Instituto de Computação
Universidade Estadual de Campinas

Ontologias Folksonomizadas
Uma Abordagem para Fusão de Ontologias e Folksonomias

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Hugo Augusto Alves e aprovada pela Banca Examinadora.

Campinas, 16 de abril de 2012.

Prof. André Santanchè (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela ficha catalográfica

(Esta página deve ser o verso da página anterior mesmo no caso em que não se imprime frente e verso, i.é., até 100 páginas.)

Substitua pela folha com as assinaturas da banca

Instituto de Computação
Universidade Estadual de Campinas

Ontologias Folksonomizadas
Uma Abordagem para Fusão de Ontologias e Folksonomias

Hugo Augusto Alves[1]

Abril de 2012

Banca Examinadora:

- Prof. André Santanchè (Orientador)
- Profa. Maria da Graça Campos Pimentel
  Instituto de Ciências Matemáticas e de Computação – USP
- Prof. Ricardo da Silva Torres
  Instituto de Computação – UNICAMP
- Profa. Claudia Bauzer Medeiros
  Instituto de Computação – UNICAMP (Suplente)
- Carla Geovana do Nascimento Macário
  Embrapa Informática Agropecuária - Embrapa (Suplente)

[1]Suporte financeiro de: Bolsa do CNPq (processo 135215/2010-2) 01/07/2010–28/02/2011, Bolsa da Fapesp (processo 2010/14217-3) 01/03/2011–2012

Resumo

Um número crescente de repositórios na web se baseia em metadados na forma de rótulos (*tags*) para organizar e classificar o seu conteúdo. Os usuários destes sistemas associam livremente tags a recursos do sistema – e.g., URLs, imagens, marcadores. O termo folksonomia se refere a esta classificação coletiva, que emerge do processo de rotulação (*tagging*) realizado por usuários interagindo em ambientes sociais na web.

Uma das maiores qualidades das folksonomias é a sua simplicidade de uso pela ausência de um vocabulário controlado. Folksonomias crescem de forma orgânica, refletindo o conhecimento da comunidade de usuários. Por outro lado, esta falta de estrutura leva a dificuldades em operações de organização e descoberta de conteúdo. Melhores resultados podem ser obtidos se forem consideradas as relações semânticas entre os rótulos.

Por esta razão, vários trabalhos foram propostos com o objetivo de relacionar ontologias e folksonomias, combinando a estrutura sistematizada das ontologias à semântica latente das folksonomias. Enquanto em uma direção algumas abordagens criam "ontologias sociais" a partir dos dados das folksonomias, em outra direção algumas abordagens conectam rótulos a ontologias preexistentes. Em ambos os casos nota-se uma unidirecionalidade, ou seja, um modelo apenas dá suporte ao enriquecimento do outro. Nossa proposta, por outro lado, é bidirecional. Ontologias e folksonomias são fundidas em uma nova entidade, que chamamos de "ontologia folksonomizada", combinando aspectos complementares de ambas. O conhecimento formal e projetado das ontologias é fundido com a semântica latente dos dados sociais.

Nesta dissertação apresentamos nossa ontologia folksonomizada e seus desdobramentos. Nós introduzimos um framework formal para a análise de trabalhos relacionados, a fim de confrontá-los com a nossa abordagem. Além das melhorias nas operações de indexação e descoberta, que foram validadas em experimentos práticos, nós propomos uma técnica chamada *3E Steps* para dar suporte à evolução de ontologias usando dados de folksonomias. Nós também implementamos o protótipo de uma ferramenta para a construção de ontologias folksonomizadas e para dar suporte à revisão de ontologias.

Abstract

An increasing number of web repositories relies on tag-based metadata to organize and classify their content. The users of these systems freely associate tags with resources of the system – e.g., URLs, images, and bookmarks. The term folksonomy refers to this collective classification, which emerges from tagging carried by users interacting in web social environments.

One of the major strengths of folksonomies is their simplicity due to the absence of a controlled vocabulary. Folksonomies grow organically, reflecting the knowledge of a community of users. On the other hand, this lack of structure leads to difficulties in operations of content organization and discovery. Better results can be obtained if we take into account the semantic relations among tags.

For this reason, many proposals were developed aiming to relate ontologies and folksonomies, combining the systematized structure of ontologies to the latent semantics of folksonomies. While in one direction some approaches build "social ontologies" from folksonomic data, in the other direction some approaches connect tags to existing ontologies. In both cases they are unidirectional approaches, i.e., one model is used only to support the enrichment of the other. Our proposal, on the other hand, is bidirectional. Ontologies and folksonomies are fused in a new entity, we call "folksonomized ontology", which combines complementary aspects of both. The formal and engineered knowledge of ontologies is fused with the latent semantics of social data.

In this dissertation we present our folksonomized ontology and its outcomes. We introduce here a formal framework to analyze the related work, confronting it with our approach. Besides the improvements in indexing and discovery operations, which are validated by practical experiments, we propose a 3E Steps technique to support ontology evolvement by using folksonomic data. We also have implemented a tool prototype to build folksonomized ontologies and to support ontology review.

Agradecimentos

Agradeço ao meu orientador, Doutor André Santanchè, pela orientação e incentivo ao longo de todo o projeto.

Ao meu pai José, que ensinando-me com suas atitudes é meu maior professor e à minha mãe Neide, que com seu infinito amor me guia pela vida.

À minha irmã Lislene, que me instigou a seguir os caminhos que me levaram à UNICAMP e à minha irmã Lilian, quem primeiro me mostrou meu interesse pelas ciências.

A todos os professores, funcionários e colegas da UNICAMP, tanto na graduação quanto no mestrado. Agradeço-os por todo o apoio e companheirismo ao longo de todos estes anos.

Aos membros da banca, pelas sugestões e melhorias feitas a este trabalho.

Às agências de fomento CNPq (processo 135215/2010-2) e FAPESP (processo 2010/14217-3), e também ao projeto INCT in Web Science (CNPq 557.128/2009-9) pelo apoio.

Por fim, agradeço aos amigos, familiares e todos aqueles que de alguma maneira contribuíram para o meu crescimento, ajudando-me a chegar até aqui.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

O número de sistemas web que fornecem serviços para que seus usuários criem e compartilhem conteúdo aumenta a cada ano. Esses sistemas acumulam um grande número de itens de conteúdo e mantêm vastas redes sociais de usuários que produzem, anotam, buscam e reusam este conteúdo. Como exemplo, existem mais de 5 bilhões de imagens hospedadas no Flickr[1], uma comunidade online de armazenamento de imagens e vídeos, e mais de 180 milhões de endereços URL foram armazenados no Delicious[2], um serviço social para armazenamento e compartilhamento de endereços URL.

Para organizar, descrever e classificar esse grande volume de dados, muitos sistemas usam uma abordagem baseadas em rótulos (*tags*) na forma de palavras-chave associadas livremente aos recursos. O termo folksonomia (do inglês *folksonomy*) – criado a partir das palavras *folk* (povo) e taxonomia [43] – tem sido empregado para se referir a esta abordagem de classificação usada por sistemas web que utilizam tags para organizar e indexar o conteúdo gerado pelos seus usuários em um ambiente social. As tags são tipicamente inseridas livremente no sistema pelos usuários sem um vocabulário controlado. Em contrapartida, operações que envolvem indexação, comparação e busca baseadas em tags são usualmente bastante limitadas por se basearem apenas em análise de strings.

A aplicação de análises estatísticas sobre o conjunto de tags associadas a recursos permite, por exemplo, a inferência de correlações entre tags e tags mais usadas para a descrição de certos recursos. Tais inferências podem ser exploradas para tornar explícita a semântica latente presente em folksonomias, aprimorando operações sobre tags. Pesquisas recentes neste sentido têm buscado derivar um tipo especial de "ontologia social" a partir de folksonomias [35, 40, 42]. Entretanto, como observamos no desenvolvimento

[1] http://blog.flickr.net/en/2010/09/19/5000000000/ - acessado em novembro de 2011

[2] http://blog.delicious.com/blog/2008/11/delicious-is-5.html - acessado em novembro de 2011

deste trabalho, há um conjunto limitado de relações que podem ser inferidas a partir de folksonomias – na maioria dos casos relações de generalização/especialização – já que as tags carecem de semântica explícita. Por esta razão, muitas iniciativas recorrem a ontologias auxiliares a fim de explicitar a semântica de tags, reduzir a ambiguidade entre tags ou dar suporte à análise de similaridade [10, 11].

Em ambos os casos, as abordagens são unidirecionais, ou seja, uma das entidades envolvidas (folksonomia ou ontologia) é utilizada para melhorar a outra. Em uma direção a semântica latente das folksonomias é usada na produção de ontologias sociais, em outra direção a semântica explícita das ontologias preexistentes dão assistência a este processo de produção. Como detalharemos nesta dissertação, nenhuma das abordagens analisadas define um modelo que seja capaz de reter em seu produto final as estruturas semânticas complementares provenientes de ontologias e folksonomias.

Partindo destas constatações, percebemos a necessidade de uma abordagem bidirecional, que seja capaz de fundir ambas as estruturas semânticas em uma nova entidade. Ou seja, utilizar ambas entidades em uma combinação simbiótica em que as bases semânticas formais das ontologias são utilizadas em conjunto com o conhecimento orgânico da comunidade de usuários de folksonomias. Isto deu origem ao tema central desta dissertação, que consistiu na concepção e na construção de uma ontologia folksonomizada (*folksonomized ontology*), que combina os melhores aspectos de ambas entidades.

1.2 Objetivo e Contribuições

O objetivo desta pesquisa é desenvolver uma abordagem para a fusão de folksonomias e ontologias em ontologias folksonomizadas, a ser aplicada na melhoria de operações sobre tags e na revisão de ontologias.

A seguir são apresentadas as principais contribuições deste trabalho.

Definição de um modelo abstrato e formal para ontologias folksonomizadas: O modelo abstrato funde elementos de ontologias e folksonomias. Posteriormente ele foi formalizado.

Elaboração de um framework formal relacionado a ontologias sociais: Como parte do processo de formalização de ontologias folksonomizadas foi elaborado um framework formal para caracterizar e comparar os modelos utilizados no processo de construção de ontologias sociais a partir de folksonomias, bem como seu relacionamento com ontologias.

Algoritmo para mapeamento de tags em ontologias: O objetivo central deste trabalho é a fusão de folksonomias e ontologias. Assim, o algoritmo desenvolvido para

esta fusão é uma das principais contribuições deste trabalho.

Técnica para expandir a similaridade de Resnik para folksonomias: A métrica de similaridade de Resnik [31] foi desenvolvida utilizando *corpus* de textos para o cálculo da frequência dos conceitos. Neste trabalho, nós expandimos esse trabalho ao desenvolver uma técnica que utiliza os dados de folksonomias para o cálculo de frequência. Dessa forma obtivemos um meio de calcular similaridade entre os conceitos da ontologia folksonomizada.

Protótipo para a criação de folksonomias: Uma contribuição prática deste trabalho consiste no desenvolvimento do protótipo de coleta e de armazenamento de dados de folksonomias, bem como a montagem de ontologias folksonomizadas. Seu desenvolvimento incluiu o estudo das APIs para acesso às bases de dados do Delicious e do Flickr. De posse das informações sobre estes sistemas, desenvolvemos um protótipo de um software responsável por acessar essas folksonomias, armazenar as informações coletadas de forma integrada e utilizá-las para a construção de ontologias folksonomizadas.

Técnica e protótipo para evolução de ontologias: Partindo das observações feitas no desenvolvimento das etapas anteriores deste trabalho, percebemos a possibilidade de utilizar uma ontologia folksonomizada para dar suporte ao processo de revisão e de evolução da ontologia original. Dado que é possível ao se comparar dados extraídos da folksonomia (como co-ocorrência entre tags) com relações entre conceitos da ontologia, desenvolvemos uma técnica para suporte à revisão de ontologias e um protótipo que busca discrepâncias entre ontologias/folksonomias e as apresenta de forma gráfica.

1.3 Estrutura da Dissertação

A estrutura desta dissertação consiste em uma compilação de três artigos publicados ou submetidos para publicação. Cada artigo está disposto em um capítulo e representa uma etapa evolutiva na pesquisa. O Capítulo 2 define as ontologias folksonomizadas e mostra o processo de construção das mesmas. O Capítulo 3 apresenta um framework formal que abrange as ontologias folksonomizadas e abordagens relacionadas. O Capítulo 4 sumariza a pesquisa em ontologias folksonomizadas e apresenta nossa técnica *3E Steps* para evolução de ontologias, bem como a respectiva ferramenta. No Capítulo 5 apresentamos as conclusões desta dissertação e trabalhos futuros. Por fim, o Apêndice A detalha aspectos de implementação do sistema e da modelagem do banco de dados, com ênfase no processo de obtenção e de armazenamento de dados das folksonomias.

1.3.1 Capítulo 2

O Capítulo 2 contém o artigo **Folksonomized ontologies - from social to formal** que foi publicado no *XVII Simpósio Brasileiro de Sistemas Multimídia e Web – WebMedia 2011* [3].

Este capítulo apresenta o modelo das ontologias folksonomizadas e detalha o seu processo de criação, desde a coleta e processamento de dados, até seu algoritmo de construção. Também são descritos os testes quantitativos e qualitativos efetuados a partir do uso do protótipo implementado para validação.

1.3.2 Capítulo 3

O Capítulo 3 é formado pelo artigo **Formal Aspects of Social Ontologies and Folksonomized Ontologies** que foi submetido ao *4th International Workshop on Semantic Web Information Management – SWIM 2012* [5].

Neste capítulo apresentamos a modelagem formal não somente das ontologias folksonomizadas, como também dos trabalhos relacionados. No total são apresentados cinco modelos, desde folksonomias e ontologias, passando por ontologias sociais até as ontologias folksonomizadas. Esta modelagem abrangente permitiu uma comparação detalhada de trabalhos relacionados. Deste modo, este artigo expande e trata de forma mais sistemática os trabalhos relacionados analisados no artigo do capítulo anterior.

O processo de construção de uma ontologia folksonomizada (já descrito no capítulo anterior), ganha uma nova dimensão na perspectiva do framework formal. Este framework permite também tornar explícito o diferencial da abordagem de fusão por nós proposta.

1.3.3 Capítulo 4

O Capítulo 4 contém o artigo **Folksonomized Ontologies and the 3E Steps Technique to Suport Ontology Evolvement** que foi submetido ao *Journal of Web Semantics* [4].

Neste capítulo apresentamos uma visão sintética sobre toda a pesquisa feita sobre as ontologias folksonomizadas – com exceção do framework formal – e introduzimos uma técnica de suporte à evolução de ontologias denominada 3E Steps. Esta técnica está dividida em três grandes etapas: (i) Extração, em que as informações de folksonomias são coletadas e processadas; (ii) Enriquecimento, que envolve o mapeamento de dados da folksonomia e da ontologia, seguido da fusão de ambos na ontologia folksonomizada; (iii) Evolução, em que partindo dos dados obtidos nas etapas anteriores, desenvolvemos um protótipo capaz de propor alterações na ontologia de origem baseadas no conhecimento

latente extraído da folksonomia. O artigo tem seu enfoque principal na técnica e em como a ferramenta é usada em casos práticos.

1.3.4 Capítulo 5

O capítulo 5 contém as conclusões desta dissertação. Nele apresentamos as contribuições da pesquisa, assim como as principais possíveis extensões e trabalhos futuros.

1.3.5 Apêndice A

No Apêndice A detalhamos o trabalho desenvolvido para a obtenção e armazenamento dos dados de folksonomias. O conteúdo do apêndice é uma versão preliminar de um relatório técnico do Instituto de Computação da Unicamp [6].

Neste apêndice, apresentamos a modelagem da base de dados para o armazenamento de folksonomias, apresentando trabalhos relacionados e modelos usados por sistemas web que subsidiaram nosso modelo unificada. Também apresentamos detalhes de implementação da ferramenta que acessa serviços web para obter folksonomias. Neste sentido, descrevemos detalhes de acesso aos serviços do Delicious e Flickr.

Capítulo 2

Folksonomized Ontologies – from social to formal

2.1 Introduction

The popularization of web-based systems offering services for content storage, indexation and sharing fostered a rapid growth of content available on-line. There are more than 5 billion images hosted on Flickr[1] and more than 180 million URL addresses on Delicious[2]. These systems increasingly rely on tag-based metadata to organize and index all the amount of data. The tags are provided by users usually connected in social networks, who are free to use any word as tag; there is no central control. The term folksonomy – combining the words "folk" and "taxonomy" [43] – has been used to characterize the product which emerges from this tagging in a social environment.

Any operation involving indexation, classification or discovery of content in these web-based systems will require a comparison among the involved tags. In this topic, there are approaches ranging from a pure lexical or statistical comparison of words to a richer semantic analysis of relations, by associating tags to formal ontologies. In many contexts, this semantic directed approach will enable machines to better classify, rank, disambiguate and discover tags, enriching the systems and the user experience. Recent investigations explore this relationship in different directions, for example: (i) by deriving ontologies from folksonomies [35, 42]; (ii) by manually or automatically connecting tags to ontologies [11, 10]. In either case, there is still a unidirectional perspective, in which a model takes advantage of the other.

This work addresses a fusion perspective. The proposed *folksonomized ontology* synthesizes complementary roles of ontologies and folksonomies. In one direction, the knowledge

[1] http://blog.flickr.net/en/2010/09/19/5000000000/

[2] http://blog.delicious.com/blog/2008/11/delicious-is-5.html

systematically organized and formalized in ontologies is "folksonomized", i.e., the latent semantics from the folksonomic tissue is extracted and fused to ontologies. On the other, the folksonomized ontologies are explored to enhance operations involving tags, e.g., content indexation and discovery. The folksonomic data fused to an ontology will tune it up to contextualize inferences over the repository.

Our approach was validated by a tool we developed, which extracts tags from Delicious and Flickr, fusing them in the WordNet [28] ontology. WordNet is a lexical database of English, having a formalized thesaurus, which can be used as ontology. The resulting folksonomized ontology shows better results when applied to content discovery.

This paper is organized as follows. In Section 2.2 we discuss the basis of our work. We present our solution in Section 2.3 and the experimental results in Section 2.4. In Section 2.5 we confront our approach with related work and we conclude and discuss the future work in Section 2.6.

2.2 Folksonomies, Ontologies and Similarity

In this section we summarize some related work which subsidized our research.

2.2.1 Folksonomies and Ontologies

In folksonomy-based systems, users can attach a set of tags to resources. These tags are not tied to any centralized vocabulary, so the users are free to create and combine tags. Some strengths of folksonomies are their easiness of use and the fact that they reflect the vocabulary of their users [26]. In a first glimpse, tagging can transmit the wrong idea of a poor classification system. However, thanks to its simplicity, users are producing millions of correlated tags. It is a shift from classical approaches – in which a restricted group of people formalize a set of concepts and relations – into a social approach – in which the concepts and their relations emerge from the collective tagging [34]. In order to perform a systematic folksonomy analysis, to subsidize the extraction of its potential semantics, researchers are proposing models to represent its key aspects. Gruber [15] models a folksonomy departing from its basic "tagging" element, defined as the following relation:

$$Tagging(object, tag, tagger, source) \tag{2.1}$$

In which *object* is the described resource, *tag* is the tag itself – a string containing a word or combined words –, *tagger* is the tag's author, and *source* is the folksonomy system, which allows to record the tag provenience (e.g., Delicious, Flickr etc.).

In order to formalize a folksonomy Mika [27] departs from a tripartite graph with hyperedges. There are three disjoint sets representing the vertices:

$$T = \{t_1, \ldots, t_k\}, U = \{u_1, \ldots, u_l\}, R = \{r_1, \ldots, r_m\} \tag{2.2}$$

In which the sets T, U and R correspond to tags, users and resources sets respectively.

A folksonomy system is a set of annotations A relating these three sets:

$$A \subseteq T \times U \times R \tag{2.3}$$

The folksonomy itself is a tripartite hypergraph:

$$H(T) = \langle V, E \rangle \tag{2.4}$$

In which $V = T \cup U \cup R$, and $E = \{\{t, u, r\} \mid (t, u, r) \in A\}$

The folksonomy analysis can be simplified and directed by reducing this tripartite hypergraph into three bipartite graphs: TU relating tags to users, UR relating users to resources and TR relating tags to resources [27]. A graph TT is a relevant extension of this model for representing relations between tags. It allows to represent the co-occurrence of tags. The same approach can be applied to the user and resource sets.

The Gruber's classical definition of ontology as "an explicit specification of a conceptualization" [14] synthesizes its key aspect as an intentionally systematized – or engineered [27] – specification. According to Shirky [34], contrasting to ontologies, in tag-based approaches the organization derives from an organic work. It is a shift from a binary categorization approach – in which a concept A "is" or "is not" part of a category B – to a probabilistic approach – in which a percentage of people relates A to B. Gruber [15], on the other hand, claims that folksonomies and ontologies should not be seen as opposite but rather as complementary, and he proposes a TagOntology – a common ontology for tagging. As we will present in this paper, we share Gruber's view of complementary roles, expanding the perspective to introduce a fusion (bidirectional) approach, in which folksonomies meet "classical" ontologies. Kim et al. [18] described three areas where the association of ontologies and folksonomies can improve the systems, namely: knowledge representation sophistication, facilitation of knowledge exchange and machine-processable. Moreover, this association can improve the tag query and disambiguation, visualization of tag clusters and tag suggestion to users [35].

2.2.2 Similarity and Information Content

One way to explore the semantics – formalized in ontologies and potential in folksonomies – involves matching and similarity. There are many applications, such as, ontology engineering, information integration and web query answering where matching operations play

a central role [13]. When tags are compared, matching operations can be organized in two main broad categories: lexical/syntactic and semantic. Lexical/syntactic approaches are mainly based on the proximity of spelling words and their derivations (e.g., conjugations). One example of this category is the edit distance, as the popular approach proposed by Levenshtein [21].

To go beyond the spelling, semantic approaches relate words to a respective semantic representation – a concept. The matching is evaluated by analyzing semantic relationships among concepts, e.g., equivalence, generalization, specialization etc. This approach can lead to better search results or expand the opportunity for discovery, by finding and ranking similar or related results. It can also subsidize better recommendation systems for tag definition. In this context, ontologies are increasingly being adopted to formalize the semantics of concepts and their relationships.

A challenge in semantic matching is how to weight the relevance of relationships when similarities are confronted. Consider a practical example of a program looking for the concepts similar to `judge`. The output will be a set of concepts ranked according their similarity. Two possible similar concepts in the example could be `district attorney` or `child`. Like a `judge`, the former is an official functionary and the latter is a person. To rank them by similarity it is necessary to define which concept is more similar to `judge`.

In order to put this comparison in a context, let us consider a classical abstraction of an ontology as a graph, in which each vertex (node) is a concept and each edge is a relationship between two concepts. A comparison supported by this ontology considers that the compared terms are connected by a path. Figure 2.1 illustrates the three previous concepts as they appear in the WordNet ontology. In the figure, circles represent concepts and edges subsumption relationships – lower concepts specialize upper ones.

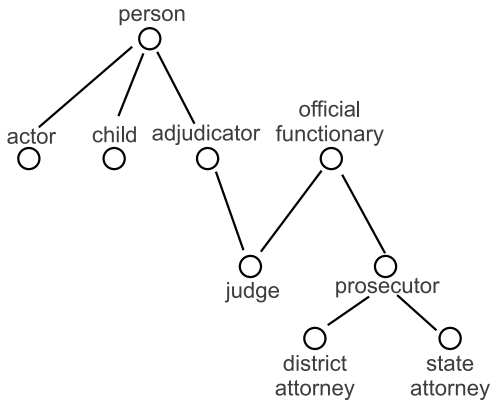

Figure 2.1: Subsumption ontology showing the relationships among compared concepts.

Many approaches to calculate semantic similarity based on ontologies were developed and we will further present some relevant techniques.

Path-based

A naive method to evaluate the semantic similarity between two nodes in an ontology is by measuring the shortest path separating them. This is equivalent to the distance metric in a *is-a* (subsumption) semantic net, defined by Rada et al. [30]: $distance(c_1, c_2)$ = minimum number of edges between c_1 and c_2. The similarity then calculated as:

$$sim_{rada} = [1 + distance(c_1, c_2)]^{-1} \tag{2.5}$$

As showed in [31], this approach is highly influenced by the level of detail applied to describe branches of the ontology, i.e., branches better detailed can contain longer paths than other, in spite of the similarity distance, leading to biased evaluations. For example, the comparison of `judge` with `child` (3 edges) results in the same similarity of `district attorney` compared to `judge` (3 edges). One way to overcome this limitation is by weighting the edges, leading to the problem of how to determine the weights. According to Jiang and Conrath [17] there are ontology aspects, such as depth of nodes and type of links, which can be used to define these weights.

Depth-relative

One way to enhance the path-based comparisons is by analyzing the most specialized common ancestor shared by two nodes in the ontology. It is founded in specific kinds of taxonomic ontologies based on subsumption relationships among terms, as the example of Figure 2.1. Observations showed that siblings sharing an ancestor deep in a hierarchy are more closely related than those sharing an ancestor higher in the hierarchy [38]. Therefore, Wu and Palmers [45] propose the following metric:

$$sim_{wp}(c_1, c_2) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3} \tag{2.6}$$

In which c_3 is the least (most specialized) common ancestor of both c_1 and c_2, N_1 is the number of nodes on the path from c_1 to c_3, N_2 the number of nodes between c_2 and c_3, and N_3 the number of nodes between c_3 and the ontology root.

To improve the depth-relative metrics, Shickel and Faltings [33] proposed the OSS metric, based on an A-Priori Score APS computation of all concepts in an ontology. Then, a distance metric is defined from two coefficients (generalization and specialization) calculated from the APS value.

Content-based

Besides the ontology topology, there are approaches showing that comparisons can be improved by analyzing also the content of the ontology concepts. Resnik proposed an

approach based on *information content* [31] applied to subsumption ontologies. Assuming that each concept in this kind of ontology is a class representing a set of instances, the probability of a given instance to belong to a more specific class – e.g., `child` – is lower than the probability to belong to a more general one – e.g., `person`. While the probability decreases, the information about more specific classes increases – a necessary consequence of their specialization. *Information Content* (IC) is a measure created to evaluate this increase of information about something. Let the probability of a given concept `c` be `p(c)`, then the IC of `c` is `-log p(c)` [32].

In order to illustrate Resnik's IC-based approach to evaluate the similarity among terms, let us return to the example involving the similarity ranking among `judge` and two other concepts: `district attorney` and `child`. The first step is to find the most specialized concept shared by `judge` and `district attorney`, which is `official functionary`, as by `judge` and `child`, which is `person`. Intuitively, we can infer that the probability of an instance to belong to `official functionary` is smaller than the probability of an instance to belong to `people`; conversely the IC is higher. In this type of ontology, when two concepts derive from the same generalization they share its characteristics, therefore, `judge` is more similar to `district attorney` than to `child`, since the former has higher IC. Therefore, the Resnik [31] similarity metric was defined as follows:

$$sim_{res}(c_1, c_2) = \max_{c \in S(c_1,c_2)} [-\log p(c)] \tag{2.7}$$

In which the set S contains all concepts that subsume both c_1 and c_2. Experiments in [31] demonstrated that this approach produces better results than the counting edges approach and is not influenced by unbalances in ontology detailing. There are many other approaches exploring probabilities to improve similarity evaluation such as Lin [23] and Jiang and Conrath [17].

All of these probability-based approaches lead to an extra challenge: how to evaluate the probability of each concept of an ontology. Resnik's strategy is based on counting words extracted from a corpus of documents. As will be further detailed, our work expands Resnik proposal in three directions:
(i) proposing a strategy for calculating probabilities and IC of concepts based on tags employed in social networks to describe content;
(ii) defining multiple context-driven IC for each concept;
(iii) applying IC and co-occurrence data to review the ontology.

2.3 Folksonomized Ontologies

As observed in the previous section, ontologies and folksonomies can play complementary roles. Nevertheless, existing proposals usually are unidirectional, attaching folksonomy's

tags to ontologies or, conversely, producing ontologies from folksonomies. In this section we describe our fusion approach, which takes advantage of both ontologies and folksonomies, producing a synthesis. This fusion results in a *folksonomized ontology*, which we define as an ontology aligned with terms of a folksonomy and enriched with their contextual data. By contextual data we mean data which emerges from a statistical analysis of a folksonomy, e.g. tag frequency, co-occurrence and information content.

In one direction the folksonomized ontology, which is aligned with tags, drives richer semantic-based matching, categorization and tag suggestion. In the other direction, contextual data will be used to review and improve the ontology. The Figure 2.2 schematizes the roles played by an ontology and a folksonomy in a folksonomized ontology building. The ontology was previously engineered to formalize concepts and typed relationships, e.g., is-a, same-as, part-of. Concepts and relationships in folksonomies, on the other hand, are inferred by statistical analysis over tags and their co-relations. They are not typed, as in ontologies, but carry substantial contextual data, which subsidizes "weighting" concepts and relationships. The resulting folksonomized ontology is a new entity that fuses the best of both worlds, having typed and "weighted" concepts and relationships.

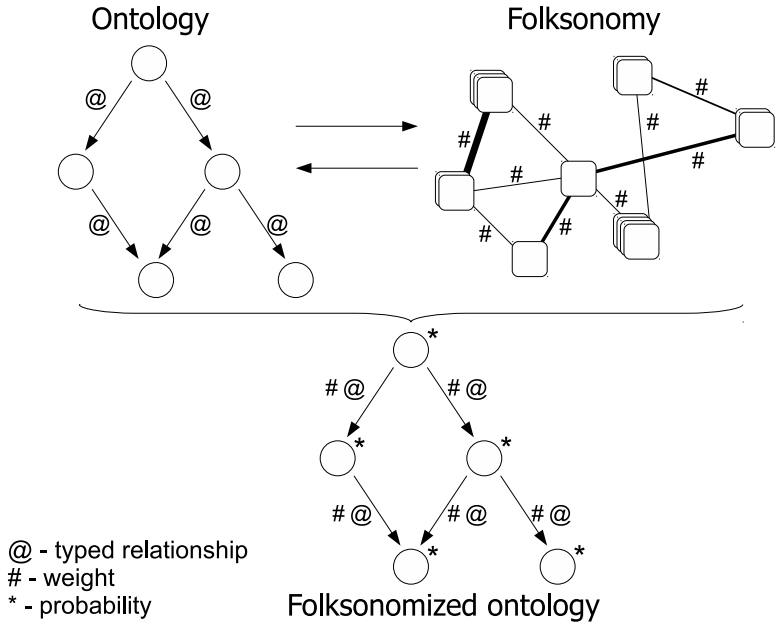

Figure 2.2: Folksonomized Ontology

A practical tool was developed in this research, apt to build folksonomized ontologies and use them for tag searching and discovery, as to ontology review and improvement. Figure 2.3 summarizes the cycle of the folksonomic ontology building and use. It starts

collecting data from folksonomy systems (step 3.1), e.g., Delicious and Flickr, which are processed, filtered and grouped as concepts (concept-group) (step 3.2). Concept-groups are mapped to concepts in ontologies (step 3.3). The probability and IC for each concept-group, as the co-occurrence of concept-groups, are calculated and fused to the ontology, obtaining our folksonomized ontology (step 3.4). The step (3.5) is an ongoing work in this research; it confronts statistical data extracted from a folksonomy with the structure of an ontology, in order to subsidize ontology review and improvement.

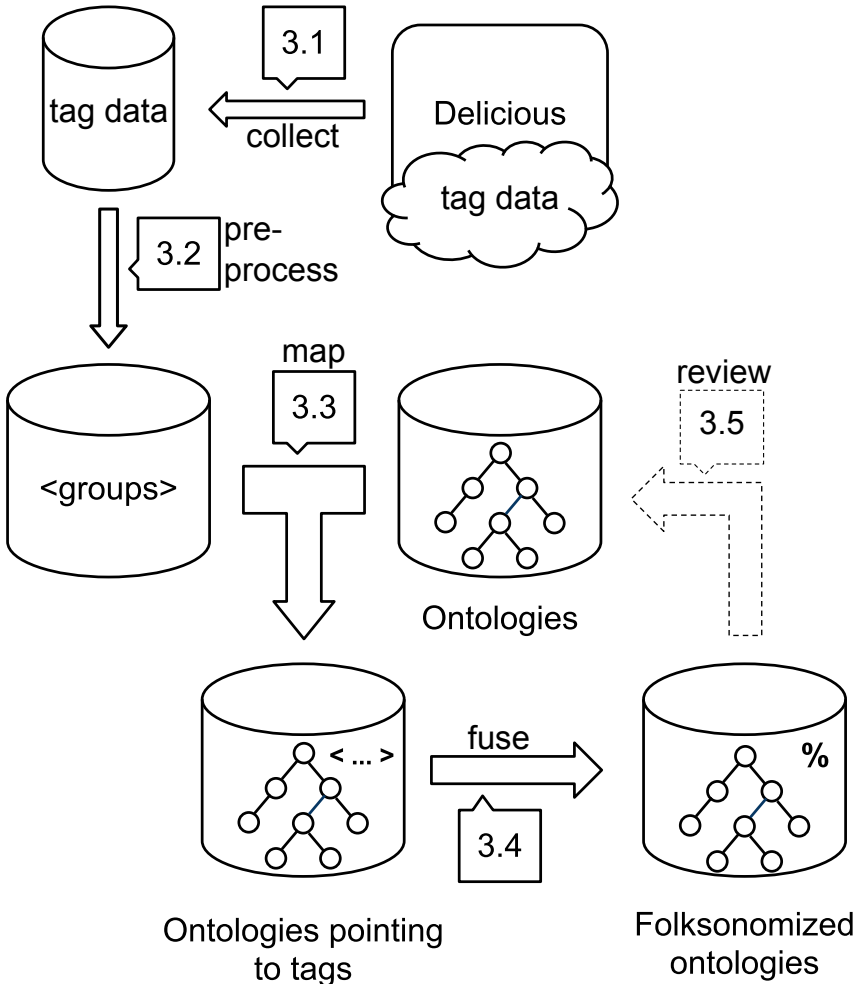

Figure 2.3: Folksonomized ontology building and use

The step (3.2) involves preprocessing algorithms, e.g., to adjust punctuation mismatches and to group tags. Since our contribution is not focused in these preprocessing algorithms, but rather in the subsequent steps, we implemented established algorithms, which will not be compared to related work. Moreover, we adopted the same preprocessing algorithms when comparing our approach to related work. In the following subsections

each step of the process illustrated in Figure 2.3 will be detailed.

2.3.1 Collecting Tag Data

Web-based content portals offer web service interfaces to access their data. The tag data collecting module (step 3.1) access these web services to select and retrieve tags and their metadata, which are stored in a database. Due to the heterogeneity in the APIs, this module was designed to be customizable and it was tested in Delicious and Flickr systems. To better obtain the emergent properties of the semantics extracted from folksonomies, this module was designed to afford large datasets. They are stored as triples of resources, users and tags, including their relations. Statistical data – e.g., co-occurrence between tags – were computed and stored during data collection, avoiding extra post-processing work. The updating process is incremental, i.e., it collects and stores just the differences of previous processings.

2.3.2 Tag Processing

In order to avoid the interference of wrong spelled tags or similar problems, unusual tags – with less than five occurrences – were eliminated to improve the quality of the data set. This procedure produces a collateral effect, since it also filters correct tags having a high IC value, due to their low-frequency. Therefore, we consider this a preliminary approach. In a future work, we intend to study the impact that low-frequency tags have in the results and if they should be kept or deleted.

The next step involves grouping tags referring to the same term. For instance, the tags *tip* and *tips* are tightly connected and represent the same term. The grouping algorithm is divided in two steps: (i) **punctuation analysis** – groups tags differing only in punctuation signs; (ii) **morphological analysis** – group tags by morphological relatedness.

A common approach in tagging systems is to delimit tags by spaces. In order to represent multiple word tags, users resort to different strategies, e.g., concatenating words with or without separating signs. By analyzing the similarity of tags without the punctuation we could group tags like *search-engine*, *search_engine*, and *searchengine*. These tags are clearly very close to each other and represent different user approaches for using multiple word tags. So, all punctuation signs of tags were removed, allowing to group tags that became equal without punctuation.

The morphological analysis and grouping go beyond spelling comparisons, considering morphological variations as singular and plural tags, or tags of different verb tenses. The algorithm retrieves morphological variations of tags from the WordNet ontology, grouping them together.

2.3.3 Mapping tags to ontology terms

The next step, of mapping tags to ontology concepts, is not a simple task, due to the lack of semantic information related to the tags. The tags cannot be directly mapped based on their words, since the same word can have multiple meanings in the ontology. In WordNet, for instance, a word can have multiple senses, called synsets, which are differentiated through identifiers combining the original word plus two affixes. The first one is a character that describes the synset type (namely noun, verb, adjective, or adverb) and the second one is a sequential number to differentiate each meaning. For instance, the synset *dog.n.01* represents a noun and it is one of the synsets for the word *dog*.

To find out which synset corresponds of each tag, we developed a technique that encompasses the relation of the WordNet synsets and tag co-occurrences, divided in three steps: (i) group key election; (ii) co-occurrence selection; (iii) group key mapping. They are further detailed.

Group key election. In the previous stage, tags referring to the same term were grouped. In this step, a "group key" is elected to represent each of these groups. Since all variations of a tag in each group are considered referring to the same term, it is necessary to select the most significant to represent the group. Since the WordNet will be the target of the tag mapping, it is also used in the group key election process. By analyzing morphological derivations of words in the WordNet, it is possible to determine which word is the root in each group. This tag is elected the group key. There are exceptional cases in which it is not possible to fetch a root word for a given group. We implemented a preliminary solution in which the first tag in the group is elected. We are planning to implement a better approach for exceptional cases as a future work.

Co-occurrence selection. In order to put tag keys in a context, they are linked to related tags having highest co-occurrence values. Considering a group containing n tags. For each tag t in this group, the selection algorithm initially fetches the h tags having highest co-occurrence with t. The result is a set of $n \times h$ co-occurrences. Then, the algorithm selects the s tags with the highest co-occurrence values in this resulting set.

Group key mapping. The last step involves mapping group keys to WordNet's synsets. Consider a tag t, a group key, to be mapped to a synset and a set C containing the tags having the highest co-occurrence related to t (obtained in the last step). Consider a group S containing all synset candidates for mapping. Our algorithm evaluates the distance of each synset s of S compared with t, in the following way: (i) the set C must have a minimum set of tags already mapped to synsets; this minimum is defined by a threshold constant *minmap*; (ii) the similarity of a given s is calculated by the sum of the distances of all c already mapped to s; (iii) since there is no IC data yet, a path-based similarity algorithm is applied. The synset s with the highest sum is the target of the mapping.

A tag group will only be processed if a minimum of the elements in the corresponding co-occurrence list had already been processed and mapped. Since the algorithm always selects a synset based on tags already mapped, it was necessary to create a starting set of tags manually mapped, to work as seeds. Algorithm 1 presents a pseudo-code of the tag mapping.

Algorithm 1 The algorithm to map group keys to synsets

Input: G: set of groups keys (tags)
Input: minmap: minimum co-occurrence mapping
Output: S: set of group keys (tags) mapped to synsets

```
1:  S ← {}
2:  while ∃ t in G — fit(t) do
3:     t ← choose(G)
4:     cooc_list ← getcooc(t)
5:     list ← {}
6:     for all synset s in synsets(t) do
7:        for all element e in cooc_list do
8:           include (s,sim(s, synmap(e), coocval(t, e))) in list
9:        end for
10:    end for
11:    S[t] ← max(list)
12:    remove t from G
13: end while
```

The functions used in Algorithm 1 are:

choose(G) Returns a tag t in G in which $fit(t) = true$.

fit(t) Returns true if the co-occurrence list related to the tag t has at least *minmap* elements already mapped.

getcooc(t) Returns the co-occurrence list for the tag t, having the highest co-occurrence values and already mapped to a synset.

synsets(t) Returns all possible synsets for a given tag.

synmap(t) Returns a synset already mapped to a tag.

coocval(t_1, t_2) Returns the co-occurrence value between t_1 and t_2.

sim(s_1, s_2, e) Calculates the path-based similarity between the two synsets (s_1 and s_2) multiplied by the co-occurrence value (e).

max(list) Returns the synset having the highest similarity value in the list.

In the best scenario this algorithm stops when it maps all tags. However, depending on the starting seeds and the *minmap* value, it is possible that it will not converge and the algorithm will stop in the absence of eligible tags to process. In this case, the result is a partial mapping set.

After this step, a subset of WordNet ontology is mapped to tags of the folksonomy. However, there are WordNet concepts that do not point to tags. They are classified here as *virtual nodes* and the ones that point to tags are the *real nodes*. For instance, the term `entity.n.01` is the root of the ontology and does not point to any group of tags, a *virtual node*.

2.3.4 Fusing

After the mapping process, it is possible to calculate the information content (IC) of each ontology concept. Our algorithm starts by setting frequency values collected from the folksonomy in the *real nodes*. Each node change reflects in every predecessor node. The frequency is calculated by using the occurrences of the mapped tags.

This strategy considers that when users associate tags to resources, they are also associating the respective generalizations. For instance, when a user tags a resource with the tag *"judge"*, he is implicitly tagging this resource with the tag *"person"*. Since each tag frequency reflects in its predecessors, it is necessary to avoid counting twice when the same resource is tagged by a user with tags having a subsumption relationship – e.g., *"judge"* and *"person"*. These frequencies subsidize the calculus of probability and IC for each node.

2.4 Practical Experiments

Among our practical experiments, in this section we will focus the presentation on Delicious data, due to the nature of its resources – URL addresses – which are better suited to compare with related work, as shown in the evaluation section. For the experiment discussed in this section, we have collected and stored a total of 1,049,422 triples of resources, users and tags, including their relations.

2.4.1 Similarity Algorithm

After calculating the IC values, we implemented some similarity and distance metrics like sim_{lin} [23], $dist_{jiang}$ [17] and sim_{res} [31] to validate our proposal. Considering that sim_{res} is a basis algorithm and sim_{lin}, $dist_{jiang}$ variations over it, our focus here will be the sim_{res} implementation.

Since Resnik's similarity metric is relative, it requires at least three terms: one pivot and two other terms to be ranked. Let's consider the pivot *graphic* and the comparing terms *picture* and *freeware*. With the sim_{res} we obtained, as expected, that *picture* is more similar to *graphic* than *freeware*.

In order to evaluate our folksonomized ontology in similarity operations, we conducted two groups of comparisons further described: (i) folksonomized ontology versus ontology and co-occurrence; (ii) folksonomy versus document emergent semantics.

2.4.2 Ontology and co-occurrence

We developed a qualitative analysis in tag comparison by confronting our proposal with: the WordNet ontology without folksonomized data and using path-based similarity algorithms; just tags and their co-occurrence statistics.

To present our considerations, we selected three representative cases of compared tag pairs: *graphics* and *inspiration*; *war* and *conflict*; *bible* and *christian*.

The terms *graphics* and *inspiration* have a high co-occurrence (41% of the maximum co-occurrence value for *graphics*), but low similarity in the path-based algorithm, since the terms are relatively far in the ontology (too much edges). The similarity based on folsksonomized ontologies was more accurate in this case, since it does not rely solely on the ontology topology.

In the case of *war* and *conflict*, there are no co-occurrence value, because *conflict* does not exist in the tags dataset. But it exists in the ontology as a *virtual node* and has a high similarity with the term *war*. This example shows that with our folksonomized ontology it is possible to find similar terms and suggest them to the users, even if they do not exist yet in the tag dataset – a feature that a standalone folksonomy is not able to offer.

The pair *bible* and *christian*, however, shows a situation in which the co-occurrence has better results than the folksonomized ontology. Even having a high co-occurrence value (there is no tag with more co-occurrence with *bible* than *christian*), any ontology-based comparison of similarity (folksonomized or not) will return zero. The reason is that in WordNet the only common parent of these two terms is *"entity"*, the root of the ontology, leading to a zero similarity. The folksonomy points to a strong relationship between the terms and it is a valuable information, which can be used to support the ontology review, as shown in Subsection 2.4.4.

2.4.3 Document emergent semantics

In order to evaluate the potential semantics extracted from folksonomies, this second group of comparisons confronts data extracted from tags with those extracted from web pages. Since our tags were extracted from Delicious, each tag is related to a web page address (URL). Our experiment fetched approximately 4,500 web pages pointed by Delicious tags. The analysis of the pages content adopted the same technique used by Resnik, i.e., counting the words in the corpus to compute the word frequency.

Besides the IC computed by using the word count of pages, the rest of the process adopted the same algorithms of our solution. The resulting enriched ontology was used to the comparisons of which results we further present a qualitative and a quantitative analysis. The terms sim_{tag} and sim_{wc} will be used to refer the Resnik's similarity algorithm applied to our folksonomized ontology and to the other enriched ontology respectively.

Qualitative analysis

In this analysis we compiled a list with 100 triples containing: a pivot and two comparing terms. For each triple we manually marked the term that we judged to be more similar to the pivot and then the sim_{wc} and sim_{tag} were applied to the list.

The result of this analysis is that both similarities had a rate of 90% of conformity compared to our judgment. Both similarity algorithms had equivalent behaviors, i.e., both differed of our judgment in the same triples. This result shows that both approaches achieved a good conformity rate and indicates a possible tendency to be explored, that in many contexts the semantics extracted from tags describing pages can avoid the analysis of the whole pages. In this preliminary analysis the results were confronted with our judgment, but the validation process will address users in future work.

Quantitative analysis

In the quantitative analysis, the two ontologies were confronted in exhaustive comparisons. For either ontology, a routine compared each concept with all other concepts of the same ontology. Since the similarity algorithm requires a third pivot concept, the pivot was randomly chosen in the ontology. The same comparison was made in parallel in both ontologies and the results were compared. To minimize the randomic effect of the pivot, the same algorithm was ran 100 times. The average number of different results obtained in similarity comparisons corresponds to 0,02% of the total of triples analyzed. Therefore, we conclude that both approaches are equivalent.

One could argue the differential of evaluating tags compared to the classic approach based on documents word counting. Besides the previous mentioned conclusion, pointing to the observation that tags could produce equivalent semantic results with less effort, since they are more focused in relevant aspects, tags are also available in a wide range of content management systems, which do not have text documents to be analyzed. In Flickr, for instance, the resources are pictures, thus it is not possible to use the approach of counting words. The folksonomized ontology can be tailored to each context by switching the folksonomy. Therefore, it is possible to consider a folksonomized ontology for pictures, other for links and so on. The same approach can be used to customize ontologies to specific domains.

2.4.4 Supporting the Ontology Review

Departing from the observations of this research, we envisage that folksonomized ontologies can support the review and improvement of the ontologies used as foundations. This can lead to a symbiotic cycle, in which folksonomized ontologies help to improve the underlying ontologies which, in turn, will improve the results of the folksonomized ontology.

Figure 2.4 shows a graph generated by a tool we are developing to review ontologies. The nodes in the graph represent concepts in the ontology. Nodes connected by arrows represent relations by concepts explicit in the ontology. Nodes connected by edges without arrows – e.g., *bible.n.01* and *christian.n.01* – represent concepts in the ontology without formal explicit relationships, having a high co-occurrence in the folksonomy. The thickness of the edge is proportional to the intensity of the correlation. It can signalize a missing relevant relationship, to be considered in the ontology review. This is a preliminary result of an ongoing work. Many other inferred data can be presented to support ontologies review.

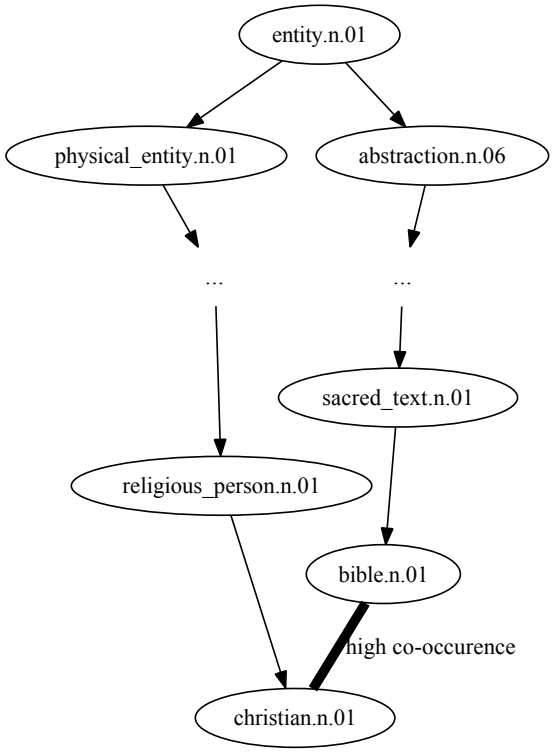

Figure 2.4: Example of folksonomy relationship absent in the ontology

2.5 Related Work

we selected four relevant works among the initiatives relating folksonomies to ontologies to compare with our approach.

Specia and Motta [35] aimed to build ontologies from folksonomy data. They first preprocessed the tags, eliminating the non-usual ones, and they created clusters of related tags, using co-occurrence information. Finally, they identified the relationships between these clusters using sources such as Google, Wikipedia and ontology bases. Damme et al. [42] proposed a system to group tags and associate them with ontologies. They used lexical resources, like Leo Dictionary, WordNet, Google and Wikipedia in the preprocessing step. A statistical analysis is applied to group tags in clusters.

Some steps followed by these works were followed by ours as well. The tag preprocessing, for instance, is a step that our work share with both. Our step of mapping tags into ontology terms differs from both. We focused in the folksonomy and ontology data, instead of looking for external sources. Different from both works, our approach takes fully advantage of the preexisting semantics in the underlying ontologies, instead of building a new ontology from scratch.

Cantador et al. [10] proposed a technique to filter tags, classifying them in categories, in order to infer the semantics of the classified tags to map them to knowledge bases like WordNet and Wikipedia. To find which category a given tag belongs, the authors resort to direct association or natural language processing heuristics. Cattuto et al. [11] applied existent ontologies, specifically WordNet, to find similarities between tags. However, their mapping approach do not group similar tags, resorting to a simple word comparison to find equivalent WordNet concepts. Our approach goes beyond, mapping groups of tags to synsets semantically related, even if syntactically they are not.

All of these related approaches are unidirectional, i.e., they produce ontologies from folksonomies or, conversely, use ontologies to assist tag relations in folksonomies. The major difference in our fusion approach is the symbiotic combination, in which ontologies support tag comparison and, on the other hand, folksonomies enrich (folksonomize) ontologies, improving their inferences and supporting ontologies review. In this sense, ontologies in our approach are not limited to be a tool to improve folksonomies. On the other hand, our approach will always require a preexisting ontology in the intended domain. Which can limit its application in some scenarios.

2.6 Conclusion and Future Work

Folksonomy-based systems have been largely adopted on the web, due to their flexibility and easiness of use. However, these systems have limited search mechanisms, based on

lexical comparisons of tags. On the other hand, formal categorizations, as ontologies, require a big effort to be built and maintained and do not take advantage of the potential semantics, which emerges in an organic way from social tagging systems.

To face this problem, this paper presents our approach to build a folksonomized ontology, an ontology fused with a folksonomy. It is a symbiotic combination, taking advantage of both semantic organizations. Ontologies provide a formal semantic basis, which is contextualized by folksonomic data, improving operations over tags based in ontologies. Conversely, the folksonomized ontologies can be also used as tools to analyze the ontology quality and to help the process of ontology evolution, showing the discrepancies between the emergent knowledge of a community and the formal representation of this knowledge in the ontology.

We are working to expand our research in the following directions: (i) to develop an interchangeable folksonomic dataset, providing different customizations of the ontology, according the context; (ii) to use other similarity algorithms and statistical data; (iii) to run tests in specialized contexts applying domain ontologies; (iv) to extend the solution to consider other relations in the ontology (besides the generalization and specialization); (v) to improve our tool for ontology evaluation and review; (vi) to measure and evaluate the costs and impact of our approach in current folksonomies.

Chapter 3

Folksonomized Ontologies and their formal aspects

3.1 Introduction

A growing number of web systems offer services for content storage, indexing, and sharing. Those systems usually have a huge number of users and large datasets. For instance, the photo sharing system Flickr has more than 5 billion images hosted[1] and there are more than 180 million URL addresses on Delicious[2]. Most of these systems use tag-based social networks to organize and index the stored content. Their users associate free-form tags with each resource, without a central vocabulary. The term folksonomy – combining the words "folk" and "taxonomy" [43] – has been used to characterize the product which emerges from this tagging in a social environment.

In order to analyze, index and classify their content, web systems compare tags attached to resources. Instead of considering the semantics of each tag in the comparison, tag-based systems usually rely on string matching approaches. While ontologies are increasingly adopted to enrich tag semantics, one common problem with the proposals to associate tags with formal ontologies concerns their unidirectionality, i.e., ontologies improve tag semantics, or the implicit/potential semantics of folksonomies is extracted to produce ontologies.

In a previous paper, we proposed a fusion approach, called *folksonomized ontology* (FO), which goes beyond this unidirectional perspective [3]. In one direction, the ontologies are "folksonomized", i.e., the latent semantics from the folksonomic tissue is extracted and fused to ontologies. On the other direction, the knowledge systematically organized and formalized in ontologies gives structure to the folksonomic semantics, enhancing op-

[1] `http://blog.flickr.net/en/2010/09/19/5000000000/` - retrieved on November, 2011

[2] `http://blog.delicious.com/blog/2008/11/delicious-is-5.html` - retrieved on November, 2011

erations involving tags, e.g., content indexing and discovery. The folksonomic data fused to an ontology will tune it up to contextualize inferences over the repository.

In a previous paper [3] we focused in the presentation of our folksonomized ontology, its respective tool [4] and validation tests. However, the scenario involving ontologies, folksonomies + derived ontologies (which we call social ontologies) and their relations still lacks a formal characterization, in order to answer open questions as:

- What is the abstract model behind social ontologies derived from folksonomies?
- How this model is related to ontologies?
- From the model point of view, how the initiatives explore the relationship between ontologies and folksonomies?

This chapter contributes by defining a formal framework to describe ontologies, folksonomies, social ontologies, and approaches to combine them, including our folksonomized ontology. Through this framework, we bring to a formal context – for the first time – the debate confronting them.

Through this formal perspective, this paper aims to explicit the limitations of unidirectional relations between folksonomies and ontologies, as well as the demand for our fusion approach and its strengths in:

Tag disambiguation: by finding groups of related tags and mapping them to ontology concepts, the FO can be applied to disambiguate tags and find the ones that are more related, going beyond statistical analyses by using semantic similarity metrics.

Tag suggestion: the current folksonomy systems consider only co-occurrence information to suggest related tags to users; a FO has a richer set of semantic relations among concepts, supporting suggestion of tags that were not used together before – folksonomies cannot do that.

Semantic similarity: a FO can support the computation of semantic similarity between concepts and, by extension, between tags; so, they can expand the usual techniques that focus only at syntactical similarity and co-occurrence of tags, achieving better results in discovery operations.

Ontology evolvement: a FO can be used to find missing relations in ontologies; the high co-occurrence between two groups of tags, and their corresponding concepts, can indicate a necessary relation in the ontology, if it does not exist yet.

The paper is organized as follows. In section 3.2 we introduce our formal framework. In section 3.3 we discuss the semantic foundations of this work. In section 3.4 we present a formal perspective of related work. In the section 3.5 the folksonomized ontologies are defined and formalized. In section 3.6 we conclude and discuss the future work.

3.2 Formal Framework

In order to substantiate our analysis of related work and to make explicit the main characteristics and the differential of our approach, we have defined an abstract framework composed by a set of models. Although there is related work which formalize individual models presented here – e.g., folksonomies and ontologies – as far as we know, this paper contributes as the first initiative to embrace a wider scenario, including approaches aimed to relate folksonomies and ontologies. It also contributes as a tool to explicit the characteristics of each approach and its differentials (see next section).

The starting point is an existing model of folksonomies proposed by Mika [27]. He departs from three fundamental sets to define folksonomies: A = actors (users annotating), T = tags, O = annotated objects (e.g., images, bookmarks). In tag-based annotation systems, users tag objects, defining ternary associations. Therefore, a folksonomy system is a set of annotations $F \subseteq A \times T \times O$. This folksonomy can be alternatively modeled as three bipartite graphs, representing associations between actors and tags (AT); tags and objects (TO); actors and objects (AO).

There are three different classes of nodes in these models: N_t, N_{ts}, and N_c. Each node of the N_t class is a single tag. A node of the N_{ts} class, on the other hand, represents a set of tags that share the same meaning – a *tagset*. Figure 3.1 illustrates a transition from a graph in which the nodes are tags – members of N_t – to a graph in which nodes are tagsets – members of N_{ts}. As illustrated in the figure, many edges connecting tag nodes of the original graph are combined in a single edge connecting tagset nodes. The N_{ts} nodes are depicted in the figures of the models in gray. Finally, each node of the N_c class is a concept of an ontology. The focus here is in the semantics assigned to each node instead of the label.

In our framework we defined five abstract models $M1$ to $M5$ aiminng to model aspects of folksonomies and their relations with ontologies. They are further detailed:

$M1$ (Figure 3.2) models co-occurrences in a folksonomy – i.e., the relations between tags that were used together – as a tuple (G_T, W_C), where $G_T = \langle T, E_T \rangle$ is an undirected graph with vertex set T formed by tags (members of the N_t class) and edge set E_T representing co-occurrences of tags. W_C is a weighting function $W_C : E_T \rightarrow \mathbb{N}$, producing a weight related to each edge, corresponding to the number of co-occurrences of the respective tags annotating the same object.

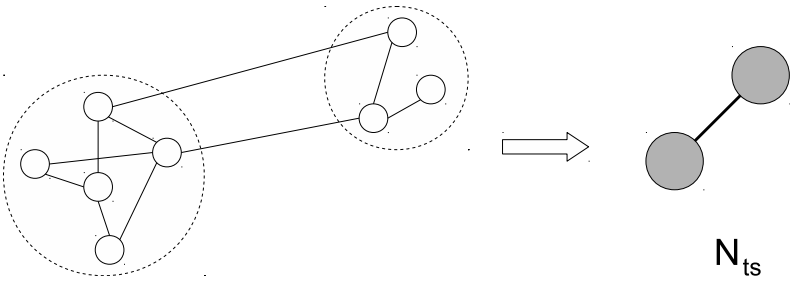

Figure 3.1: Tagset nodes.

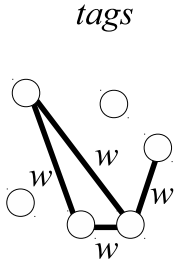

Figure 3.2: Model M1

$M1$ is a graph that represents the relatedness among tags. It is the raw material used by many proposals to synthesize ontologies. There are several approaches to define the relatedness [16]. They are mostly variations of co-occurrences of tags annotating resources.

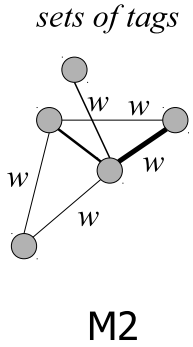

Figure 3.3: Model M2

$M2$ (Figure 3.3) models tagsets and their co-occurrences as a tuple (G_S, W_O), where $G_S = \langle S, E_S \rangle$ is an undirected graph with vertex set S formed by tagsets (members of

the $N_t s$ class) and edge set E_S representing co-occurrences of tagsets. W_O is a weighting function $W_O : E_S \rightarrow \mathbb{N}$, producing a weight related to each edge, corresponding to the number of co-occurrences of the respective tagsets annotating the same object.

Since tags with the same meaning are grouped together in $M2$, it is nearer to the way ontologies organize concepts. Many proposals use this model to relate tagsets with concepts from ontologies [35, 40].

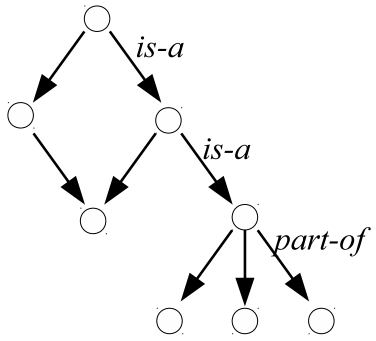

Figure 3.4: Model M3

$M3$ (Figure 3.4) is a simplified model of an ontology, represented as a tuple (G_O, RT, F_{RT}), where $G_O = \langle C, E_R \rangle$ is a directed graph with vertex set C formed by concepts (members of the N_c class) and arc set E_R representing relations between concepts. RT is a set of relation types between concepts. F_{RT} is a function $F_{RT} : E_R \rightarrow RT$, which associates a type with each relation (arc).

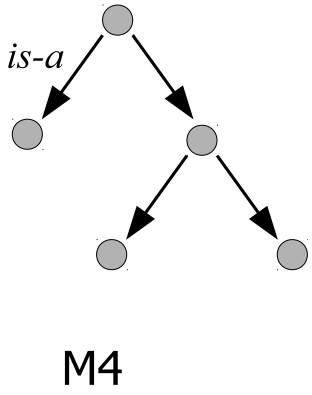

Figure 3.5: Model M4

$M4$ (Figure 3.5) models tagsets and their typed relations as a tuple (G_S, RT, F_{RT}), where $G_S = \langle S, E_R \rangle$ is a directed graph with vertex set S formed by tagsets (members

of the N_ts class) and arc set E_R representing relations between tagsets. RT is a set of relation types between tagsets. F_{RT} is a function $F_{RT} : E_R \rightarrow RT$, which associates a type with each relation (arc).

$M4$ models "social ontologies", similar to the Kotis et al. proposal [19] in the sense of the social aspect built-in. Compared to $M3$, $M4$ has tagsets (members of the N_{ts} class) in each vertex, rather than concepts. The term "social ontology" will be adopted here, contrasting with folksonomy, to emphasize this structure – mostly derived from a folksonomy – which makes explicit many relations among tags and whose structure resembles ontologies.

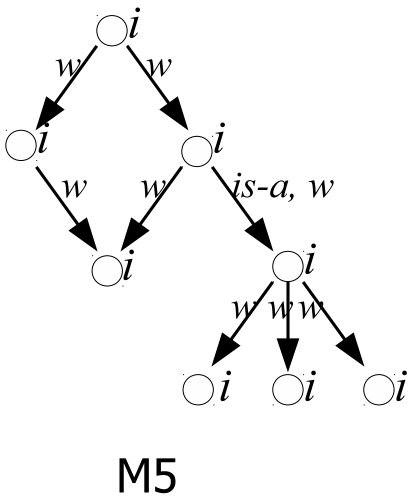

Figure 3.6: Model M5

$M5$ (Figure 3.6) models our proposed folksonomized ontology; it is derived from $M3$, incorporating semantic data extracted from folksonomies. This model is further detailed in Section 3.5.

3.3 Semantic Similarity

The similarity evaluation is in the core of any comparison mechanism and is a fundamental notion in this work. In this section we discuss the semantic similarity in ontologies or semantic networks, under the perspective of our models presented in the previous section. Departing from a scenario in which users use free-form tags as the main mechanism to classify content in folksonomy-based systems, the main challenge addressed in this work concerns how to improve the semantic interpretation of these tags to support operations of indexing, comparison, classification etc. We consider two main sources of semantics in

this scenario: (i) the implicit semantics derived from these folksonomies; (ii) the semantics imported from external ontologies and related to tags.

A practical scenario to employ this semantics – formalized in ontologies and potential in folksonomies – involves matching and similarity comparison. There are many applications, such as ontology engineering, information integration, and web query answering in which matching operations play a central role [13]. Matching operations applied to tag comparison can be organized in two main broad categories: lexical/syntactic and semantic. Lexical/syntactic approaches are mainly based on the proximity of spelling words and their derivations (e.g., conjugations). One example of this category is the edit distance, as the popular approach proposed by Levenshtein [21].

A challenge in semantic matching is how to weight the relevance of relations among concepts when confronting similarities. Consider a practical example of a program looking for the concepts similar to `judge`. The output will be a set of concepts ranked according their similarity. Two possible similar concepts in the example could be `district attorney` or `child`. Like a `judge`, the former is an official functionary and the latter is a person. A prerequisite to rank them by similarity is to define which concept is more similar to `judge`.

In order to put this comparison in a context, let us consider a model $M3'$ derived from $M3$. In $M3'$ all relation types are subsumption relations, i.e., the function $F_{RT} : E_R \rightarrow RT$ will always produce the type IS_A, member of RT. The ontology illustrated in Figure 3.7 follows the model $M3'$ and lower concepts specialize upper ones. A comparison supported by this ontology considers that there is a path connecting the compared terms. Figure 3.7 illustrates the three previous concepts as they appear in the WordNet ontology [28].

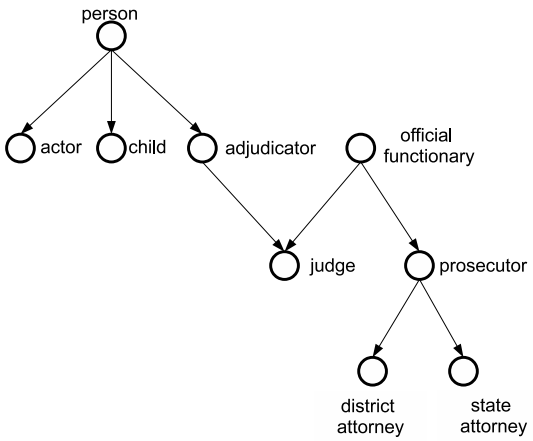

Figure 3.7: Subsumption ontology showing the relations among compared concepts - M3 model.

Beyond the primal comparison strategies founded on edge counting (like the one pro-

posed by Rada et al. [30]) and graph topology analysis (Wu and Palmer [45], for instance), there are approaches showing how to improve the comparisons taking into account the content of the ontology concepts. Resnik proposed an approach based on *information content* [31] applied to ontologies following the $M3'$ model. In this kind of ontology, each concept is a class representing a set of instances. If there is an edge connecting the vertex B (class B) to A (class A), then A subsumes B; B is a subclass and A is a superclass in the relation. A superclass is more general than a subclass. Therefore, a superclass will comprise more instances than its subclasses. As a consequence, a given instance has less probability of belonging to a subclass – e.g., `child` – than to a superclass – e.g., `person`. While the probability decreases, the information about more specific classes increases – a necessary consequence of their specialization. *Information Content* (*ic*) is a measure created to evaluate this increase of information about something. Let the probability of a given concept `c` be `p(c)`, then the *ic* value of `c` is defined as `-log p(c)` [32].

In order to illustrate Resnik's *ic*-based approach to evaluate the similarity among terms, let us apply it to the example involving the similarity ranking among `judge` and two other concepts: `district attorney` and `child`. The first step is to find the most specialized concept shared by `judge` and `district attorney`, which is `official functionary`, and shared by `judge` and `child`, which is `person` (see Figure 3.7). Intuitively, we can infer that an instance has less probability of belonging to `official functionary` than to `people`; conversely the *ic* value is higher. In an $M3'$ ontology a superclass will gather together only the common characteristics of all subclasses. `judge` is more similar to `district attorney` than to `child`, since `official functionary` has higher *ic* value than `person`. As `judge` and `district attorney` derive from a superclass with higher *ic*, they will have more commonalities. Therefore, Resnik [31] defines his similarity metric as follows:

$$sim_{res}(c_1, c_2) = \max_{c \in S(c_1,c_2)} [-\log p(c)] \tag{3.1}$$

In which the set S contains all concepts that subsume both c_1 and c_2. Experiments in [31] demonstrated that this approach produces better results than the counting edges approach and is not influenced by unbalances in ontology detailing.

In order to apply Resnik's similarity metric, ontologies following the $M3$ or $M3'$ models must also represent probability or *ic* related to each concept. Our approach, the folksonomized ontology (model $M5$) encompasses the *ic* value retrieved from folksonomies. It departs from $M1$, processing it to obtain $M2$, and then it fuses $M2$ and $M3$, combining the social knowledge of the former and the structure of the later, resulting in the enriched $M5$. All this process is further detailed in the Section 3.5.

3.4 Related Work

In this section we discuss the related work, from the perspective of our formal framework, using the models introduced in Section 3.2.

Gruber's classical definition of ontology as "an explicit specification of a conceptualization" [14] synthesizes its key aspect as an intentionally systematized – or engineered [27] – specification. According to Shirky [34], tag-based approaches differ from ontologies since tags organization derives from an organic work. It is a shift from a binary categorization approach – in which a concept A "is" or "is not" part of a category B – to a probabilistic approach – in which a percentage of people relates A to B. Gruber [15], on the other hand, claims that folksonomies and ontologies should not be seen as opposite but rather as complementary, and he proposes a TagOntology – a common ontology for tagging. As we will present in this paper, we share Gruber's view of complementary roles. Nevertheless, existing proposals usually are unidirectional, attaching folksonomy's tags to ontologies or, conversely, producing ontologies from folksonomies.

The main purpose of this section is to show the way related work explore the relation between folksonomies and ontologies. The summary of our analysis is presented in Table 3.1. Column 1 relates the authors analyzed in this section; column 2 sumarizes the path of the models followed by each approach; column 3 indicates auxiliary resources and models used in the process; column 4 defines the role of the auxiliary resources and models mentioned in column 3. In the last row, we present our fusion approach of folksonomized ontologies. Our main argument here concerns the unidirectionality of the initiatives, i.e., they use folksonomies as the main raw material and ontologies as auxiliary.

All analyzed proposals depart from the model $M1$, since the co-occurrences of tags is a metric to express the latent semantics of folksonomies. From these co-occurrences, Cattuto et al. [11] calculated several measures of tag relatedness by using an auxiliary ontology, the WordNet ($M3$). They do not group related tags in tagsets; each individual tag of $M1$ is associated with a synset in the WordNet ontology. Synsets are sets of synonyms that play an equivalent role of concepts in ontologies. The similarity of the related synsets are then transferred to the respective tags.

Specia et al. [35] proposed a technique to map clusters of tags to ontology concepts, in order to make explicit the semantics of the tag space. They departed from $M1$ creating clusters of high-related tags (tagsets) and relating them to produce $M2$, using co-occurrence information. The relations between these clusters were aligned with auxiliary external resources like Wikipedia, Google, and ontology bases – following the semantic standards ($M3$) – to produce $M4$. Those resources were used to improve the folksonomic data, mainly making explicit the semantics of the tags in the model $M1$.

In a similar way Tesconi et al. [40] used external resources, namely Wikipedia, and

Table 3.1: Work Comparison.

Authors	Path	Auxiliary	Auxiliary Role
Cattuto et al. [11]	tags (M1) → ontology	M3 - Wordnet	Measure tag relatedness (M1)
Specia et al. [35]	tags (M1) → tagsets (M2) → ontology (M4)	M3, Google, Wikipedia	Explicit semantics (M1)
Tesconi et al. [40]	tags (M1) → tagsets (M2) → ontology (M4)	M3 - WordNet, YAGO, Wikipedia	Disambiguate tags (M1)
Damme et al. [42]	tags (M1) → tagsets (M2) → ontology (M4)	M3 - WordNet, Google, Wikipedia	Derive ontologies (M4)
Cantador et al. [10]	tags (M1) → tagsets (M2) → ontology (M4)	M3 - WordNet, Wikipedia	Explicit semantics (M1)
Bang et al. [8]	tags (M1) → tagsets (M2) / ontology (M4)		
Heymann et al. [16]	tags (M1') → ontology (M4)		
Limpens et al. [22]	tags (M1) → ontology (M4)		Propose and review tags (M1)
Alves et al. [3]	(tags/tagsets (M1/M2) ↔ ontology (M3)) → FO (M5)		

ontologies ($M3$) like WordNet and YAGO [37]. Their objective was disambiguate tags, "semantifying" them. They developed an algorithm to disambiguate tags, grouping them by sense. The output of this algorithm is an entity like the model $M2$. Its tagsets are finally linked to Wikipedia categories and ontology concepts, producing $M4$.

Damme et al. [42] aimed to use folksonomy data ($M1$) to build and to maintain ontologies. They employ lexical resources, like Leo Dictionary, WordNet, Google, and Wikipedia, to enrich the results of a preprocessing step, in which the tagsets are prepared and cleaned, resulting in $M2$. Then they map tagsets of $M2$ to concepts of $M3$ (ontologies). The relations of $M3$ are mapped back to $M2$, in order to produce $M4$. Finally, the folksonomy's community validates the resulting $M4$.

Cantador et al. [10] proposed a mechanism to filter and classify tags, producing $M2$. Then, they mapped these tagsets of $M2$ to knowledge bases like WordNet and Wikipedia, to discover the corresponding semantic entities. Different from previous approaches, in order to map $M2$ to $M4$ they predefined a set of possible categories and relation types among tagsets. In order to do so, they used direct association or natural language processing heuristics.

Bang et al. [8] proposed the concept of "structurable tags", in which tags can be linked through relations, allowing basic inference operations. They expanded the model $M1$, allowing users to create two types of relations between tags: inclusion and synonymy. These types of relations support the transformation of folksonomic data into more semantic models. Thanks to the synonymy relation, the system transforms the data into the

model $M2$, grouping the tags with the same meaning. On the other hand, the inclusion relation led to an hierarchical organization, as a simplified $M4$.

Heymann et al. [16] proposed an algorithm to build a graph $M4$ departing from a variation of $M1$, in which the edges are unweighted. It first aggregates tags in *tag vectors*, in which the $v_{t_l}[o_m]$ corresponds to the number of times that the tag t_l annotates the object o_m. In the resulting unweighted $M1$, the vertexes will be the tags, and there will be an edge for pair of tags whose relatedness is above a threshold. The resulting graph, without weights and maintaining just the relevant edges, contains a "latent hierarchical taxonomy". It is captured by an algorithm that builds a subsumption hierarchy, derived from the centrality of each node in the graph.

3.4.1 Changing Users' Behavior

Many initiatives involve changing the way users tag resources. Tanasescu et al. [39] propose an approach in which tags can be also applied to annotate other tags, producing a network of "inter-annotated" tags. It adds an ontology-like approach to produce knowledge by using tags. This approach can reduce the ambiguity of the folksonomic data, because two tags with the same spelling but different meanings would have different describing tags. Their relations are expressed as triples, linking them to semantic web standards, like RDF [20].

For [22] the $M4$ is only a starting point for an annotation system based on controlled tags, in which it can propose new tags and review third-party tags. The system tracks each proposition and review in RDF, allowing multiple and even conflicting views of descriptions of resources. This way of enhancing the tagging process can be interpreted as a halfway between free tags and more formal description systems, e.g., ontologies.

The purpose of this work is to explore the social tagging approach as it is, without modifications. Therefore, the works of [39, 22] is out of the scope of this comparison.

3.4.2 Unidirectional Approach

As can be observed in our synthesis of related work, all approaches follow almost the same path, producing social ontologies ($M4$) from data extracted from folksonomies. Ontologies appear as adjuncts, making the semantics of tags explicit and helping operations of tag disambiguation and similarity evaluation. Nevertheless, the rich structure of the ontology is not appropriated and the produced $M4$ social ontology is limited to those simple relations – usually subsumption relations – which can be inferred from tags. Our proposal, described in the next section, overcomes this limitation.

3.5 Folksonomized Ontologies

In this section we describe our *folksonomized ontology* (FO). This section summarizes the main characteristics previously presented in [3] from a new point of view. It describes our FO from the perspective of the formal framework, introduced in Section 3.2, and confronts it with related work following the same perspective.

Our fusion approach presented here takes advantage of both ontologies and folksonomies to produce a synthesis. This fusion results is a *folksonomized ontology* (FO), which we define as an ontology aligned with terms of a folksonomy and enriched with their contextual data. By contextual data we mean data which emerges from a statistical analysis of a folksonomy, e.g., tag frequency, co-occurrence and information content. In one direction the folksonomized ontology, which is aligned with tags, drives richer semantic-based matching, categorization, and tag suggestion. In the other direction, contextual data will be used to enrich, review, and improve the ontology. In order to present our proposed model and to contrast it with related work, we will further present its formalization, defined as model $M5$ in Figure 3.6.

A FO is defined as a tuple (G, RT, F), where $G = \langle V, E \rangle$ is a directed graph with vertex set V formed by ontology concepts (members of the N_c class) and arc set E representing relations between these concepts, and RT is a set of relation types between concepts. F is a set of functions, they are: $\mathcal{F}_1$ is a weighting function $\mathcal{F}_1 : E \to \mathbb{N}$ where the weight of the relation is derived from the total of co-occurrences between tags represented by the respective concepts, the function $\mathcal{F}_2 : E \to RT$ defines the type of the relation as in ontologies (see $M3$) – in its first version, presented here, all relations are subsumptions, but the model is extensible to other types of relations –, and the function $\mathcal{F}_3 : V \to \mathbb{R}$ associates the *information content* (ic) related to each concept, calculated by $ic(c) = -\log p(c)$, where $p(c)$ is the probability of a given concept c. This ic value also derives from a statistical analysis of the folksonomy and will substantiate computations of semantic similarity between the concepts using, for example, Resnik similarity [31].

Figure 3.8 schematizes the roles played by an ontology and a folksonomy in the proccess of building a folksonomized ontology. It departs from an ontology ($M3$), which was previously engineered to formalize concepts and typed relations, e.g., is-a, same-as, part-of. Concepts and relations in folksonomies, on the other hand, are inferred by statistical analysis over tags and their co-relations. They are not typed, as in ontologies, but carry substantial contextual data, which substantiate "weighting" concepts and relations. A refined folksonomy, presented in Figure 3.8, follows the $M2$ model, aggregating tags that share the same meaning in tagsets. The resulting folksonomized ontology is a new entity that fuses the best of both worlds, having typed and "weighted" concepts and relations. To this purpose, tagsets are previously aligned to concepts.

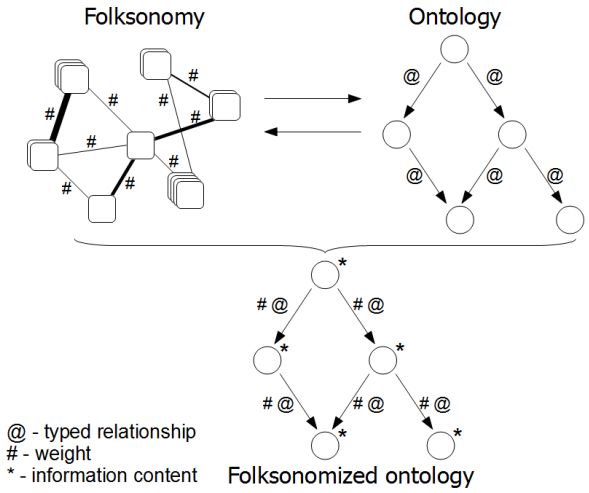

Figure 3.8: Folksonomized Ontology

A practical tool was developed in this research, apt to build folksonomized ontologies and use them for tag searching and discovery, as to ontology review and improvement. Figure 3.9 summarizes the cycle of the folksonomized ontology building and use. See [3] for a detailed description.

It starts collecting data from folksonomy systems, e.g., Delicious and Flickr. The output of this step is a folksonomy modeled as $M1$. The following step processes, filters, and groups the tags as tagsets (model $M2$). Tagsets are mapped to concepts in preexisting ontologies (modeled as $M3$). The probability and *ic* values for each tagset, as the co-occurrence of tagsets, are calculated and fused with the ontology, obtaining our folksonomized ontology (model $M5$). The review step is an ongoing work in this research; it confronts statistical data extracted from a folksonomy with the structure of an ontology, in order to support ontology review and improvement.

In order to illustrate how folksonomized ontologies can improve operations in tag-based systems, let us consider an example of a user looking for bookmarks related to "district attorney" in a system (e.g., Delicious). There are many relevant bookmarks which are not marked with this specific tag. Current folksonomy-based approaches (based on models $M1$, $M2$, and $M4$) will expand the search only considering related tags with a high co-occurrence value with *"district attorney"*, and will not consider other important relations that were not frequently tagged together. Some approaches that use $M3$ ontologies to find similarities among tags, cannot apply improved similarity algorithms, as proposed by Resnik [31] (see Section 3.3), due to the lack of statistical data related to the ontology. Our FO will offer both: relations engineered in ontologies and captured from folksonomies fused together. The statistical data will support better similarity comparisons, as proposed by Resnik.

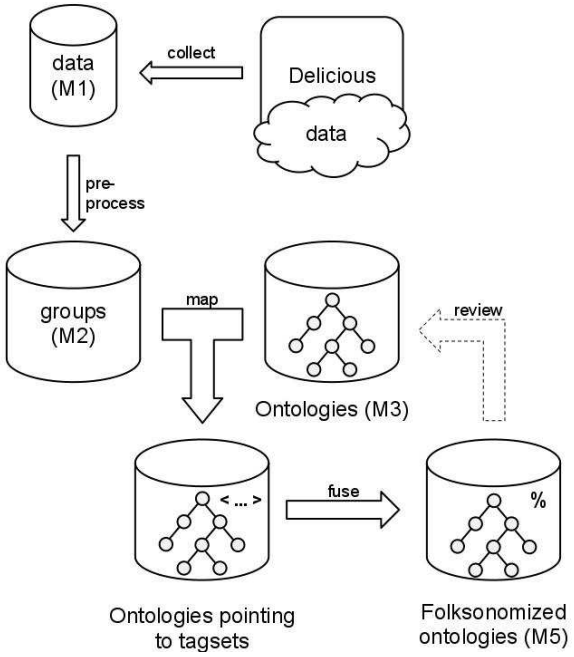

Figure 3.9: Folksonomized ontology building and use

Users looking for highly frequent subjects will have a similar problem. Consider a user who types the tag *"mathematics"* in the search box. Since there are millions of bookmarks tagged as mathematics, a similarity algorithm can be applied to rank bookmarks which have more tags similar or related to the subject. The same approach can be applied for tag suggestion when the user is adding a new bookmark. In this case, systems like Delicous suggest other tags that were used in the same bookmark. For example, the tags *"math"*, *"maths"*, *"interactive"*, and *"numeracy"* are returned in the Delicious system. A FO-based system can go beyond, by suggesting co-related terms even when they are not used together before in the same bookmark, i.e., even though they are not connected in $M1$ and $M2$, e.g., *"science"* and *"geometry"* would also be suggested by the system.

Existing approaches to integrate folksonomies and ontologies are based on mapping tags or tagsets to ontology concepts. The relations among tags mapped to concepts can be derived from co-occurrence analysis of the tags, or from the relations that already exist in the ontology. As observed in the previous section, the final product of existing approaches is a "social ontology" $M4$. The concepts of this model are limited to those extracted from tags aligned to ontologies. Preexisting concepts from ontologies, not present in the folksonomies, will not be present in $M4$. The semantics of ontologies enriches the analysis of tags in a unidirectional way, i.e., statistical data from folksonomies are not used to improve the similarity analysis in the ontologies. The FO, on the other hand, preserves preexisting ontology nodes that cannot be mapped to tagsets. They are explored to do

inferences which are not possible in related work. Moreover, the FO model $M5$ represents more than $M4$ relations among concepts, capturing weights of relations and probabilities, to support better inferences (see Section 3.3).

A FO is built on the assumption that the semantics of folksonomies can be also applied to refine the ontology itself. For this reason, in one direction FOs support suggestion of related tags that do not appear together in the folksonomy annotations; in inverse direction, other relevant aspect that emerges from the folksonomized approach is the possibility of verifying a relation that does not exist in ontology, but is strong in the folksonomy. This information can be used to evolve and improve ontologies.

3.6 Conclusion

This paper contributed in three important issues concerning FOs, firstly presented in [3]. It introduces a formal framework to support a formal presentation of models addressing folksonomies, ontologies, social ontologies and their relationship. This framework substantiated a characterization and comparison of related work, including the FO. As far as we know, this is the first initiative to produce such a framework and to compare the models adopted by the related work from a formal perspective.

We also presented in a formal perspective the advantages of using the folksonomized ontologies compared to related work, due to its hybrid approach fusing folksonomies and ontologies. It is a symbiotic combination, taking advantage of both semantic organizations. Ontologies provide a formal semantic basis, which is contextualized by folksonomic data, improving operations over tags based on ontologies. Conversely, the folksonomized ontologies can also be used as tools to analyze the ontology quality and to help the process of ontology evolution, showing the discrepancies between the emergent knowledge of a community and the formal representation of this knowledge in the ontology.

We have implemented a practical experiment with 1,049,422 extracted from Delicious. Our prototype can build a FO, restricted to generalization relationships. Future work include: (i) to expand the folksonomized model to include other relations (besides the generalization); (ii) to run tests in specialized contexts applying domain ontologies; (iii) to improve our tool for ontology evaluation and review.

Chapter 4

Folksonomized Ontologies and the 3E Steps Technique to Suport Ontology Evolvement

4.1 Introduction

An ontology, as a shared conceptualization, expresses a consensus among people, conducting to a consensus among machines. There are several strategies to look for a consensus, e.g., a selected group of representatives and/or specialists designs an ontology incorporating a consensual perspective of a given domain [41]; tools extract latent semantics from a body of digital artifacts produced by many people (a statistical consensus), automatically deriving it to an ontology [25]. In the second case, a promissory source of latent semantics comes from social tagging mechanisms offered by a great number of web systems for content storage, indexation and sharing. It is based on the free-form tags that the users can associate to each resource, without the need of a central vocabulary. In this context, the photo sharing system Flickr has more than 5 billion images hosted and there are more than 180 million URL addresses on Delicious. Beyond a collection of tags produced by individuals, systems promote interactions among people, resources and their tags – e.g., frequency-based tag suggestion, recommendation of related tags. The term folksonomy – combining the words "folk" and "taxonomy" [43] – has been used to characterize the product which emerges from this tagging in a social environment.

Gruber's classical definition of ontology as "an explicit specification of a conceptualization" [14] synthesizes its key aspect as an intentionally systematized – or engineered [27] – specification. According to Shirky [34], tag-based approaches differ from ontologies since tags organization derives from an organic work. It is a shift from a binary categorization approach – in which a concept A "is" or "is not" part of a category B – to a probabilistic

approach – in which a percentage of people relates A to B. Gruber [15], on the other hand, claims that folksonomies and ontologies should not be seen as opposite but rather as complementary, and he proposes a TagOntology – a common ontology for tagging.

As we will present in this chapter, we share Gruber's view of complementary roles. A folksonomy can represent a perspective of a wider group, but the semantics extracted from the implicit relations among tags are rather simple. An ontology is usually built by a more restrict group, but has the richness of an engineered product. There are initiatives towards exploring the interplay between folksonomies and ontologies. However, one common problem concerns their unidirectionality, i.e., in one direction there are proposals to use ontologies to improve tags' semantics, in the other direction there are proposals to extract the implicit/potential semantics of folksonomies in order to produce ontologies.

Differently from traditional techniques, we proposed a fusion approach, called *folksonomized ontology* (FO), which goes beyond this unidirectional perspective [3]. In one direction, the ontologies are "folksonomized", i.e., the latent semantics from the folksonomic tissue is extracted and fused to ontologies. In the other direction, the knowledge systematically organized and formalized in ontologies gives structure to the folksonomic semantics, enhancing operations involving tags, e.g., content indexation and discovery. The folksonomic data fused to an ontology will tune it up to contextualize inferences over the repository.

Beyond the advantages of FOs we have shown in previous papers [3, 5] – concerning enhanced tag disambiguation, tag suggestion and semantic similarity – they can be used to support the review and enhancement of ontologies. The social semantics – produced by a wide group of persons in their concrete needs of classification –, offers relevant information for the restrict group of specialists, which can use them to enhance and update their ontologies. In this sense, while a FO embeds the relations among ontologies and folksonomies, it can make explicit concepts and relations extracted from the folksonomies, which are not present or contrast with a given ontology.

This chapter focus on a technique we propose to support ontology review and enhancement, which we call *3E Steps*: Extraction, Enrichment and Evolution. It is founded on our work concerning the folksonomized ontology [3] and its abstract framework [5], which constitute the two first steps of the technique. In order to validate our proposal, we developed a tool to extract folksonomic data from Flickr and Delicious and to integrate them into the WordNet ontology [28]. The data is further used in a visual tool – first presented in this chapter – that supports ontology review and enhancement.

In our point of view, our *3E Steps* technique opens an interesting field of applying latent semantics, socially produced by wide communities, to improve engineered ontologies. Related work addressing ontologies and folksonomies does not explore the full potential of this interaction, since they do not retain the richness of the semantics from both sources.

This chapter is organized as follows. In Section 4.2, we discuss the basis of our work. In Section 4.3, we present our *3E Steps* technique. In Section 4.4, we describe the tool developed. In Section 4.5, we present the related work. In the Section 4.6, we conclude and discuss the future work.

4.2 Ontologies, Folksonomies and Similarity

4.2.1 Ontologies and Semantic Similarity

Due to the wide spectrum of possible interpretations to the term ontology [44], we start this section by presenting an abstract model that represents our perspective to an ontology in this work. It is a simplified graph-based model of an ontology, which is the underlying model adopted by most of the related work, as we detail in [5].

An ontology is represented as a tuple (G_O, RT, F_{RT}), where $G_O = \langle C, E_R \rangle$ is a directed graph with vertex set C formed by concepts and arc set E_R representing relations between concepts. RT is a set of relation types between concepts. F_{RT} is a function $F_{RT} : E_R \rightarrow RT$, which associates a type with each relation (arc). See more details of this model and its relationship with other models – e.g., folksonomies and social ontologies – in [5].

One way to explore the semantics – formalized in ontologies and potential in folksonomies – involves matching and similarity. There are many applications, such as, ontology engineering, information integration and web query answering where matching operations play a central role [13]. When tags are compared, matching operations can be organized in two main broad categories: lexical/syntactic and semantic. Lexical/syntactic approaches are mainly based on the proximity of spelling words and their derivations (e.g., conjugations). One example of this category is the edit distance, as the popular approach proposed by Levenshtein [21].

To go beyond the spelling, semantic approaches relate words to a respective semantic representation – a concept. The matching is evaluated by analyzing semantic relationships among concepts, e.g., equivalence, generalization, specialization etc. This approach can lead to better search results or expand the opportunity for discovery, by finding and ranking similar or related results. It can also subsidize better recommendation systems for tag definition. In this context, ontologies are increasingly being adopted to formalize the semantics of concepts and their relationships.

A challenge in semantic matching is how to weight the relevance of relationships when similarities are confronted. Consider a practical example of a program looking for the concepts similar to `judge`. The output would be a set of concepts ranked by their similarity to the input concept – `judge`. Two possible similar concepts in the output are `district attorney` and `child`.

In order to put this comparison in a context, let us consider the abstract model presented in the beginning of this section. A comparison supported by this kind of ontology considers that the compared terms are connected by a path. Figure 4.1 illustrates the three previous concepts as they appear in the WordNet ontology, an ontology that meets our abstract model. In the figure, circles represent concepts (C in the abstract model) and arcs represent subsumption relations (E_R in the abstract model, associated to the is-a RT by the F_{RT} function) – lower concepts specialize upper ones.

As can be seen in the figure, in the same way as a `judge`, a `district attorney` is an official functionary. A `child` is a person, just like `judge`. To rank them by similarity it is necessary to define which concept is more similar to the concept `judge`.

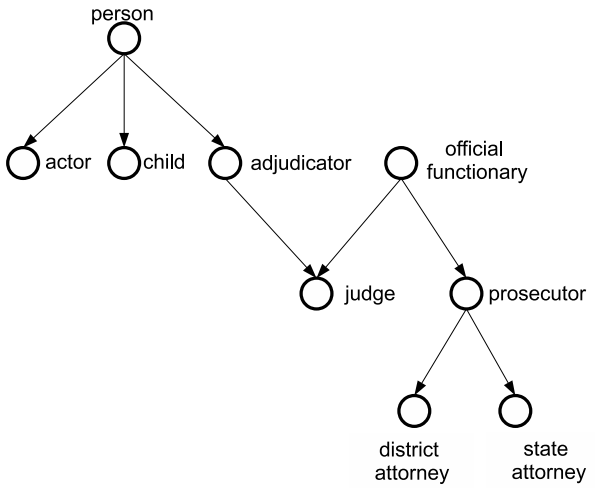

Figure 4.1: Subsumption ontology showing the relationships among compared concepts.

Many approaches to calculate semantic similarity based on ontologies were developed and we will further present some relevant techniques.

Path-based

A naive method to evaluate the semantic similarity between two nodes in an ontology is by measuring the shortest path separating them. This is equivalent to the distance metric in a *is-a* (subsumption) semantic net, defined by Rada et al. [30]: $distance(c_1, c_2)$ is the minimum number of edges between c_1 and c_2. The similarity then calculated as:

$$sim_{rada} = [1 + distance(c_1, c_2)]^{-1} \tag{4.1}$$

As showed in [31], this approach is highly influenced by the level of detail applied to describe branches of the ontology, i.e., branches better detailed can contain longer paths than other, in spite of the similarity distance, leading to biased evaluations. For example,

the comparison between `judge` and `child` (3 edges) – illustrated in Figure 4.1 – results in the same similarity of `district attorney` compared to `judge` (3 edges). One way to overcome this limitation is by weighting the edges, leading to the problem of how to determine the weights. According to Jiang and Conrath [17] there are ontology aspects, such as depth of nodes and type of links, which can be used to define these weights.

Depth-relative

One way to enhance the path-based comparisons is by analyzing the most specialized common ancestor shared by two nodes in the ontology. It is founded on specific kinds of taxonomic ontologies based on subsumption relationships among terms, as the example of Figure 4.1. Observations showed that siblings sharing an ancestor deep in a hierarchy are more closely related than those sharing an ancestor higher in the hierarchy [38]. Therefore, Wu and Palmers [45] propose the following metric:

$$sim_{wp}(c_1, c_2) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3} \tag{4.2}$$

In which c_3 is the least (most specialized) common ancestor of both c_1 and c_2, N_1 is the number of nodes on the path from c_1 to c_3, N_2 the number of nodes between c_2 and c_3, and N_3 the number of nodes between c_3 and the ontology root.

To improve the depth-relative metrics, Shickel and Faltings [33] proposed the OSS metric, based on an A-Priori Score *APS* computation of all concepts in an ontology. Then, a distance metric is defined from two coefficients (generalization and specialization) calculated from the *APS* value.

Content-based

Besides the ontology topology, there are approaches showing that comparisons can be improved by analyzing also the content of the ontology concepts. The content drives the weighting of concepts, which in turn supports similarity algorithms. Resnik proposed an approach based on *information content* [31] applied to subsumption ontologies. Assuming that each concept in this kind of ontology is a class representing a set of instances, the probability of a given instance to belong to a more specific class – e.g., `child` – is lower than the probability to belong to a more general one – e.g., `person`. While the probability decreases, the information about more specific classes increases – a necessary consequence of their specialization. *Information Content* (IC) is a measure created to evaluate this increase of information about something. Let the probability of a given concept `c` be `p(c)`, then the IC of `c` is `-log p(c)` [32].

In order to illustrate Resnik's IC-based approach to evaluate the similarity among terms, let us return to the example involving the similarity ranking among `judge` and two

other concepts: `district attorney` and `child`. The first step is to find the most specialized concept shared by `judge` and `district attorney`, which is `official functionary`, as by `judge` and `child`, which is `person`. Intuitively, we can infer that the probability of an instance to belong to `official functionary` is smaller than the probability of an instance to belong to `people`; conversely the IC is higher. In this type of ontology, when two concepts derive from the same generalization they share its characteristics, therefore, `judge` is more similar to `district attorney` than to `child`, since the former has higher IC. Therefore, the Resnik [31] similarity metric was defined as follows:

$$sim_{res}(c_1, c_2) = \max_{c \in S(c_1, c_2)} [-\log p(c)] \tag{4.3}$$

In which the set S contains all concepts that subsume both c_1 and c_2. Experiments [31] demonstrated that this approach produces better results than the counting edges approach and is not influenced by unbalances in ontology detailing. There are many other approaches exploring probabilities to improve similarity evaluation such as Lin [23] and Jiang and Conrath [17].

All of these probability-based approaches lead to an extra challenge, which is addressed in this work: how to evaluate the probability of each concept of an ontology. Resnik's strategy is based on counting words extracted from a corpus of documents.

4.2.2 Folksonomies

In folksonomy-based systems, users can attach a set of tags to resources. These tags are not tied to any centralized vocabulary, so the users are free to create and combine tags. Some strengths of folksonomies are their easiness of use and the fact that they reflect the vocabulary of their users [26]. In a first glimpse, tagging can transmit the wrong idea of a poor classification system. However, thanks to its simplicity, users are producing millions of correlated tags. It is a shift from classical approaches – in which a restricted group of people formalize a set of concepts and relations – into a social approach – in which the concepts and their relations emerge from collective tagging [34]. In order to perform a systematic folksonomy analysis, to subsidize the extraction of its potential semantics, researchers are proposing models to represent its key aspects. Gruber [15] models a folksonomy departing from its basic "tagging" element, defined as the following relation:

$$Tagging(object, tag, tagger, source) \tag{4.4}$$

In which *object* is the described resource, *tag* is the tag itself – a string containing a word or combined words –, *tagger* is the tag's author, and *source* is the folksonomy system, which allows to record the tag provenance (e.g., Delicious, Flickr etc.).

Table 4.1: Symbols for types and values in FOs.

Symbol	Description	Abstract Model
#	Weight of the relation derived from the total of co-occurrences between tags.	$\mathcal{F}_1$
@	Type of the relation.	$\mathcal{F}_2$
$*$	Frequency of the node, or one of its derivatives: probability or information content (IC).	$\mathcal{F}_3$

4.3 3E Steps Technique

In this section, we describe our *3E Steps* technique – Extraction, Enrichment and Evolution – to review and enhance ontologies, as well as, its relation with our approach to fuse an ontology and a folksonomy, the *folksonomized ontology* (FO). The FO concept and model as well as the two first steps of the technique – Extraction and Enrichment – are based on works previously described in [3, 5]. Since they are essential parts of *3E Steps* technique, they are summarized here.

4.3.1 Folksonomized Ontology

We define a folksonomized ontology as an ontology aligned with terms of a folksonomy and enriched with their contextual data. By contextual data we mean data which emerges from a statistical analysis of a folksonomy, e.g. tag frequency, co-occurrence and information content. The ontology to be folksonomized can be of any kind, which meets the abstract model presented in the beginning of Section 4.2.1. The choice of the domain covered by the ontology and the folksonomy have direct impacts in the results. The results will be as good as the overlap between their domains.

In one direction, the FO, which is aligned with tags, drives richer semantic-based matching, categorization and tag suggestion. In the other direction, contextual data is used to review and improve the ontology. Figure 4.2 schematizes the roles played by an ontology and a folksonomy in a folksonomized ontology building. The symbols in the figure are described in Table 4.1, whose column *Abstract Model* will be further explained.

Following Figure 4.2, the ontology was previously engineered to formalize concepts and typed relationships, e.g., is-a, same-as, part-of. Concepts and relationships in folksonomies, on the other hand, are inferred by statistical analysis over tags and their co-relations. They are not typed, as in ontologies, but carry substantial contextual data, which subsidizes "weighting" concepts and relationships. The resulting folksonomized ontology is a new entity that fuses the best of both worlds, having typed and "weighted" concepts and relationships.

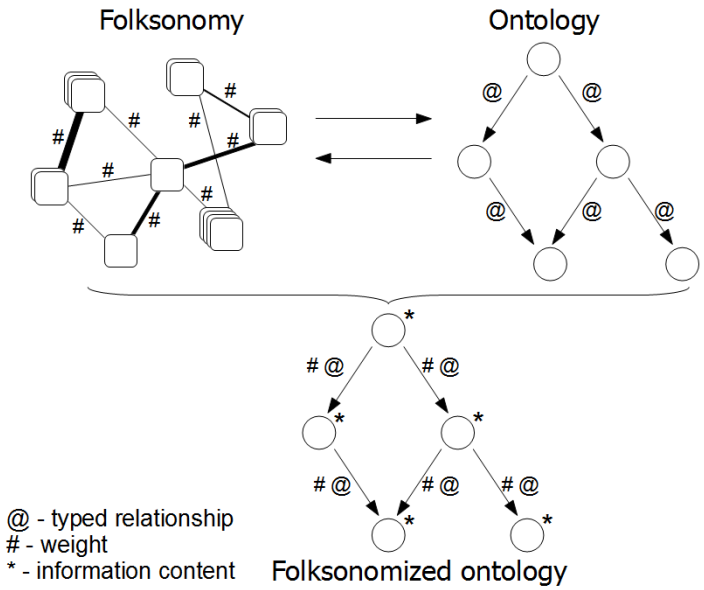

Figure 4.2: Folksonomized Ontology

In a previous paper, we defined an abstract framework to make explicit the underlying models addressed in related work concerning folksonomies, ontologies and their relationships. They supported our abstract model for folksonomized ontologies [5]. In this subsection, we summarize this model, which will be the basis to present our technique and respective tool.

A FO is defined as a tuple $(G, RT, \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3)$, where $G = \langle V, E \rangle$ is a directed graph with vertex set V formed by ontology concepts and arc set E representing relations between these concepts, RT is a set of relation types between concepts, $\mathcal{F}_1$ is a weighting function $\mathcal{F}_1 : E \to \mathbb{N}$ where the weight of the relation is derived from the total of co-occurrences between tags represented by the respective concepts, the function $\mathcal{F}_2 : E \to RT$ defines the type of the relation as in ontologies and the function $\mathcal{F}_3 : V \to \mathbb{R}$ associates the *information content* (ic) related to each concept, calculated by $ic(c) = -\log p(c)$, where $p(c)$ is the probability of a given concept c. This ic value also derives from a statistical analysis of the folksonomy and will substantiate computations of semantic similarity between the concepts using, for example, Resnik similarity [31]. Table 4.1 relates the functions presented in this abstract model (last column) to the symbols of FO diagrams.

In our practical experiments, the $\mathcal{F}_2$ function is still limited to subsumption relations. However, for the sake of generality, we distinguish our design (abstract model) from our implementation, which is an instance of the abstract model. This distinction predisposes the design to future expansions of its implementation, which we plan in future develop-

ments.

Our work expands Resnik proposal in three directions:
(i) proposing a strategy for calculating probabilities and IC of concepts based on tags employed in social networks to describe content;
(ii) defining multiple context-driven IC for each concept;
(iii) applying IC and co-occurrence data to review and enhance the ontology.

In the following sections, we will describe our technique, illustrated in Figure 4.3, involving three steps: *Extraction* – the folksonomy data are mined in order to collect the metadata used in the next step; *Enrichment* – the latent semantics from the folksonomic tissue is extracted and fused with ontologies, and it comprises the map and fuse phases; and *Evolution* – the ontology managers could analyze the FO data and visualize the cases in which the collaborative knowledge indicates that the ontology needs to be reviewed and/or enhanced. The first two steps (gray boxes) are divided in phases illustrated inside the gray boxes.

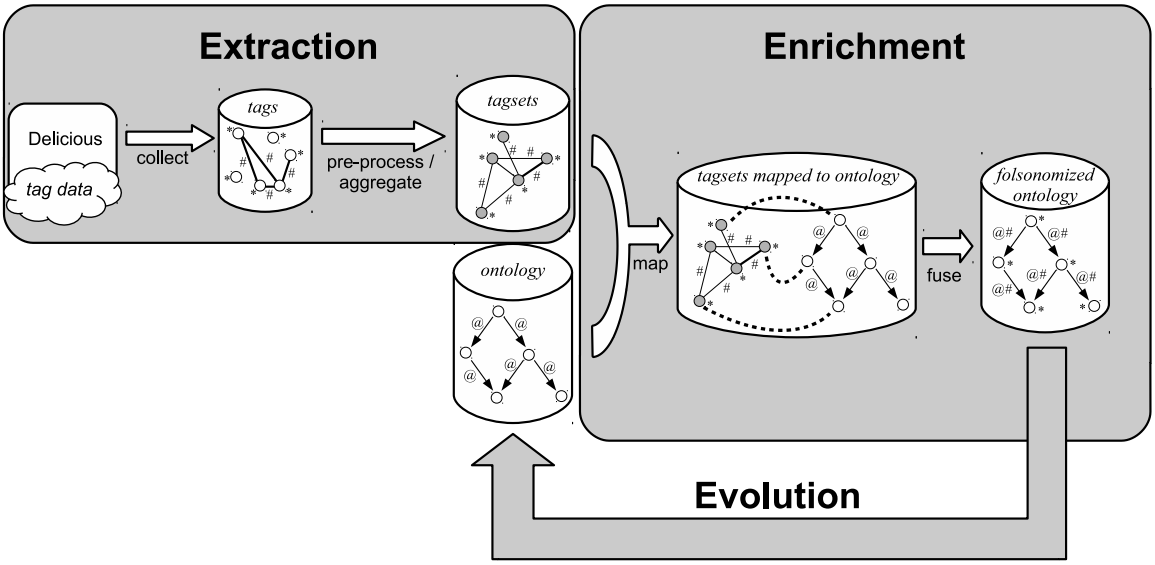

Figure 4.3: 3E Steps Technique

At the end of each phase in the figure, there is a storage symbol representing the intermediate data produced by the respective phase. Inside each storage, there is a graph diagram synthesizing the main elements captured by the respective phase. They will be further detailed in the description of each phase.

4.3.2 Extraction

This step is organized in two phases: collect and pre-process/aggregate. The collect phase involves accessing external systems in order to retrieve tag data from primary sources. The pre-process/aggregate phase cleans the data and aggregates tags according to their meaning in tagsets.

Web-based content portals offer web service interfaces to access their data (APIs). Due to the heterogeneity in the APIs, the tag data collecting module was designed to be customizable and it was tested in Delicious and Flickr systems. It access these web services to select and retrieve tags and their metadata, which are stored in a database.

In order to better obtain the emergent properties of the semantics extracted from folksonomies, this module was designed to afford large datasets. They are stored as triples of resources, users and tags, including their relations. Statistical data were computed and stored during data collection, avoiding extra post-processing work. These data – co-occurrence between tags and frequency (used to calculate probability and information content) – feeds subsequent phases that compute the values addressed by the functions $\mathcal{F}_1$ and $\mathcal{F}_3$ of a FO, previously in the formal model. The updating process is incremental, i.e., it collects and stores just the differences of a previous execution.

The output of the collect phase stored in the database is a collection of tags – represented as vertexes of a graph in Figure 4.3 –, their relations – represented as edges –, the frequency of each tag and co-occurrence between tags. The meaning of each symbol – "#", "$*$" and "@" – is described in Table 4.1. Data concerning users and resources are stored as well.

In the pre-processing phase, unusual tags were eliminated to improve the quality of the tag set. We classify as unusual those tags with low number of occurrences, i.e., $\leq$ a constant LO. The value of LO varies according to the size of the data set and the domain. In our practical experiments, we achieved the best results with $LO = 4$. The amount of unusual tags corresponded to less than 5% of the total. Those tags are, in most of the cases, wrong spelled tags – e.g., *folsonomy* – or personal tags – e.g., *toread*.

After this pre-processing phase, we aggregated tags that refer to the same term. For instance, the tags *tip* and *tips* are tightly connected and represent the same term. The grouping algorithm is divided in two steps: *marks analysis* – the algorithm groups tags differing only by special characters, e.g., "_", "-", ".", etc –; and *morphological analysis* – it groups tags by morphological relatedness.

In order to represent multiple-word tags, users resort to different strategies, e.g., concatenating words with or without separating signs. By analyzing the similarity of tags without the special characters we group tags like *search-engine*, *search_engine*, and *searchengine*. These tags are clearly very close to each other and represent different user approaches for using multiple-word tags. So, all special characters of tags were removed,

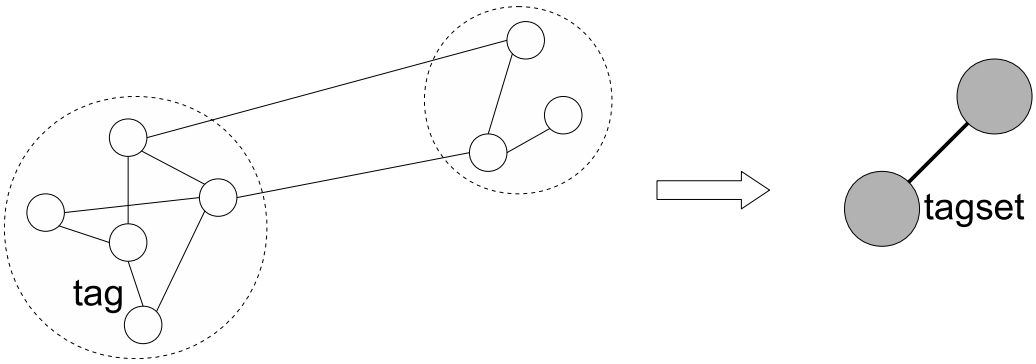

Figure 4.4: Tagset nodes.

allowing to group tags that became equal without punctuation.

The morphological analysis and grouping go beyond spelling comparisons, considering morphological variations as singular and plural tags, or tags of different verb tenses. The algorithm retrieves morphological variations of tags from the WordNet ontology, grouping them together.

The output of the pre-process/aggregate phase is a collection of tagsets – represented as vertexes of a graph in Figure 4.3 –, their relations, frequencies and co-occurrences – represented as edges, asterisks and sharps respectively (see Table 4.1). The gray node of a *tagset* represents a set of tags that share the same meaning. Figure 4.4 illustrates a transition from a graph in which the nodes are tags to a graph in which nodes are tagsets. As illustrated in the figure, many edges connecting tag nodes of the original graph are combined in a single edge connecting tagset nodes.

Each tagset represents a folksonomic concept. Tagsets will be the concept units to be confronted to concepts in the ontology. In the Evolution step, tagsets will play two important roles in ontology review and enhancement. In an external perspective, each tagset is an atom, embedding a concept shared by a community. In an internal perspective, a tagset encapsulates a network of interrelated tags concerning a shared meaning. Both perspectives are explored in the Evolution step, when confronting the folksonomic perception with ontologies.

4.3.3 Enrichment

This step is organized in two phases: map and fuse. The map phase involves mapping tagsets produced in the previous phase to concepts of an ontology. The fuse phase involves fusing the ontology to the folksonomic data to produce the folksonomized ontology.

The map phase is not a simple task, due to the lack of semantic information related to the tagsets. The tagsets cannot be directly mapped based on their words, since the same

word can have multiple meanings in the ontology. In WordNet, for instance, a word can have multiple senses, called synsets, which are differentiated through identifiers combining the original word plus two affixes. The first one is a character that describes the synset type (namely noun, verb, adjective, or adverb) and the second one is a sequential number to differentiate each meaning. For instance, the synset *dog.n.01* represents a noun and it is one of the synsets for the word *dog*.

To find out which concept – or which synset in Wordnet – corresponds of each tagset, we developed a technique that encompasses the relation of concepts (WordNet synsets) and tag co-occurrences, divided in three steps: (i) tagset key election – a tag of each tagset is chosen to represent it; (ii) co-occurrence selection – the co-occurrence values of the tags are selected; (iii) tagset key mapping – finally, each tagset is mapped to an ontology concept.

Consider a tag t, a tagset key, to be mapped to a synset and a set C containing the tags having the highest co-occurrence related to t (obtained in the pre-process step). Consider a group S containing all synset candidates for mapping. Our algorithm evaluates the distance of each synset s of S compared with t, in the following way: (i) the set C must have a minimum set of tags already mapped to synsets; this minimum is defined by a threshold constant *minmap*; (ii) the similarity of a given s to t is calculated by the sum of the distances of all c already mapped to s; (iii) since there is no IC data yet, a path-based similarity algorithm is applied. The synset s with the highest sum is the target of the mapping.

A tag group will only be processed if a minimum of the elements in the corresponding co-occurrence list had already been processed and mapped. Since the algorithm always selects a synset based on tags already mapped, it was necessary to create a starting set of tags manually mapped, to work as seeds. This initial selection of seeds influences the coverage of the mapping process; there is a tradeoff between the effort to manually map tags to concepts and the coverage obtained with the algorithm. See further details of the mapping algorithm at [3]. To the best of our knowledge, this algorithm has many innovative characteristics. In order to go beyond individual similarity analysis, it explores the network of relations among terms. The quality of the association between tagsets and concepts of an ontology, produced by this phase, will be essential to better analyze the ontology by confronting it with folksonomic data.

The result of this phase is illustrated in Figure 4.3. The graph of tagsets, their relations and statistical data, produced in the previous phase, is mapped to the graph of an ontology. The ontology represented as a graph – following the abstract model defined in Section 4.2.1 – is presented in the figure, the vertexes represent concepts and the edges their relations. The meanings of the symbols "@", "#" and "$*$" are defined in Table 4.1. The product of the map phase is a set of edges (dashed lines) mapping tagsets to concepts.

Each of these edges stores the degree of similarity between the tagset and the concept, to be further used in the ontology analysis.

The fusion phase combine the data from the previous phase to produce a single unified Folksonomized Ontology. The resulting graph presented in Figure 4.3 is expanded in Figure 4.2. It departs from the ontology and enriches it with the statistical data obtained in the previous phases. Therefore, concepts – which were mapped to tagsets – are enriched with information content (asterisks) and their relations are enriched with co-occurrence rates (sharps) – see Table 4.1.

Resuming the formal model presented in Section 4.3.1, the graph G, the RT set and the function $\mathcal{F}_2$ are derived from the preexisting ontology. The functions $\mathcal{F}_1$ and $\mathcal{F}_3$ represent the enrichment computed from folksonomies.

Besides its value in ontology analysis, we showed in an experimental evaluation [3] – involving 1,049,422 triples of resources retrieved from Delicious – that the enrichment provided by FOs can be explored to improve similarity analysis among concepts. The statistical data associated to the concepts support enhanced comparisons by applying the Resnik approach [31].

4.3.4 Evolution

During the implementation of the two previous steps we observed that our approach to relate folksonomies and ontologies produced a rich data set, which can support ontology review and enhancement:

A popular tagset without a respective concept in the ontology. It can indicate a candidate to a new concept to be added in the ontology.

A strong relation between two tagsets that has no correspondent relation between the respective concepts. It can indicate some important relation not represented in the ontology.

Tagsets embed rich information about relations among tags and concepts. A tagset aggregates many tags around a meaning. Its internal network of relations and the connections they have with the concepts in the ontology are rich sources for the analysis of how words are related to the meaning of concepts.

Therefore, the two previous steps were incorporated in our *3E Steps* technique, in which the third step is the evolution of the ontology. This is also an important step, because it leads to a symbiotic cycle, in which folksonomized ontologies help to tune up the underlying ontologies which, in turn, will improve the results of the folksonomized ontology itself.

4.4 Visual Review/Enhancement Tool

In order to support ontology evolvement, we developed a tool to visually explore the interplay between the latent semantics of the folksonomy system and a given ontology. It is important to point out that our tool is built upon the idea of offering more information to the managers of the ontology. It is designed to explore data and to suggest changes, but does not apply any automatic modification and does not offer support for ontology editing.

In this section, we describe the tool we developed to support ontology review and enhancement. In Subsection 4.4.1, we present the implementation details of the tool. In Subsection 4.4.2, we summarize its main characteristics. In Subsection 4.4.3, we describe the process of reviewing and enhancing ontology relations. In Subsection 4.4.4, we present practical examples of this process. In Subsection 4.4.5, we describe the process of analysis inside a tagset with examples.

4.4.1 Implementation Aspects

Figure 4.5 presents an architectural diagram of the tool. These are the main modules:

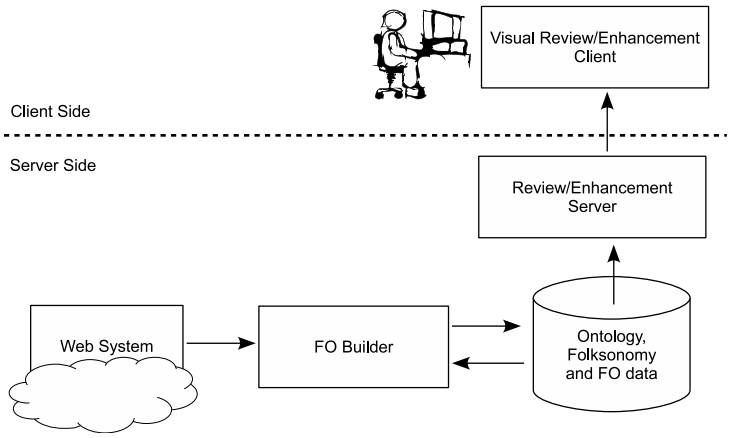

Figure 4.5: Architectural diagram of the review/enhancement tool

Web System Web-based repositories aimed at sharing content, links or metadata – e.g., Delicious and Flickr – which use tag-based classification mechanisms.

FO Builder Responsible for collecting folksonomy data from web systems. It implements the two first steps of the *3E Steps* technique (see Figure 4.3). The module is implemented in python and uses SQLite to manage the database. It is designed to be extensible, i.e., it allows developers to extend it to work with different kinds

of web systems. The default implementation works with Flickr and Delicious. The Delicious extension adopts a third-party library, the DeliciousAPI [29]. The module retrieves and stores the data in an incremental way, i.e., only the difference from the previous processing is stored, saving processing resources. The data is further filtered, cleaned, and homogenized as described in the *3E Steps* technique. Finally, the module maps tagsets to ontology concepts and fuse them to produce a FO.

Review/Enhancement Server The interactive module – responsible for visually presenting the data to the user – is designed to run over the browser. It is organized in two sub-modules, a server module implemented in python and a client module implemented in HTML + JavaScript. The server sub-module is responsible for the following operations: (i) it builds the HTML + JavaScript module from a template and dispatches to the client (web browser); (ii) it interacts with the database retrieving and filtering relevant information for the client.

Visual Review/Enhancement Client This module uses a third-party library, JavaScript InfoVis Toolkit [9] to provide the interactive visualizations. It is responsible for visually presenting data to the end-user. This tool is detailed in the next sub-section.

4.4.2 Visual Review/Enhancement

Figure 4.6 shows a screenshot of the main screen of the Visual Review/Enhancement Client, illustrated in Figure 4.5. It is organized in two main areas: control panel – displayed in the bottom side – and an interactive graph area – displayed in the top side. The control panel can switch among three possible sub-panels: navigation, details and history.

In Figure 4.6, the *Details Panel* is selected. In the interactive graph area, a segment of an FO is displayed, centered in the *physical_entity* node. The tool generates an interactive graphical representation of the segment. In this representation, the user can drag the nodes, zoom, and pan the visualization. When a node is clicked, the information associated with it is showed in the *Details Panel*. The data can be explored in two modalities: relations and concepts. They are detailed in the next subsections.

4.4.3 Analyzing Relations

The goal of this modality is to analyze the relations among concepts in the ontology and confront them with relations captured from the folksonomy. Figure 4.7 shows a typical graph presented for analysis. It derives from our abstract model presented in

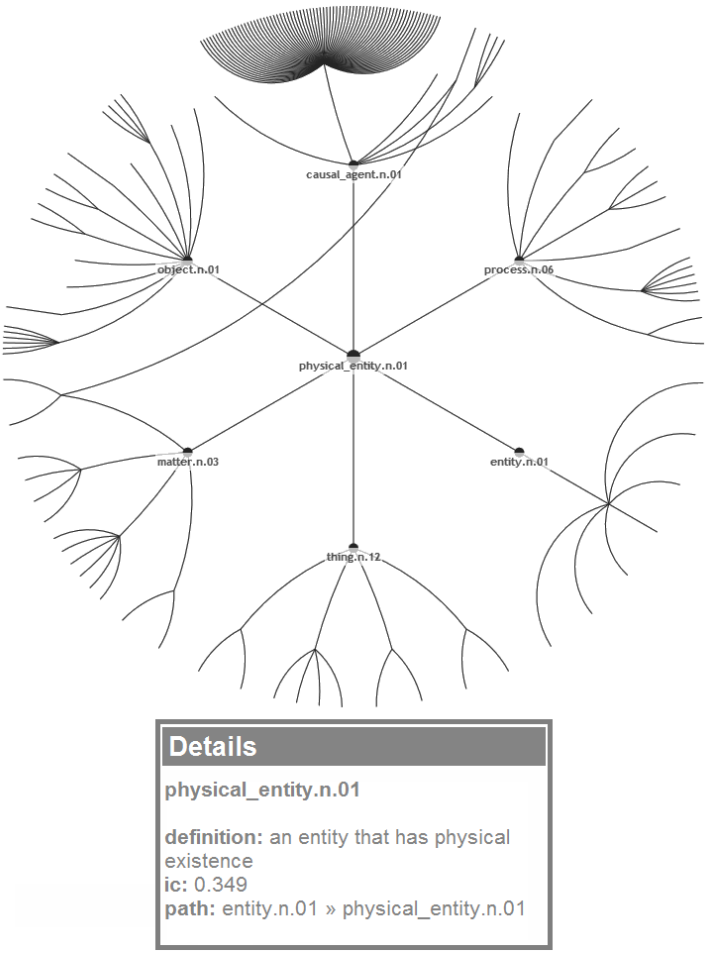

Figure 4.6: FO Visualization

Subsection 4.3.1. Its elements are described in Table 4.2. There are three parameters that control the details of the visualization: more nodes, virtual nodes, and edge.

As mentioned before, the Details panel shows the correspondent information of the selected node. It shows co-occurrence and *ic* values – representing the functions of the formal model $\mathcal{F}_1$ and $\mathcal{F}_3$, respectively.

There are two approaches for navigation: overview or compare. In the overview approach, the user can freely navigate in the entire FO tree, by using the interactive focus provided by the hyperbolic tree. In the compare approach, the user will analyze a given pair of concepts, the path of relations that connects both and the relations with near concepts. Therefore, the first step for an analysis in this approach is to select that pair of concepts.

When the user selects the compare approach to navigate, it is possible to manually assign two concepts of the FO or enter in the assisted mode, in which the tool finds

Table 4.2: Description of the elements of a graph to visualize and to analyze relations.

Element		**Description**
Graph		Visually presents a clip of the graph G of the formal model.
Vertex		All vertexes are members of the V set of the formal model.
	Rounded black	Concept of the originating ontology fused to a correspondent tagset coming from the originating folksonomy. It contains a value of information content for the vertex, defined in the formal model by the function $\mathcal{F}_3$.
	Square gray	*Virtual node* – concept of the originating ontology which has no correspondent tagset in the originating folksonomy. They are necessary in the FO to maintain its topology. For example, two concepts A and B coming from the originating ontology have correspondent tagsets in the originating folksonomy. The path between A and B in the originating ontology includes a third concept C in the middle. This concept has no correspondent tagset in the folksonomy. Therefore, C is included in the FO to keep the topology, but it is considered a virtual node.
	Rectangular	Focus of a given analysis – an analysis can focus on the comparison of two concepts, which will be presented as rectangular vertexes.
	Rounded big	Considering that the FO shows subsumption relations, when two concepts are the focus of the analysis (rectangular), for each concept, the tool will show the path to the most specialized ancestor shared by both. This ancestor is highlighted as a big rounded node. The analysis of a common ancestor is fundamental to define the similarity between two concepts, as described in Section 4.2.
Edge		All regular edges are members of the set E of the formal model.
	Thin	Regular edge – relation between two concepts of the originating ontology fused to a correspondent relation between two tagsets coming from the originating folksonomy. Each edge has a weight (of the relation) and is typed, defined in the formal model by the functions $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively. As mentioned before, the current implementation of the tool handles only subsumption relations coming from the ontology, but it was designed to afford other relations in future versions.
	Thick	Strong relation between two tagsets of the originating folksonomy which has no correspondent relation in the originating ontology. Strong relations in the folksonomy are those having high co-occurrence value. The thickness of the edge is proportional to the amount of co-occurrences. This type of edge can indicate important relations discovered in the originating folksonomy, which can trigger a review and/or an enhancement of the originating ontology.

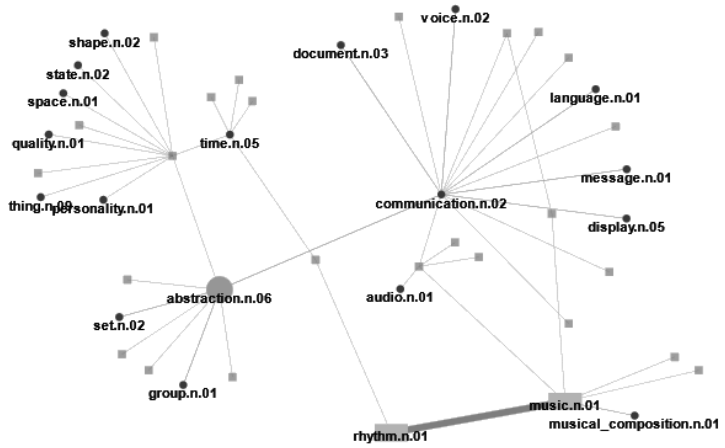

Figure 4.7: Graph to visualize and to analyze relations.

two candidates to review. The basic principle of the assisted mode is to look for relevant discrepancies among data coming from the ontology and from the folksonomy. The current version can find two concepts in the FO that have a weak similarity in the originating ontology, but have a strong connection in the originating folksonomy.

In the assisted mode, the tool runs the following process to automatically two candidates for analysis:

- A set of candidate concepts is selected to be tested. The size of this set can be configured.

- The tool creates an analysis tree with a branch of the original FO for each selected candidate. This branch includes all concepts (and the respective relations) that can be reached departing from the candidate concept, in a given customizable depth.

- The next step involves finding two nodes with low similarity – considering the path coming from the FO – and high co-occurrence – considering the data extracted from the folksonomy.

- For each concept in a given branch, the tool tests if its co-occurrence with the candidate concept is higher than a threshold. If so, the distance of both is tested.

- If the two conditions – higher co-occurrence and long distance (low similarity) – are satisfied, the pair of concepts is selected to be analyzed in the visual tool.

4.4.4 Practical Examples of Relations Analysis

In this subsection, we will show practical examples of the tool, illustrating its support for ontology enhancement and the improvements achieved by the use of FOs.

Figure 4.8 shows a visualization generated by the tool. The pair of concepts to be analyzed in the visualization is *(bible, christian)*, and the common ancestor – *entity* – is highlighted as well. This example shows a scenario in which the tool can be used to improve the ontology. The concepts *bible* and *christian* are separated by a long path and their common ancestor, *entity*, is the most general concept – the root – in the ontology. Any ontology-based approach to compare the terms – see Section 4.2.1 – will return a low similarity, due to the long path and the generic common ancestor, which has zero of information content. This was also observed in our practical experiments confronting our approach to related work.

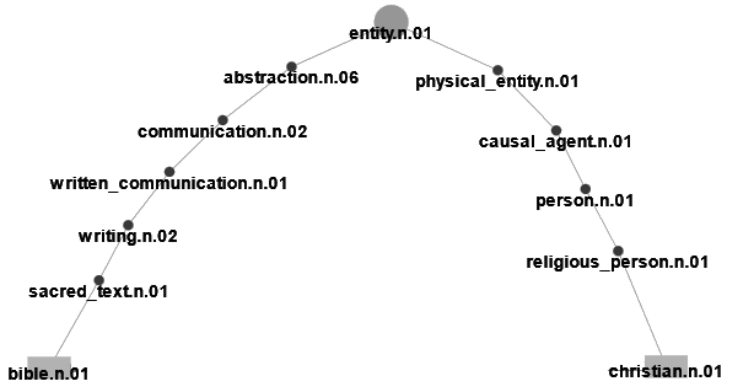

Figure 4.8: Default Visualization

When the parameter *edge* is activated, the tool draws a strong edge between bible and christian, as they have a strong co-occurrence in the originating folksonomy – see Figure 4.9. This edge does not mean that both nodes should have a direct link, but just emphasizes that something must be reviewed or improved in the ontology, considering the observations of the folksonomy.

In this scenario, the ontology managers – facing the task of enhancing the ontology – can use the tool to find and visualize how and where the ontology could be improved. If they need more nodes in the visualization, they can use the respective parameter to increase the number of nodes.

Figure 4.10 shows the resulting visualization when *More Nodes* and *Virtual Nodes* parameters are selected. This parameter highlights the real nodes making the virtual nodes (see Table 4.2) less visible. The distinction between regular and virtual nodes makes explicit the contrast between concepts shared by the ontology and the folksonomy,

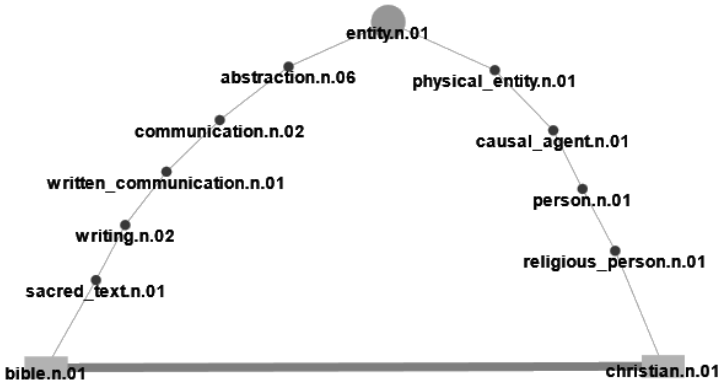

Figure 4.9: Visualization with more nodes, virtual nodes, and edge

and concepts of the ontology not present in the folksonomy. This is an useful synthesis to analyze the popular use of concepts present in the ontology.

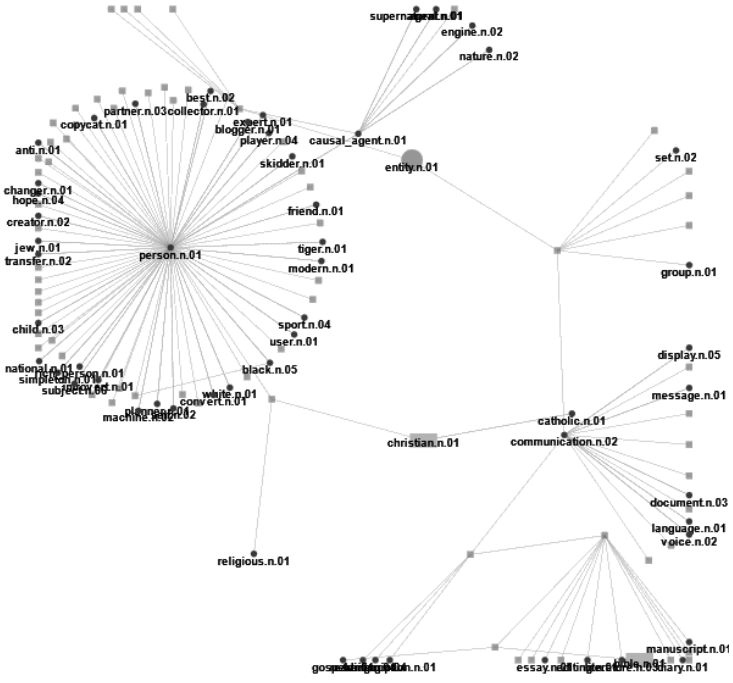

Figure 4.10: Visualization with more nodes and virtual nodes

Besides the review/enhancement process, the visualization tool can be used to inspect the improvements of the FOs, when confronted with traditional ontologies. In Figure 4.11, the tool is focusing on the pair of nodes *(graphics, inspiration)*. This pair has a high relation in the folksonomy, but they are separated by a great distance in the ontology. Our practical experiments showed that when statistical data – information content and

co-occurrences – embedded in the FOs are explored in the similarity algorithm, they achieve better results than ontologies alone. This case shows that the FO is an entity that can support improvements in the operations over its data, because it can use more than one source of semantics – folksonomies and ontologies.

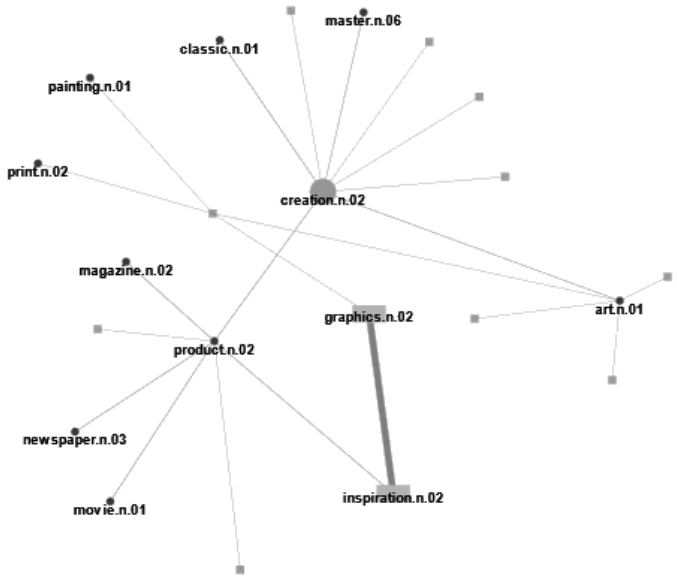

Figure 4.11: Visualization: Improvement of FOs

The following example shows another improvement of the FO in the opposite direction. In Figure 4.12, the tool is focusing on the pair of nodes *(war, conflict)*. *Conflict* is a concept coming from the originating ontology, which has no corresponding tagset in the folksonomy. *Conflict* is a virtual node, inserted to keep the topology of the ontology. *Conflict* is a virtual node, and in the traditional folksonomies it would not be considered. Since the FO considers virtual nodes in the comparison algorithm, even if they do not appear in the folksonomy, it will achieve better results in this kind of scenarios when compared with traditional folksonomies.

4.4.5 Inside Concepts

As mentioned in Subsection 4.4.2, the second approach to review/enhance ontologies allows the user to inspect inside a tagset and its relation with other tags in the folksonomy.

As described in Section 4.3.1, in order to relate tags to concepts of an ontology, the tag is not evaluated alone. There is a network of relations among a tag and other tags, which have high co-occurrence with it. This network is essential to provide a context to the tag. It is the basis to relate a tagset to a given concept.

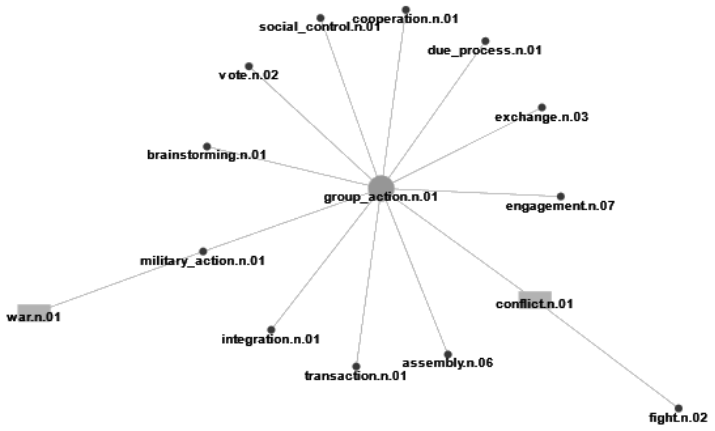

Figure 4.12: Visualization: Improvement of virtual nodes

A graphical presentation of a tag and its co-occurrences is therefore a rich source of information. Figure 4.13 shows that presentation. Table 4.3 describes the elements of the figure.

The intensity evaluation of relations among tags considers also transitive relations among tags.

4.5 Related Work

Our work has some common aspects with the related work presented here. We process sets of tags to create clusters of high-related tags and we use knowledge bases to map those clusters to ontologies as well. The main limitation observed in related work concerns their unidirectionality. Proposals to automatically build ontologies from scratch, based on data coming from texts or folksonomies produce rather simple structures and do not explore the richness of preexisting engineered ontologies. Proposals which explore the interplay between folksonomies and ontologies are limited to map tags or sets of tags to ontologies. Nevertheless, they do not enrich the ontology itself with the statistical data extracted from folksonomies.

Beyond this unidirectional enrichment, we proposed a fusion approach. The result is the Folksonomized Ontology, which combines semantics from both contexts. These characteristics are also the differential of our *3E Steps* technique, which explores the information derived of this function to support ontology evolvement.

Many approaches to automatically or semi-automatically develop ontologies were proposed. Some of them aim at discovering relations and building ontologies from a given corpus of texts [25, 24]. Alternative approaches adopt folksonomic data [42, 35], instead

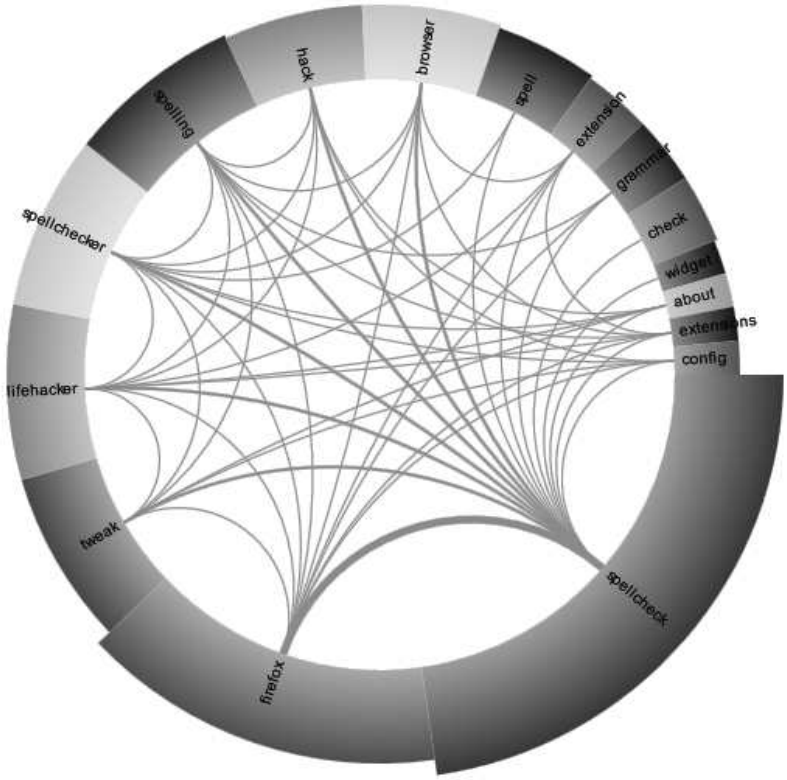

Figure 4.13: Diagram to diagram to inspect a tagset

of texts. In either case, the ontologies are built from scratch. These approaches contrast of our proposal, which does not build new ontologies, but departs from existing ones and takes advantage of their structure to build a new entity, which is an enriched (folksonomized) ontology.

Damme et al. [42] employ folksonomic data and lexical/sematic resources, like Leo Dictionary, Wordnet, Google and Wikipedia, to build and to maintain ontologies. They aggregate sets of tags, mapping them to ontology concepts. The relations of those ontologies are mapped back to the folksonomy, in order to produce a social ontology. One important aspect of their proposal is the mechanism to validate the ontology, in which the community that produced the folksonomy validates the results by accepting or discarding the proposed concepts.

They employ stemming algorithms to clean the folksonomic data, specially plural nouns and conjugated verbs. This operation occurs in the pre-process phase of the extraction step – as in *3E Steps* – aiming to improve the quality of the data, grouping the tags that have strong relations.

Specia et al. [35] proposed a technique to map clusters of tags to ontology concepts,

Table 4.3: Description of the elements of the diagram to inspect a tagset.

Element	**Description**
Label	Tag of a folksonomy. The figure always shows more than ten tags. One of the tags is the focus of the analysis, nine tags are those which have more co-occurrences with it, and the remaining tags are the ones obtained transitively from the nine others.
Highlighted label	The highlighted label (the largest thickness in the graphic – spellcheck in the example) is the tag which is the focus on the analysis. In the example, is the spellcheck tag. It is also the key tag of a given tagset (see Section 4.3.1).
Line connecting labels	Co-occurrence relation between two labels.
Label thickness	The thickness is the space occupied by the label in the circle. It defines the intensity of the relation of a given tag with the focus of the analysis, i.e., the number o co-occurrence with these two tags.

making explicit the semantics of the tag space. They depart from a set of tags, creating clusters of high-related tags, using co-occurrence information. The relations between these clusters are aligned with external resources like Wikipedia, Google and ontology bases to produce an ontology. Those resources were used to improve the folksonomic data, mainly making explicit the semantics of the tags.

In the step of pre-processing, they group morphologically similar tags using Levenshtein similarity metric [21]. As the authors point out, this technique could find minor variations (*cat* and cats, and *san_francisco*, *sanfrancisco* and *san.francisco* are the examples given by them). But it is important to mention that the use of this technique could led to undesired results. For example, the distance between the pair of words *range* and *orange* is the same as the distance between the pair *orange* and *oranges*. While the latter have much semantic relatedness, the former does not. For this reason, we used stemming algorithms to group tags.

Angeletou [7] proposed a tool, called FLOR, to perform semantic enrichment on folksonomic data. The first enrichment step is connecting tags to semantic entities and then connecting those entities directly to resources managed by the system. Connecting tags to semantic entities is divided into three sub-steps: (i) lexical processing, in which the decision of which tags are meaningful is made; and the normalization step, selecting lexical representations for each tag; (ii) semantic expansion, in which the tags are processed in order to disambiguate their meanings, using WordNet; (iii) semantic enrichment, where the tags are finally mapped to ontology concepts – ontologies that were found by querying web repositories.

Our approach to map ontology concepts and tags are very close to this tool. For example, both approaches organize the tags into tagsets, i.e., sets of tags with strong relations. Both approaches use WordNet to extract the latent semantics of the folksonomic data. But, differently of our approach, the FLOR tool aim to annotate resources annotated by tags with semantic entities. We focused on mapping folksonomic data on concepts of a single ontology, enriching it in the process.

Cattuto et al. [11] calculated several measures of tag relatedness, based in their co-occurences, mapping them to WordNet synsets (sets of synonyms). They do not group related tags; each individual tag of the folksonomy is associated with a concept in the WordNet ontology. Synsets are sets of synonyms that play an equivalent role of concepts in ontologies. The similarity of the related synsets are then transferred to the respective tags. The step Enrichment of our approach have similar objectives, but are not based only in the co-ocurrences of the tags, but in the topology of the target ontology and the relations between its concepts as well.

Cantador et al. [10] proposed a mechanism to filter and classify tags, producing a graph of clusters. Then, they mapped these clusters to knowledge bases, like WordNet and Wikipedia, aiming to discover the corresponding semantic entities. Different from previous approaches, in order to map clusters to ontologies, they predefine a set of possible categories and relation types among tag sets, using direct association or natural language processing heuristics.

They build categories and them classify tags into them. Our approach, on the other hand, does not classify tags in predefined categories, but map them to ontologies concepts. In this way, we can build more malleable entities, apt to easy expansion, in order to accommodate new tags and relations.

Like many of the related work, Tesconi et al. [40] used external resources, namely Wikipedia, and ontologies like WordNet and YAGO [37]. Their objective was disambiguate tags, "semantifying" them. They developed an algorithm to disambiguate tags, grouping them by sense. Its tagsets are finally linked to Wikipedia categories and ontology concepts, producing social ontologies.

Therefore, they cannot take into account tags that does not have a direct map to an external resource – e.g., Wikipedia concept. As folksonomies allow users to create tags as they wish, it is likely that new terms would emerge. In our approach, tags without direct map to ontology concepts would still be considered because they could be grouped to ones that are mapped.

Bang et al. [8] proposed the concept of "structurable tags", in which tags can be linked through relations, allowing basic inference operations. They expanded the folksonomic model, allowing users to create two types of relations between tags: inclusion and synonymy. These types of relations support the transformation of folksonomic data into

semantically richer models. Due to the synonymy relation, the system can group tags with the same meaning. On the other hand, the inclusion relation led to an hierarchical organization, as a simplified ontology.

In the *3E Steps*, the users are not forced to change their natural use of folksonomies. The extra effort of creating relations between the tags is responsibility of the system, in an automated way.

Heymann et al. [16] proposed an algorithm to build a graph departing from folksonomic data. It first aggregates tags in *tag vectors*, in which the $v_{t_l}[o_m]$ corresponds to the number of times that the tag t_l annotates the object o_m. In the resulting unweighted graph, the vertexes will be the tags, and there will be an edge for each pair of tags whose relatedness is above a threshold. The resulting graph, without weights and maintaining just the relevant edges, contains a "latent hierarchical taxonomy". It is captured by an algorithm that builds a subsumption hierarchy, derived from the centrality of each node in the graph.

As ontologies become bigger and more complex, the process of evolving them requires an increasing effort. As pointed out by Ding et al. [12], almost all techniques to evolve ontologies require manual intervention. According to Stumme et al. [36], even though the ontology evolving process requires human experts, in order to address the increasing complexity of modern and large ontologies, tools to support the evolving process are necessary. This observation motivated our work to propose a technique and a related tool to support ontology evolvement.

4.6 Conclusion

This chapter presented our *3E Steps* technique to review and enhance ontologies and our approach to build and use a folksonomized ontology (FO) in this context. A FO is a hybrid entity fusing folksonomies and ontologies. It is a symbiotic combination, taking advantage of both semantic organizations. Ontologies provide a formal semantic basis, which is contextualized by folksonomic data, improving operations over tags based on ontologies. Conversely, the FOs were used as tools to analyze the ontology and to support the process of ontology evolvement, showing the discrepancies between the emergent knowledge of a community and the formal representation of this knowledge in the ontology.

In this chapter, we described the *3E Steps* : Extraction, Enrichment, and Evolution. Extraction is the step where the semantic information is collected from the folksonomies and processed. In the Enrichment step, we combine the two entities, building a third one, with the best of both worlds. Finally, Evolution is the step where the folksonomized ontology is used to support the review and enhancement in the original ontology, closing the circle. A review and enhancement tool was presented in this chapter.

In our point of view, the work presented here opens an interesting field of applying la-

tent semantics, socially produced by wide communities, to improve engineered ontologies. Related work addressing ontologies and folksonomies does not explore the full potential of this interaction, due to their unidirectionality. Our fusion approach explores the symbiotic complementarity of ontologies and folksonomies.

Future work include: (i) to expand the folksonomized model to include other relations (besides the generalization); (ii) to run tests in specialized contexts applying domain ontologies; (iii) to expand our tool that allows the visualization of the individual tags inside a cluster, improving the observation of the interrelation between the tags.

We consider the technique and the respective tool presented in this chapter a first step that opens a perspective to review and enhance ontologies. We oversee many other relevant information that we can obtain from folksonomies, which can be computed, fused to ontologies and displayed in our tool. Nevertheless, there is still a wide range of possibilities to visually explore the available data.

Chapter 5

Conclusão

Os sistemas de tagging obtiveram um grande sucesso em aplicações sociais na web devido à sua facilidade de uso e flexibilidade. Por outro lado, operações de organização, indexação e busca encontram limitações devido à carência de semântica explícita no corpo de tags das folksonomias. Diversas iniciativas buscaram usar o conhecimento latente das folksonomias para construir ontologias sociais, eventualmente utilizando ontologias preexistentes para dar suporte ao processo.

Diferente dos trabalhos relacionados, propomos aqui uma abordagem de fusão, em que folksonomias e ontologias são combinadas dando origem a uma terceira entidade que chamamos de ontologias folksonomizadas.

5.1 Contribuições

O desafio deste trabalho foi demonstrar que é possível a fusão de ontologias e folksonomias, a fim de se explorar de uma forma mais completa seus papéis complementares em relação aos trabalhos relacionados. Para atingi-lo, desenvolvemos modelos, algoritmos, técnicas e protótipos, cada um perfazendo uma contribuição diferente. São elas:

Abordagem de fusão de ontologias e folksonomias: Esta é a contribuição central deste trabalho e envolveu um modelo de ontologia folksonomizada e uma técnica para integração de ontologias e folksonomias. Dentro desta contribuição podemos destacar: o processo para coleta de tags; o algoritmo de mapeamento de tags e conceitos de ontologias, bem como as estatísticas sobre o corpo de tags; a integração de métricas nas ontologias folksonomizadas, tais como os valores de conteúdo de informação, expandindo o trabalho de Resnik [31] para o contexto de folksonomias.

Ferramenta para coleta de dados de folksonomias: O protótipo desenvolvido permite a coleta de dados de folksonomias e o seu armazenamento em um banco de

dados. Os módulos para acesso ao flickr e ao delicious foram desenvolvidos em uma abordagem extensível, e é possível desenvolver módulos para outras folksonomias e acoplá-los ao protótipo.

Definição de um framework formal para ontologias sociais: Desenvolvemos neste trabalho um framework formal abrangendo as ontologias folksonomizadas, além das abordagens a elas relacionadas. Tal framework subsidiou a análise comparativa entre as diferentes abordagens dos trabalhos relacionados.

Metodologia e ferramenta de suporte à evolução de ontologias: As ontologias folksonomizadas apresentam a possibilidade de utilizá-las para oferecer suporte para evolução de ontologias. Assim desenvolvemos uma técnica e um respectivo protótipo de ferramenta para suporte a revisão e evolução de ontologias [4].

5.2 Extensões

Existem extensões deste trabalho em dois âmbitos: teórico e de implementação. Nesta seção descrevemos as principais extenções:

Expandir o modelo da ontologia folksonomizada: Atualmente, o modelo da ontologia folksonomizada prevê a representação de qualquer tipo de relacionamento. No protótipo atualmente implementado, porém, foi considerado apenas o relacionamento de generalização / especialização como abordagem inicial. Uma possível extensão do trabalho é a inserção de diferentes tipos de relacionamento e a análise do impacto delas na aplicação das ontologias folksonomizadas.

Expandir a ferramenta de coleta de dados: A ferramenta de coleta e armazenagem de dados implementada funciona em módulos e foi projetada para a expansão pela adição de novos módulos, de modo que possa ser aplicada em outras folksonomias. A ferramenta atual possui módulos desenvolvidos para acesso ao flickr e delicious [6].

Aplicação em ontologias e folksonomias de domínio: Os testes desenvolvidos foram feitos levando em consideração consultas genéricas, com o objetivo de testar as vantagens da nossa proposta em ambientes gerais de uso. Uma possível expansão do trabalho é a aplicação das técnicas e abordagens desenvolvidas em folksonomias de domínio, combinando com o uso de ontologias destes domínios específicos.

Avaliar os custos e impactos da aplicação: Isto envolve medir as variações em tempo de consulta, volume de dados armazenados e complexidade de manutenção, com o objetivo de verificar a viabilidade de uso das ontologias folksonomizadas em ambientes reais.

Apêndice A

Retrieving and Storing Data from Folksonomies

A.1 Introduction

The popularization of web-based systems offering services for content storage, indexing and sharing fostered a rapid growth of content available on-line. There are more than 5 billion images hosted on Flickr[1] and more than 180 million URL addresses on Delicious[2]. These systems increasingly rely on tag-based metadata to organize and index all the amount of data. The tags are provided by users connected in social networks, who are free to use any word as tag; there is no central control. The term folksonomy – combining the words "folk" and "taxonomy" [43] – has been used to characterize the product which emerges from this tagging in a social environment.

In order to analyze, index and classify their content, web systems compare tags attached to resources. Instead of considering the semantics of each tag in the comparison, tag-based systems usually rely on string matching approaches. While ontologies are increasingly adopted to enrich tags semantics, one common problem with the proposals to associate tags to formal ontologies concerns their unidirectionality, i.e., ontologies improve tags semantics, or the implicit/potential semantics of folksonomies is extracted to produce ontologies.

Differently from traditional techniques, we proposed a fusion approach, called *folksonomized ontology* (FO), which goes beyond this unidirectional perspective [3]. In one direction, the ontologies are "folksonomized", i.e., the latent semantics from the folksonomic tissue is extracted and fused to ontologies. On the other direction, the knowledge systematically organized and formalized in ontologies gives structure to the folksonomic

[1] `http://blog.flickr.net/en/2010/09/19/5000000000/` - retrieved on November, 2011

[2] `http://blog.delicious.com/blog/2008/11/delicious-is-5.html` - retrieved on November, 2011

semantics, enhancing operations involving tags, e.g., content indexation and discovery. The folksonomic data fused to an ontology will tune it up to contextualize inferences over the repository.

In our fusion approach both ontologies and folksonomies are enriched in the process. This symbiosis is explored to:

Tag disambiguation: by finding groups of related tags and mapping them to ontology concepts, the FO can be applied to disambiguate tags and find the ones that are more related, going beyond statistical analyses by using semantic similarity metrics.

Tag suggestion: the current folksonomy systems consider only co-occurrence information to suggest related tags to users; a FO has a richer set of semantic relations among concepts, supporting suggestion of tags that were not used together before – folksonomies cannot do that.

Semantic similarity: a FO can support the computation of semantic similarity between concepts and, by extension, between tags; so, they can expand the usual techniques that focus only at syntactical similarity and co-occurrence of tags, achieving better results in discovery operations.

Ontology evolvement: a FO can be used to find missing relations in ontologies; the high co-occurrence between two groups of tags, and their corresponding concepts, can indicate a necessary relation in the ontology, if it does not exist yet.

In order to build a practical tool to validate our proposal, we have implemented a software module to access and collect data from folksonomy-based web systems. We confronted the model adopted by each system with models proposed in the literature, in order to propose a generic model to represent and store the collected data.

The goal of this appendix is to detail this work. In Section A.2 we synthesize related work concerning formal models to represent folksonomies. In Section A.3 we discuss implementation aspects of our module, which interacts with these systems to collect and store folksonomic data. In Section A.4 we analyse the approach adopted by folksonomy-based systems to represent and store their folksonomies, including their Application Programming Interfaces (APIs). The systems that will be analysed here are Delicious [1] and Flickr [2].

A.2 Formal Model for Folksonomies

In folksonomy-based systems, users can attach a set of tags to resources. These tags are not tied to any central vocabulary, so the users are free to create and combine tags.

Some strengths of folksonomies are their easiness of use and the fact that they reflect the vocabulary of their users [26]. In a first glimpse, tagging can transmit the wrong idea of a poor classification system. However, thanks to its simplicity, users are producing millions of correlated tags. It is a shift from classical approaches – in which a restricted group of people formalize a set of concepts and relations – into a social approach – in which the concepts and their relations emerge from the collective tagging [34]. In order to perform a systematic folksonomy analysis, to subsidize the extraction of its potential semantics, researchers are proposing models to represent its key aspects. Gruber [15] models a folksonomy departing from its basic "tagging" element, defined as the following relation:

$$Tagging(object, tag, tagger, source) \tag{A.1}$$

In which *object* is the described resource, *tag* is the tag itself – a string containing a word or combined words –, *tagger* is the tag's author, and *source* is the folksonomy system, which allows to record the tag provenience (e.g., Delicious, Flickr etc.).

In order to formalize a folksonomy Mika [27] departs from a tripartite graph with hyperedges. There are three disjoint sets representing the vertices:

$$T = \{t_1, \ldots, t_k\}, U = \{u_1, \ldots, u_l\}, R = \{r_1, \ldots, r_m\} \tag{A.2}$$

In which the sets T, U and R correspond to tags, users and resources sets respectively.

A folksonomy system is a set of annotations A relating these three sets:

$$A \subseteq T \times U \times R \tag{A.3}$$

The folksonomy itself is a tripartite hypergraph:

$$H(T) = \langle V, E \rangle \tag{A.4}$$

In which $V = T \cup U \cup R$, and $E = \{\{t, u, r\} \mid (t, u, r) \in A\}$

The folksonomy analysis can be simplified and directed by reducing this tripartite hypergraph into three bipartite graphs: TU relating tags to users, UR relating users to resources and TR relating tags to resources [27]. A graph TT is a relevant extension of this model for representing relations between tags. It allows to represent the co-occurrence of tags. The same approach can be applied to the user and resource sets.

A.3 Implementation

In this section we describe the tool we have implemented to retrieve data from Delicious and Flickr, as well as the database model. The implementation adopted the python

language[3]. The data was stored by using the SQLite[4] database manager.

The access of Flickr data required the implementation of the code to handle its protocol and to treat the results of the requests. The module to retrieve the data from Delicious, on the other hand, adopted a third-party library: DeliciousAPI[5].

During the development, we faced an unexpected behavior of the library. The reason was a change in the structure of Delicious' pages. This is still a challenge to be faced in web services research, mainly in public web services: whenever servers change their interfaces, the clients will break if there is not backwards compatibility. In order to fix it, we developed a patch to adjust the access, contributing to the community to fix the error[6].

A.3.1 Database Model

As mentioned before, our database model results from a comparative analysis of related work and models adopted by folksonomy based systems. The logical modeling is depicted in the Figure A.1. There are three main entities – `User`, `Resource` and `Tag` – following the model presented in Section A.2. The `Resource` entity was specialized to better characterize its representation in the systems Flickr (`Photo`) and Delicious (`URL`). As the database was designed to simultaneously afford tags of many systems, the `Source` entity keeps track of the origin of the tag. This is an important information, since our algorithms were designed to work with a single folksonomy system each time.

The physical modeling is based on the logical one. However, it includes control tables and flags to indicate: users already processed, resources already visited, and so on. Some auxiliary tables – prefixed by `count_` – record the counts of occurrences or co-occurrences of analyzed items; they will support statistics produced by our system. It is composed of 13 tables, as we further detail:

control: records control data related to the execution of the process, like the timestamps of the last requisition to the systems. `(name TEXT, value TEXT)`

count_resource: records the count of each resource. `(rid INTEGER, sid INTEGER, count INTEGER)`

count_rt: records the count of each pair (resource, tag). `(rid INTEGER, tid INTEGER, sid INTEGER, count INTEGER)`

count_tag: records the count of each tag. `(tid INTEGER, sid INTEGER, count INTEGER)`

[3] `http://www.python.org/`
[4] `http://www.sqlite.org/`
[5] `https://github.com/quuxlabs/DeliciousAPI`
[6] `https://github.com/quuxlabs/DeliciousAPI/commit/1cea76941797d6807ac8411b0e8437aa92a35aa5`

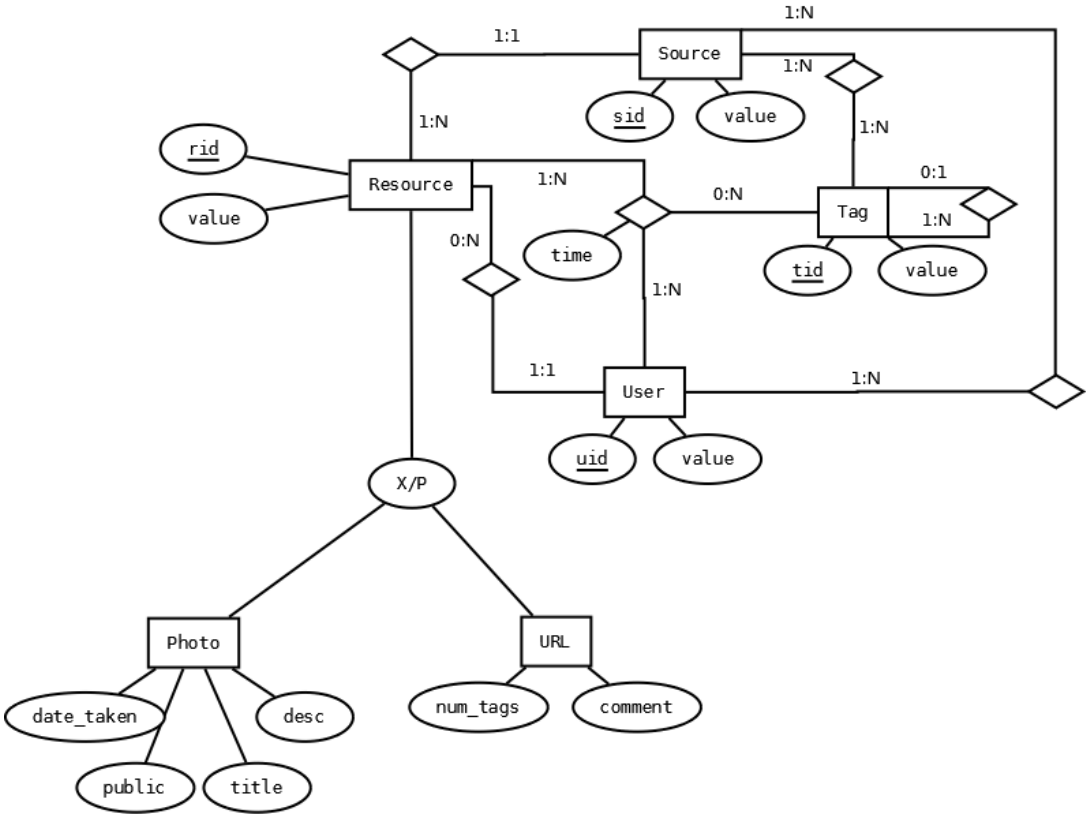

Figure A.1: Logical modeling of the database

count_tt: records the count of each pair of tags. `(t1 INTEGER, t2 INTEGER, sid INTEGER, count INTEGER)`

count_tu: records the count of each pair (tag, user). `(tid INTEGER, uid INTEGER, sid INTEGER, count INTEGER)`

count_ur: records the count of each pair (user, resource). `(uid INTEGER, rid INTEGER, sid INTEGER, count INTEGER)`

count_user: records the count of each user. `(uid INTEGER, sid INTEGER, count INTEGER)`

resource: records each resource, which has an internal id `rid` and a reference to the specific identification in the source system – e.g., a URL for Delicious or a Flickr internal identifier – in the `value` field. `(rid INTEGER PRIMARY KEY, sid INTEGER, value TEXT, done NUMERIC)`

source: records each source, which has an internal id `sid` and the specification of the source (e.g., delicious or flickr) in the `value` field. `(sid INTEGER, value TEXT)`

tag: records each tag, assigning an internal id `tid` to each one. `(tid INTEGER, sid INTEGER, value TEXT)`

tagging: records each tagging triple (resource, tag, user) in a given source. `(sid INTEGER, uid INTEGER, rid INTEGER, tid INTEGER, time TEXT)`

user: records each user, assigning for each value (string of the username) an internal id. `(uid INTEGER, sid INTEGER, value TEXT, done NUMERIC)`

The complete schema of the database is as follows:

```
1  CREATE TABLE source (sid INTEGER PRIMARY KEY, value TEXT);
2  CREATE TABLE control (name TEXT, value TEXT);
3  CREATE TABLE user (uid INTEGER PRIMARY KEY, sid INTEGER, value TEXT COLLATE NOCASE,
       done NUMERIC, FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(sid, value));
4  CREATE TABLE tag (tid INTEGER PRIMARY KEY, sid INTEGER, value TEXT COLLATE NOCASE,
       FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(sid, value));
5  CREATE TABLE resource (rid INTEGER PRIMARY KEY, sid INTEGER, value TEXT COLLATE
       NOCASE, done NUMERIC, FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(sid,
       value));
6  CREATE TABLE tagging (sid INTEGER, uid INTEGER, rid INTEGER, tid INTEGER, time TEXT,
       FOREIGN KEY(sid) REFERENCES source(sid), FOREIGN KEY(uid) REFERENCES user(uid),
       FOREIGN KEY(rid) REFERENCES resource(rid), FOREIGN KEY(tid) REFERENCES tag(tid),
       UNIQUE(sid, uid, rid, tid));
7  CREATE TABLE 'count_tu' (tid INTEGER, uid INTEGER, sid INTEGER, count INTEGER,
       FOREIGN KEY(tid) REFERENCES tag(tid), FOREIGN KEY(uid) REFERENCES user(uid),
       FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(tid, uid, sid));
8  CREATE TABLE 'count_ur' (uid INTEGER, rid INTEGER, sid INTEGER, count INTEGER,
       FOREIGN KEY(uid) REFERENCES user(uid), FOREIGN KEY(rid) REFERENCES resource(rid),
       FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(uid, rid, sid));
9  CREATE TABLE 'count_rt' (rid INTEGER, tid INTEGER, sid INTEGER, count INTEGER,
       FOREIGN KEY(rid) REFERENCES resource(rid), FOREIGN KEY(tid) REFERENCES tag(tid),
       FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(rid, tid, sid));
10 CREATE TABLE 'count_tt' (t1 INTEGER, t2 INTEGER, sid INTEGER, count INTEGER, FOREIGN
       KEY(t1) REFERENCES tag(tid), FOREIGN KEY(t2) REFERENCES tag(tid), FOREIGN
       KEY(sid) REFERENCES source(sid), UNIQUE(t1, t2, sid));
11 CREATE TABLE 'count_tag' (tid INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(tid)
       REFERENCES tag(tid), FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(tid, sid));
12 CREATE TABLE 'count_user' (uid INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(uid)
       REFERENCES user(uid), FOREIGN KEY(sid) REFERENCES source(sid), UNIQUE(uid, sid));
13 CREATE TABLE 'count_resource' (rid INTEGER, sid INTEGER, count INTEGER, FOREIGN
       KEY(rid) REFERENCES resource(rid), FOREIGN KEY(sid) REFERENCES source(sid),
       UNIQUE(rid, sid));
```

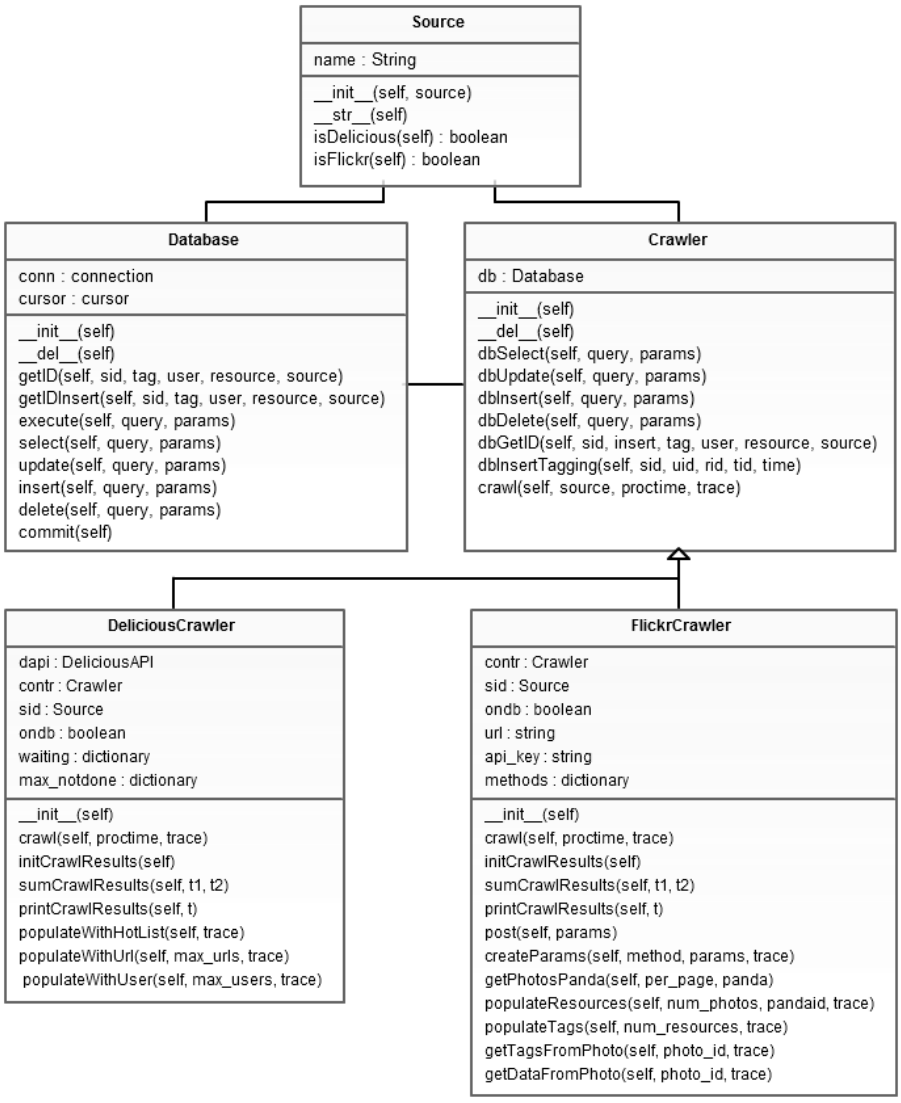

Figure A.2: Class diagram

A.3.2 Tool Model

Figure A.2 presents a UML diagram of the main classes of our tool. The five classes depicted in the figure are:

Source: represents a specific folksonomy system.

Database: abstracts and centralizes all database operations.

Crawler: abstract class whose instances represent a crawler a its operations.

DeliciousCrawler: implements the crawler operations for the Delicious context.

FlickrCrawler: implements the crawler operations for the Flickr context.

The `Crawler` abstract class makes simpler to extend the system, since it standardize the API to the crawler mechanism. Each new system will require only a new implementation extending `Crawler`.

A.3.3 Source Code

In this section we present the source of the tool, according the model presented in the previous section. Each block contains a class and comments explaining its functionality.

```
1  import sys                                # print without \n
2  import urllib                             # http request and manipulation
3  import sqlite3                            # database
4  import time, datetime                     # time processing
5  from xml.etree.ElementTree import parse # parser xml
6  import deliciousapi                       # delicious data access
7  import random
8
9  """ Object Source - encapsulates the current folksonomy system
10 """
11 class Source(object):
12     def __init__(self, source = 'all'):
13         if source in ['all', 'flickr', 'delicious']:
14             self.name = source
15         else:
16             self.name = 'all'
17
18     def __str__(self):
19         return self.name
20
21     """ true if the source is delicious
22     """
23     def isDelicious(self):
24         return self.name == 'delicious'
25
26     """ true if the source is flickr
27     """
28     def isFlickr(self):
29         return self.name == 'flickr'
```

```
30 """ Object Database - encapsulates the database operations
31 """
```

```
32 class DataBase(object):
33
34     """ constructor - try to create the tables if they don't exist
35         dbfile - name of the database file
36     """
37     def __init__(self, dbfile = 'folk.sqlite'):
38         self.conn = sqlite3.connect(dbfile)
39         self.conn.text_factory = str
40         self.cursor = self.conn.cursor()
41         self.cursor.execute('CREATE TABLE IF NOT EXISTS user (uid INTEGER PRIMARY
               KEY, sid INTEGER, value TEXT COLLATE NOCASE, done NUMERIC, FOREIGN
               KEY(sid) REFERENCES source(sid), UNIQUE(sid, value))')
42         self.cursor.execute('CREATE TABLE IF NOT EXISTS tag (tid INTEGER PRIMARY KEY,
               sid INTEGER, value TEXT COLLATE NOCASE, FOREIGN KEY(sid) REFERENCES
               source(sid), UNIQUE(sid, value))')
43         self.cursor.execute('CREATE TABLE IF NOT EXISTS source (sid INTEGER PRIMARY
               KEY, value TEXT COLLATE NOCASE)')
44         self.cursor.execute('CREATE TABLE IF NOT EXISTS resource (rid INTEGER PRIMARY
               KEY, sid INTEGER, value TEXT COLLATE NOCASE, done NUMERIC, FOREIGN
               KEY(sid) REFERENCES source(sid), UNIQUE(sid, value))')
45         self.cursor.execute('CREATE TABLE IF NOT EXISTS tagging (sid INTEGER, uid
               INTEGER, rid INTEGER, tid INTEGER, time TEXT, FOREIGN KEY(sid) REFERENCES
               source(sid), FOREIGN KEY(uid) REFERENCES user(uid), FOREIGN KEY(rid)
               REFERENCES resource(rid), FOREIGN KEY(tid) REFERENCES tag(tid))')
46         self.cursor.execute('CREATE TABLE IF NOT EXISTS control (name TEXT, value
               TEXT)')
47         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_tag (tid INTEGER, sid
               INTEGER, count INTEGER, FOREIGN KEY(tid) REFERENCES tag(tid), FOREIGN
               KEY(sid) REFERENCES source(sid))')
48         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_user (uid INTEGER, sid
               INTEGER, count INTEGER, FOREIGN KEY(uid) REFERENCES user(uid), FOREIGN
               KEY(sid) REFERENCES source(sid))')
49         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_resource (rid INTEGER,
               sid INTEGER, count INTEGER, FOREIGN KEY(rid) REFERENCES resource(rid),
               FOREIGN KEY(sid) REFERENCES source(sid))')
50         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_tu (tid INTEGER, uid
               INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(tid) REFERENCES
               tag(tid), FOREIGN KEY(uid) REFERENCES user(uid), FOREIGN KEY(sid)
               REFERENCES source(sid))')
51         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_ur (uid INTEGER, rid
               INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(uid) REFERENCES
               user(uid), FOREIGN KEY(rid) REFERENCES resource(rid), FOREIGN KEY(sid)
               REFERENCES source(sid))')
52         self.cursor.execute('CREATE TABLE IF NOT EXISTS count_rt (rid INTEGER, tid
               INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(rid) REFERENCES
               resource(rid), FOREIGN KEY(tid) REFERENCES tag(tid), FOREIGN KEY(sid)
```

```
            REFERENCES source(sid))')
53          self.cursor.execute('CREATE TABLE IF NOT EXISTS count_tt (t1 INTEGER, t2
                INTEGER, sid INTEGER, count INTEGER, FOREIGN KEY(t1) REFERENCES tag(tid),
                FOREIGN KEY(t2) REFERENCES tag(tid), FOREIGN KEY(sid) REFERENCES
                source(sid))')
54          self.conn.commit()
55
56
57      """ destructor - close the resources
58      """
59      def __del__(self):
60          self.cursor.close()
61          self.conn.close()
62
63
64      """ find the id of the given entity
65          if more than one entity is given, the order [tag; user; resource; source] is
                important
66          return the id or 'None' if failed
67      """
68      def getID(self, sid, tag = None, user = None, resource = None, source = None):
69          res = None
70          try:
71              if tag is not None:
72                  self.cursor.execute('select tid from tag where value = ? and sid =
                        ?', (tag.lower(), sid,))
73              elif user is not None:
74                  self.cursor.execute('select uid from user where value = ? and sid =
                        ?', (user.lower(), sid,))
75              elif resource is not None:
76                  self.cursor.execute('select rid from resource where value = ? and sid
                        = ?', (resource.lower(), sid,))
77              elif source is not None:
78                  self.cursor.execute('select sid from source where value = ?',
                        (source.lower(),))
79              res = self.cursor.fetchone()
80              if res is not None: res = res[0]
81          except:
82              pass
83          return res
84
85
86      """ return the id of the given entity, if it isn't in the database, insert and
            return the id
87          if more than one entity is given, the order [tag; user; resource; source] is
                important
```

```
88          return a pair <id, already>:
89             id - the id of the given entity - None if failed
90             already - [boolean] True if the entity was already in the db; False
                   otherwise - None if failed
91      """
92      def getIDInsert(self, sid, tag = None, user = None, resource = None, source =
            None):
93          # try to find the id in database
94          id = self.getID(sid, tag = tag, user = user, resource = resource, source =
                source)
95          if id is not None: return id, True
96
97          # if it isn't, try to insert
98          try:
99              if tag is not None:
100                 self.cursor.execute('insert into tag values(NULL, ?, ?)', (sid,
                        tag.lower(),))
101                 self.conn.commit()
102                 return self.getID(sid, tag = tag), False
103             elif user is not None:
104                 self.cursor.execute('insert into user values(NULL, ?, ?, 0)', (sid,
                        user.lower(),))
105                 self.conn.commit()
106                 return self.getID(sid, user = user), False
107             elif resource is not None:
108                 self.cursor.execute('insert into resource values(NULL, ?, ?, 0)',
                        (sid, resource.lower(),))
109                 self.conn.commit()
110                 return self.getID(sid, resource = resource), False
111             elif source is not None:
112                 self.cursor.execute('insert into source values(NULL, ?)',
                        (source.lower(),))
113                 self.conn.commit()
114                 return self.getID(sid, source = source), False
115         except Exception,e:
116             print '[!]', e
117             return None, None
118
119
120     """ execute the given query and return the results of it
121     """
122     def execute(self, query, params):
123         try:
124             self.cursor.execute(query, params)
125             self.conn.commit()
126             return self.cursor.fetchall()
```

```
127         except Exception,e:
128             print '[!]', e
129             return None
130 
131 
132     """ execute the given select query and return the results of it
133     """
134     def select(self, query, params):
135         try:
136             self.cursor.execute(query, params)
137             return self.cursor.fetchall()
138         except Exception,e:
139             print '[!]', e
140             return None
141 
142     """ execute the given update query
143         if the parameter 'commit' is False, the commit is delayed
144     """
145     def update(self, query, params, commit = True):
146         try:
147             self.cursor.execute(query, params)
148             if commit: self.conn.commit()
149         except Exception,e:
150             print '[!]', e
151 
152 
153     """ execute the given insert query
154         if the parameter 'commit' is False, the commit is delayed
155     """
156     def insert(self, query, params, commit = True):
157         try:
158             self.cursor.execute(query, params)
159             if commit: self.conn.commit()
160         except Exception,e:
161             print '[!]', e
162 
163 
164     """ execute the given delete query
165         if the parameter 'commit' is False, the commit is delayed
166     """
167     def delete(self, query, params, commit = True):
168         try:
169             self.cursor.execute(query, params)
170             if commit: self.conn.commit()
171         except Exception,e:
172             print '[!]', e
```

```
173
174     """ execute the commit in the database
175     """
176     def commit(self):
177         self.conn.commit()
```

```
178 """ Object Crawler - encapsulates the crawler operations
179 """
180 class Crawler(object):
181
182     """ constructor - sets the database
183     """
184     def __init__(self, db = None):
185         if db is None:
186             self.db = DataBase()
187         else:
188             self.db = db
189
190
191     """ destructor - deletes the database
192     """
193     def __del__(self):
194         del self.db
195
196
197     """ calls the select of the current database
198     """
199     def dbSelect(self, query, params):
200         return self.db.select(query, params)
201
202
203     """ calls the update of the current database
204     """
205     def dbUpdate(self, query, params):
206         self.db.update(query, params)
207
208
209     """ calls the delete of the current database
210     """
211     def dbDelete(self, query, params):
212         self.db.delete(query, params)
213
214
215     """ get the id of the given entity.
216         the optional parameter 'insert' indicates if is necessary to insert the
            entity in the database
```

```
217     """
218     def dbGetId(self, sid, insert = False, tag = None, user = None, resource = None,
            source = None):
219         if not insert:
220             return self.db.getID(sid, tag, user, resource, source)
221         else:
222             return self.db.getIDInsert(sid, tag, user, resource, source)
223
224
225     """ insert a 'tagging object' (a triple user, resource, tag associated with a
            source) in the database
226     """
227     def dbInsertTagging(self, sid, uid, rid, tid, time = None):
228
229         # no parameter (except 'time') can be 'None'
230         if sid is None or uid is None or rid is None or tid is None:
231             print 'invalid values', sid, uid, rid, tid
232             return
233
234         # transaction [begin] - delay commit until transaction ends
235         # update the counters -
236         #   if there's no entity in db: counter = 1; else counter += 1.
237         r = self.db.select('select count from count_tag where tid = ? and sid = ?',
                (tid, sid))
238         if r == []: self.db.insert('insert into count_tag values (?, ?, 1)', (tid,
                sid), commit = False)
239         else: self.db.update('update count_tag set count = ? where tid = ? and sid =
                ?', (int(r[0][0]) + 1, tid, sid), commit = False)
240
241         r = self.db.select('select count from count_user where uid = ? and sid = ?',
                (uid, sid))
242         if r == []: self.db.insert('insert into count_user values (?, ?, 1)', (uid,
                sid), commit = False)
243         else: self.db.update('update count_user set count = ? where uid = ? and sid =
                ?', (int(r[0][0]) + 1, uid, sid), commit = False)
244
245         r = self.db.select('select count from count_resource where rid = ? and sid =
                ?', (rid, sid))
246         if r == []: self.db.insert('insert into count_resource values (?, ?, 1)',
                (rid, sid), commit = False)
247         else: self.db.update('update count_resource set count = ? where rid = ? and
                sid = ?', (int(r[0][0]) + 1, rid, sid), commit = False)
248
249         r = self.db.select('select count from count_tu where tid = ? and uid = ? and
                sid = ?', (tid, uid, sid))
```

```
250         if r == []: self.db.insert('insert into count_tu values (?, ?, ?, 1)', (tid,
                uid, sid), commit = False)
251         else: self.db.update('update count_tu set count = ? where tid = ? and uid = ?
                and sid = ?', (int(r[0][0]) + 1, tid, uid, sid), commit = False)
252
253         r = self.db.select('select count from count_ur where uid = ? and rid = ? and
                sid = ?', (uid, rid, sid))
254         if r == []: self.db.insert('insert into count_ur values (?, ?, ?, 1)', (uid,
                rid, sid), commit = False)
255         else: self.db.update('update count_ur set count = ? where uid = ? and rid = ?
                and sid = ?', (int(r[0][0]) + 1, uid, rid, sid), commit = False)
256
257         r = self.db.select('select count from count_rt where rid = ? and tid = ? and
                sid = ?', (rid, tid, sid))
258         if r == []: self.db.insert('insert into count_rt values (?, ?, ?, 1)', (rid,
                tid, sid), commit = False)
259         else: self.db.update('update count_rt set count = ? where rid = ? and tid = ?
                and sid = ?', (int(r[0][0]) + 1, rid, tid, sid), commit = False)
260
261         # insert the 'tagging object'
262         self.db.insert('insert into tagging values(?, ?, ?, ?, ?)', (sid, uid, rid,
                tid, time), commit = False)
263
264         # get all tags in the same post
265         tags = self.db.select('select distinct tid from tagging where rid = ? and uid
                = ? and sid = ?', (rid, uid, sid))
266
267         for _t in tags:
268             t = _t[0]
269             # convention - id t1 is always less than id t2
270             if tid < t:
271                 t1 = tid
272                 t2 = t
273             else:
274                 t1 = t
275                 t2 = tid
276
277             # update the counter of tag - tag relation
278             r = self.db.select('select count from count_tt where t1 = ? and t2 = ?
                    and sid = ?', (t1, t2, sid))
279             if r == []:
280                 self.db.insert('insert into count_tt values (?, ?, ?, 1)', (t1, t2,
                        sid), commit = False)
281             else:
282                 self.db.update('update count_tt set count = ? where t1 = ? and t2 = ?
                        and sid = ?', (int(r[0][0]) + 1, t1, t2, sid), commit = False)
```

```
283
284         # transaction [end]
285         self.db.commit()
286
287     """ the main method that get the data
288         the parameter 'proctime' is the minimum amount of time processing. The actual
                time may be (and usually is) greater.
289     """
290     def crawl(self, source = Source(), proctime = 10, trace = False):
291
292         # create the crawler object of the given source
293         if source.isDelicious():
294             dc = DeliciousCrawler(self)
295             try: dc.crawl(proctime = proctime, trace = trace)
296             except Exception,e:
297                 print e
298                 pass
299         elif source.isFlickr():
300             fc = FlickrCrawler(self)
301             try: fc.crawl(proctime = proctime, trace = trace)
302             except Exception,e:
303                 print e
304                 pass
305         # all sources
306         else:
307             start = time.time()
308             dc = DeliciousCrawler(self)
309             fc = FlickrCrawler(self)
310             td = dc.initCrawlResult()
311             tf = fc.initCrawlResult()
312
313             # run until the timeout
314             while True:
315                 try:
316                     # minimum processing time
317                     t = dc.crawl(proctime = 1, trace = trace)
318                     td = dc.sumCrawlResults(td, t)
319                 except Exception,e:
320                     print e
321                     pass
322                 try:
323                     # minimum processing time
324                     t = fc.crawl(proctime = 1, trace = trace)
325                     tf = fc.sumCrawlResults(tf, t)
326                 except Exception,e:
327                     print e
```

```
                    pass

                # verify if the timeout has ben reached
                end = time.time()
                delta = end - start
                if (delta / 60) > proctime: break

            # print the results
            if trace:
                print '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$'
                print 'Total time:', datetime.timedelta(seconds = delta)
                dc.printCrawlResults(td)
                fc.printCrawlResults(tf)
```

```
""" Object DeliciousCrawler - encapsulates the delicious crawler operations
"""
class DeliciousCrawler(object):

    """ constructor - initialize the variables
    """
    def __init__(self, contr):

        # api object
        self.dapi = deliciousapi.DeliciousAPI(user_agent = "folklicious v0.1.5")
        # controller - the main crawl object
        self.contr = contr
        # source id
        self.sid, ondb = self.contr.dbGetId(None, insert = True, source = 'delicious')
        # waiting time for each data category
        self.waiting = {'hotlist': 1800, 'user': 300, 'url': 30}
        # max elements in the 'not done' list
        self.max_notdone = {'resources': 200}

    """ the main method that get the data
        the parameter 'proctime' is the minimum amount of time processing.
        the actual time may be (and usually is) greater.
    """
    def crawl(self, proctime = 10, trace = False):

        # start the processing
        start = time.time()

        # verify if more than x seconds have been elapsed since the last
        # requisition to hotlist - delicious restriction
        oldtime = self.contr.dbSelect("select value from control where name =
            'del.hotlist.timestamp'", ())
```

```
372         old = int(oldtime[0][0])
373         interval = time.time() - old
374         if interval < self.waiting['hotlist']: # 30min
375             if trace: print 'wait more', (self.waiting['hotlist'] - interval),
                    'seconds to call populateWithHotList'
376             size_hot = 0
377         else:
378             # populate the hotlist
379             size_hot = self.populateWithHotList(trace)
380             # set the time of the requisition
381             self.contr.dbUpdate('update control set value = ? where name =
                    "del.hotlist.timestamp"', (int(round(time.time())),))
382
383         # initialize the variables
384         waiting = 0
385         size_url = 0
386         size_usr = 0
387         count_pop_user = 0
388
389         # process until timeout
390         while True:
391
392             # 'progress bar'
393             for i in range(10, waiting):
394                 sys.stdout.write('.')
395                 time.sleep(1)
396                 if i == waiting - 1: print ''
397             count_pop_user += 1
398
399             # verify if more  than x seconds have been elapsed since the last
400             # requisition to hotlist - delicious restriction
401             oldtime = self.contr.dbSelect("select value from control where name =
                    'del.url.timestamp'", ())
402             old = int(oldtime[0][0])
403             interval = time.time() - old
404             if interval < self.waiting['url']: # 15min
405                 if trace: print 'wait more', (self.waiting['url'] - interval),
                        'seconds to call populateWithUrl'
406                 waiting += 1
407             else:
408                 # populate with url as seed
409                 size_url += self.populateWithUrl(1, trace)
410
411             # only populate with user if tried to populate with url 20 times
412             if count_pop_user == 20:
413
```

```
414                 # verify if more  than x seconds have been elapsed since the last
415                 # requisition to hotlist - delicious restriction
416                 oldtime = self.contr.dbSelect("select value from control where name =
                        'del.user.timestamp'", ())
417                 old = int(oldtime[0][0])
418                 interval = time.time() - old
419                 if interval < self.waiting['user']: # 15min
420                     if trace: print 'wait more', (self.waiting['user'] - interval),
                            'seconds to call populateWithUser'
421                     waiting += 1
422                 else:
423                     # populate with user as seed
424                     size_usr += self.populateWithUser(1, trace)
425
426                 # reset the counter
427                 count_pop_user = 0
428
429             # verify if the minimum processing time has been reached
430             end = time.time()
431             delta = end - start
432             if (delta / 60) > proctime:
433                 if trace:
434                     print 'del :: time reached', datetime.timedelta(seconds=delta)
435                     print 'del ::', size_hot, 'new resources from hotlist'
436                     print 'del ::', size_url, 'new tuples'
437                     print 'del ::', size_usr, 'new urls'
438                 break
439
440         # return the elapsed time, and the amount of elements
441         return (delta, size_hot, size_url, size_usr)
442
443     """ initialize the results
444     """
445     def initCrawlResult(self):
446         return (0, 0, 0, 0)
447
448     """ sum two sets of results
449     """
450     def sumCrawlResults(self, t1, t2):
451         return (t1[0] + t2[0], t1[1] + t2[1], t1[2] + t2[2], t1[3] + t2[3])
452
453     """ print the results
454     """
455     def printCrawlResults(self ,t):
456         print '$del :: total time', datetime.timedelta(seconds=t[0])
457         print '$del ::', t[1], 'new resources from hotlist'
```

```
458         print '$del ::', t[2], 'new tuples'
459         items = self.contr.dbSelect('select count(*) from resource where sid = ? and
                done = 0', (self.sid,))
460         print '$del ::', t[3], 'new urls,', items[0][0], 'to be processed'
461
462     """ populate the db using the hotlist
463     """
464     def populateWithHotList(self, trace = False):
465         if trace: print 'del :: populateWithHotList'
466
467         # get the resources not processed yet
468         notdone = self.contr.dbSelect('select count(*) from resource where sid = ?
                and done = 0', (self.sid,))[0][0]
469         if notdone > self.max_notdone['resources']:
470             if trace: print 'There are %d (%d) resources notdone.' % (notdone,
                    self.max_notdone['resources'])
471             return 0
472         # get the URLs in hotlist
473         urls = self.dapi.get_urls()
474         if trace: print len(urls), 'urls retrieved'
475         count = 0
476
477         # for each URL
478         for u in urls:
479             # try to insert in db
480             rid, ondb = self.contr.dbGetId(self.sid, insert = True, resource = u)
481             if not ondb:
482                 count += 1
483                 if trace: print 'inserted', u
484             elif trace: print 'already inserted', u
485         if trace: print count, 'new urls'
486
487         # return the number of new objects stored
488         return count
489
490     """ populate the db using the URL
491     """
492     def populateWithUrl(self, max_urls = 10, trace = False):
493         if trace: print 'del :: populateWithUrl - max_urls:', max_urls
494
495         # get the resources not processed yet
496         res = self.contr.dbSelect('select value from resource where done = 0 and sid
                = ? limit ?', (self.sid, max_urls,))
497         total = len(res)
498         if total > max_urls: total = max_urls
499         if trace: print total, 'urls'
```

```
500         curr = 1
501         count = 0
502 
503         # process each resource ...
504         for _r in res:
505             # ... until reach the maximum
506             if curr > max_urls: break
507 
508             r = _r[0]
509             if trace: print 'resource %03d/%03d' %(curr, total)
510 
511             # get the bookmarks associated with that URL
512             # with 'max_bookmarks=0' all of them are retorned, but it consumes more
                    time
513             meta = self.dapi.get_url(r, max_bookmarks=0)
514             # store the requisition time
515             self.contr.dbUpdate('update control set value = ? where name =
                    "del.url.timestamp"', (int(round(time.time())),))
516 
517             # process the bookmarks
518             total_bookmarks = len(meta.bookmarks)
519             if trace: print 'resource %s - %d bookmarks' %(r, total_bookmarks)
520             if total_bookmarks > 0 :
521                 curr_bookmarks = 0
522                 for b in meta.bookmarks:
523                     curr_bookmarks += 1
524                     # [user, taglist, comment, time]
525                     user = b[0]
526                     taglist = b[1]
527                     timestamp = b[3]
528 
529                     # user id
530                     uid, ondb = self.contr.dbGetId(self.sid, insert = True, user =
                            user)
531                     # resource id
532                     rid, ondb = self.contr.dbGetId(self.sid, insert = True, resource
                            = r)
533 
534                     # failed to insert
535                     if uid is None or rid is None:
536                         break
537 
538                     for t in taglist:
539                         count += 1
540                         # tag id
```

```
541                         tid, ondb = self.contr.dbGetId(self.sid, insert = True, tag =
                                t)
542                         # insert the tagging object
543                         self.contr.dbInsertTagging(self.sid, uid, rid, tid,
                                str(timestamp))
544                         if trace:
545                             print 'ids:', uid, rid, tid
546                             print 'inserted', (user, r[:60], t, str(timestamp))
547                             print '[%03d/%03d resources][%03d/%03d bookmarks][%4d
                                    inserts]' %(curr, total, curr_bookmarks,
                                    total_bookmarks, count)
548
549                 # set the resource as done
550                 self.contr.dbUpdate('update resource set done = 1 where value = ? and
                        sid = ?', (r, self.sid,))
551             else:
552                 # set the resource as done - no tags
553                 self.contr.dbUpdate('update resource set done = 1 where value = ? and
                        sid = ?', (r, self.sid,))
554                 if trace: print 'no tags -> done', r[:75]
555             curr += 1
556
557         # return the number of new objects stored
558         return count
559
560     """ populate the db using the URL
561     """
562     def populateWithUser(self, max_users = 10, trace = False):
563         if trace: print 'del :: populateWithUser - max_users:', max_users
564
565         # get the resources not processed yet
566         notdone = self.contr.dbSelect('select count(*) from resource where sid = ?
                and done = 0', (self.sid,))[0][0]
567         # if there are more resources than the maximum, don't try to populate with
                user
568         if notdone > self.max_notdone['resources']:
569             if trace: print 'There are %d (%d) resources not done.' % (notdone,
                    self.max_notdone['resources'])
570             return 0
571
572         # get the users not processed yet
573         users = self.contr.dbSelect('select value from user where done = 0 and sid =
                ? limit ?', (self.sid, max_users,))
574         total = len(users)
575         if total > max_users: total = max_users
576         if trace: print total, 'users'
```

```
577         curr = 1
578         count = 0
579 
580         # process each user ...
581         for _u in users:
582             # ... until reach the maximum
583             if curr > max_users: break
584 
585             u = _u[0]
586             if trace: print 'user %03d/%03d' %(curr, total)
587 
588             try:
589                 # get the bookmarks associated with that user
590                 meta = self.dapi.get_user(u, max_bookmarks = 10)
591                 # store the requisition time
592                 self.contr.dbUpdate('update control set value = ? where name =
                        "del.user.timestamp"', (int(round(time.time())),))
593             except:
594                 # if there was an error, mark that resource as already processed and
                        continue to the next one
595                 if trace: print 'Delicious error: user', u, 'marked as done'
596                 self.contr.dbUpdate('update user set done = 1 where value = ? and sid
                        = ?', (u, self.sid,))
597                 continue
598 
599             if len(meta.bookmarks) > 0 :
600                 for b in meta.bookmarks:
601                     # [url, taglist, title, comment, time]
602                     r = b[0]
603                     # resource id
604                     rid, ondb = self.contr.dbGetId(self.sid, insert = True, resource
                            = r)
605                     if not ondb:
606                         count += 1
607                         if trace: print 'inserted', r, 'id:', rid
608                     else:
609                         if trace: print 'already inserted', r, 'id:', rid
610                     if trace: print '[%03d/%03d] %d inserts' %(curr, total, count)
611 
612             # set the user as done
613             self.contr.dbUpdate('update user set done = 1 where value = ? and sid =
                    ?', (u, self.sid))
614             curr += 1
615 
616         # return the number of new objects stored
617         return count
```

```
618 """ Object FlickrCrawler - encapsulates the flickr crawler operations
619 """
620 class FlickrCrawler(object):
621
622     """ constructor - initialize the variables
623     """
624     def __init__(self, contr):
625         # controller - the main crawl object
626         self.contr = contr
627         # source id
628         self.sid, ondb = self.contr.dbGetId(None, insert = True, source = 'flickr')
629         # base URL
630         self.url = 'http://api.flickr.com/services/rest/'
631         # api key - REPLACE WITH YOUR OWN API KEY
632         self.api_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
633         # api methods
634         self.methods = {
635             'echo':     'flickr.test.echo',
636             'pInfo':    'flickr.photos.getInfo',
637             'pPList':   'flickr.people.getPublicPhotos',
638             'uPList':   'flickr.contacts.getPublicList',
639             'uPPhotos': 'flickr.photos.getContactsPublicPhotos',
640             'pRecent':  'flickr.photos.getRecent',
641             'tHot':     'flickr.tags.getHotList',
642             'tPhoto':   'flickr.tags.getListPhoto',
643             'panda':    'flickr.panda.getPhotos',
644         }
645
646     """ the main method that get the data
647         the parameter 'proctime' is the minimum amount of time processing.
648         the actual time may be (and usually is) greater.
649     """
650     def crawl(self, proctime = 10, trace = False):
651
652         # start the processing
653         start = time.time()
654
655         # initialize the variables
656         size_tag = 0
657         size_res = 0
658         count_pop_res = 0
659
660         # process until timeout
661         while True:
662             count_pop_res += 1
```

```
663
664             # populate the db with tags
665             size_tag += self.populateTags(1, trace = trace)
666
667             # only populate with user if tried to populate with tags 20 times
668             if count_pop_res == 20:
669                 size_res += self.populateResources(1, trace = trace)
670                 count_pop_user = 0
671
672             # verify if the minimum processing time has been reached
673             end = time.time()
674             delta = end - start
675             if (delta / 60) > proctime:
676                 if trace:
677                     print 'fli :: time reached', datetime.timedelta(seconds=delta)
678                     print 'fli ::', size_tag, 'new tuples'
679                     print 'fli ::', size_res, 'new resources'
680                 break
681
682         # return the elapsed time, and the amount of elements
683         return (delta, size_tag, size_res)
684
685     """ initialize the results
686     """
687     def initCrawlResult(self):
688         return (0, 0, 0)
689
690     """ sum two sets of results
691     """
692     def sumCrawlResults(self, t1, t2):
693         return (t1[0] + t2[0], t1[1] + t2[1], t1[2] + t2[2])
694
695     """ print the results
696     """
697     def printCrawlResults(self, t):
698         print '$fli :: total time', datetime.timedelta(seconds=t[0])
699         print '$fli ::', t[1], 'new tuples'
700         items = self.contr.dbSelect('select count(*) from resource where sid = ? and
                done = 0', (self.sid,))
701         print '$fli ::', t[2], 'new resources,', items[0][0], 'to be processed'
702
703     """ execute a post request
704     """
705     def post(self, params):
706         return urllib.urlopen(self.url, params)
707
```

```
708     """ create the params object
709     """
710     def createParams(self, method, params, trace = False):
711         params['method'] = method
712         params['api_key'] = self.api_key
713         if trace: print params
714         return urllib.urlencode(params)
715 
716     """ get random photos - this service is called panda in flickr
717     """
718     def getPhotosPanda(self, per_page = None, panda = None):
719         # choose the panda - 'ling ling', 'hsing hsing', and 'wang wang'
720         if panda is None: panda = random.randint(1, 3)
721         if panda == 1:
722             lst = {'panda_name': 'ling ling'}
723         elif panda == 2:
724             lst = {'panda_name': 'hsing hsing'}
725         else:
726             lst = {'panda_name': 'wang wang'}
727         if per_page is not None:
728             lst['per_page'] = per_page
729 
730         # return the result
731         return self.post(self.createParams(self.methods['panda'], lst))
732 
733     """ populate the db using resources
734     """
735     def populateResources(self, num_photos, pandaid = None, trace = False):
736         if trace: print 'fli :: populateResources - num_photos:', num_photos,
              'pandaid:', pandaid
737 
738         # the limit is 500 photos
739         if num_photos <= 500:
740 
741             # xml with photos from pandas
742             pxml = self.getPhotosPanda(num_photos, pandaid)
743             # get the information
744             root = parse(pxml).getroot().find('photos')
745             interval = root.get('interval')
746             lastupdate = root.get('lastupdate')
747             photos = root.findall('photo')
748             total = len(photos)
749             curr = 1
750             count = 0
751 
752             # for each photo of the xml
```

```
753             for p in photos:
754                 # insert in the db
755                 pid = p.get('id')
756                 rid, ondb = self.contr.dbGetId(self.sid, insert = True, resource =
                        pid)
757                 if not ondb:
758                     count += 1
759                 if trace: print '%s: %03d/%03d' % (rid, curr, total)
760                 curr += 1
761 
762             # flickr gives the lastupdate time and the waiting interval
763             wait = time.time() - (int(lastupdate) + int(interval))
764             # so, verify if need to wait
765             if wait > 0:
766                 if trace: print 'waiting', wait, 'before next requisition'
767                 time.sleep(wait)
768 
769         # more than 500 photos - multiple requests
770         else:
771             n = 0
772             r = 0
773             while n < num_photos:
774                 r += self.populateResources(500)
775                 n += 500
776             return r
777 
778         # return the number of new objects stored
779         return count
780 
781     """ populate the db using tags
782     """
783     def populateTags(self, num_resources = 10, trace = False):
784         if trace: print 'fli :: populateTags - num_resources:', num_resources
785 
786         # get the resources not processed yet
787         res = self.contr.dbSelect('select rid from resource where done = 0 and sid =
                ? limit ?', (self.sid, num_resources,))
788         curr = 1
789         if len(res) > num_resources:
790             total = num_resources
791         else:
792             total = len(res)
793         count = 0
794 
795         # process each resource ...
796         for _r in res:
```

```
797             rid = _r[0]
798
799             # ... until reach the maximum
800             if curr > total:
801                 break
802
803             if trace: print 'resources: %03d/%03d' % (curr, total)
804
805             # get the data from photo
806             data = self.getDataFromPhoto(rid, trace = trace)
807             notags = True
808
809             for d in data:
810                 # user id
811                 uid, ondb = self.contr.dbGetId(self.sid, insert = True, user =
                        d['user'])
812                 # tag id
813                 tid, ondb = self.contr.dbGetId(self.sid, insert = True, tag =
                        d['tag'])
814
815                 # insert the tagging object
816                 self.contr.dbInsertTagging(self.sid, uid, rid, tid)
817                 count += 1
818                 if trace: print '%d new tuple: %s' % (count, (self.sid, uid, rid,
                        tid,))
819                 notags = False
820
821             # set the resource as done
822             self.contr.dbUpdate('update resource set done = 1 where rid = ? and sid =
                    ?', (rid, self.sid,))
823             if trace: print 'rid:', rid, 'set done - notags:', notags
824             curr += 1
825
826         # return the number of new objects stored
827         return count
828
829     """ return the xml file with the tags of a given photo
830     """
831     def getTagsFromPhoto(self, photo_id):
832         return self.post(self.createParams(self.methods['tPhoto'], {'photo_id':
                photo_id}))
833
834     """ return the data from a given photo
835     """
836     def getDataFromPhoto(self, photo_id, trace = False):
837
```

```
838         # get the tags
839         tags = self.getTagsFromPhoto(photo_id)
840         root = parse(tags).getroot()
841         stat = root.get('stat')
842         if stat == 'fail' and trace: print '[!] Flickr error:',
                root.find('err').get('msg'), 'photo_id', photo_id
843
844         # prepare the result with all pairs {user; tag}
845         result = []
846         for t in root.findall('photo/tags/tag'):
847             result.append({'user': t.get('author'), 'tag': t.text})
848
849         # return the result data
850         return result
```

```
851 # the main execution
852 if __name__ == "__main__":
853     # create the crawler object
854     c = Crawler()
855     # execute a crawl operation in all sources and with minimum 60 minutes
856     c.crawl(Source(), proctime=60, trace=True)
857
858     # example of crawl in delicious with minimum 45 minutes
859     # c.crawl(Source('delicious'), proctime = 45, trace = True)
860     # example of crawl in flickr with minimum 15 minutes
861     # c.crawl(Source('flickr'), proctime = 15, trace = True)
862     # delete the object and release all resources
863     del c
```

A.4 Folksonomy Systems

A.4.1 Flickr

Flickr is an online community and an image and video hosting website. It has a large user community and huge amount of resources (mainly images). Whenever authors uploads an image they insert tags to describe it. The main search mechanism of Flickr is based on tags.

Flickr offers a web service API to access its data by non-commercial applications. One important aspect of the API is that all data should be codified by using the UTF-8 standard. If the API receives a sequence in any codification but UTF-8, it assumes that the codification is ISO-8859-1 and then transforms it to UTF-8. Any other codification could result in incorrect data. In the following subsections we detail aspects of the Flickr

API.

Authentication

The services can be accessed in non-authenticated or authenticated modes. There are API methods that are only available to the authenticated users. The authentication process is described in the Flickr OAuthAPI webpage[7]. In our study we focused on public data. Therefore, we did not use the authenticate methods.

Definitions

Services' protocols and data objects exchanged by Flickr and clients follow a set of basic standard definitions further detailed.

Dates[8] There are two types of dates: *taken* – the date when the photo was taken; *posted* – the date when the photo was posted on the system. *Taken* dates must follow the MySQL 'datetime' format (e.g., 2004-11-29 16:01:26) and they have 4 levels of accuracy:
0 Y-m-d H:i:s
4 Y-m
6 Y
8 circa...

On the other hand, *posted* dates are always in the unix timestamp format, i.e., an unsigned integer specifying the number of seconds since Jan 1st 1970 GMT.

Buddyicons[9] Buddyicon is a 48x48 pixel image that represent the user – an "avatar". It is necessary to inform the user's NSID (user id), icon server and icon farm to access the buddyicon of a user.

If the icon server parameter is a number greater than zero. A URL to request the icon takes the format:

```
http://farm{icon-farm}.static.flickr.com/{icon-server}/
buddyicons/{nsid}.jpg
```

There is also a URL to request the default buddyicon:

```
http://www.flickr.com/images/buddyicon.jpg
```

[7] http://www.flickr.com/services/api/auth.oauth.html
[8] http://www.flickr.com/services/api/misc.dates.html
[9] http://www.flickr.com/services/api/misc.buddyicons.html

URLs[10] The photo URL is built by using the following parameters: photo ID, server ID, farm ID, and secret. The URL takes the format:

```
http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
or
http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_[mstzb].jpg
or
http://farm{farm-id}.static.flickr.com/{server-id}/
{id}_{o-secret}_o.(jpg|gif|png)
```

The second URL affords one of the following size suffixes (indicated in the URL by brackets):
s – small square 75x75
t – thumbnail, 100 on longest side
m – small, 240 on longest side
- – medium, 500 on longest side
z – medium 640, 640 on longest side
b – large, 1024 on longest side
The last URL is specific for images in the original size:
o – original image, either a jpg, gif or png, depending on source format

Tags[11] When a photo has tags, the format of the tag field in XML is as follows:

```
<tag id="1234" author="12037949754@N01" raw="woo yay">wooyay</tag>
```

The parameters are:
id – The photo id.
author – The NSID of the user who added the tag.
raw – The "raw" version of the tag - as entered by the user. This version can contain spaces and punctuation.
tag-body – The "clean" version of the tag – as processed by Flickr.

Access

API keys are necessary to access and use the flickr data. The process to obtain those keys is the following:

1. Register to Flickr (as a user).
2. Go to the API request page (`http://www.flickr.com/services/apps/create/apply/`).

[10] `http://www.flickr.com/services/api/misc.urls.html`
[11] `http://www.flickr.com/services/api/misc.tags.html`

3. Choose the appropriate option (in this work we used "Non-Commercial").

4. Fill in the form.

In the end of this process the API keys are generated.

Request Protocols

There are three request protocols: REST[12], XML-RPC[13], and SOAP[14]. We adopted REST since it is the simplest option and it meets our needs.

The following pair requisition/response illustrates a REST requisition on Flickr. In this case we are using a fictitious API key.

```
http://www.flickr.com/services/rest/?method=flickr.test.echo&format=rest
&foo=bar&api_key=cc4094c55264c02ec2a83001b95a0837

<rsp stat="ok">
  <method>flickr.test.echo</method>
    <format>rest</format>
    <foo>bar</foo>
  <api_key>cc4094c55264c02ec2a83001b95a0837</api_key>
</rsp>
```

Response Formats

There are five response formats: REST[15], XML-RPC[16], SOAP[17], JSON[18], and PHP[19]. Again, we chose the REST format.

API Methods

In this section we briefly describe the API methods that are of most relevant to this work. The complete list of methods can be viewed in the API page (`http://www.flickr.com/services/api/`).

auth: methods to authenticate the app.

[12] `http://www.flickr.com/services/api/request.rest.html`
[13] `http://www.flickr.com/services/api/request.xmlrpc.html`
[14] `http://www.flickr.com/services/api/request.soap.html`
[15] `http://www.flickr.com/services/api/response.rest.html`
[16] `http://www.flickr.com/services/api/response.xmlrpc.html`
[17] `http://www.flickr.com/services/api/response.soap.html`
[18] `http://www.flickr.com/services/api/response.json.html`
[19] `http://www.flickr.com/services/api/response.php.html`

- flickr.auth.checkToken[20]: returns the credentials attached to a token
- flickr.auth.getFrob[21]: returns the frob to be used in authentication
- flickr.auth.getFullToken[22]: returns the full token from a mini-token
- flickr.auth.getToken[23]: returns the token from a frob

contacts

- flickr.contacts.getList[24]: returns the contact list from a user
- flickr.contacts.getPublicList[25]: returns the public contact list from a user (doesn't need authentication)

galleries

- flickr.galleries.getInfo[26]: returns the information of a gallery
- flickr.galleries.getList[27]: returns the list of galleries
- flickr.galleries.getListForPhoto[28]: returns the list of galleries that contains a given photo
- flickr.galleries.getPhotos[29]: returns the photos of a given gallery

interestingness

- flickr.interestingness.getList[30]: returns the most interesting photos of a specific date

machinetags

- flickr.machinetags.getNamespaces[31]: returns a list of unique namespaces
- flickr.machinetags.getPairs[32]: returns a list of unique namespace and predicate pairs

[20] http://www.flickr.com/services/api/flickr.auth.checkToken.html
[21] http://www.flickr.com/services/api/flickr.auth.getFrob.html
[22] http://www.flickr.com/services/api/flickr.auth.getFullToken.html
[23] http://www.flickr.com/services/api/flickr.auth.getToken.html
[24] http://www.flickr.com/services/api/flickr.contacts.getList.html
[25] http://www.flickr.com/services/api/flickr.contacts.getPublicList.html
[26] http://www.flickr.com/services/api/flickr.galleries.getInfo.html
[27] http://www.flickr.com/services/api/flickr.galleries.getList.html
[28] http://www.flickr.com/services/api/flickr.galleries.getListForPhoto.html
[29] http://www.flickr.com/services/api/flickr.auth.getToken.html
[30] http://www.flickr.com/services/api/flickr.interestingness.getList.html
[31] http://www.flickr.com/services/api/flickr.machinetags.getNamespaces.html
[32] http://www.flickr.com/services/api/flickr.machinetags.getPairs.html

- flickr.machinetags.getPredicates[33]: returns a list of unique predicates
- flickr.machinetags.getRecentValues[34]: returns the most recent machinetags
- flickr.machinetags.getValues[35]: returns a list of unique values for a namespace and predicate

panda

- flickr.panda.getList[36]: returns a list of pandas (photo services)
- flickr.panda.getPhotos[37]: returns a list of photos of the given panda

people

- flickr.people.findByEmail[38]: returns a user's NSID, given his/her email address
- flickr.people.findByUsername[39]: returns a user's NSID, given his/her username
- flickr.people.getInfo[40]: gets information about a user
- flickr.people.getPhotos[41]: returns photos from the given user's photostream
- flickr.people.getPhotosOf[42]: returns a list of photos containing a particular Flickr member
- flickr.people.getPublicPhotos[43]: gets a list of public photos for the given user

photos

- flickr.photos.getAllContexts[44]: returns all visible sets and pools the photo belongs to
- flickr.photos.getContactsPhotos[45]: fetches a list of recent photos from the calling users' contacts

[33] http://www.flickr.com/services/api/flickr.machinetags.getPredicates.html
[34] http://www.flickr.com/services/api/flickr.machinetags.getRecentValues.html
[35] http://www.flickr.com/services/api/flickr.machinetags.getValues.html
[36] http://www.flickr.com/services/api/flickr.panda.getList.html
[37] http://www.flickr.com/services/api/flickr.panda.getPhotos.html
[38] http://www.flickr.com/services/api/flickr.people.findByEmail.html
[39] http://www.flickr.com/services/api/flickr.people.findByUsername.html
[40] http://www.flickr.com/services/api/flickr.people.getInfo.html
[41] http://www.flickr.com/services/api/flickr.people.getPhotos.html
[42] http://www.flickr.com/services/api/flickr.people.getPhotosOf.html
[43] http://www.flickr.com/services/api/flickr.people.getPublicPhotos.html
[44] http://www.flickr.com/services/api/flickr.photos.getAllContexts.html
[45] http://www.flickr.com/services/api/flickr.photos.getContactsPhotos.html

- flickr.photos.getContactsPublicPhotos[46]: fetches a list of recent public photos from a users' contacts
- flickr.photos.getContext[47]: returns next and previous photos for a photo in a photostream
- flickr.photos.getCounts[48]: gets a list of photo counts for the given date ranges for the calling user
- flickr.photos.getInfo[49]: gets information about a photo. The calling user must have permission to view the photo
- flickr.photos.getPerms[50]: gets permissions for a photo
- flickr.photos.getRecent[51]: returns a list of the latest public photos uploaded to flickr
- flickr.photos.getSizes[52]: returns the available sizes for a photo. The calling user must have permission to view the photo
- flickr.photos.getUntagged[53]: returns a list of your photos with no tags
- flickr.photos.getWithGeoData[54]: returns a list of your geo-tagged photos
- flickr.photos.getWithoutGeoData[55]: returns a list of your photos which haven't been geo-tagged
- flickr.photos.recentlyUpdated[56]: returns a list of your photos that have been recently created or which have been recently modified
- flickr.photos.search[57]: returns a list of photos matching some criteria

photos.licenses

- flickr.photos.licenses.getInfo[58]: fetches a list of available photo licenses for Flickr

[46] `http://www.flickr.com/services/api/flickr.photos.getContactsPublicPhotos.html`
[47] `http://www.flickr.com/services/api/flickr.photos.getContext.html`
[48] `http://www.flickr.com/services/api/flickr.photos.getCounts.html`
[49] `http://www.flickr.com/services/api/flickr.photos.getInfo.html`
[50] `http://www.flickr.com/services/api/flickr.photos.getPerms.html`
[51] `http://www.flickr.com/services/api/flickr.photos.getRecent.html`
[52] `http://www.flickr.com/services/api/flickr.photos.getSizes.html`
[53] `http://www.flickr.com/services/api/flickr.photos.getUntagged.html`
[54] `http://www.flickr.com/services/api/flickr.photos.getWithGeoData.html`
[55] `http://www.flickr.com/services/api/flickr.photos.getWithoutGeoData.html`
[56] `http://www.flickr.com/services/api/flickr.photos.recentlyUpdated.html`
[57] `http://www.flickr.com/services/api/flickr.photos.search.html`
[58] `http://www.flickr.com/services/api/flickr.photos.licenses.getInfo.html`

photosets

- flickr.photosets.getContext[59]: returns next and previous photos for a photo in a set
- flickr.photosets.getInfo[60]: gets information about a photoset
- flickr.photosets.getList[61]: returns the photosets belonging to the specified user
- flickr.photosets.getPhotos[62]: gets the list of photos in a set

reflection

- flickr.reflection.getMethodInfo[63]: returns information for a given flickr API method
- flickr.reflection.getMethods[64]: returns a list of available flickr API methods

tags

- flickr.tags.getClusterPhotos[65]: returns the first 24 photos for a given tag cluster
- flickr.tags.getClusters[66]: returns a list of tag clusters for the given tag
- flickr.tags.getHotList[67]: returns a list of hot tags for the given period
- flickr.tags.getListPhoto[68]: gets the tag list for a given photo
- flickr.tags.getListUser[69]: gets the tag list for a given user (or for the user currently logged)
- flickr.tags.getListUserPopular[70]: gets the popular tags for a given user (or for the user currently logged)
- flickr.tags.getListUserRaw[71]: gets the raw versions of a given tag (or all tags) for the user currently logged

[59] http://www.flickr.com/services/api/flickr.photosets.getContext.html
[60] http://www.flickr.com/services/api/flickr.photosets.getInfo.html
[61] http://www.flickr.com/services/api/flickr.photosets.getList.html
[62] http://www.flickr.com/services/api/flickr.photosets.getPhotos.html
[63] http://www.flickr.com/services/api/flickr.reflection.getMethodInfo.html
[64] http://www.flickr.com/services/api/flickr.reflection.getMethods.html
[65] http://www.flickr.com/services/api/flickr.tags.getClusterPhotos.html
[66] http://www.flickr.com/services/api/flickr.tags.getClusters.html
[67] http://www.flickr.com/services/api/flickr.tags.getHotList.html
[68] http://www.flickr.com/services/api/flickr.tags.getListPhoto.html
[69] http://www.flickr.com/services/api/flickr.tags.getListUser.html
[70] http://www.flickr.com/services/api/flickr.tags.getListUserPopular.html
[71] http://www.flickr.com/services/api/flickr.tags.getListUserRaw.html

- flickr.tags.getRelated[72]: returns a list of tags 'related' to a given tag, based on clustered usage analysis

test

- flickr.test.echo[73]: a testing method which echo's all parameters back in the response
- flickr.test.login[74]: a testing method which checks if the caller is logged and then returns his/her username
- flickr.test.null[75]: null test

urls

- flickr.urls.getGroup[76]: returns a url pointing to a group's page
- flickr.urls.getUserPhotos[77]: returns a url pointing to a user's photos
- flickr.urls.getUserProfile[78]: returns a url pointing to a user's profile
- flickr.urls.lookupGallery[79]: returns a gallery info
- flickr.urls.lookupGroup[80]: returns a group NSID, given the url pointing to a group's page or photo pool
- flickr.urls.lookupUser[81]: returns a user NSID, given the url pointing to a user's photos or profile

API Examples

In this section we show some examples of the API. In each example there are two blocks, one containing the request and other with the response.

Getting Photo Information

[72] `http://www.flickr.com/services/api/flickr.tags.getRelated.html`
[73] `http://www.flickr.com/services/api/flickr.test.echo.html`
[74] `http://www.flickr.com/services/api/flickr.test.login.html`
[75] `http://www.flickr.com/services/api/flickr.test.null.html`
[76] `http://www.flickr.com/services/api/flickr.urls.getGroup.html`
[77] `http://www.flickr.com/services/api/flickr.urls.getUserPhotos.html`
[78] `http://www.flickr.com/services/api/flickr.urls.getUserProfile.html`
[79] `http://www.flickr.com/services/api/flickr.urls.lookupGallery.html`
[80] `http://www.flickr.com/services/api/flickr.urls.lookupGroup.html`
[81] `http://www.flickr.com/services/api/flickr.urls.lookupUser.html`

```
http://api.flickr.com/services/rest/?method=flickr.photos.getInfo
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&photo_id=120292580

/**
  api_key (required): api key
  photo_id (required): photo id
  secret (optional): if correct the permission check is not performed
**/
```

```
<rsp stat="ok">
<photo id="120292580" secret="fca8637ab6" server="47" farm="1"
    dateuploaded="1143731314" isfavorite="0" license="5" rotation="0"
    originalsecret="fca8637ab6" originalformat="png" views="10163" media="photo">
    <owner nsid="67526850@N00" username="Alex Osterwalder" realname="Alexander
        Osterwalder" location="Genvea, Switzerland" />
    <title>Web2.0 Business Model Characteristics</title>
    <description>The outcome of a short late-night brainstorming session on the
        characteristics of a Web2.0 business model. The reflections are based on what
        I write at my (...)</description> <visibility ispublic="1" isfriend="0"
        isfamily="0" />
    <dates posted="1143731314" taken="2006-03-30 22:08:34" takengranularity="0"
        lastupdate="1202967678" />
    <editability cancomment="0" canaddmeta="0" />
    <usage candownload="1" canblog="0" canprint="0" canshare="0" />
    <comments>2</comments>
    <notes />
    <tags>
        <tag id="2017715-120292580-380852" author="67526850@N00" raw="business model"
            machine_tag="0">businessmodel</tag>
        <tag id="2017715-120292580-11227" author="67526850@N00" raw="web2.0"
            machine_tag="0">web20</tag>
        <tag id="2017715-120292580-2956157" author="67526850@N00" raw="business model
            innovation" machine_tag="0">businessmodelinnovation</tag>
        <tag id="2017715-120292580-2956158" author="67526850@N00" raw="business model
            ontology" machine_tag="0">businessmodelontology</tag>
        <tag id="2017715-120292580-2109580" author="67526850@N00" raw="osterwalder"
            machine_tag="0">osterwalder</tag>
    </tags>
    <urls>
        <url type="photopage">
          http://www.flickr.com/photos/osterwalder/120292580/
        </url>
    </urls>
</photo>
</rsp>
```

Photo Galleries Information

```
http://api.flickr.com/services/rest/?method=flickr.galleries.getListForPhoto
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&photo_id=2080242123&per_page=5

/**
  api_key (required): api key
  photo_id (required): photo id
  per_page (optional): number of galleries in result. default 100 - max 500
  page (optional): the page of the result. default 1
**/
```

```
<rsp stat="ok">
<galleries total="19" page="1" pages="4" per_page="5" photo_id="2080242123">
    <gallery id="25845796-72157624218638653"
        url="http://www.flickr.com/photos/25891118@N06/galleries/72157624218638653"
        owner="25891118@N06" primary_photo_id="2080242123" date_create="1277335620"
        date_update="1278257567" count_photos="4" count_videos="0"
        primary_photo_server="2209" primary_photo_farm="3"
        primary_photo_secret="55c93c007d"> <title>Nature</title>
        <description />
    </gallery>
    <gallery id="51165959-72157624161029199"
        url="http://www.flickr.com/photos/fer10/galleries/72157624161029199"
        owner="51198098@N04" primary_photo_id="4691319257" date_create="1276662136"
        date_update="1276906396" count_photos="14" count_videos="0"
        primary_photo_server="4007" primary_photo_farm="5"
        primary_photo_secret="f411f6ba4e">
        <title>Bellezas</title>
        <description>Expectacular</description>
    </gallery>
    <gallery id="1344252-72157623919289749"
        url="http://www.flickr.com/photos/68196577@N00/galleries/72157623919289749"
        owner="68196577@N00" primary_photo_id="4536144000" date_create="1273628141"
        date_update="1278099389" count_photos="15" count_videos="0"
        primary_photo_server="4032" primary_photo_farm="5"
        primary_photo_secret="49a59c20ff">
        <title>WHAT?!</title>
        <description />
    </gallery>
    <gallery id="15624814-72157623792903801"
        url="http://www.flickr.com/photos/15646144@N07/galleries/72157623792903801"
        owner="15646144@N07" primary_photo_id="2080242123" date_create="1272049888"
        date_update="1272052268" count_photos="1" count_videos="0"
        primary_photo_server="2209" primary_photo_farm="3"
        primary_photo_secret="55c93c007d">
        <title>For Mobile</title>
```

```
        <description />
    </gallery>
    <gallery id="20945644-72157623529610741"
        url="http://www.flickr.com/photos/20966974@N07/galleries/72157623529610741"
        owner="20966974@N07" primary_photo_id="2080242123" date_create="1269051584"
        date_update="1269051616" count_photos="1" count_videos="0"
        primary_photo_server="2209" primary_photo_farm="3"
        primary_photo_secret="55c93c007d">
        <title>Fall</title>
        <description />
    </gallery>
</galleries>
</rsp>
```

Public List of User Contacts

```
http://api.flickr.com/services/rest/?method=flickr.contacts.getPublicList
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&user_id=67526850@N00

/**
  api_key (required): api key
  user_id (required): photo id
  per_page (optional): number of result items. default 1000 - max 1000
  page (optional): the page of the result. default 1
**/
```

```
<rsp stat="ok">
<contacts page="1" pages="1" per_page="1000" perpage="1000" total="19">
    <contact nsid="28404674@N00" username="(^_^) wellwin" iconserver="41"
        iconfarm="1" ignored="0" />
    <contact nsid="38075047@N00" username="dgray_xplane" iconserver="32" iconfarm="1"
        ignored="0" />
    <contact nsid="27009262@N00" username="Dion Hinchcliffe" iconserver="7"
        iconfarm="1" ignored="0" />
    <contact nsid="80095026@N00" username="fanstone" iconserver="53" iconfarm="1"
        ignored="0" />
    <contact nsid="80739942@N00" username="keystone1111" iconserver="3128"
        iconfarm="4" ignored="0" />
    <contact nsid="46752978@N00" username="kisco" iconserver="34" iconfarm="1"
        ignored="0" />
    <contact nsid="92455005@N00" username="laurenthaug" iconserver="4" iconfarm="1"
        ignored="0" />
    <contact nsid="89529267@N00" username="LynetteRadio" iconserver="40" iconfarm="1"
        ignored="0" />
    <contact nsid="92518516@N00" username="modahome" iconserver="120" iconfarm="1"
        ignored="0" />
```

```
    <contact nsid="73314839@N00" username="Naaaif" iconserver="110" iconfarm="1"
        ignored="0" />
    <contact nsid="20056291@N00" username="nicolasnova" iconserver="4059"
        iconfarm="5" ignored="0" />
    <contact nsid="21296916@N03" username="Paul Hughes: Design Thinking"
        iconserver="2186" iconfarm="3" ignored="0" />
    <contact nsid="54412022@N00" username="publicmind" iconserver="17" iconfarm="1"
        ignored="0" />
    <contact nsid="46557603@N00" username="Ralf Beuker" iconserver="2386"
        iconfarm="3" ignored="0" />
    <contact nsid="49147885@N00" username="squidish" iconserver="21" iconfarm="1"
        ignored="0" />
    <contact nsid="36112663@N00" username="tangyg" iconserver="0" iconfarm="0"
        ignored="0" />
    <contact nsid="34862120@N08" username="think.smith" iconserver="3126"
        iconfarm="4" ignored="0" />
</contacts>
</rsp>
```

Public Photos of User Contacts

```
http://api.flickr.com/services/rest/?method=flickr.photos.getContactsPublicPhotos
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&user_id=67526850@N00

/**
  api_key (required): api key
  user_id (required): user id
  count (optional): number of photos. default 10 - max 50. only used if without
      parameter 'single_photo'
  just_friends (optional): if 1 returns only photos of family and friends
  single_photo (optional): only returns the last photo of each contact
  include_self (optional): if 1 includes photos of the user (specified in 'user_id')
  extras (optional): extra information (license, date_upload, date_taken, owner_name,
      icon_server, original_format, last_update)
**/
```

```
<rsp stat="ok">
<photo id="120292580" secret="fca8637ab6" server="47" farm="1"
    dateuploaded="1143731314" isfavorite="0" license="5" rotation="0"
    originalsecret="fca8637ab6" originalformat="png" views="10163" media="photo">
    <owner nsid="67526850@N00" username="Alex Osterwalder" realname="Alexander
        Osterwalder" location="Genvea, Switzerland" />
    <title>Web2.0 Business Model Characteristics</title>
    <description>The outcome of a short late-night brainstorming session on the
        characteristics of a Web2.0 business model. The reflections are based on what
```

```
        I write at my &lt;a
        href=&quot;http://business-model-design.blogspot.com&quot;&gt;business model
        design blog&lt;/a&gt;</description>
    <visibility ispublic="1" isfriend="0" isfamily="0" />
    <dates posted="1143731314" taken="2006-03-30 22:08:34" takengranularity="0"
        lastupdate="1202967678" />
    <editability cancomment="0" canaddmeta="0" />
    <usage candownload="1" canblog="0" canprint="0" canshare="0" />
    <comments>2</comments>
    <notes />
    <tags>
        <tag id="2017715-120292580-380852" author="67526850@N00" raw="business model"
            machine_tag="0">businessmodel</tag>
        <tag id="2017715-120292580-11227" author="67526850@N00" raw="web2.0"
            machine_tag="0">web20</tag>
        <tag id="2017715-120292580-2956157" author="67526850@N00" raw="business model
            innovation" machine_tag="0">businessmodelinnovation</tag>
        <tag id="2017715-120292580-2956158" author="67526850@N00" raw="business model
            ontology" machine_tag="0">businessmodelontology</tag>
        <tag id="2017715-120292580-2109580" author="67526850@N00" raw="osterwalder"
            machine_tag="0">osterwalder</tag>
    </tags>
    <urls>
        <url type="photopage">
          http://www.flickr.com/photos/osterwalder/120292580/
        </url>
    </urls>
</photo>
</rsp>
```

Latest Public Photos

```
http://api.flickr.com/services/rest/?method=flickr.photos.getRecent
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&per_page=10

/**
  api_key (required): api key
  extras (optional): extra information (description, license, date_upload,
      date_taken, owner_name, icon_server, original_format, las_update, geo, tags,
      machine_tags, o_dims, views, media, path_alias, url_sq, url_t, url_s, url_m,
      url_o)
  per_page (optional): number of result items. default 100 - max 500
  page (optional): page of the result. default 1
**/
```

```
<rsp stat="ok">
<photos page="1" pages="100" perpage="10" total="1000">
    <photo id="4771876711" owner="30428372@N05" secret="94ef60dcfa" server="4098"
        farm="5" title="Picture0136" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4771876775" owner="10047346@N00" secret="5cd4161426" server="4122"
        farm="5" title="Aberdeenshire" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4771876795" owner="29221546@N07" secret="4b06f6eb86" server="4080"
        farm="5" title="P1030381" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4771876809" owner="89235411@N00" secret="30b600dcd9" server="4123"
        farm="5" title="P1000277" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4771876827" owner="51617540@N07" secret="085439dc86" server="4095"
        farm="5" title="01winery1" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4772515510" owner="58562067@N00" secret="d1e84f605b" server="4134"
        farm="5" title="IMG_5164" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4772515590" owner="50585245@N06" secret="db914e1f92" server="4079"
        farm="5" title="DSC00558" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4772515614" owner="10887912@N03" secret="cc143872a3" server="4116"
        farm="5" title="IMG_2268" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4772515634" owner="32128624@N05" secret="f171de864e" server="4093"
        farm="5" title="DSC00216" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="4772515640" owner="73657575@N00" secret="71e526d11e" server="4118"
        farm="5" title="Therion @ GMM 2010" ispublic="1" isfriend="0" isfamily="0" />
</photos>
</rsp>
```

Most Popular Tags

```
http://api.flickr.com/services/rest/?method=flickr.tags.getHotList
&api_key=f629fbcf316fbea8611ca0b2d33f2ea7&period=week

/**
  api_key (required): api key
  period (optional): time period of result. 'day' (default) or 'week'
  count (optional): number of result items. default 20 - max 200
**/
```

```
<rsp stat="ok">
<hottags period="week" count="20">
    <tag score="100">me2mobileme2photo</tag>
    <tag score="100">canadaday2010</tag>
    <tag score="100">tdf10</tag>
    <tag score="100">japanexpo</tag>
    <tag score="100">happybirthdayamerica</tag>
    <tag score="100">animeexpo2010</tag>
    <tag score="100">zurifascht</tag>
```

```
    <tag score="100">cosfest</tag>
    <tag score="100">glasto2010</tag>
    <tag score="100">huracanalex</tag>
    <tag score="100">macysfireworks</tag>
    <tag score="100">canadadayfireworks</tag>
    <tag score="100">happy4th</tag>
    <tag score="100">peachtreeroadrace</tag>
    <tag score="100">jul10</tag>
    <tag score="100">redwhiteandboom</tag>
    <tag score="100">goodwoodfestivalofspeed2010</tag>
    <tag score="100">rondevanfrankrijk</tag>
    <tag score="100">marincountyfair</tag>
    <tag score="100">stpaulscarnival</tag>
</hottags>
</rsp>
```

A.4.2 Delicious

Delicious is a social bookmarking service on the web launched in 2003, i.e., a service where its users can save and share bookmarks (URL adresses). It is important to stress that our study was done before the acquisition of delicious by AVOS Systems. Thus, the following content might be outdated.

Authentication

It is possible to access public data from Delicious in an anonymously way, by using the web feeds (a data format to publish content) service of the system. On the other hand, in order to access private data, the requests must be authenticated by using the OAuth – an open protocol to enable an application to access end user information from a Web service. The process of authentication is described in OAuth Authorization Flow web page[82].

OAth Python Library There is a python library that supports Oauth authentication: oauth2[83].

Feeds

To access public data from Delicious, there are read-only data feeds[84], which adopted in our study, since our work is focused on public data. The response of feed requests comes

[82] http://developer.yahoo.com/oauth/guide/oauth-auth-flow.html
[83] http://github.com/simplegeo/python-oauth2
[84] http://delicious.com/help/feeds

in two possible formats – RSS[85] and JSON[86].

Update Rate
Due to practical reasons, the Delicious system does accept update requests of the feeds very often. RSS feeds, for instance, may not be updated more than twice an hour. Requesting data more often than allowed by the system may result in HTTP 503 errors, indicating either that the requests were blocked or throttled by the servers.

Feeds Available
All feeds follow this base URL prefix:

```
http://feeds.delicious.com/v2/{format}/
```

Where the placeholder {`format`} is the feed format: `rss` or `json`.

The following parameters are accepted:

?count = {1..1000} limit the results – default (15).

?plain or ?fancy disable or enable HTML content.

?callback=js call allows the inclusion of a wrapper call. Only JSON data.

Additional placeholders used in URLs further described are:

{format} rss or json.

{username} user's login name on delicious

{tag+[tag+...+tag]} tag or intersection of tags.

{url md5} MD5 hash of a URL.

{key} security key that allows view private data.

URL Patterns for Feeds

- Recent bookmarks:

  ```
  http://feeds.delicious.com/v2/{format}/recent
  ```

- Recent bookmarks by tag:

[85] `http://en.wikipedia.org/wiki/RSS_(protocol)`
[86] `http://json.org/`

```
http://feeds.delicious.com/v2/{format}/tag/{tag[+tag+...+tag]}
```

- Bookmarks for a specific user:

```
http://feeds.delicious.com/v2/{format}/{username}
```

- Private bookmarks for a specific user:

```
http://feeds.delicious.com/v2/{format}/{username}?private={key}
```

- Bookmarks for a specific user by tag(s):

```
http://feeds.delicious.com/v2/{format}/{username}/{tag[+tag+...+tag]}
```

- Private bookmarks for a specific user by tag(s):

```
http://feeds.delicious.com/v2/{format}/{username}/{tag[+tag+...+tag]}
?private={key}
```

- Public summary information about a user:

```
http://feeds.delicious.com/v2/{format}/userinfo/{username}
```

- A list of all public tags for a user:

```
http://feeds.delicious.com/v2/{format}/tags/{username}
```

- A list of related public tags for a user/tag combination:

```
http://feeds.delicious.com/v2/{format}/tags/{username}/{tag[+tag+...+tag]}
```

- Bookmarks from subscriptions of a given user:

```
http://feeds.delicious.com/v2/{format}/subscriptions/{username}
```

- Private feed for of third-party suggested bookmarks for a given user:

```
http://feeds.delicious.com/v2/{format}/inbox/{username}?private={key}
```

- Bookmarks from network members of a given user:

```
http://feeds.delicious.com/v2/{format}/network/{username}
```

- Bookmarks from network members of a given user by tag:

```
http://feeds.delicious.com/v2/{format}/network/{username}/{tag[+tag+...+tag]}
```

- A list of network members of a given user:

```
http://feeds.delicious.com/v2/{format}/networkmembers/{username}
```

- Recent bookmarks for a URL:

```
http://feeds.delicious.com/v2/{format}/url/{url md5}
```

- Summary information about a URL:

```
http://feeds.delicious.com/v2/json/urlinfo/{url md5}
```

A.5 Conclusion

Folksonomies maintained by web systems are an important source of information. In order to access and manage them these web systems usually provide web APIs. As a partial result of a research we are conducting concerning folksonomies, we have developed the tool presented here, which can access, retrieve and store data from folksonomies.

In this paper we showed the tool we developed, detailing the strategy to access folksonomy based systems. We also showed our work of a unified tag database, derived from the comparison of related work and models adopted by web systems.

Bibliography

[1] Delicious. `http://www.delicious.com`. Retrieved on November, 2011.

[2] Flickr. `http://www.flickr.com`. Retrieved on November, 2011.

[3] Hugo Alves and André Santanchè. Folksonomized Ontologies - from social to formal. In *XVII Brazilian Symposium on Multimedia and the Web*, pages 58–65, 2011.

[4] Hugo Alves and André Santanchè. Folksonomized Ontologies and the 3E Steps Technique to Suport Ontology Evolvement. *Journal of Web Semantics*, 2012. Submitted.

[5] Hugo Alves and André Santanchè. Formal Aspects of Social Ontologies and Folksonomized Ontologies. In *4th International Workshop on Semantic Web Information Management*, 2012. Submitted.

[6] Hugo Alves and André Santanchè. Retrieving and Storing Data from Folksonomies. Technical Report IC-12-15, Institute of Computing – UNICAMP, `http://www.ic.unicamp.br/~reltech/2012/12-15.pdf`, May 2012.

[7] Sofia Angeletou. Semantic enrichment of folksonomy tagspaces. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 889–894, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Benjamin Huynh Kim Bang, Eric Dané, and Monique Grandbastien. Merging Semantic and Participative Approaches for Organising Teachers' Documents . In Joseph Luca and Edgar R. Weippl, editors, *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, pages 4959–4966, Vienna, Austria, June 2008. AACE.

[9] Nicolas Garcia Belmonte. `http://thejit.org/`, 2011. Retrieved on November, 2011.

[10] Iván Cantador, Ioannis Konstas, and Joemon M. Jose. Categorising social tags to improve folksonomy-based recommendations. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9:1–15, March 2011.

[11] Ciro Cattuto, Dominik Benz, Andreas Hotho, and Gerd Stumme. Semantic Grounding of Tag Relatedness in Social Bookmarking Systems. In *The Semantic Web – ISWC 2008, Proc.Intl. Semantic Web Conference 2008*, volume 5318 of *LNAI*, pages 615–631, Heidelberg, 2008. Springer.

[12] Ying Ding and Schubert Foo. Ontology Research and Development Part 2 - A Review of Ontology Mapping and Evolving. *Journal of Information Science*, 28(5):375–388, 2002.

[13] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[14] Thomas Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[15] Thomas Gruber. Ontology of Folksonomy: A Mash-up of Apples and Oranges. *International Journal on Semantic Web & Information Systems*, 3(2):1–11, 2007.

[16] Paul Heymann and Hector Garcia-Molina. Collaborative Creation of Communal Hierarchical Taxonomies in Social Tagging Systems. Technical Report 2006-10, Computer Science Department, Standford University, April 2006.

[17] J. Jiang and D. Conrath. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In *Proceedings of the International Conference Research on Computational Linguistics*, pages 19–33, Taiwan, 1997.

[18] Hak Lae Kim, Simon Scerri, John G. Breslin, Stefan Decker, and Hong Gee Kim. The state of the art in tag ontologies: a semantic model for tagging and folksonomies. In *DCMI '08: Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 128–137. Dublin Core Metadata Initiative, 2008.

[19] Konstantinos Kotis, Panos Alexopoulos, and Andreas Papasalouros. Towards a Framework for Trusting the Automated Learning of Social Ontologies. In Yaxin Bi and Mary-Anne Williams, editors, *Knowledge Science, Engineering and Management*, volume 6291 of *Lecture Notes in Computer Science*, pages 388–399. Springer Berlin / Heidelberg, 2010.

[20] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF). Model and Syntax Specification. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, 1999. Retrieved on November, 2011.

[21] Vladimir Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[22] Freddy Limpens, Fabien Gandon, and Michel Buffa. Helping online communities to semantically enrich folksonomies. *Web Science Conference*, 2010.

[23] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[24] Alexander Maedche, Er Maedche, and Raphael Volz. The Ontology Extraction Maintenance Framework Text-To-Onto. In *Proceedings of the ICDM'01 Workshop on Integrating Data Mining and Knowledge Management*, 2001.

[25] Alexander Maedche and Stefen Staab. Semi-Automatic Engineering of Ontologies from Text. In *Proceedings of the 12th Internal Conference on Software and Knowledge Engineering. Chicago, USA, July, 5-7, 2000*. KSI, 2000.

[26] Adam Mathes. Folksonomies - cooperative classification and communication through shared metadata. `http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html`, December 2004. Retrieved on April, 2011.

[27] Peter Mika. Ontologies Are Us: A Unified Model of Social Networks and Semantics. *Journal of Web Semantics*, 5(1):5–15, 2007.

[28] George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[29] Michael G. Noll. `https://github.com/quuxlabs/DeliciousAPI`, 2008. Retrieved on November, 2011.

[30] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.

[31] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In C. Raymond Perrault, editor, *Proceedings of the 14th IJCAI*, pages 448–453, Montréal (Canada), 1995.

[32] Sheldon M. Ross. *A First Course in Probability*. Macmillan, 1976.

[33] Vincent Schickel-Zuber and Boi Faltings. Inferring user's preferences using ontologies. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, pages 1413–1418. AAAI Press, 2006.

[34] Clay Shirky. Ontology is Overrated: Categories, Links, and Tags. `http://www.shirky.com/writings/ontology_overrated.html`, 2005. Retrieved on May, 2011.

[35] Lucia Specia and Enrico Motta. Integrating Folksonomies with the Semantic Web. In *Proceedings of the European Semantic Web Conference (ESWC2007)*, volume 4519 of *LNCS*, pages 624–639, Berlin Heidelberg, Germany, July 2007. Springer-Verlag.

[36] Gerd Stumme, Rudi Studer, and York Sure. Towards an Order-Theoretical Foundation for Maintaining and Merging Ontologies. In *Verbundtagung Wirtschaftsinformatik 2000. F. Bodendorfand M. Grauer (eds.)*, pages 136–149. Shakerand Aachen, 2000.

[37] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[38] Michael Sussna. Word Sense Disambiguation for Free-text Indexing Using a Massive Semantic Network. In *Proceedings of the second international conference on Information and knowledge management*, CIKM '93, pages 67–74, New York, NY, USA, 1993. ACM.

[39] Vlad Tanasescu and Olga Streibel. Extreme Tagging: Emergent Semantics through the Tagging of Tags. In Peter Haase, Andreas Hotho, Luke Chen, Ernie Ong, and Philippe Cudre Mauroux, editors, *Proceedings of the International Workshop on Emergent Semantics and Ontology Evolution (ESOE2007) at ISWC/ASWC2007, Busan, South Korea*, November 2007.

[40] Maurizio Tesconi, Francesco Ronzano, Andrea Marchetti, and Salvatore Minutoli. Semantify del.icio.us: Automatically Turn your Tags into Senses. In *First Workshop on Social Data on the Web (SDoW2008)*, 2008.

[41] Mike Uschold and Michael Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Sharing and Review*, 11(2):93–136, June 1996.

[42] Céline Van Damme, Martin Hepp, and Katharina Siorpaes. FolksOntology: An Integrated Approach for Turning Folksonomies into Ontologies. In *Proceedings of the ESWC Workshop "Bridging the Gap between Semantic Web and Web 2.0"*, pages 57 – 70. Springer, 2007.

[43] Thomas Vander Wal. Folksonomy. `http://vanderwal.net/folksonomy.html`, 2007. Retrieved on April, 2011.

[44] C. Welty, F. Lehmann, G. Gruninger, and M. Uschold. Ontology: Expert Systems All Over Again?, 1999.

[45] Zhibiao Wu and Martha Palmer. Verb Semantics And Lexical Selection. In *Proc. of the 32nd Annual Meeting on Association for Computational Linguistics*, pages 133–138, 1994.