Department of Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava

Tagging system for the blog.matfyz.sk portal

(Diploma thesis)

Juraj Ďuďák

Thesis advisor: Mgr. Juraj Frank

Bratislava 2011

62cf4fe1-369f-4f33-9b3d-0973fc369700

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Juraj Ďuďák
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický

Názov : Tagging system for the blog.matfyz.sk portal

Cieľ : The goal is to design and implement a tagging system that will be used on the blog.matfyz.sk portal. Specific goals: - Analyse the requirements for a modern tagging system. - Develop tag-based navigation for blog.matfyz.sk considering all aspects of this portal. - Create an administration panel including tag statistics. - Expand old tagging model to allow all users to tag content. - Implement a portlet for tag suggestions.

Anotácia : Tagging is a very effective and popular way of categorizing web content. However, there are some common problems with tagging. The goal of this thesis is to design and implement a new tagging system for the matfyz.sk portal which will eliminate these problems and enhance the overall user experience with the portal.

Vedúci : Mgr. Juraj Frank
Konzultant : RNDr. Martin Homola, PhD.

Dátum zadania: 21.10.2010

Dátum schválenia: 28.10.2010

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

študent

vedúci práce,
konzultant práce

Dátum potvrdenia finálnej verzie práce, súhlas s jej odovzdaním (vrátane spôsobu sprístupnenia)

vedúci práce, konzultant práce

By this I declare that I wrote this diploma thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

. . . . . .

I would like to thank my supervisor and consultant for many suggestions and corrections, my family for their support, my friends who helped me stay sane and all the players of my tagging game. Special thanks and congratulations to the best player, Dada Halásová.

Contents

Abstract

Collaborative tagging is widely used on the web and is well known to web developers, but mainly as a supportive way to categorise content and provide minor navigation. We describe that tags provide much more information and can be used to create full-feature navigation. When designing relational tagbrowser for blog.matfyz.sk, we experienced many difficulties with old tagging system design, so we designed a new system, with all the properties of modern tagging tools, but also suitable for our portal. Expected properties of the system were extensively studied and statistics were made to ensure suitability of our system. Tagging system we created is very modular, with modules capable of creating many sorts of tag-based navigation and dealing with problems related to free-nature of tags. We also experimented with ways of implementing tag suggestions, to aid users of portal to use more tags. Resulting system is faster and easier to use than the old one, also provides more possible navigation features.

Keywords: tag, tagging system, web application, tagcloud, folksonomy

Chapter 1

Introduction

Tagging is a process is which we assign tags (keywords) to pieces of information. Tag is a descriptive term that helps to define information value of object it is assigned to. Nowadays, tagging is very popular, since it is strongly associated with Web 2.0 and collaborative web applications, where large amounts of data created by users have to be categorised. Using tagging, categorisation and resource management is done by users themselves. Tagging is also very important for machine processing; tags generated by users often describe basic properties and semantics of the object. Tags create so called folksonomy [Mat04], a user generated informal taxonomy without strict hierarchical structure. Folksonomy is defining relations, much like ontology. Ontology is a formal description of things, their types, and relations in some domain of interest [Sow00].

The main goal of tagging is resource management. We need content to be easily searchable, processed by machine, and most of all browsable. Many times users do not really know what they are looking for, in which case tags can point them to category containing resources they are interested in (e.g. some "recipe").

Tagging is widely used on many different web portals, they form tagclouds

and are used for simple navigation and searching. Tagcloud is a list of tags that visualises tag importance and other data, for example tag relations. But tags provide much more information and many interesting visualisations can be done.

1.1 Motivation

There are many papers on tagging [Mik05], [SAG06], [DL07], [MH07], [ZXS06], but most of them take for example Flickr(www.flickr.com) data and work with it offline, to demonstrate some techniques. Our goal was to develop a tagging system, that will be actually used on real web portal. During our work on relational tag browser for blog.matfyz.sk [ak09], we experienced many problems associated with old tagging system design. Mainly all the operations were too slow, and many tagclouds were precomputed a cached. Also, the system was simple in design but very complicated to really understand what it was doing. There were no IDs, tags were stored as strings with complicated matching algorithms working with them. One whitespace character makes two string different, but in case of tags, they are equal.
To be able to create relational tagcloud, we also introduced relations and shortly after we discovered recent article [Mar09] explaining many different similarity measures.

1.2 Contribution

Our goal was to create easy to use system, where many experimental technologies can be easily tested. Model of this system should come from theoretical papers on tagging, but also from practical experience gained in previous work. After extensive research about this problem we summarized all the

properties of good tagging system, but also properties specific for our main domain, blog.matfyz.sk portal. We have created very modular system that consists of two parts, one dealing with problems with free nature of tags, the other, core of our system, in responsible for generating tag based navigation. This way we are easily able to alter system parts and try new technologies. Our idea was to evaluate all the queries, sort them according to their frequency and design a system that minimizes time needed to serve most frequent queries. We also experimented with way of creating relations between words that used game similar to Google Image Labeler [Goo07]. The structure we have created is usable for tag suggestions, but with the data we provided, more research can be done.

1.3 Outline

First chapter contains details of theoretical background of our problem. Here we introduced model of tagging system and operations on this model, we described forms of tag-based navigation and finally summarized important properties of a good tagging system. Chapter 3 describes blog.matfyz.sk portal, since it is necessary to understand design of this portal, to understand our system design, presented in Chapter 4. Following chapters describe main parts of this system, CloudMaker responsible for generating tagclouds and Tagregator for dealing with tag related problems. Chapter 7 describes our tagging game and presents results of this experiment.

Chapter 2

Tagging

Tag is a non-hierarchical meta-data keyword assigned to a piece of information (resource) in order to mark its semantics, information value, of describe other personal aspects of the object. Tags are mainly created by users, in order to help themselves, or other users to categorize some information, for future reference, searching, or simply express personal opinion. The nature of tags is their non-hierarchical structure, that makes them very useful to use for example on the internet, where predefined hierarchies are almost impossible to make. We can look on an easy example of directories. Say Bob works mainly in two main languages, Java and PHP. He works in school, but also he has a job. He bought a new computer and is about to make a new directory structure. But he finds it a little bit confusing, because he has 2 possible choices (Figure 2.1). He can choose any of those, with its own pros and cons (navigation, performing batch operations). But this simple example shows that hierarchies are often questionable, and large data hierarchies hard to make. Tagging solves this problem, Bob can tag his project folders with tags "work", "school", "PHP" or "Java". Then when he wants to see for example all his school projects made in PHP, he filters all projects and selects only those with both "school" and "PHP" tags. Unluckily for Bob,

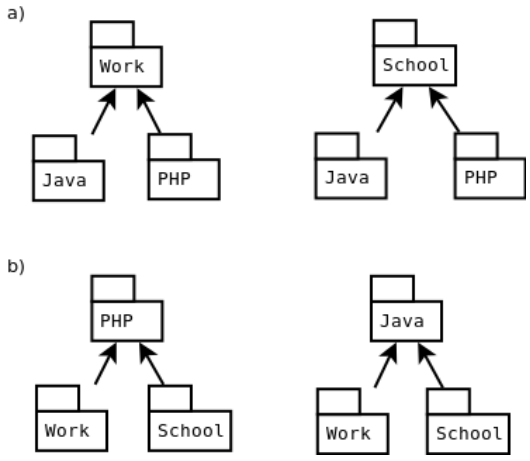


Figure 2.1: Bobs dilemma: he has to choose from 2 different hierarchies, but which one is better?

even tagging systems have their disadvantages.
Tagging is mainly collaborative process, and strongly depends on willingness of users. It is time consuming for one user to tag all the data he wants to work with, but many users can cooperate to achieve a common goal. But this also bring another well known problem of tagging; usage of natural language. Any word (in general any string) can be used as a tag, and thus users can choose different words to describe common aspect. Synonyms and also homonyms are also a part of a tagging systems and should be dealt with. In this his work, a method partially solving this problem using canonical representations is described.

2.1 Formal definition

To describe algorithms and components used in our tagging system, it is necessary to give basic definition of tagging system properties and operations on this system. Tagging system [Mik05] consists of users $U = \{u_1, u_2, ..., u_k\}$,

set of resources $R = \{r_1, r_2, .., r_m\}$ and a set of tags $T = \{t_1, t_2, .., t_m\}$. The process of tagging can therefore be defined as creating a triple $a \in A$, where A is a set of annotations $A \subseteq U \times R \times T$. This annotations create a tagging system, which can be represented as a triparted hypergraph:

$$\mathcal{S} = \langle V, E \rangle, V = U \cup R \cup T, E = A$$

This special kind of graph also describes our intuition. As we can see on

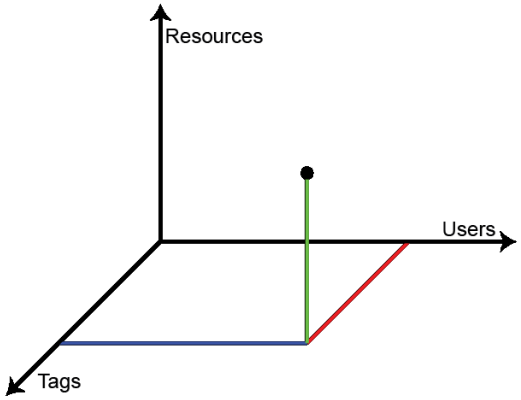


Figure 2.2: Axes represent Users, Resources and Tags. Annotation is a single point in this space.

Figure 2.2, it represents a 3-dimensional space (tagspace).

This tagging system is very simple and does not provide any information about quality of resource, user or tag. In the tagging system implemented in this thesis, author expanded the model and added strength of triplet association.

2.2 Basic operations and similarities

We will use different kinds of operations borrowed from different fields to work with our tagging system. Since our system consists of large quantities

of triplets, in order to present them to user, we need to throw away some of the information from the 3-dimensional model. There are many methods for achieving this [Mar09], but we have chosen only a few, mainly because they serve our purpose well. To demonstrate effect of our operations, we will use sample dataset of triplets, written in he table below.

user	resource	tag
Alice	XML	markup
Alice	XML	web
Bob	XML	language
Bob	Web 2.0	web
Cyril	CSS	language
Cyril	XML	language

2.2.1 Projection

Projection π is a operation from relational algebra, that is defined on relations. Suppose we have a n-ary relation, defined as follows:

$$R \subseteq C_1 \times C_2 \times \ldots \times C_n \tag{2.1}$$

We will calls sets $C_1, C_2 \ldots C_n$ columns. Projection π selects only those columns which we need:

$$\pi_{C_1,C_2 \ldots C_k}(R) = \{(c_1, c_2 \ldots c_k) \,|\exists c_{k+1}, c_{k+2} \ldots c_n \quad (c_1, c_2 \ldots c_k, c_{k+1} \ldots c_n) \in R\} \tag{2.2}$$

Using projection on our sample data, thus calculating $\pi_{resource,tag}(\mathcal{S})$, we obtain pairs of resources and tags.

resource	tag
XML	markup
XML	web
XML	language
Web 2.0	web
CSS	language

This operation is very useful for different purposes, but mostly for searching, when what we exactly need is a table of resource-tag pairs.

2.2.2 Distributional aggregation

Aggregation is an operation, which groups several data rows together and creates a single value containing their count. There are basically 2 types of aggregation, weighted by Shannon information (log odds), or very simple direct aggregation. For example to create a system-wide tagcloud, all we have to do is to aggregate across users and resources to obtain number of usage for each tag. we will denote tag weight $w(t)$.

$$w(t) =\mid \{(u, r, t) \mid (u, r, t) \in \mathcal{S}, u \in U, r \in R\} \mid \tag{2.3}$$

tag	weight
language	3
web	2
markup	1

To make a tag cloud for a resource, we only need to aggregate across the users. This time we will use log odds, to demostrate this technique. From information theory, we know that information content I of an outcome of random variable x can be enumerated using very simple formula

$$I(x) = \log \frac{1}{P(x)} = -\log P(x) \tag{2.4}$$

Where P is probability of random variable outcome x. In our case, we can derive similar formula

$$w_r(t) = -\log P(t) = -\log \frac{\mid \{(u,r,t) \mid (u,r,t) \in \mathcal{S}, u \in U\} \mid}{\mid \{(u,r,t') \mid (u,r,t') \in \mathcal{S}, u \in U, t' \in T\} \mid} \quad (2.5)$$

The final formula inside logarithm may look a bit complicated, but it is just a fraction of annotations containing selected tag over all the annotations.
Aggregation is the main and most common method for creating tag navigation. In small scale (tagcloud of individual users) linear aggregation is better choice since it preserves ratios of tag occurrences. On the other hand, in large scale, logarithmic aggregation is good choice; it reduces large differences between tag weights.

2.2.3 Similarity

To make our 3-dimensional model of tagspace complete, we have to define user-user, tag-tag and resource-resource relations. From all the possibilities [Mar09] we have chosen matching similarity, which is defined as number of common appearances. For example tags are related if they tag a common resource, and the strength of relation is based on a size of set of resources that are tagged with both tags.

$$\sigma_{tag}(t_1, t_2) = \mid \{r \mid (u_1, r, t_1), (u_2, r, t_2) \in \mathcal{S}, u_1, u_2 \in U\} \mid \quad (2.6)$$

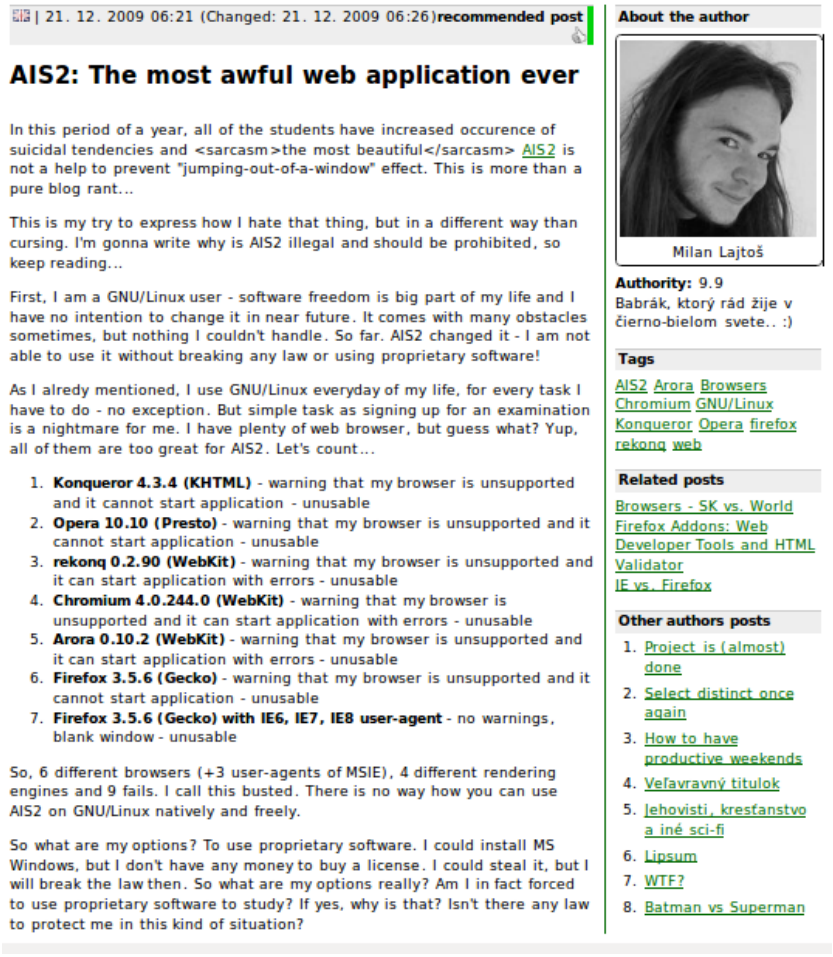
2.3 Navigation

Our model of tagspace can be viewed as a 3-dimensional space, with relations providing us distances for users, resources and tags. But when browsing for content, user can look on the whole system, or choose user/resource/tag he is interested in. This is a standard for almost every blog portal. For every point selection we are able to visualize current space point, present

interesting content and use relations to give user further options for point selection. According to current page, we can provide:

- **User page**
  - list of related users - users using related tags
  - list of resources created by selected user
  - tagcloud consisting of user added tags, tag sizes are only measured from user tags
- **Resource page**
  - list of related resources according to related tags
  - list of other resources created by author of selected resource
  - tagcloud consisting of tags assigned to pivot resources
- **Tag page**
  - tagcloud of tags related to selected tag
  - list of resources tagged with selected tag
  - list of users using selected tag

This way our user is not only browsing for resources, but he is also able to find new interesting topics and authors.
Demonstration of this principle can be seen on Figure 2.3. This image shows a page from blog.matfyz.sk portal. This is a page that is shown to users when they read article. Note that tag navigation contains tags of this article, other posts by this user, and related posts. This way we have provided many paths to find new interesting content.



| 21. 12. 2009 06:21 (Changed: 21. 12. 2009 06:26) **recommended post**

AIS2: The most awful web application ever

In this period of a year, all of the students have increased occurence of suicidal tendencies and <sarcasm>the most beautiful</sarcasm> AIS2 is not a help to prevent "jumping-out-of-a-window" effect. This is more than a pure blog rant...

This is my try to express how I hate that thing, but in a different way than cursing. I'm gonna write why is AIS2 illegal and should be prohibited, so keep reading...

First, I am a GNU/Linux user - software freedom is big part of my life and I have no intention to change it in near future. It comes with many obstacles sometimes, but nothing I couldn't handle. So far. AIS2 changed it - I am not able to use it without breaking any law or using proprietary software!

As I alredy mentioned, I use GNU/Linux everyday of my life, for every task I have to do - no exception. But simple task as signing up for an examination is a nightmare for me. I have plenty of web browser, but guess what? Yup, all of them are too great for AIS2. Let's count...

1. **Konqueror 4.3.4 (KHTML)** - warning that my browser is unsupported and it cannot start application - unusable
2. **Opera 10.10 (Presto)** - warning that my browser is unsupported and it cannot start application - unusable
3. **rekonq 0.2.90 (WebKit)** - warning that my browser is unsupported and it can start application with errors - unusable
4. **Chromium 4.0.244.0 (WebKit)** - warning that my browser is unsupported and it can start application with errors - unusable
5. **Arora 0.10.2 (WebKit)** - warning that my browser is unsupported and it can start application with errors - unusable
6. **Firefox 3.5.6 (Gecko)** - warning that my browser is unsupported and it cannot start application - unusable
7. **Firefox 3.5.6 (Gecko) with IE6, IE7, IE8 user-agent** - no warnings, blank window - unusable

So, 6 different browsers (+3 user-agents of MSIE), 4 different rendering engines and 9 fails. I call this busted. There is no way how you can use AIS2 on GNU/Linux natively and freely.

So what are my options? To use proprietary software. I could install MS Windows, but I don't have any money to buy a license. I could steal it, but I will break the law then. So what are my options really? Am I in fact forced to use proprietary software to study? If yes, why is that? Isn't there any law to protect me in this kind of situation?

About the author

Milan Lajtoš

Authority: 9.9
Babrák, ktorý rád žije v čierno-bielom svete.. :)

Tags

AIS2 Arora Browsers Chromium GNU/Linux Konqueror Opera firefox rekonq web

Related posts

Browsers - SK vs. World
Firefox Addons: Web Developer Tools and HTML Validator
IE vs. Firefox

Other authors posts

1. Project is (almost) done
2. Select distinct once again
3. How to have productive weekends
4. Veľavravný titulok
5. Jehovisti, kresťanstvo a iné sci-fi
6. Lipsum
7. WTF?
8. Batman vs Superman

Figure 2.3: Reading an article on blog.matfyz.sk portal. You can see tag based navigation to the right.

2.4 Tag Clouds

Tag clouds are visual representations of tagspace, or a point in it. Usually they are lists of words sorted alphabetically, and the importance of each tag is shown with font size or color. [MH07] These typical tagclouds are common part on many web sites and services, such as image sharing or blogging. But when we look closer, we can distinguish more types of tagclouds, depending on their visualisation.

2.4.1 Inline tagclouds

Inline tagclouds are the most simple and oldest kind of a cloud. They are only a list of words, with a strength assigned to each of this words, to represent importance.

On Figure 2.4 we can the effect of aggregation type on tag clouds. Both clouds are generated using same data. Cloud on the left was created using linear distributional aggregation, cloud on the left uses logarithmic distributional aggregation. We can see that usage of logarithms help to make differences smaller. As we will further explain, tag distribution is not linear. To create such a cloud, we need to use aggregation of any kind to produce a list of tags and their importance t. Then, it is very important to normalize these importances, so that the final tagcloud has font sizes s in a certain range. Say $s_{\min}, s_{\max}$ are minimum and maximum font sizes in our cloud. We have to find minimum and maximum importance $t_{\min}, t_{\max}$. Then, font size for each tag s_i can be calculated easily:

$$s_i = s_{\min} + \frac{(s_{\max} - s_{\min})\,(t_i - t_{\min})}{t_{\max} - t_{\min}}$$

Similarly, we can calculate a color from certain range to use as importance visualisation.

Figure 2.4: Example of inline tagclouds.

2.4.2 Tagclouds with arbitrary positions

Inline tags have one big disadvantage, each line is as high as the highest tag in it, so sometimes we want to make tagclouds with arbitrary positions. Also, we can use the extra dimension to present another aspect of tagclouds, such as relations.

Tagclouds with arbitrary positions are much more difficult to generate than ordinary inline clouds, where all the positions and line breaks are calculated by browser. In fact, finding good position of tags often leads to NP-complete problems such as rectangle packing [HK09]. Another problem is displaying these clouds in a browser. In general, browsers were made for displaying text, so these clouds have to be represented as HTML tables or divs with CSS positioning. Also, we can export the tagcloud as an image. External plugins, such as Flash are also possible choice. Another interesting option is usage of Scalable Vector Graphics(SVG) [W3C10], a modern format based

on XML, developed by W3C. SVG has native support in most browsers and there are also plugins for browsers without SVG support.

Figure 2.5: Tagcloud with arbitrary positions. This tagcloud from wikipedia shows Web 2.0 themes

2.4.3 Relational tagclouds

Special kind of tagcloud with arbitrary positions is relational tagcloud. Apart from tag importance, this type of tagcloud visualises relations between tags. These relations help users to navigate by further exploring tagspace. Relations are visualised using distance between tags, or by displaying tagcloud as a graph and using graph edges as relations between them. This graph kind of tagcloud was developed and tested as a part of bachelor thesis [ak09]. Also, for this work the model was reviewed and SVG was chosen to be the best format. The main idea behind this graph generation is using force directed layout. This algorithm uses physical modelling to iteratively find best graph layout.

Given a graph $G = (V, E)$, this approach deals vertices as electrically charged particles, and edges as springs. Algorithm simulates dynamics of this system over time. Since vertices are electrically charged, they repel each other

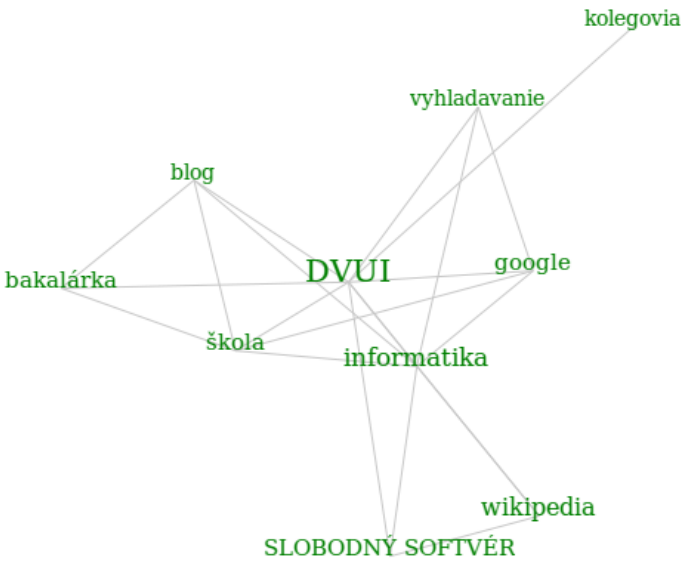


Figure 2.6: Relational tagcloud rendered using SVG.

according to simplified Coulumb law

$$F = \frac{q_1 q_2}{r^2}$$

Where q_1, q_2 are vertice charges and r is vertice distance. And with edges as springs, they are contracting and pushing nodes closer to each other according to Hooke's law

$$F = k\mathbf{x}$$

Where $\mathbf{x}$ is distance from equilibrium position, k is spring constant, in our case same constant for all edges. This equations provide systems energy, and constants have to be properly set up, so that there exists a equilibrium between these forces. In every iteration, we have to calculate all the forces

that apply to vertices:

$$F(v) = \sum_{v' \in V; v' \neq v} CoulombRepulsion(v, v') + \sum_{e \in E; v \in e} HookeAtraction(v, e)$$

where

$$CoulombRepulsion(u, v) = \frac{charge(u) \cdot charge(v)}{distance(u, v)^2}$$

$$HookeAtraction(u, e) = weight(e) \cdot (length(e) - baseLength(e))$$

This calculations result in running time $O(V^2 + E)$ for one iteration. Iter-

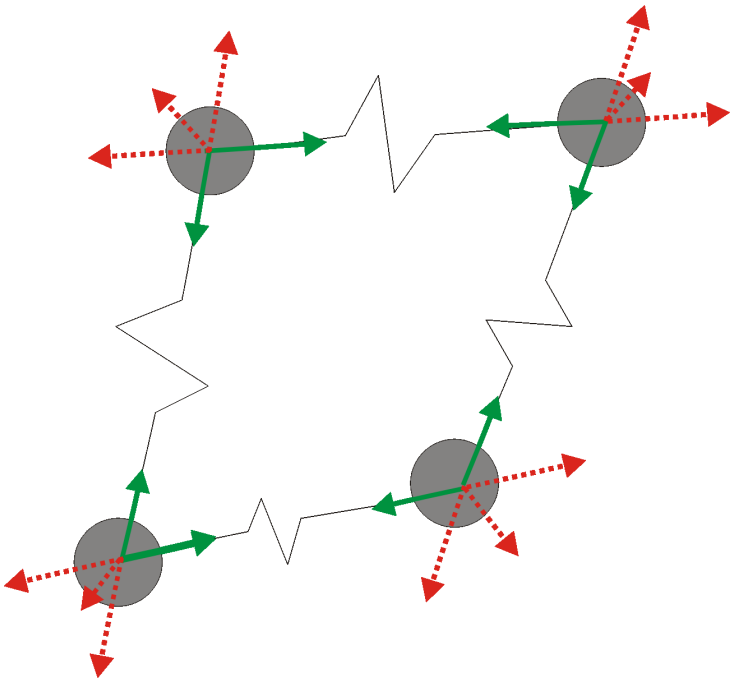


Figure 2.7: Graph layout dynamics. Red dotted lines represent repulsive forces, green lines represent spring forces.

ation means recalculating system dynamics over time, constant or variable, and moving vertices according to forces. Number of iterations depend on model, in this case, best solution was to stop simulation, when total kinetic

energy was smaller than some ϵ.
All forms of tagclouds were used in tagging system described later in this work, each one was selected for different purposes. Inline tagclouds are ideal as a side-portlets, since they do not take much space. When more sophisticated mean of navigation is needed, relation cloud is displayed.

2.5 Properties of good tagging system

Tagging, with its unhierarchical nature, is a very interesting choice for navigation, since usually websites use tree like structures. In order for tagging system to be a useful in means of navigation, it needs to follow 2 main principles, proper fragmentation and proper clusterisation.

Property 1 *Clusterisation needs users to reuse tags often, so that categories created by tags are large enough, and they form a top level navigation*

Property 2 *Fragmentation needs users to use specific tags, that are not reused often. Tag based navigation need to have small specific categories that lead users to precise category they are interested in.*

Figure 2.8 shows us categorization of resourses, assigned following tags:

resource	tags
Cat	animal, pet
Lion	animal, africa
Hyena	animal, africa, canine
Dog	animal, pet, canine
Sunflower	plant

As we can see, categories overlap and thus they can not be represented by simple tree-like structure. But there are larger and smaller categories. Tags like “animal” that cover a large set of concepts are useful to aim users to

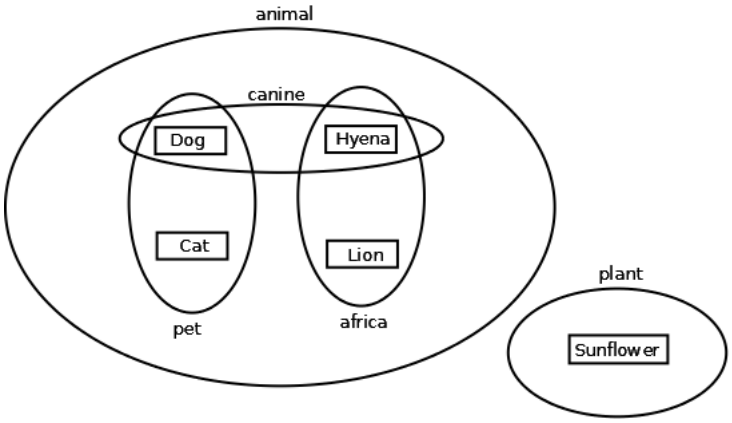


Figure 2.8: Clustering and fragmentation example.

topics they are interested in. Tag "animal" defines the largest category of resources, that are somehow related and in a tagcloud, this tag would be the larger than others, what is exactly what we are want to achieve, top-level navigation with general tags. Smaller categories, can then guide user to very specific categories using relational navigation. Further on, we will focus on tag suggestion system, that will try to follow both of these principles and suggest both tags from the folksonomy created by tags used within tagging system, but also tags from external sources, so that users does not always use the same tags, creating large categories without internal fragmentation.

Chapter 3

Blog portal

To test and demonstrate algorithms associated with tagging, a whole new tagging system was implemented on blog.matfyz.sk portal. Blog.matfyz.sk is a school-project that is being developed and administered by students and staff of Faculty. This portal is available for anyone, users only need to register or use openID account to write articles. Moreover, several courses use blogging and this portal as a supportive tool for teaching, therefore a significant number of blog articles is created by students and there is even a course in which students write their own blog layout using XSLT.

3.1 Ranking

Almost every blog portal uses ranking mechanisms to help users find good content. An easy way is to take number of views as a quality measure, but this favours authors that write many articles, even with lower quality, since there is no way to say that article is bad, and it takes a longer time to distinguish between good and bad. Another problem is sharing the article as much as possible, so that users go to article and raise its rank without even reading it. Very common way is to use scoring system, which allows

users to evaluate an article they have read. There are different algorithms [Koh08] to calculate scores of articles and also score of users - so called karma. Portal uses Eigenrumor algorithm [TN06], which is based on Pagerank and calculates both user karma and article score.

3.2 Technology overview

Portal is built on PHP [Gro] scripting language, extensively using object-oriented programming. Mainly for academic reasons, many XML based technologies are used. Data is stored in Native XML Database Sedna [sed07], which is a open-source project that is still under development, and data from this portal help developers fix bugs. PHP is used to generate data, that is then processed by XSL to produce final output. Furthermore, usage of XSL allows users of webdesign course to make their own stylesheets. Whole system is very modular and uses modules (pluggable components) to generate both data and output style.

Part of the system responsible for calculating user and article rank is written in C, to make it as fast as possible. Usage of javascript or other technologies that are not fully supported is limited, and there is always fallback for users with browser without support of given technology.

3.3 Portal architecture

Since many students write their thesis and otherwise work on developing the portal, it was necessary to rewrite blogportal code to make it more modular and easy to manage. This reconstruction was done as a part of thesis [Rej10]. New system has a horizontal separation, and thus there are no “super” classes responsible for too many functions of the portal.
The system’s core is a class called Controler, with only one purpose: gather

data from the modules and display them to users. Controler has a list of many different modules registered to it. Special kind of modules that are used in this design are called portlets. Portlet contains code logic for several use cases and provides interface for basic operations required by core (mainly for output generation) [Rej10]. Each portlet implements interface called Portlet, with 4 methods, all given the same set of parameters containing GET and POST data. These required methods are:

- **init** - used for initializing all the necessary parts of the portlet, and processing input
- **getXML** - this method is probably the most important in portlet. It returns XML data that is all gathered by Controler. Typically, it is divided into cases according to current page and provides all the data needed from this portlet.
- **getXSLList** - this method provides XSL stylesheets needed to present data returned from getXML. This method returns an array of stylesheets. We need to make sure that our XSL template matches exactly the XML we need. This is solved by wrapping the XML to "module" element with attribute name set to portlets name. XSL contains template that matches exactly this node and everything works as planned.
- **getXSLParams** - apart from XSL stylesheet list, we can define parameters that will be given to XSL processor during transformation

Controler creates XML by gathering all the XML data from the portlets, than loads all the necessary XSL stylesheets, sets up the processor and simply writes the output of the transformation to user (Figure 3.1). Each part of the system such as article, tagcloud, search bar is made as a portlet. But this also brings some problems, mostly with code-sharing. In our case, this problem was solved with another software engineering pattern called Singleton, as we

will further describe in Chapter 4. The goal was to keep the system modular, but without making previous mistakes - writing the same code more than once.

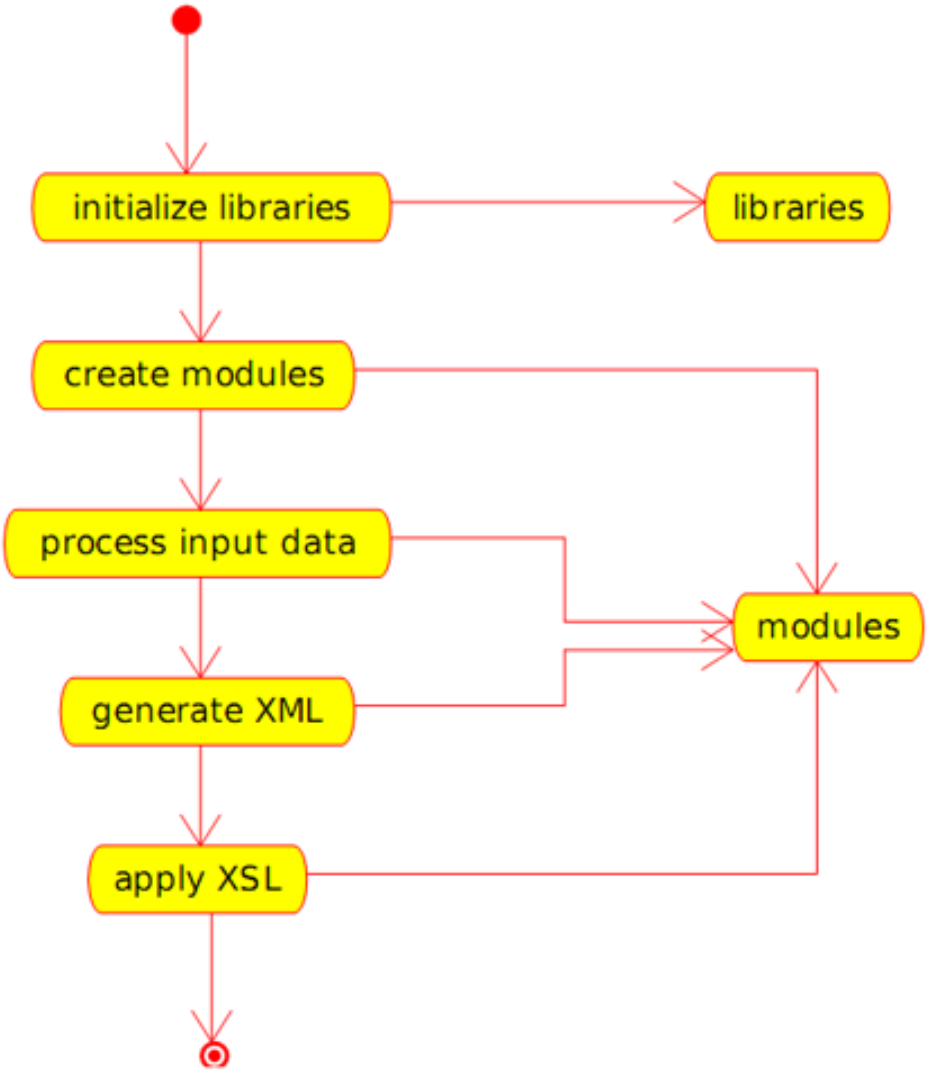


Figure 3.1: Aplication flow

3.4 Data model

Portal uses Sedna, a Native XML Database, mostly because XML is natural way to represent blog data [Koh08]. But this format was found not very effective for working with tags, so a new data format was created for this purpose. Data in Sedna is stored in aspect to user, each user node contains basic information about the user, and users blog. Blog is basically a list of articles written by user. Tags are associated with articles, which makes them accessible for alternation, but displaying a whole-portal tagcloud took a lot of time to compute and moreover calculating relations was very complicated and time consuming.
For this particular task, relational database was a better solution, as we will describe later.

3.5 Tag properties

To create a tagging system specially for this blog portal, we have to adjust basic system properties according to current state of tags. We have made following statistics on 28.1.2011 on a Slovak tags on blog.matfyz.sk portal.
To get a basic view on a size of the system, number of tags, resources are necessary. To the date, there were 3529 published posts. Authors of articles (resources) has assigned 2834 tags, so in average, 0.8 tag per post. But when we look only on articles containing at least one tag (1218 articles) we get a ratio of 2.33, what is more some users tend to add 20 or even more tags to their articles. On Figure 3.2 we can see how many atricles there are with given number of tags. Articles with 0 tags are not shown here. Some users add up to 27 tags to their own articles. This statistic could be improved by allowing all the users to tag resources, because some authors tend to forget about tags, or even assign not very suitable tags.

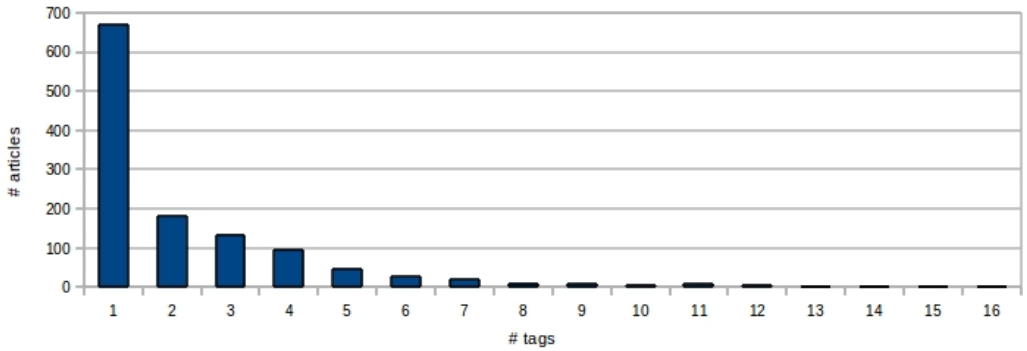


Figure 3.2: Number of posts with certain number of tags assigned to it. Up to 27 tags were assigned to an article, but only 3 articles had more than 16 tags.

There were 966 unique tags in use, so average tag reuse was about 2.93, a satisfying number, showing that tagclouds can be made on this data. To choose from different aggregation methods, we can look on a distribution of tag counts. Maximum of 291 uses was achieved for tag "matfyz" which is not very surprising. Very interesting was a exponential distribution of tags, the reason why it is good to use logarithmic aggregation methods for creating tagclouds. Exponential distribution is also important because it shows that with size of tag created categories, number of those categories decreases exponentially. This is the feature of tree-like navigations (if categories contain at least 2 subcategories, tree size grows exponentially) and good thing is that similar structure emerged in our system as well. Another interesting fact is that this usage of tags like "matfyz", that is assigned to almost a fourth of all the articles with tags has made this tags almost useless for navigation. This is exactly the reason why we need to encourage users to add new tags, not just reuse old tags. These statistics were taken in account when we were designing the tagging system. Exponential distribution of tag is the main reason for usage of logarithmic aggregation when creating portal-wide

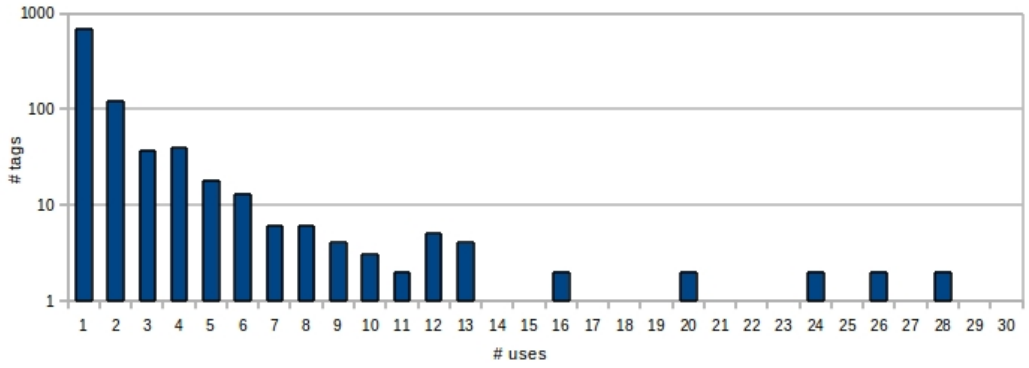


Figure 3.3: Tag reuse. y-axis is in logarithmic scale, so we can see the exponential distribution. There were even tags that were used more than 100 times, but only 9 tags were used more than 30 times.

tagclouds.

Chapter 4

Tagging system overview

Now, after we have introduced main aspects of tagging systems in general, and also properties and technological background of blog.matfyz.sk, it is time to summarize all the requirements and properties of tagging system we need:

- easy to use and fast way of generating tagclouds. As we discussed, almost every page on the portal will have a tagcloud of some kind on it. Cloud generation is the main goal of tagging system, thus we will have to make it very fast.
- tag administration. Portal is not very large in scale, so we cannot entirely rely on large quantity of tags and we will have to build administration of tags, where we can create canonical representations of tags, stop tags, and also sometimes we will have to merge tags into one.
- tag suggestion. As we discussed, we will have to somehow motivate users to use as many tags as possible and thus give us more data we can process and create even better navigation.
- keep everything as modular as possible. Since many students develop parts and modules for this portal, it is hard to keep everything working together.

- introduce tag weight. Users and articles on the portal have score/karma assigned to them, we need to have weighted tags as well. Using weighted tags we can favour users with higher tag quality.

These few items served as guidelines when designing the system. We will start explaining main parts of the system from top to bottom, also we will describe their cooperation, since system parts work as black boxes. Their internal implementation is described in chapters named after them. Firstly we will describe expanded tagging system definition.

4.1 Our tagging system definition

Common definition of tagging system given in the beginning of Chapter 2 does not serve all the requirements we made. Here, all the tag associations are taken as equal, but since our portal is rather smaller in scale, we distinguish between users, and we want to have weighted associations. This can easily be solved by expanding model from hypergraph to weighted hypergraph. All the sets and graph stay the same, we only add a function W defined on set of annotations $W : A \to \mathcal{R}^+$. This function tell us how important is this tag association and all aggregation methods and similarities are expanded from counting associations to summs of weights of associations. For simplicity, we will often use weighted triplets or weighted annotations (u, r, t, w) where $(u, r, t) \in A$ and $W((u, r, t)) = w$.

4.2 System decomposition

Tagging system that we described, has basically two logical functions. Outputting clouds and solving tag related problems. This was the main idea we used when designing system modules. We implemented 2 modules *Tagregator* and *CloudMaker*. Tagregator is responsible for dealing with tag problems,

and CloudMaker has a structure with all these problems solved and is only responsible for generating tag-based navigation. Modules were designed to work as black boxes, which was not always possible, but the main idea was to create universal classes that can be used with any project. With this in mind we added third module *Tagging* responsible for communication of 2 main modules with the portal. Tagging is a singleton [EGV94] class, which means there is always only one instance of it in processing. This was necessary, because portal uses portlets, and to generate page, we often need more that one tag-based portlet. This way all the portlets use the same instance of Tagging system. In class constructor, Tagging creates instances of both Tagregator and CloudMaker, so they also share singleton property of Tagging.

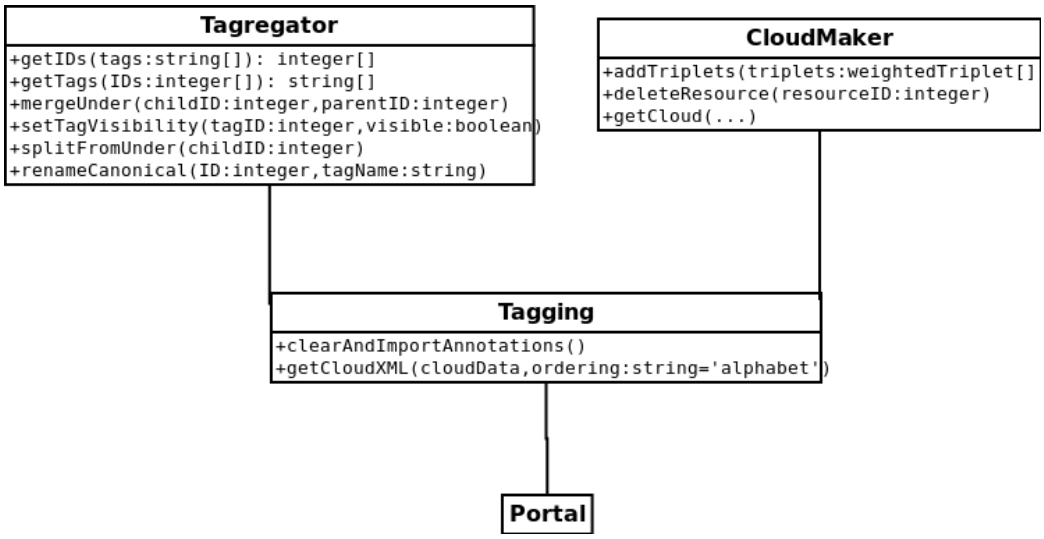


Figure 4.1: Tagging system with its 2 modules

4.3 System use-cases and interfaces

We will discuss some basic use-case algorithms and describe implementation details of whole system, with CloudMaker and Tagregator as black boxes, with their input and output methods. Queries done within tagging system can be sorted by their frequency:

1. generating clouds, filtering data
2. adding annotations, removing annotations
3. tag administration operations (merging tags, setting stopwords)

Number 1 is obvious, this is done each time user interacts with portal, number 2 has to be before number 3, because without new annotations, there is no need to do administration operations.
Our system should serve most frequent operations as fast as possible, and operations like tag administration can take up more time. Following this principle, our CloudMaker will contain cached annotations, already processed by Tagregator. After a administration operation (these are not very frequent), we will re-cache CloudMaker data. This property also makes our system a super-structure working over portals existing tagging system, since everything works basically the same way as before, we only add extra functionality. All the basic queries can be written as simple algorithms. As an example we describe most common queries.

Inputting annotations:
Input: array of weighted annotations (u, r, t, w)

1. create array of tags from the annotations and give it to Tagregators method $getIDs$, which result in array of canonical tag IDs
2. add annotations to CloudMaker using $addTriplets$, but instead of tag, use tag ID returned from Tagregator

Getting tagcloud:
Input: parameters of cloud to generate

1. use appropriate method of CloudMaker to get tag IDs and weights, in most cases *getCloud*

2. create array of IDs and give it to Tagregators method *getTags*, which result in array of canonical tags

3. make weighted tag array using matching canonical tags

Output: cloud data
Deleting or editing resources is done by CloudMakers method *deleteResource* followed by *addTriplets* if necessary.
As we said only problem is tag administration. Changing name of canonical tag is again cheap, since CloudMaker only stores IDs, but merging/stopping tags require re-caching whole annotation database from raw structures (portals xml database). Tests shown that this importing operation takes about 5 seconds, so there is no need to worry about this solution.
Reader may find interesting that we only store IDs in CloudMaker, which may look as a waste of time, we need a extra operations after generating tagcloud, but we actually save a lot. Tags in general are arbitrary strings, so storing and operating over MySQL database would be both time and space consuming. In our model, CloudMaker only stores IDs of users, resources and tags. So after CloudMaker generates cloud data, class Tagging is responsible for finding string representation and sorting result. This sorting operation is very fast, since it only works with cloud, which has a maximum number of items assigned (about 50 for usability).

4.4 Languages

Portal uses both Slovak and English language, so it was necessary to think of language variants when designing the system. We completely separated tags for both languages, and what is more, system automaticaly installs when used on new language. Portal initializes Tagging class with parameters including language as a string. When creating instances of both CloudMaker and Tagregator, we create tables for chosen language, if they do not exist. All the work is then done with tables for chosen language. This way it is very easy to integrate our system even with other possible languages.

Chapter 5

CloudMaker

Cloudmaker is a system part responsible for generating tagclouds and relations of tags/users/resources. Main reason that led to developing this class, was to only work with data that is relevant in process of tag-navigation. This class was developed in a very abstract way, so that it can be used in another projects. We do not give any limitations for users, resources or tags, they can be IDs or strings, but in our case, we decided to use IDs, since strings are not very good choice for computation.

In our extended model, annotations are weighted triplets, shortly written as tuple (u, r, t, w). These annotations represent points in discreet 3 dimensional space, with weight assigned to them. CloudMaker uses a mySQL database, with only one table, containing these annotations. This data model is very natural and easy to use. We abstract from all the other aspects of tagging system, and only look on the hypergraph edges - annotations. This natural representation of data allows us to do our computations very straightforward. What is more, when we look on this system we can see that now all parts are equal, we do not distinguish between users, resources or tags, they are all equal parts of annotation. This property is very important, because we are able to use same functionality for generating tagclouds, but also user

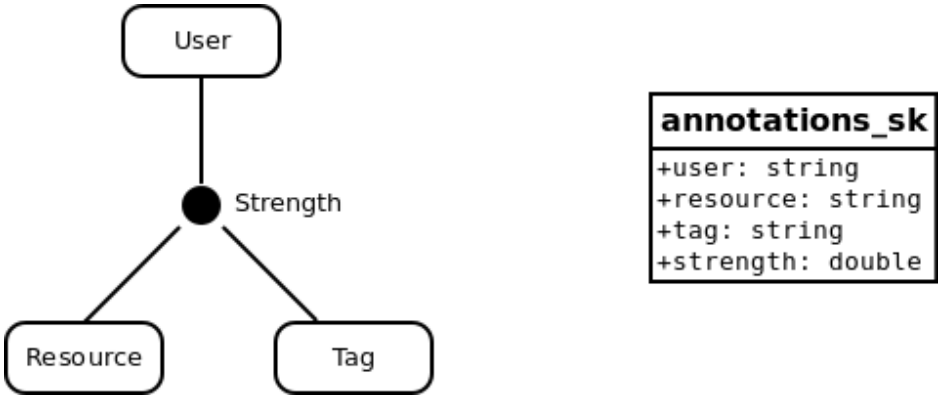


Figure 5.1: Annotation with added strength. Database simply consist of rows storing each single annotation.

"clouds" and resource "clouds".

5.1 Method getCloud

Core of this module is a method called *getCloud*. This method is responsible for retrieving weighted lists of users/resources/tags. In pseudo-code, header of the method looks like

```
getCloud (data, method, viewUser, user, resource, tag, limit)
```

This method builds up mySQL query, that gives result with almost all the properties we need. First, we will describe parameters used and then give examples of parameters setting and resulting query.

- *user*, *resource* and *tag* parameters defining filters for selecting annotations. These parameters can even be arrays, so we can select all annotations with tag/resource/user set to given values.

- *data* is either user, resource or tag. After filtering annotations, we group them along column *data*. This parameter tells the system what kind of list we need.
- *limit* says how many items at most will our list contain.

5.2 Examples

Now to demonstrate these queries, let us look on basic examples, portal-wide tagcloud and user tagcloud.
Portal-wide tagcloud is the most simple kind of tagcloud, we take all the annotations, aggregate them and then take top lets say 50 tags. Our query follows:

```
SELECT tag, SUM (strength) AS strength FROM annotations_sk
GROUP BY tag HAVING SUM (strength)>=1
ORDER BY SUM (strength) DESC LIMIT 0,50;
```

Making user tagcloud is very similar, we only add user filter:

```
SELECT tag, SUM (strength) AS strength FROM annotations_sk
WHERE user=[input user] GROUP BY tag HAVING SUM (strength)>=1
ORDER BY SUM (strength) DESC LIMIT 0,50;
```

We can easily imagine similar queries used to list all the users using given tag etc.
After retrieving list of data, aggregated lineary, we can use other aggregation method, by setting parameter *method*. We have implemented linear and logarithmic aggregation, but other methods can be easily implemented as well.
The most beautiful property of this system was found when rewriting code for relational tagbrowser. This particular task requires some very specific

data. For input tag, our task is to output graph consisting of tags with strongest relations with input tag, but we also need relations between these tags. Retrieving relations is very straightforward, when we use matching similarity. Matching similarity is defined as number of common resources. For input tag, our simple algorithm selects all resources tagged with input tag:

```
SELECT resource FROM annotations_sk WHERE tag=[input tag];
```

And then calls *getCloud* to get tags from annotations filtered by this resource list:

```
SELECT tag, SUM (strength) AS strength FROM annotations_sk
WHERE resource IN [list of resources] GROUP BY tag
HAVING SUM (strength)>=1 ORDER BY SUM (strength) DESC LIMIT 0,10;
```

Relations between chosen tags use the same algorithm, but we add tag filter to retrieve only tags which we need.

5.3 Making portlets

Simplicity of our natural database model and universality of *getCloud* method allows us to create all sorts of tag based navigation, filter users and resources. This part of our system was developed as first, and became center of whole system. Portal requires many different portlets containing tag navigation, all these portlets were created within a day of programming and we even suggested and created new portlets, for example related posts. Full feature tag navigation introduced in bachelor thesis [ak09] looked like a very uneasy task, since problems asociated with tagging rapidly slowed down development. When we assume that data stored in CloudMaker is already processed

and our only task is to visualy present this data, development becomes very fast.

Chapter 6

Tagregator

Free nature of tagging creates many problems, users think of the same keyword, but write it differently. To deal with this problem, tag administration system was developed for the portal [Fra09]. The idea of the system was to assign canonical representations to tags and ability to merge tags. We followed the main idea and designed a new module to use in our tagging system.

6.1 Module overview

Main goal of this module is to assign canonical representations to tags. Operations can be divided into 3 logical parts:

1. for given list of tags, our module returns list of canonical IDs, to be stored in CloudMaker

2. for given list of canonical IDs, our module returns tags as strings

3. module has to provide interface for tagging administration, including renaming tags, merging tags, splitting merged tags and hiding tags.

First two operations should be as fast as possible, these are performed very often. The last one is done only by administrators and only once in a while, so it can take up more time. Final design of this class uses only one table to store data (Figure 6.1). Each tag is assigned a unique ID; its string value;

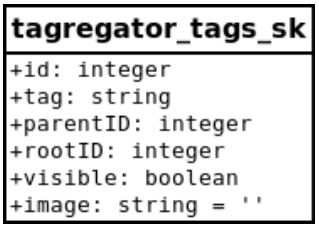


Figure 6.1: Data model of Tagregator

$parentID$, $rootID$ for merging operations; visiblity flag and for future reference we added image link. In future we would like to introduce image tags, but this also brings problems when generating cloud. Setting tag visibility is very obvious and straight forward, merging and renaming tags is a bit more complicated.

6.2 Tag merging

We have introduced tree-structures to memorize merging history. Using this structure we can easily merge tags, but also we are able to undo this operation by splitting tags and returning system to previous state. In order to create this structure, for each tag we memorize ID of a parent tag, but also ID of root tag, so that we can quickly find canonical representation of any tag.

Figure 6.2 demonstrates merging of tags . On part a) we can see situation before merge. Part b) demonstrates situation after tag merge, where new canonical representation of tag B is set to C. This operation also changes canonical representation ($rootID$) of tags that has tag B as their current canonical. Algorithms for tag merging and splitting can be easily written

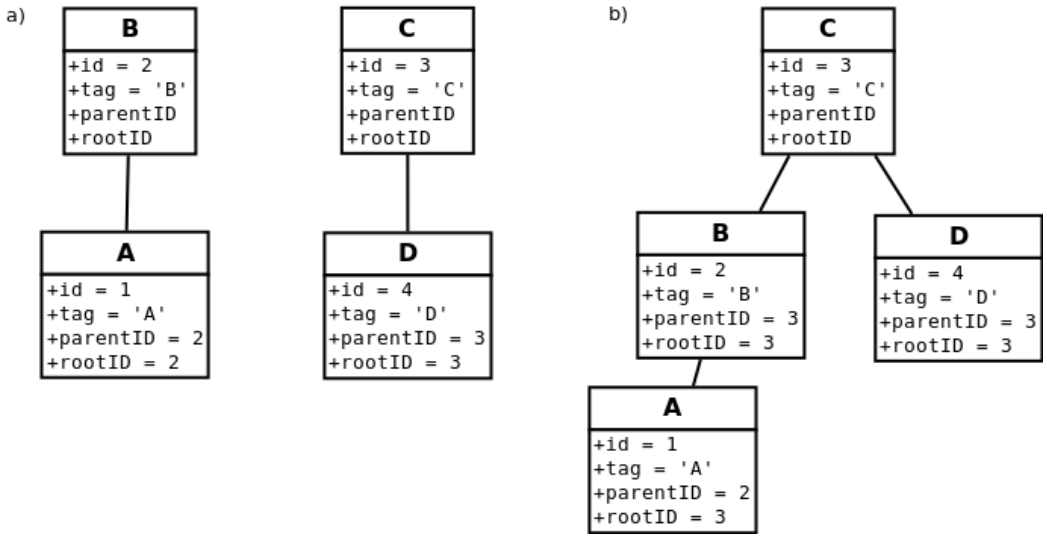


Figure 6.2: Merging tags in Tagregator

down:

Merging A under B:

1. $A.parentID = B.id$
2. $A.rootID = B.id$
3. for each tag C with $C.rootID == A.id$ set $C.rootID = B.id$

Splitting A from under B:

1. clear $A.parentID$
2. clear $A.rootID$
3. for each tag C in tree under A set $C.rootID = A.id$

As we can see, setting root ID is simple when we are merging tags (one direct SQL update), but splitting requires us to traverse through whole tree structure. This was considered and was not found as a problem, since spitting

operations are not very frequent, since they are only used to undo previous wrong decisions when merging tags.
Renaming of tag is done by merging tag under new canonical tag, with our desired name. This creates more entries in tags table, but keeps our system simple to administer.

6.3 Queries

Presented solution tries to minimize time needed to perform queries, but also make queries as easy as possible. As we discussed, most common operation is getting names of canonical tags. Every time we are creating tagcloud, CloudMaker returns array of tag IDs and it is necessary to look for their string representations. This operation is done using only one simple SQL select.

```
SELECT tag FROM tagregator_tags_sk WHERE id=[ID];
```

To get canonical ID for a tag (this is done when storing tags), we need to perform at most three queries. We look into table containing tags for our input tag. If we find no match, we perform second query, add input tag to our database. We select details T for input tag, and we need to look if $T.rootID$ is set to different tag. If so, we select this canonical representation of our tag, and make it our T. Last step is to look if tag is visible ($T.visible$). If $T.visible$ is true, we return ID of this tag. Else we return -1. This is important property when working with arrays. As we wrote earlier, we give Tagregator array of tags and it returns array of IDs to store in CloudMaker. When some of the tags can be hidden, they are not added to final array. This way we can directly add all the IDs to CloudMaker, hidden tags are filtered here. Figure 6.3 demonstrates the algorithm. You can see that there is a path that requires 4 queries, but it involves adding tag and than looking

for its canonical representation, which is clearly not possible, when we just added the tag.

Apart from background for tag-based operations, we developed administration panel to work with canonical representations of tags. Features are very simple and follow operations we introduced, merging/splitting tags and setting visibility. Our tag administration interface also features a possibility to re-cache data in CloudMaker, which is necessary after performing administrative operations.

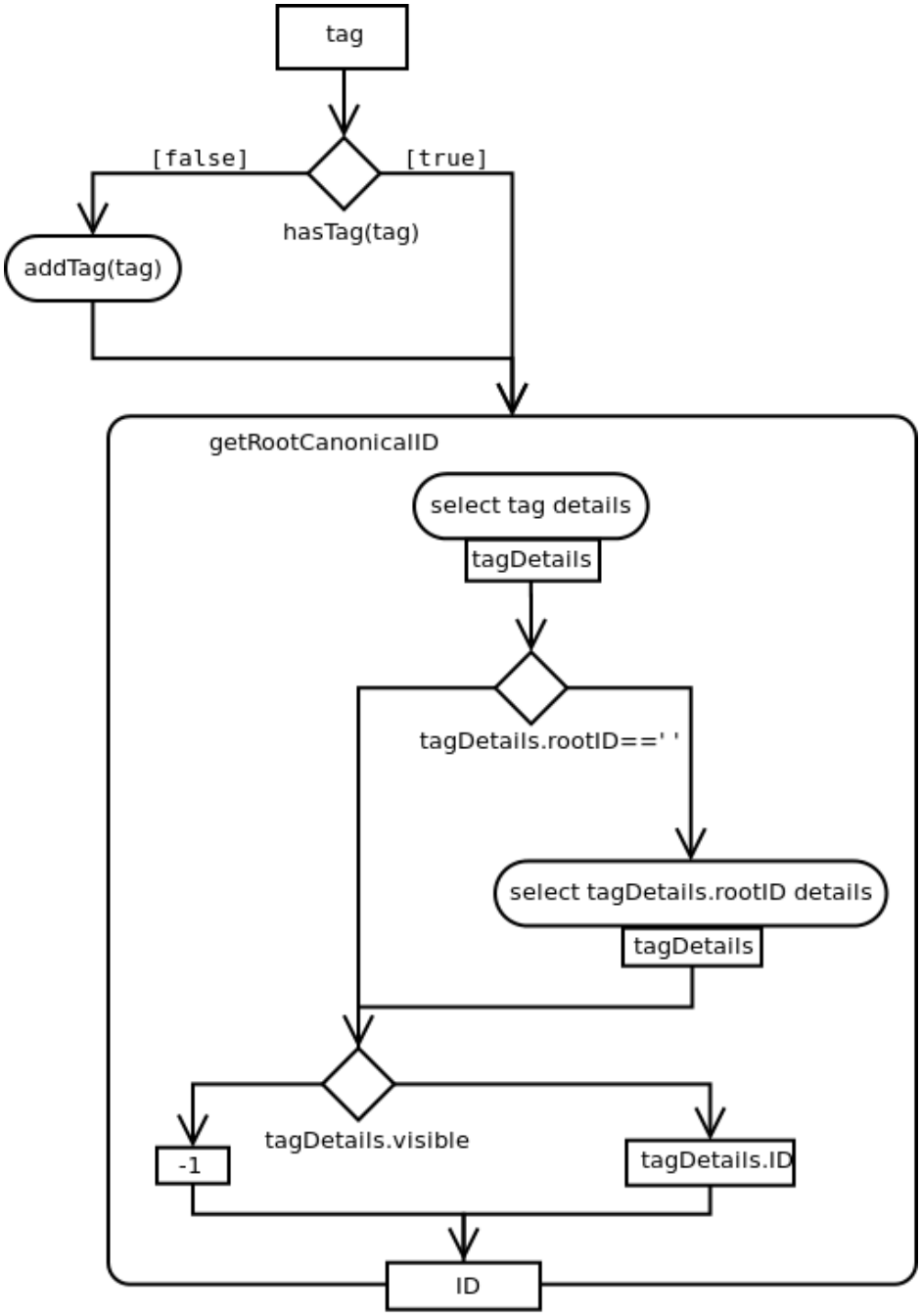


Figure 6.3: Tagregator getIDs operation. For an input tag, we need its canonical ID

Chapter 7

Tagging game

To provide good tag suggestions for users, many systems are using existing tag created folksonomy [ZXS06]. This approach is good for maintaining tagging vocabulary of the system, but it does not further expand vocabulary. To follow principles of the good tagging system, we will use an external precomputed ontology, and after user inputs a tag, we can suggest that he can also use tags related to his tag from the ontology. This way, our tagging system will expand in natural way, with ease of use.
For this specific task, we need a large corpus of words and their relations. We designed a game, similar to Google Image Labeler, so that this ontology is user generated. Our game is playable on http://apps.facebook.com/asociacie.

7.1 Google Image Labeler

In 2006, Google introduced a game, in which 2 players were connected randomly, and they were shown a picture. Their goal was to type in a words describing the picture. When players used common keyword to describe the picture (without seeing their partners guesses), they were assigned some points and new picture was generated. The game run for 120 seconds, and

users were also enabled to skip word, so that they can try as many words as possible in the time limit. Generated this way, keywords were considered good quality and were used for Google Image Search. The game was very popular, and created a large database of tagged images. Tag quality varied, users tried to simply tag the images with for example "x", and when they matched with their partner, they continued this way to achieve maximum score, without creating any useful tags. Later, Google introduced new scoring policy, the more specific keyword you use, the more points you get. Despite many problems, this idea was a great example of human computation [vA07], and deserves much attention for our work.

7.2 Asociácie (Associations)

Google used users to tag images for them, and when we further think about their idea, it can be used for any kind of resource, even keywords. This way, we created game called "Asociácie" (Associations), in which users are shown a word and their task is to write a word that is somehow associated to this word. Game was developed in flash and PHP, it was deployed on Facebook, to use social features of this web to easily find new players for our game. Game was tested only for Slovak language, since Slovak is the language on which we focus in implementation part of this thesis. Also, there exist large ontologies for English, but Slovak language is lacking enough good works of this kind.
Game can be divided into several logical parts:

- **welcome screen** containing information about top scores, games and game rules.
- **queue** of players waiting for game co-players.
- **actual game** divided into rounds (different words).

- **game results** with sharing options. These results show game score and also position in Top games. Later version of game also shows co-players suggestions. This feature was requested by players.

7.3 Implementation

The core part of the system was written in PHP, with data stored in mySQL database. PHP served for communication with database, which is crucial in this type of application. Game client was written in Flash, Actionscript version 3. We have decided to use Flash, to its asynchronous nature, and ease of programming. Since we only had a provided web server, we were unable to establish socket connection. Thus we had to look for a different way to create real-time communication with server. This problem was solved using brute-force, Flash client was periodically sending requests to PHP. Communication was done using XML files, and GET parameters. Flash was simply requesting URLs containing XML data.

7.3.1 Player identification

Since game runs as a Facebook application, players does not need to register, they only need to give basic permissions this application. Players were stored in mySQL database, there as a unique identifier we used their Facebook ID. Early versions did not store player name, but API we used was rather slow, so it was necessary to store player name. This name was updated each time player started this application. This way we could easily create high-score tables, start game etc. without having to use API.

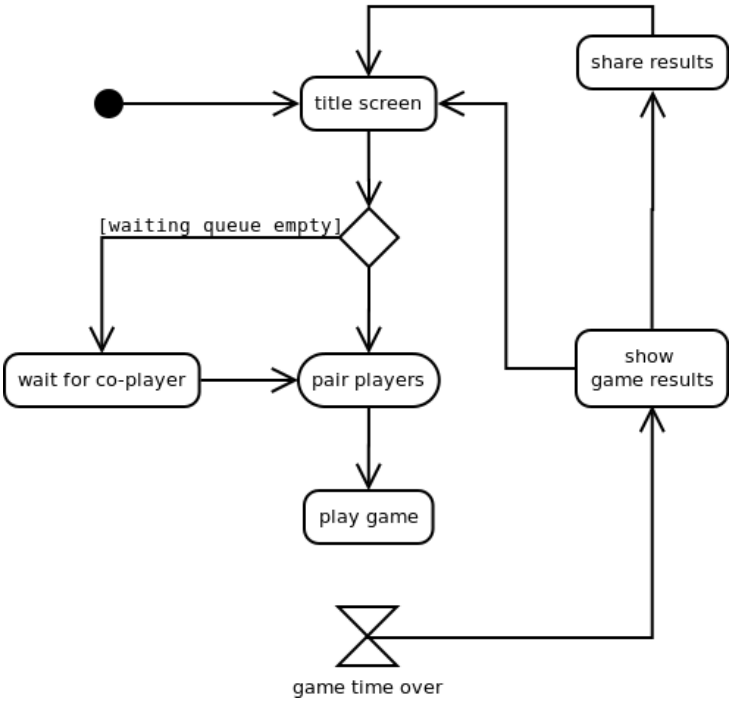


Figure 7.1: Aplication states and flow

7.3.2 Pairing players

In order to start the game, it was necessary to find 2 players willing to play and pair them up. Basic idea was to use a waiting for game queue, a database table containing IDs of players that want to play the game. When a user want to play a game, we look into this table: if it is empty we add this player, else there is a co-player for him and after one extra check the game can start. Later we realized that when a player is added to waiting for game queue a leaves application (this often happens when no one is playing) he stays in this queue and he is paired up, but the game can not start. This bug was solved by using keep alive signals. In queue table, apart from user ID, we also store a timestamp that says when was the last time user pinged server that he is looking for partner. When we look into waiting for game queue,

we first delete all players with old timestamp - players that started looking for a co-player but left without success. After this step everything goes as planned. When two players are paired up, we create a new game for them.

7.3.3 Running game

Game data are stored in database table Games. When creating new game instance, we mark a timestamp, so that we can measure game time. We select a word (details later) and than both players game clients call scripts *getGameData* and *sendWord*. Script *getGameData* is called periodically, to get time left, actual word, number of co-players suggestions and whether or not co-player wants to skip actual word. Players own suggestions were send using *sendWord*. All control mechanisms are server side, moreover as said before, also time measurement is server side, and game runs until *getGameData* returns that game has finished. After playing the game, players can share their score on their Facebook profile, and thus spread the game some more.

7.3.4 Word database

Words for games are stored in same database as players suggestions. Each word is assigned unique id, status, number of uses as game word and number of times players agreed to skip this word. Word status can be *new*, *approved* or *banned*. When a player uses some word for the first time it is added to database with status set to *new*. Then in administrator panel, admin can change this status to *approved*, which means it can be used as a game word, or *banned*, so that the word does not count anywhere.

Figure 7.2: Image of the running game

7.4 Results

First game was played on 27th October 2010, and by the day of writing (1st February 2011) more than 600 games were played, creating many relations between keywords. Top players of the game have successfully guessed about a thousand words. The corpus of words had 1683 different words, and 9265 suggestions were made. To visualise this data, we used a library called graphviz (http://www.graphviz.org/), especially a part called neatto, that is capable of drawing graphs using spring layout, similar to algorithm implemented for relational tagcloud. Taking in account all the relations, and even a stronger part rendered not well-arranged graph, so we used only those

relations that at least 3 users assigned between words. Figures in the end of this chapter show parts of this visualisation.

Tagging game is a partial success, the application worked well, only problem was with getting players to play it. Huge disadvantage is that two players are needed for this game, and since there were not as many players, you had to find a co-player yourself. The structure we have created can be used for tag suggestion, but it is not very large in scale, and thus it does not exactly suite need of blog.matfyz.sk, which has a very specific domain. Articles are mainly mathematics and computer science oriented and our general structure does not cover these topics well. Anyway, we consider this experiment a success, we have created almost 10000 relations between words. We have a lot of data that can be further analysed, and since users are from Facebook, some details like gender or age can be used to create new interesting statistics.

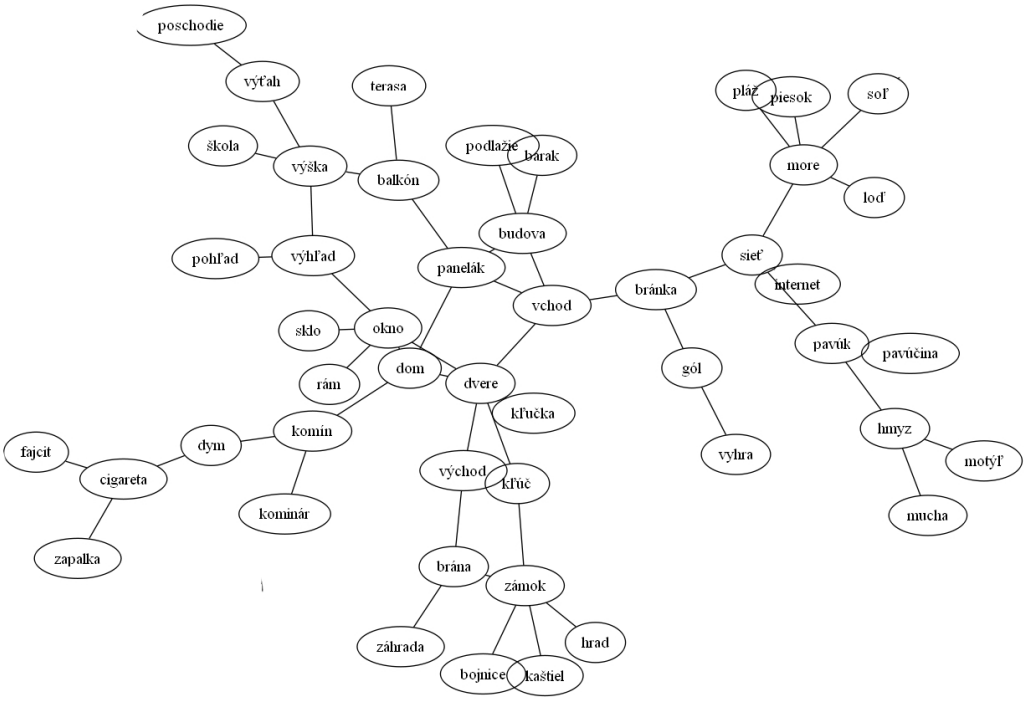


Figure 7.3: Part of a graph created by players of our game. A Slovak speaking user can see nice semantic relations between words.

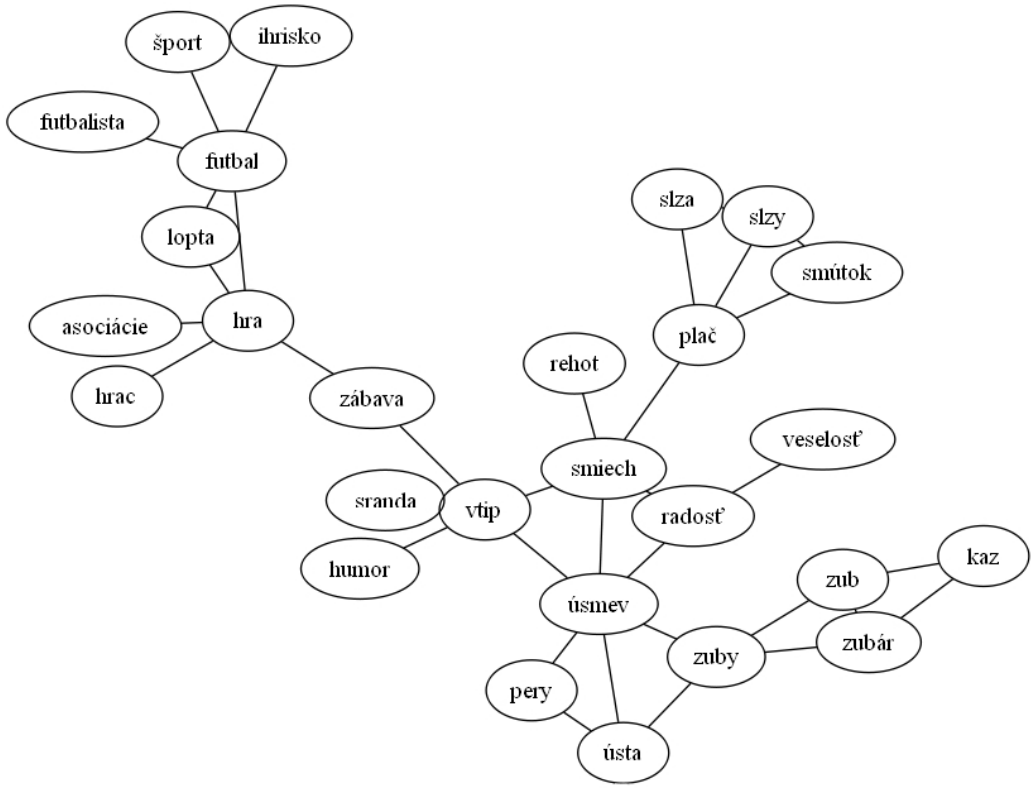


Figure 7.4: Another part of a graph. We can see that even semantically very different words can be connected by a path.

Chapter 8

Conclusion

In the first chapters we have introduced tagging as a modern way of navigation on web-portals. We described tagging system and operations within this system to give reader insight on the power of tagging.
Our main goal was to develop a tagging system for blog.matfyz.sk. Very much afford was given to analysis of problem and design, since all the smaller goals rely on quality of tagging system. We have studied modular design of this portal and created application that fits this design requirements. Also, tag properties of portal were studied, since proper distribution of tags is very important for working tagging system. The system we have created is very modular, easy to understand and provides many useful method that can be used in further development. This is the most important feature of all, ability to alter or add system parts for testing new technologies.
In old tagging system, we worked directly with Sedna database, and each time we wanted to create a tagcloud, it was necessary to find canonical representations of tags and then aggregate them. This operation was very time consuming, and poorly designed. Administration operations does not occur very often, compared to tagcloud generation, and thus our solution which uses precomputed tags is much more effective. All the tagcloud generation

is now performed real time, even complicated operation, like list of related tags needed for .

Our model of tagging system does not differentiate tags, users of resources. All the parts are taken as equal, so by implementing functionality for tag-clouds, we have done much more. We are able to create list of related resources, cloud of users using related tags etc. The extension of standard tagging system model allows us to use weighted tags, using which we are able to measure quality of tag annotations. This way we can distinguish between authors and readers tagging articles, but what is more we are able to use weighted tags everywhere. Our future idea is a tagging tool allowing users to add weighted tags for their articles, and that way really express category or semantics of the data. For example article about web-application AIS can be tagged with tags: “AIS” with weight 0.8, “web” with weight 0.1 and “browsers” with weight 0.1. This way we can really see that article about browsers with “browsers” weight 0.7 is a more relevant resource for tag “browsers” than our article about AIS.

Bibliography

[ak09] Juraj Ďuďák. *Visualization of tagspace*, 2009. Bachelor thesis.

[BGN08] Scott Bateman, Carl Gutwin, and Miguel Nacenta. Seeing things in the clouds: the effect of visual features on tag cloud selections. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, HT '08, pages 193–202, New York, NY, USA, 2008. ACM.

[DL07] Owen Kaser Daniel Lemire. *Tag-Cloud Drawing: Algorithms for Cloud Visualization*. 2007.

[EGV94] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

[Fra09] Juraj Frank. *Towards enhanced effectivity and usability of tagging*. Master's thesis, Comenius University, 2009.

[Goo07] Google. *http://images.google.com/imagelabeler/*, 2007.

[Gro] The PHP Group. *http://php.net/*.

[Gru07] Thomas Gruber. Ontology of folksonomy: A mash-up of apples and oranges. *International Journal on Semantic Web Information Systems*, 3(1):1–11, 2007.

[HH07] H. Shepherd H. Halpin, V. Robu. *The Complex Dynamics of Collaborative Tagging*. In *Proceedings of the 16th International World Wide Web Conference*, 2007.

[HK09] Eric Huang and Richard E. Korf. New improvements in optimal rectangle packing. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 511–516, 2009.

[JS08] Michael Cardew-Hall James Sinclair. *The folksonomy tag cloud: when is it useful?* In *Journal of Information Science*, volume 34, pages 15–39, 2008.

[Koh08] Anton Kohutovič. *blog.matfyz.sk - community blog portal*. Master's thesis, Comenius University, 2008.

[Mar09] Benjamin Markines. *Evaluating Similarity Measures for Emergent Semantics of Social Tagging*. *Proceedings of the 18th international conference on World wide web*, pages 641–650, 2009.

[Mat04] Adam Mathes. *Folksonomies - Cooperative Classification and Communication Through Shared Metadata*. 2004. `http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html`.

[MH07] Mark T. Keane Martin Halvey. *An Assessment of Tag Presentation Techniques*. 2007. `http://www2007.org/htmlposters/poster988/`.

[Mik05] Peter Mika. *Ontologies Are Us: A Unified Model of Social Networks and Semantics*. *Lecture Notes in Computer Science*, Volume 3729/2005:522–536, 2005.

[Rej10] Martin Rejda. *Modular Redesign of The blog.matfyz.sk Portal*. Master's thesis, Comenius University, 2010.

[RGMM07] A. W. Rivadeneira, Daniel M. Gruen, Michael J. Muller, and David R. Millen. Getting our head in the clouds: toward evaluation studies of tagclouds. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 995–998, New York, NY, USA, 2007. ACM.

[SAG06] Bernardo A. Huberman Scott A. Golder. *Usage patterns of collaborative tagging systems. Journal of Information Science*, pages 198–208, April 2006.

[sed07] *Sedna - native XML database*. 2007. `http://www.modis.ispras.ru/sedna/`.

[Sow00] John F. Sowa. *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2000.

[TN06] FUJIMURA KO TANIMOTO NAOTO. *The EigenRumor Algorithm for Calculating Reputation of Information Resources in Electronic Communities. IPSJ SIG Technical Reports*, 2005(NO.3(DPS-121 GN-54)):37–42, 2006.

[vA07] Luis von Ahn. Human computation. In *Proceedings of the 4th international conference on Knowledge capture*, K-CAP '07, pages 5–6, New York, NY, USA, 2007. ACM.

[W3C10] W3C. *http://www.w3.org/Graphics/SVG/*, 2010.

[YHm06] Víctor Herrero-solana A Yusef Hassan-montero. Improving tagclouds as visual information retrieval interfaces. In *Merída, InSciT2006 conference*, 2006.

[ZXS06] Jianchang Mao Zhichen Xu, Yun Fu and Difu Su. *Towards the Semantic Web: Collaborative Tag Suggestions. 15th International World Wide Web Conference*, 2006.

Abstrakt

Kolaboratívne tagovanie je dnes na webe veľmi rozšírené a známe webdizajnérom, ktorí ho však používajú len na pomocnú kategorizáciu obsahu a druhoradú navigáciu. V práci popisujeme silu tagovania, ktorá nám dáva omnoho viac informácií a umožňuje vytvoriť plnohodnotnú navigáciu. Počas práce na relačnom tagbrowsri pre blog.matfyz.sk sme zaznamenali veľa problémov so starým tagovacím systémom, preto sme navrhli nový systém, so všetkými vlastnosťami moderných tagovacích nástrojov a zároveň vhodný pre náš portál. Požadované vlastnosti systému boli dôkladne preskúmané, takisto boli nad databázou tagov urobené štatistiky na utvrdenie správnosti dizajnu nového systému pre portál. Vytvorený tagovací systém je veľmi modulárny, skladá sa z modulov na tvorbu rôznych druhov tagovej navigácie a na riešenie problémov s voľným tvarom tagov. V práci experimentujeme s novým spôsobom sugescie tagov, čím chceme zvýšiť počet tagov v systéme. Nami vytvorený systém je výrazne rýchlejší a poskytuje nové možnosti navigácie pomocou tagov.

Keywords: tag, tagovací systém, webová aplikácia, tagcloud, folksonómia