berlin, Heidelberg, November 2007. Springer Verlag.
- [10] S. Auer, S. Dietzold, and T. Riechert. Ontowiki -a tool for social, semantic collaboration. In I. C. et al. (Eds.), editor, *Proceedings of 5th International Semantic Web Conference, Nov 5th-9th, Athens, GA, USA, LNCS 4273*, Berlin Heidelberg, 2006. Springer-Verlag.
- [11] D. Aumüller and S. Auer. Towards a semantic wiki experience – desktop integration and interactivity in wiksar. In S. Decker, J. Park, D. Quan, and L. Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, pages 212 – 217, Galway, Ireland, November 2005.
- [12] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logics Handbook: Theory, Implementations, and Applications*. Cambridge University Press, 2003.
- [13] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
- [14] J. Bacher, R. Hoehndorf, and J. Kelso. Bowiki: Ontology-based semantic wiki with abox reasoning. In C. L. 0002, S. Schaffert, H. Skaf-Molli, and M. Völkel, editors, *SemWiki*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [15] J. E. Bardram. I love the system — i just don’t use it! In *GROUP ’97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 251–260, New York, NY, 1997. ACM Press.
- [16] T. Berners-Lee. Linked data. World wide web design issues, July 2006.
- [17] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.
- [18] S. Bindelli, C. Criscione, C. A. Curino, M. L. Drago, D. Eynard, and G. Orsi. Improving search and navigation by combining ontologies and social tags. In *1st International Workshop on Ambient Data Integration*, 2008. (to appear).

- [19] C. Bizer. D2r map - a database to rdf mapping language. In *WWW (Posters)*, 2003.
- [20] C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the web, 2007.
- [21] C. Bizer and A. Seaborne. D2rq - treating non-rdf databases as virtual rdf graphs. In *ISWC2004 (posters)*, November 2004.
- [22] S. Bloehdorn, O. Görlitz, S. Schenk, and M. Völkel. Tagfs - tag semantics for hierarchical file systems. In *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06)*, Graz, Austria, September 6-8, 2006, September 2006.
- [23] U. Bojars, A. Passant, J. G. Breslin, and S. Decker. Social network and data portability using semantic web technologies. In *2nd Workshop on Social Aspects of the Web (SAW 2008) at BIS2008*, pages 5–19, 2008.
- [24] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):article 11, October 2007.
- [25] S. Braun, A. Schmidt, A. Walter, G. Nagypál, and V. Zacharias. Ontology maturing: a collaborative web 2.0 approach to ontology engineering. In N. F. Noy, H. Alani, G. Stumme, P. Mika, Y. Sure, and D. Vrandečić, editors, *CKC*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
- [26] J. Breslin, A. Harth, U. Bojars, and S. Decker. Towards Semantically-Interlinked Online Communities. In *Second European Semantic Web Conference, ESWC 2005, Heraklion,, 2005*.
- [27] D. Bricklin. The cornucopia of the commons: How to get volunteer labor, August 2000. <http://www.bricklin.com/cornucopia.htm>.
- [28] S. L. Bryant, A. Forte, and A. Bruckman. Becoming wikipedia: Transformation of participation in a collaborative online encyclopedia. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 1–10, New York, NY, USA, 2005. ACM Press.
- [29] M. H. Butler, J. Gilbert, A. Seaborne, and K. Smathers. Data conversion, extraction and record linkage using xml and rdf tools in project simile. research report, August 2004.

- [30] D. Calvanese, G. D. Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier Science Publishers, 2001.
- [31] A. Cheyer and J. Levy. A collaborative programming environment for web interoperability. In Völkel and Schaffert [152].
- [32] N. A. Chinchor. Proceedings of the Seventh Message Understanding Conference (MUC-7) named entity task definition. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, page 21 pages, Fairfax, VA, April 1998. version 3.5, http://www.itl.nist.gov/iaui/894.02/related_projects/muc/.
- [33] M. Crispin. Internet message access protocol - version 4rev1, March 2003. <http://tools.ietf.org/html/rfc3501>.
- [34] C. Curino, G. Orsi, and L. Tanca. X-som: A flexible ontology mapper. In *DEXA Int. Workshop on Semantic Web Architectures For Enterprises (SWAE)*, 2007.
- [35] C. Curino, E. Quintarelli, and L. Tanca. Ontology-based information tailoring. In *Proc. IEEE of 2nd Int. Workshop on Database Interoperability (InterDB 2006)*, pages 5–5, Atlanta, USA, April 2006.
- [36] C. V. Damme, T. Coenen, and E. Vandijck. Turning a corporate folksonomy into a lightweight corporate ontology. In W. Abramowicz and D. Fensel, editors, *BIS*, volume 7 of *Lecture Notes in Business Information Processing*, pages 36–47. Springer, 2008.
- [37] C. V. Damme, M. Hepp, and K. Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*, pages 57–70, 2007.
- [38] F. Dawson and D. Stenerson. Internet calendaring and scheduling core object specification (icalendar). RFC 2445 (Proposed Standard), November 1998.
- [39] K. Dello, E. P. B. Simperl, and R. Tolksdorf. Creating and using semantic web information with makna. In Völkel and Schaffert [152].

- [40] J. Domingue, M. Dzbor, and E. Motta. Collaborative semantic web browsing with magpie. In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 388–401. Springer, 2004.
- [41] F. Echarte, J. J. Astrain, A. Córdoba, and J. E. Villadangos. Ontology of folksonomy: A new modelling method. In S. Handschuh, N. Collier, T. Groza, R. Dieng, M. Sintek, and A. de Waard, editors, *SAAKM*, volume 289 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
- [42] M. Eirinaki, D. Mavroeidis, G. Tsatsaronis, and M. Vazirgianis. Introducing semantics in web personalization: The role of ontologies. In M. Ackermann, B. Berendt, M. Grobelnik, A. Hotho, D. Mladenic, G. Semeraro, M. Spiliopoulou, G. Stumme, V. Svátek, and M. van Someren, editors, *EWMF/KDO*, volume 4289 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2005.
- [43] J. Engeström. Why some social network services work and others don’t — or: the case for object-centered sociality, April 2005. http://www.zengestrom.com/blog/2005/04/why_some_social.html [accessed 10-Oct-2008].
- [44] Y. Engeström. *Learning by expanding: an activity-theoretical approach to developmental research*. Orienta-Konsultit Oy., Helsinki, 1987.
- [45] M. Erdmann, A. Maedche, H. Schnurr, and S. Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In P. Buitelaar and K. H. (eds.), editors, *Proc. of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*. Morgan Kaufmann, August 2000.
- [46] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
- [47] D. Eynard. Powerbrowsing, 2005. <http://davide.eynard.it/malawiki/PowerBrowsingEn>. Accessed 10-Sep-2008.
- [48] D. Eynard. Using semantics and user participation to customize personalization. Technical report, HP Labs, 2008.

- [49] D. Eynard and M. Colombetti. Exploiting user gratification for collaborative semantic annotation. In *Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*, 2008.
- [50] D. Eynard, J. Recker, and C. Sayers. An imap plugin for squirrel-rdf. Technical report, HP Labs, 2007.
- [51] C. Fellbaum. *WordNet – An Electronic Lexical Database*. MIT Press, 1998.
- [52] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
- [53] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [54] D. Flejter, editor. *Mailing Lists Meet The Semantic Web*, 2007.
- [55] N. E. Fuchs, K. Kaljurand, and G. Schneider. Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In G. Sutcliffe and R. Goebel, editors, *FLAIRS Conference*, pages 664–669. AAAI Press, 2006.
- [56] D. Gillmor. *We the Media: Grassroots Journalism by the People, for the People*. O’Reilly Media , Inc., Sebastopol, CA, August 2004. Chapter 2: The Read-Write Web.
- [57] M. Gladwell. *The Tipping Point: How Little Things Can Make a Big Difference*. Back Bay Books, January 2002.
- [58] S. Golder and B. A. Huberman. The structure of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, April 2006.
- [59] T. Gruber. Ontology of folksonomy: A mash-up of apples and oranges, 2005. <http://tomgruber.org/writing/ontology-of-folksonomy.htm>.
- [60] T. Gruber. Tagontology - a way to agree on the semantics of tagging data, 2005. <http://tomgruber.org/writing/tagontology-tagcamp-talk.pdf>.

- [61] S. Guelich, S. Gundavaram, and G. Birznieks. *CGI programming with Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, second edition, 2000.
- [62] N. Gulley. In praise of tweaking: a wiki-like programming contest. *interactions*, 11(3):18–23, 2004.
- [63] V. Haarslev and R. Möller. Description logic systems. In *The Description Logic Handbook*, chapter 8, pages 282–305. Cambridge University Press, 2003.
- [64] P. Haase, M. Ehrig, A. Hotho, and B. Schnizler. Personalized information access in a bibliographic peer-to-peer system. In S. Staab and H. Stuckenschmidt, editors, *Peer-to-Peer and SemanticWeb, Decentralized Management and Exchange of Knowledge and Information*, pages 143–158. Springer, 2006.
- [65] T. Hammond, T. Hannay, B. Lund, and J. Scott. Social bookmarking tools (i): A general review. *D-Lib Magazine*, 11, Apr 2005.
- [66] T. Heath and E. Motta. Revyu.com: A reviewing and rating site for the web of data. In K. Aberer, K.-S. Choi, N. F. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 895–902. Springer, 2007.
- [67] J. Hendler. Semantics and the network effect: A little semantics goes a long way. In *SemGrail 2007 Workshop*, Redmond, Washington, USA, June 2007. Position paper.
- [68] M. Hepp, K. Siorpaes, and D. Bachlechner. Harvesting wiki consensus: Using wikipedia entries as vocabulary for knowledge management. *IEEE Internet Computing*, 11(5):54–65, 2007.
- [69] B. Horowitz. Creators, synthesizers, and consumers, 2006. <http://www.elatable.com/blog/2006/02/17/creators-synthesizers-and-consumers/> [accessed 16-Oct-2008].
- [70] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

- [71] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
- [72] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From *SHIQ* and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [73] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Trend detection in folksonomies. In Y. S. Avrithis, Y. Kompatsiaris, S. Staab, and N. E. O’Connor, editors, *Proc. First International Conference on Semantics And Digital Media Technology (SAMT)*, volume 4306 of *LNCS*, pages 56–70, Heidelberg, dec 2006. Springer.
- [74] P. M. Johnson. A glossary of political economy terms: Incentive, 2005. <http://www.auburn.edu/~johnspm/gloss/> [accessed 13-Oct-2008].
- [75] J. Kahan and M.-R. Koivunen. Annotea: an open rdf infrastructure for shared web annotations. In *WWW ’01: Proceedings of the 10th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2001. ACM Press.
- [76] N. Kennedy. Sniff browser history for improved user experience, 2008. <http://www.niallkennedy.com/blog/2008/02/browser-history-sniff.html>; accessed 08-Sep-2008.
- [77] H. L. Kim, J. G. Breslin, S.-K. Yang, and H.-G. Kim. Social semantic cloud of tag: Semantic model for social tagging. In N. T. Nguyen, G. Jo, R. J. Howlett, and L. C. Jain, editors, *KES-AMSTA*, volume 4953 of *Lecture Notes in Computer Science*, pages 83–92. Springer, 2008.
- [78] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Journal of Web Semantics: Science, Services and Agents on the World Wide We*, 2(1):49–79, 2004.
- [79] P. Kollock. The economics of online cooperation: Gifts and public goods in cyberspace. In M. Smith and P. Kollock, editors, *Communities in Cyberspace*. Routledge, London, 1998.
- [80] E. Kroski. The hive mind: Folksonomies and user-based tagging, Dec 2005. <http://infotangle.blogspot.com/2005/12/07/the-hive-mind-folksonomies-and-user-based-tagging/>.

- [81] M. Krötzsch, S. Schaffert, and D. Vrandečić. Reasoning in semantic wikis. In G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P.-L. Patranjan, and R. Tolksdorf, editors, *Reasoning Web*, volume 4636 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2007.
- [82] M. Krötzsch, D. Vrandečić, M. Völkel, H. Haller, and R. Studer. Semantic wikipedia. *Journal of Web Semantics*, DEC 2007. To appear.
- [83] T. Kuhn. Acewiki: Collaborative ontology management in controlled natural language. In C. L. 0002, S. Schaffert, H. Skaf-Molli, and M. Völkel, editors, *SemWiki*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [84] K. Kuutti. Activity theory as a potential framework for human-computer interaction research. pages 17–44, 1995.
- [85] C. Lange. Swim - a semantic wiki for mathematical knowledge management. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 832–837. Springer, 2008.
- [86] D. Laniado, D. Eynard, and M. Colombetti. A semantic tool to support navigation in a folksonomy. In *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia*, pages 153–154, New York, NY, USA, 2007. ACM Press.
- [87] D. Laniado, D. Eynard, and M. Colombetti. Using wordnet to turn a folksonomy into a hierarchy of concepts. In *Semantic Web Application and Perspectives - Fourth Italian Semantic Web Workshop*, pages 192–201, Dec 2007.
- [88] J. Lanier. Digital maoism: The hazards of the new online collectivism, 2006. http://www.edge.org/3rd_culture/lanier06/lanier06_index.html.
- [89] O. Lassila and R. Swick. Resource description framework (rdf) model and syntax specification, February 1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>.
- [90] J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, New York, 1991.

- [91] C. Leacock, M. Chodorow, and G. A. Miller. Using corpus statistics and wordnet relations for sense identification. *Computational Linguistics*, 24(1):147–165, 1998.
- [92] B. Leuf and W. Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley, 2001.
- [93] L. Liu and M. T. Oszu. *Encyclopedia of Database Systems*. Springer, 2008.
- [94] B. Lund, T. Hammond, M. Flack, and T. Hannay. Social bookmarking tools (ii): A case study - connotea. *D-Lib Magazine*, 11, Apr 2005.
- [95] C. Marlow, M. Naaman, D. Boyd, and M. Davis. Ht06, tagging paper, taxonomy, flickr, academic article, to read. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 31–40, New York, NY, USA, 2006. ACM Press.
- [96] B. Mason and S. Thomas. A million penguins research report. Technical report, Institute of Creative Technologies, De Montfort University, Leicester, UK, April 2008. <http://www.ioct.dmu.ac.uk/projects/amillionpenguinsreport.pdf>.
- [97] A. Mathes. Folksonomies - cooperative classification and communication through shared metadata. <http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html>, December 2004.
- [98] R. Mayfield. Power law of participation, 2006. http://ross.typepad.com/blog/2006/04/power_law_of_pa.html [accessed 16-Oct-2008].
- [99] S. Mazzocchi. Folksologies: de-idealizing ontologies, April 2005. <http://www.betaversion.org/~stefano/linotype/news/85/>.
- [100] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language overview. W3C recommendation, World Wide Web Consortium, February 2004.
- [101] U. A. Mejias. A delicious study, December 2004. http://ideant.typepad.com/ideant/2004/12/a_delicious_stu.html.
- [102] U. A. Mejias. Tag literacy, April 2005. http://ideant.typepad.com/ideant/2005/04/tag_literacy.html.

- [103] E. Menchen. Feedback, motivation and collectivity in a social bookmarking system, 06 2005. <http://kairosnews.org/node/4338>.
- [104] P. Merholz. Metadata for the masses, oct 2004. <http://adaptivepath.com/publications/essays/archives/000361.php>.
- [105] P. Mika. Ontologies are us: A unified model of social networks and semantics. In *International Semantic Web Conference*, LNCS, pages 522–536. Springer, 2005.
- [106] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. Owl 2 web ontology language: Profiles. W3c working draft, W3C, October 2008.
- [107] R. Newmann. Tag ontology design, 2005. <http://www.holygoat.co.uk/projects/tags/>.
- [108] M. Noll and C. Meinel. Web search personalization via social bookmarking and tagging. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, G. Schreiber, and P. Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007)*, Busan, South Korea, volume 4825 of *LNCS*, pages 365–378, Berlin, Heidelberg, November 2007. Springer Verlag.
- [109] D. A. Norman. *The design of everyday things*. Basic Books, New York, 2002.
- [110] T. O’Reilly. What is web 2.0. design patterns and business models for the next generation of software. September 2005.
- [111] E. Oren. Semperwiki: a semantic personal wiki. In S. Decker, J. Park, D. Quan, and L. Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, pages 107 – 122, Galway, Ireland, November 2005.
- [112] E. Oren, K. Möller, S. Scerri, S. Handschuh, and M. Sintek. What are semantic annotations? Technical report, DERI Galway, 2006.
- [113] H. V. D. Parunak, T. C. Belding, R. Hilscher, and S. Brueckner. Modeling and managing collective cognitive convergence. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *AA-MAS (3)*, pages 1505–1508. IFAAMAS, 2008.

- [114] A. Passant. Using Ontologies to Strengthen Folksonomies and Enrich Information Retrieval in Weblogs. In *Proceedings of the First International Conference on Weblogs and Social Media (ICWSM)*, Boulder, Colorado, March 2007.
- [115] S. Patwardhan, T. Pedersen, and S. Banerjee. Sensere-late::targetword - a generalized framework for word sense disambiguation. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 73–76, Ann Arbor, MI, June 2005.
- [116] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *AAAI*, pages 1024–1025, 2004.
- [117] E. Prud’hommeaux and A. Seaborne. Sparql query language for rdf. W3c recommendation, W3C, January 2008.
- [118] E. Quintarelli. Folksonomies: power to the people, June 2005. <http://www-dimat.unipv.it/biblio/isko/doc/folksonomies.htm>.
- [119] E. Quintarelli, L. Rosati, and A. Resmini. Facetag: Integrating bottom-up and top-down classification in a social tagging system. In *IA Summit 2007*, 2007.
- [120] C. Rahhal, H. Skaf-Molli, and P. Molli. Swooki: A peer-to-peer semantic wiki. In C. L. 0002, S. Schaffert, H. Skaf-Molli, and M. Völkel, editors, *SemWiki*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [121] L. Reeve and H. Han. Semantic annotation for semantic social networks using community resources. *AIS SIGSEMIS Bulletin*, 2(3-4):52–56, 2005.
- [122] L. H. Reeve and H. Han. Survey of semantic annotation platforms. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 1634–1638. ACM, 2005.
- [123] E. A. M. Ruiz-Casado and P. Castells. From wikipedia to semantic relationships: a semi-automated annotation approach. 2006.
- [124] M. Sabou, M. d’Aquin, and E. Motta. Using the semantic web as background knowledge for ontology mapping. In P. Shvaiko, J. Euzenat, N. F. Noy, H. Stuckenschmidt, V. R. Benjamins, and

- M. Uschold, editors, *Ontology Matching*, volume 225 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
- [125] M. Saleem. Why the wisdom of crowds fails on digg, 2006. <http://themulife.com/?p=145>. Accessed 11-Oct-2008 from its cached copy at <http://www.google.com/search?q=cache:Uj2wkQrBeCAJ:www.themulife.com/wp-trackback.php>
- [126] L. Sauermann and S. Schwarz. Gnowsis adapter framework: Treating structured data sources as virtual rdf graphs. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the ISWC 2005*, number 3729 in LNCS, page p. 1016 ff., Galway, Ireland, November 6-10, 2005 2005. Springer.
- [127] S. Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *1st International Workshop on Semantic Technologies in Collaborative Applications (STICA'06)*, Manchester, UK, June 2006.
- [128] S. Schaffert, D. Bischof, T. Bürger, A. Gruber, W. Hilzensauer, and S. Schaffert. Learning with semantic wikis. In Völkel and Schaffert [152].
- [129] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme. Mining association rules in folksonomies. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Žiberna, editors, *Data Science and Classification. Proceedings of the 10th IFCS Conf.*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 261–270, Heidelberg, July 2006. Springer.
- [130] C. Schwarm and M. Sevier. Gocollab – peer to peer document collaboration, 2005. <http://gnomejournal.org/article/31/gocollab----peer-to-peer-document-collaboration>.
- [131] T. Segaran. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly, Sebastopol, CA, USA, 2007.
- [132] H. Shepard, H. Halpin, and V. Robu. The dynamics and semantics of collaborative tagging. In *Proc. of the 1st Semantic Authoring and Annotation Workshop (SAAW2006)*, 2006.
- [133] C. Shirky. Ontology is overrated: Categories, links, and tags, 2005. http://www.shirky.com/writings/ontology_overrated.html.

- [134] D. Sifry. The state of the live web, april 2007. Technical report, Technorati, 2007.
- [135] D. Sims and R. Dornfest. Steven johnson on “emergence”, 2 2002. <http://www.oreillynet.com/pub/a/network/2002/02/22/johnson.html> [accessed 07-Oct-2008].
- [136] R. Sinha. A cognitive analysis of tagging, September 2005. http://www.rashmishinha.com/archives/05_09/tagging-cognitive.html.
- [137] R. Sinha. A social analysis of tagging, January 2006. http://www.rashmishinha.com/archives/06_01/social-tagging.html.
- [138] K. Siorpaes and M. Hepp. myontology: The marriage of ontology engineering and collective intelligence. In *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*, pages 127–138, 2007.
- [139] E. Sirin and B. Parsia. Pellet: An owl dl reasoner. In *Description Logics*, 2004.
- [140] D. Steer. Squirrelrdf, 2006. <http://jena.sourceforge.net/SquirrelRDF/>.
- [141] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.
- [142] J. Surowiecki. *The wisdom of crowds*. Doubleday, 2004.
- [143] A. Swartz. Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.
- [144] R. Tazzoli, P. Castagna, and S. E. Campanini. Towards a Semantic WikiWikiWeb. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, 2004.
- [145] C. W. Thompson, P. Pazandak, V. Vasudevan, F. Manola, M. Palmer, G. Hansen, and T. J. Bannon. Intermediary architecture: Interposing middleware object services between web client and server. *ACM Comput. Surv.*, 31(2es):14, 1999.
- [146] C. Tziviskou and M. Brambilla. Semantic personalization of web portal contents. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 1245–1246. ACM, 2007.

- [147] A. A. J. van Ossenbruggen and M. Hildebrand. Why evaluating semantic web applications is difficult. In *Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*, 2008. To appear.
- [148] T. Vander Wal. Folksonomy explanations, 2005. <http://www.vanderwal.net/random/entrysel.php?blog=1622>.
- [149] T. Vander Wal. Folksonomy, 2007. <http://vanderwal.net/folksonomy.html>.
- [150] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 379–391, Siguenza, Spain, 2002.
- [151] M. Völkel, M. Krötzsch, D. Vrandečić, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, Edinburgh, Scotland, 2006.
- [152] M. Völkel and S. Schaffert, editors. *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics*, Workshop on Semantic Wikis. ESWC2006, June 2006.
- [153] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.
- [154] T. V. Wal. Getting to know collective and collaborative, March 2008. <http://www.personalinfocloud.com/2008/03/getting-to-know.html>.
- [155] S. Weiss, P. Urso, and P. Molli. Wooki: A p2p wiki-based collaborative writing tool. In B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, editors, *WISE*, volume 4831 of *Lecture Notes in Computer Science*, pages 503–512. Springer, 2007.
- [156] E. Wenger. *Communities of Practice. Learning, meaning, and identity*. Cambridge University Press, New York, Port Chester, Melbourne, Sydney, 1998.
- [157] E. Wenger, R. McDermott, and W. Snyder. *Cultivating Communities of Practice*. Harvard Business School Press, 2002.

- [158] Wikipedia. Collaborative software — Wikipedia, the free encyclopedia, 2008. [Online; accessed 02-Sep-2008].
- [159] Wikipedia. Lamp (software bundle) — Wikipedia, the free encyclopedia, 2008. [Online; accessed 01-Sep-2008].
- [160] Wikipedia. Seigenthaler incident — Wikipedia, the free encyclopedia, 2008. [Online; accessed 01-Sep-2008].
- [161] Wikipedia. Usenet — Wikipedia, the free encyclopedia, 2008. [Online; accessed 02-Sep-2008].
- [162] F. Wu and D. S. Weld. Autonomously semantifying wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 41–50, New York, NY, USA, 2007. ACM.
- [163] W. Yung. Bring existing data to the semantic web, May 2007. <http://www-128.ibm.com/developerworks/library/x-semweb.html>.
- [164] W. Yung. Using sparql for good: Querying ldap with squirrelrdf. Blog post, May 2007. <http://wingerz.com/blog/2007/05/10/using-sparql-for-good-querying-ldap-with-squirrelrdf/>.
- [165] V. Zacharias and S. Braun. Soboleo – social bookmarking and lightweight engineering of ontologies. In N. F. Noy, H. Alani, G. Stumme, P. Mika, Y. Sure, and D. Vrandečić, editors, *CKC*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.