Vrije Universiteit Brussel

WISE – Web & Information Systems Engineering
Department of Computer Science
Faculty of Science
Vrije Universiteit Brussel

Towards Web 2.0: WSDM Modeling Support for Tagging

Master Thesis submitted in order to obtain a Master Degree in Applied Computer Sciences.

Nicolai Roovers

Academic Year 2006-2007

Promoter: Prof. Dr. Olga De Troyer

Supervisor: Dr. Sven Casteleyn

Vrije Universiteit Brussel

WISE – Web & Information Systems Engineering
Departement Informatica
Faculteit Wetenschappen
Vrije Universiteit Brussel

Modelleerondersteuning gericht op Web 2.0: Tagging in WSDM

Proefschrift ingediend met het oog op het behalen van de graad van Licentiaat in de Toegepaste Informatica.

Nicolai Roovers

Academiejaar: 2006-2007

Promotor: Prof. Dr. Olga De Troyer

Assistent: Dr. Sven Casteleyn

Abstract

The Web 2.0 has brought a large variety of people together in open communities on the Web. In these open communities, the people create information and receive information created by other community members. This evolution has resulted in the creation of large amounts of data. Since many years, techniques have been developed to organize data. However, most of these methods require special skills, skills that the large variety of people, which the Web reaches, don't have. Folksonomies deliver a solution to this problem. Folksonomies are based on a simple mechanism: by adding keywords to data, the data gets a certain meaning which helps with the further organization of the data. This simple mechanism is easy to use and allows a broad variety of people to annotate their data and, as a result, to organize their data.

This Master Thesis goes deeper into detail about folksonomies and furthermore it deals with finding out what ways there are for easily creating a folksonomy-based Web system.

Currently, methodologies are available that aid the designer with the design of a Web system. An extension is added to a certain web design methodology, WSDM (Web Semantics Design Method). This extension will facilitate the design of a folksonomy-based Web system. It is build up by a set of modeling primitives that provide the designer with the necessary semantics for modeling information, functionality and navigability, related to folksonomies.

Finally, as an illustration for these modeling primitives, a Web system will be extended with a folksonomy and its functionality.

Keywords: Conceptual Modeling, Folksonomy, Web 2.0, Web Engineering, WSDM.

Samenvatting

Het Web 2.0 heeft een uitgebreide en gevarieerde groepen samengebracht tot publieke gemeenschappen die ontstaan op het Web. Leden van die publieke gemeenschappen creëren nieuwe informatie. Deze informatie wordt vervolgens toegankelijk voor de andere leden. Deze evolutie heeft de creatie van een grote hoeveelheid data tot stand gebracht. Dit vereist een organisatie van de data. Verschillende methoden werden ontwikkeld die helpen bij de organisatie van data. Deze methoden eisen echter deskundige vaardigheden, vaardigheden waarover de gevarieerde gemeenschappen op het Web, niet beschikken. Folksonomieën brachten oplossingen voor de problemen waarmee klassieke organisatie methoden mee te kampen hadden. Folksonomieën organiseren hun data op basis van een eenvoudig mechanisme: het toevoegen van tags aan data waardoor de data van een bepaalde betekenis voorzien wordt, en op die manier georganiseerd kan worden.

In deze Thesis worden folksonomieën meer in detail behandeld en wordt nagegaan welke methoden er zijn om folksonomiegebaseerde websystemen te ontwikkelen.

Momenteel zijn er verschillende methodes ter beschikken die een ontwikkelaar van een websysteem bijstaan bij de ontwikkelen hiervan. Één van deze methodes, WSDM (Web Semantics Design Method), wordt uitgebreid met een aanvulling die speciaal voorzien is om folksonomiegebaseerde websystemen te ontwikkelen. Deze aanvulling bestaat uit een verzameling modelleer primitieven die de ontwikkelaar met de nodige semantiek voorzien bij het modelleren van informatie, functionaliteit en navigatie, gerelateerd met folksonomieën.

Tot slot, ter illustratie van de modelleer primitieven, wordt een websysteem uitgebreid met een ondersteuning voor een folksonomie.

Sleutelwoorden: Conceptueel Modelleren, Folksonomie, Web2.0, Web Engineering, WSDM.

Table of Contents

List of Figures

List of Tables

Introduction

During the last years, the World Wide Web has been one of the fastest growing media, with more and more information being created every day. The Web is a public accessible medium where everyone, with access to the Web, can publish information. This results in some information being very valuable while other information can be complete junk. We therefore need to find ways in organizing the available information on the Web, so that we improve recovery and discovery of the available and valuable information.

We can organize the information on the Internet by classifying and categorizing the information by means of metadata – data about data.

The Semantic Web has been playing an important role in the organization of information on the Web. The vision of the Semantic Web is to make the information on the Web not only readable for people, but also make it understandable and processable for automated software agents, allowing a better cooperation between people and software agents. For this purpose, the data need to be classified by means of metadata in a uniform and formal way. The formal descriptions for the data are defined by ontologies.

Ontologies describe precisely the concepts that can be represented and their relationships. However, ontologies require special modeling skills and aren't really accessible for the large public.

Since the introduction of the Web 2.0[1], the large public started creating and publishing information on the Web on so-called social web-based communities where the Web applications serve as platforms for the end-users. As a mass organizational tool, ontologies were to complex. Folksonomies brought an answer to this problem. People could categorize their information by annotating the information with simple keywords. These folksonomies are becoming a big success on the Web. It was even mentioned in the New York Times Magazine as 2005's best ideas [44].

The growing success of folksonomies makes it interesting to analyze how we can design and build such folksonomy-based Web systems.

For designing a Web system in general, methodologies have been created which aid a designer with the design of his Web system. These methodologies construct a Web system in a structured and systematic way improving the quality of the design and the Web system and

[1] Web 2.0 is a new way of looking at the World Wide Web, not by its technical specifications, but more as the idea that the end-users create the content in social platforms being served by Web applications.

the maintainability of the Web system. These methodologies provide concepts, specially created for designing a Web system and all its different aspects, e.g., the data content, the navigation structure, the presentation and layout.

We have analyzed different existing Web design methodologies and tried to find out what modeling and design concepts they offer for designing a Web system supporting a folksonomy. These concepts could provide the necessary semantics for modeling a folksonomy-based Web system, facilitating the design of such a Web system. However, after analyzing these methodologies we can conclude that no such folksonomy modeling concepts are currently available in the analyzed methodologies. A designer however can benefit from such concepts when designing a folksonomy-based Web system. Since no such concepts are currently yet available in existing methodologies and since a designer can benefit from such concepts, it would be interesting to analyze what concepts are exactly required and how we can design and create them.

The goal of this Master Thesis is first of all to give a clear overview of what a folksonomy is and afterwards to discover all its aspects, like what kind of information or functionality is involved, and based on these discoveries we will create and introduce new concepts which will aid a designer with the design of a folksonomy-based Web system.

This Master Thesis consists out of four chapters. In the **first chapter** we highlight the increase of information being available on the Web. We will first describe two classification schemes used for the Web in order to organize data. After discussing drawbacks related to these classification schemes, folksonomies are introduced. This chapter continues with a broad description of folksonomies and its structure. We will end the first chapter with Web modeling methodologies and their benefits towards the design of Web applications in the first place. Afterwards, we will analyze what these design methodologies have to offer for designing a folksonomy-based Web application. In the **second chapter** we will describe one particular design methodology, the Web Semantics Design Method (WSDM), into more detail since it will serve as the framework of this Master Thesis.

In the **third chapter** we will extend WSDM with modeling concepts aiding a designer with the design of a Web system supporting a folksonomy. First we need to analyze existing folksonomy-based Web systems in order to extract their requirements and to build up a general view of requirements related to folksonomies. Based on these requirements we will afterwards create new modeling concepts which we will add to WSDM.

In our final chapter, the **fourth chapter**, we will describe the case study we made, as a proof of concept for our extension to WSDM.

Finally, we will end this Master Thesis by describing possible future work and by giving a general conclusion.

1 Background

Before starting with the research, we performed a literature study on the subject and related issues. In this chapter we will present the result of this literature study and present the information we found which will clarify the scope of the thesis.

We start by going deeper into the expansion of the Web and how we need ways to organize the available information. For doing so, metadata is used. Here, we present a set of metadata schemes, including folksonomies. Since folksonomies are an important aspect for this Master Thesis we will give a detailed overview of the available information about folksonomies.

After finishing the literature about metadata, metadata schemes and more in detail folksonomies, we will present design methodologies which aid the designer of a Web application with the design of that Web application.

1.1 Growth of the Web

Since years, the Internet has been one of the fastest growing medium currently available. During the years 2000 and 2003, the School of Information Management and Systems at the University of California at Berkley conducted a survey [27] in which they tried to estimate how much information is created each year. In 2000 the volume of the surface Web was estimated at 20 to 50 terabytes. During the second session, the volume was measured at about 167 terabytes. This surface Web, or visible Web, is the part of the World Wide Web that is build up by static and publicly available Web sites which are indexed by search engines like Google[2], Yahoo![3] and MSN[4].

To define the size of the surface Web in terms of the amount of Web pages we refer to the work of Gulli, A and Signorini, A. [17]. During the year 2005, they estimated the surface Web size in terms of the amount of Web pages at more then 11,5 billion Web pages, based on the overlap and index size of the search engines Google, MSN, Ask/Teoma[5] and Yahoo!.

The second part of the Web, the deep Web or invisible Web, consists of information that is available through dynamically generated Web pages. These dynamically generated Web pages can't be indexed (yet) by the search engines. In order for Web pages to be searched by

[2] http://www.google.com/

[3] http://search.yahoo.com/

[4] http://search.msn.com/

[5] http://www.ask.com/

Web engines, these Web pages need to be static and linked by other Web pages. Deep Web sources, however, store their content into databases which immediately present their information by dynamically generated Web pages after a request.

The fact that the deep Web can't be indexed by search engines makes it more difficult to give a measure for size of this part of the Web. Bergman, M. [1] measured the volume of the deep Web by means of the BrightPlanet[6] search technology. The key findings of this research included that the deep Web was 400 till 550 times larger and is growing faster then the visible Web. The volume of the deep Web contained up to 7500 terabytes of information resulting into nearly 550 billion individual documents. During the second session of the survey by [27], the volume was already measured at 91.850 terabytes.

1.2 Metadata

The fast growing amount of information, available on the Web, where everyone can create and publish information, requires the need for organizing all this information. In order to organize the available data, we need data about data, or what is called *metadata*. Metadata aids the identification, description, management and location of the available information. With this metadata, we can enhance the process of resource discovery by disclosing sufficient information about data to enable users or machines to discriminate between what is relevant and what is irrelevant to a specific information need [28].

1.3 Classification

A first approach in organizing data by means of metadata is by the use of *classification*. In Library and Information Sciences[7], the term classification is used to refer to three distinct but related concepts: (1) a system of classes, ordered according to a predetermined set of principles and used to organize a set of entities; (2) a group or class in a classification system; (3) and the process of orderly and systematic assigning entities to classes in a classification system of mutually exclusive and non-overlapping classes where each entity can only be assigned to one class [22]. Our focus rests on the first description where we organize resources available on the Web by using a classification system of classes where each class provides the necessary metadata for its class members, the Web resources.

---

[6] The BrightPlanet technology is originally developed for commercial uses by BrightPlanet, http://www.brightplanet.com/

[7] **Library and Information Science** is an interdisciplinary science incorporating the humanities, law and applied science to study topics related to libraries, the collection, organization, preservation and dissemination of information resources.

The process of classification is applied by a specific set of rules determining the structure of the classes and the class relationships, resulting in a specific scheme. We will now take a look at two kinds of classification schemes which had any use in the organization of resources on the Web.

1.3.1 Taxonomy

Taxonomy, which comes from the Greek words *taxis:* 'order' and *nomos:* 'science', is a hierarchical and exclusive classification system. Examples of taxonomies are the Dewey Decimal classification for libraries [12], the Linnaean system for living things and computer file systems. The main idea of taxonomies is that a book, living thing, computer file, etc. is classified in a class and only one class within the whole taxonomy, with each class having its own set of specific features. This classification of items gives a certain meaning to these items. Furthermore a class can be subdivided into several sub-classes and a class itself can be a sub-classification of a higher leveled class and inheriting all the features of this higher leveled class. This implies parent-child relations between different classes.

A first significant attempt to bring order in the chaos of the Web was brought to the public by Yahoo! [40]. Yahoo! offered a list of available resources on the Web, by organizing them into a hierarchy of classes. As the Web started to expand, it became more and more difficult to maintain the hierarchy and the need for systematizing the hierarchy arose.

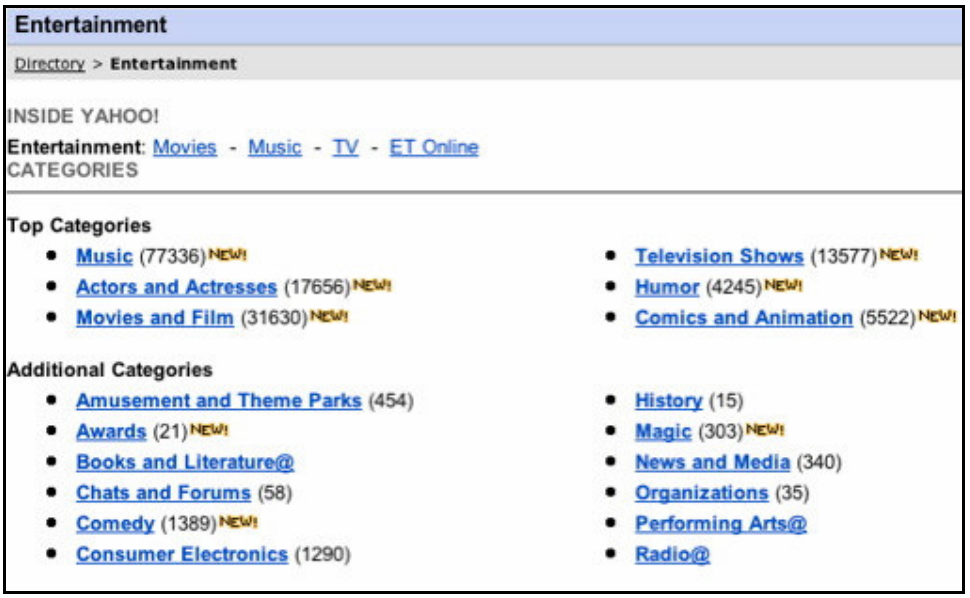

Figure 1: Yahoo's entertainment class.

1.3.2 Ontology

An ontology, a term which has its origin in philosophy, is an indication of existence. It states what can exist and therefore what can be represented. The ontology describes the domain of existence by a set of formal rules, defining the concepts that can exist in that domain and the relationships between these concepts.

The Dublin Core[8] ontology, for example, is a standardized ontology with 15 concepts for describing information resources on the Web, such as video, sound, image, combined media like Web pages, etc.

1.4 Semantic Web

The Semantic Web, which was first introduced in 2001 by Berners-Lee, T. [2], is a vision for the Web where the resources are not only readable for human beings, but also for computers to understand and automatically process them, so that human beings and computers can work in cooperation. This requires that Web resources are annotated with additional uniform metadata, providing common information about a domain that is understood and shared by human beings and computers.

Within the vision of the Semantic Web, ontologies play an important role. With the help of ontologies, it is clear for both human beings and computers what concepts exist in the domain and what relations exist between these concepts. In order to be processed by computers, the ontologies need to be defined in a formal way. For this purpose, the Web Ontology Language (OWL)[9] can be used. OWL provides an additional layer on top of XML, RDF[10] and RDF Schema[11] which facilitates expressing semantics, being used on the Web. Although there are other ontology definition languages like OIL, DAML and DAML+OIL[12], OWL is currently being recommended by the World Wide Web Consortium (W3C)[13] as a standard for defining ontologies.

[8] The Dublin Core Web site: http://dublincore.org/

[9] http://www.w3.org/TR/owl-features/

[10] http://www.w3.org/RDF/

[11] http://www.w3.org/TR/rdf-schema/

[12] http://www.w3.org/TR/daml+oil-reference

[13] http://www.w3.org/

1.5 Drawbacks

Although taxonomies and ontologies improve the organizational structuring of data, they are also subject to some disadvantages [15, 40, 47].

First off all, these metadata schemes are traditionally designed and created by professionals. These professionals are experienced in the fields of meta-modeling. But before one can model a domain (into a taxonomy or ontology), one needs to have a very good knowledge of this domain. So not only do the designers need to be experienced in the field of modeling and cataloguing, they also ought to be experienced domain users. This often requires serious education and training. But it results in high quality metadata. On the other hand, however, it is very time consuming and expensive.

The fact that the domains are modeled by professionals brings a second disadvantage along with it. The professionals modeling the domain are a small selected group of people, different from the eventual group of end users. This small group will model a domain according to their own view on this domain. The end-users on the other hand are disconnected from the design process. This means that the eventual view of the domain can be different from the end user's view on the domain. The only way for the user using the system is by changing his view on the domain.

The domain to be modeled can be very extensive. Establishing a unified model for such a large domain can be really difficult. Not all aspects can or will be modeled and the view of different designers can differ. This very often requires that a consensus needs to be made for certain domain aspects within the eventual result.

Taxonomies and ontologies resulting for highly dynamic domains also result in higher degree of maintenance. Changes in the domain need to be applied to the resulting model and current resources need to be adapted to the updated domain model. All of this requires a lot of time and effort from the designers and the more the dynamic the domain, the more time and effort is needed for the domain maintenance.

All these disadvantages don't mean we should throw away taxonomies and ontologies. We just need to measure how suitable these metadata approaches are when modeling a specific domain. When we look at suitable conditions for modeling and classification we can list them in the following characteristics:

- **Domain specific**: stable entities, small corpus, formal categories and clear edges.
- **User specific**: expert domain users, expert designers and coordinated users.

Now on the other hand when we turn this list around we get the following characteristics:

- **Domain specific**: unstable entities, large corpus, informal categories and no clear edges.

- **User specific**: amateur users, naive designers, uncoordinated users.

All these characteristics which make ontologies and taxonomies a bad fit, resemble closely the Web: largest corpus, most naïve users, no global authority, etc. [40].

The Web 2.0 brought a broad group of people together, all creating a large variety of information on the Web and all having different knowledge about that information and the Web. When we also want to organize this data by means of metadata, we saw that classical classification schemes, such as ontologies and taxonomies, wont suit. So the need raised for an easy mechanism that allows a broad public to annotate data according to their own interests and vocabulary.

In the next section we will introduce an alternative mechanism for annotating information on the Web. With this mechanism we try to eliminate some of the drawbacks that appear with classical classification schemes. Based on these drawbacks, this mechanism aims to be easy in use and to be applicable for a broad and dynamic corpus.

1.6 Folksonomy

Adding descriptive terms to information is an alternative process for annotating information with metadata. These descriptive terms, called *tags*, can be used to annotate any kind of information, e.g., Web pages, images, blogs. The end-users can chose these tags without being conformed by any predefined dictionary, taxonomy or ontology.

The organic system that flows out of this organization of information is called a *folksonomy*. The term was first introduced by Thomas Vander Wal and has been conducted from the combination of the terms *folk* and *taxonomy* [30] and is defined by:

> *Folksonomy is the result of personal free tagging of information and objects (anything with a URL) for one's own retrieval. The tagging is done in a social environment (shared and open to others). The act of tagging is done by the person consuming the information [43].*

Since the term folksonomy is partly conducted by the term taxonomy, there has been some debate about whether the term folksonomy is correct for its use [15]. As we saw in our previous chapter, a taxonomy is a classification scheme. The process of tagging resources with descriptive terms is actually *categorization*.

Categorization is the process of dividing the domain into groups of entities whose members are in some way similar to each other [22]. Based on the similarities between entities and the grouping of these entities into categories (the tags in a folksonomy), the users can find order in a complex environment. Category membership is generally more flexible and the boundaries of these categories are less clear. In contrast with classification, with single class membership for resources in classes with non-overlapping and mutual exclusive boundaries and where there exist clear relations between these classes.

The discussion about the term folksonomy has lead to the avoidance of the term by some researchers. Instead, names have been used in research like *collaborative tagging* or *social tagging*. Since we clearly defined what we mean by the term folksonomy we will therefore continue to use this term.

When we look at the definition of a folksonomy more closer, we can conclude that in a folksonomy it are the end-users who tag information and objects on the Web, which we will call the resources throughout the thesis. The creation of the metadata is thus handed to the end-users by annotating resources with tags, or tagging resources as defined by the definition. The user can pick out any kind of tag in order to help him organize resources according to his personal interests which will later on improve recovery of these resources. This aspect, which is called personal free tagging by the definition, can be shown by tags like "toread" in Delicious[14] or "me" in Flickr[15], tags which can only be understood in the context of the user.

In folksonomies we thus shift the creation of metadata for resources from professionals towards the end-users. As we saw before in taxonomies and ontologies the end-user needed to understand and adapt to the models defined by professionals before he could create consistent metadata. In folksonomies this disadvantage disappears since it is the end-user who creates the metadata by deciding what (resource) will be annotated and how (tag) it will be annotated by allowing the end-users to use their own personal vocabulary.

The principal of tagging is far more easier then the previous appointed taxonomies and ontologies [30]. Users can participate in the system without any previous training or knowledge. This lowers the barrier of entry for the end-users, which allows more people to annotate resources. This reduces time and effort costs when creating the metadata (the tags).

We can furthermore read from the definition that the act of tagging is done in a social environment. Tagging not only has a personal issue, it also serves a social purpose. The fact is that this personal organization takes place in a public community: the tags and the tagged resources are available for the public. This leads to an aggregation of information [34].

[14] http://del.icio.us/

[15] http://www.flickr.com/

A beneficial aspect of tagging is that feedback is immediate [30]. Annotating a resource with a tag, results in clustering all the resources annotated by the same tag. If this cluster is not what you expected for your resource, you are able to change or add another tag. In a personal perspective, this results in a good way for organizing your resources. We can also extend the scope to the whole community, including all the annotations on all resources given by all the users. Then in case you don't receive what you expected for a certain tag, you can adapt to the group norm, you can keep the tag and influence the group norm or both [41]. The behavior of users can therefore be related to the behavior of other users sharing the same tags and therefore the same interests which results in the improvement of discovery on new and interesting resources.

1.6.1 Folksonomy Example: Delicious

Before going further into any details about folksonomies, let us now give an existing example of a Web application with such a folksonomy. We have picked out the Web site Delicious since it is a commonly used example in research related to folksonomies [15, 30, 40, 47].

Delicious is a bookmark Web application allowing users to store their bookmarks, just like they can store their bookmarks in a Web browser. The benefit of Delicious is that these bookmarks are available from every browser.

Once users have created an account, they can start book marking Web pages. Each bookmark records the Web page's title, URL and time of creation. In addition users can also annotate the bookmarks with tags they can freely choose without any restrictions. After bookmarks have been created, a user can get an overview of all his bookmarks ordered in reverse chronological order on his personal Web page (Figure 2). Together with these bookmarks, a list is shown with all the tags given to any bookmark by the user. Selecting such a tag filters out all the bookmarks that are not annotated by the tag so that only those bookmarks are shown which are annotated by the tag.

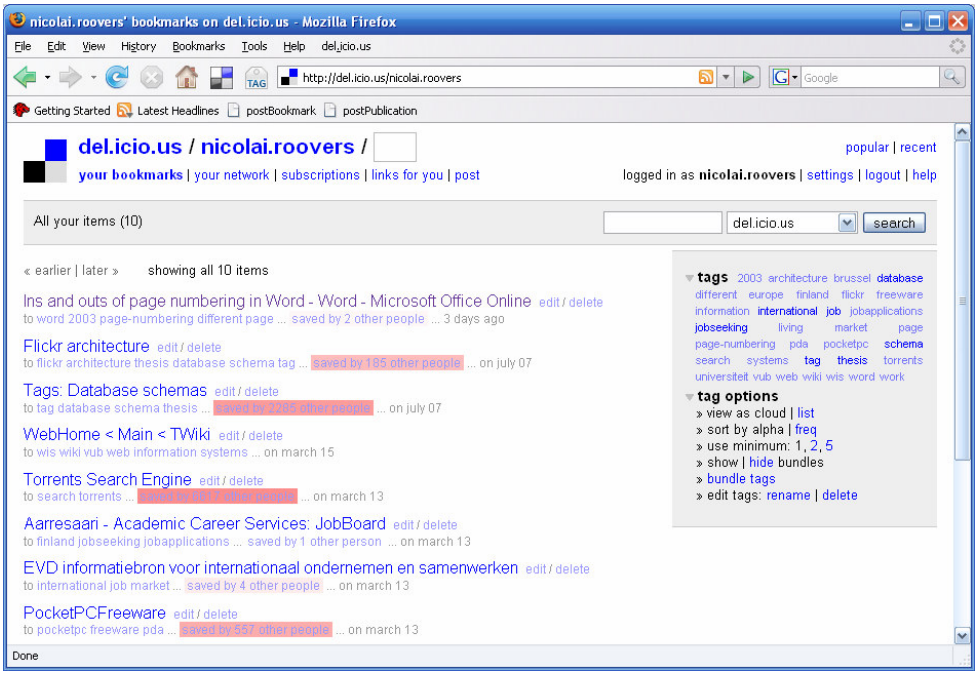

Figure 2: A personal Web page of a member using Delicious.

By being a folksonomy Web application, a user cannot only see his own bookmarks in Delicious, but also the bookmarks of other members. Each bookmark is shown together with who added it and the tags that user used for annotating the bookmark. By selecting a tag, Delicious presents a list of all available resources which are annotated with the tag. Furthermore, users can browse through other member's personal pages, viewing their bookmarks and their tags. By browsing through tags and member's pages, users can find Web sites that are of interest to them and people who share common interests.

1.6.2 Structure

A folksonomy is build up by a set of annotations in a flat namespace: there is no hierarchy and no direct specified relationships between these annotations [30]. Typically each annotation consists of three parts: a user, a resource and a tag. Therefore we can abstract a folksonomy into a set of triples

```
(user, resource, tag)
```

with each annotation defining which `user` annotated a certain `resource` with a certain `tag` [47]. We note here that annotations and tags are two different concepts. A tag is the single keyword and an annotation is what defines the metadata for a certain resource for a certain user by relating the resource, user and tag.

In the flat namespace of annotations we can derive implicit relations between different annotations [47]. Tags are usually semantically related when they are used to tag the same

resources or when they are used by users with related interests. Users may share the same interests when they use semantically related tags or when they share the same resources. Resources are semantically related when they are annotated with semantically related tags or by users sharing the same interests. This chain of semantics is embodied within the different co-occurrences among the users, resources and tags. It is from these co-occurrences that we can obtain implicit semantics that exist in a folksonomy.

1.6.3 Broad and Narrow Folksonomy

Besides the introducing and defining the term folksonomy, Thomas Vander Wal [42] also introduced two types of folksonomies: a broad folksonomy and a narrow folksonomy. We will now continue with describing these two types of folksonomies.

1.6.3.1 Broad Folksonomy

A broad folksonomy allows different end-users to annotate the same resource. As mentioned before, these end-users can tag the resources with their own tags according to their own personal interests and vocabulary. An example of a broad folksonomy is Delicious.

A broad folksonomy tends to build up a *power curve* (Figure 3) as the result from many users tagging the same resources. A power curve displays the tagging behavior of users for a certain resource. At first this power curve shows us the tags on which a large group of users seem to agree on and how they will describe the item. These tags represent the left spike in the power curve and can be extremely useful, for example when preferring tags or to extract controlled vocabularies. On the other hand we have what is called the *long-tail* which is shown on the right side of the power curve. The long tail represents small groups of users who agree on the same tag when tagging the resource. These people tagging the resource with the tag allow other people with the same vocabulary finding this resource even though their vocabulary differs from the vocabulary from the big masses.

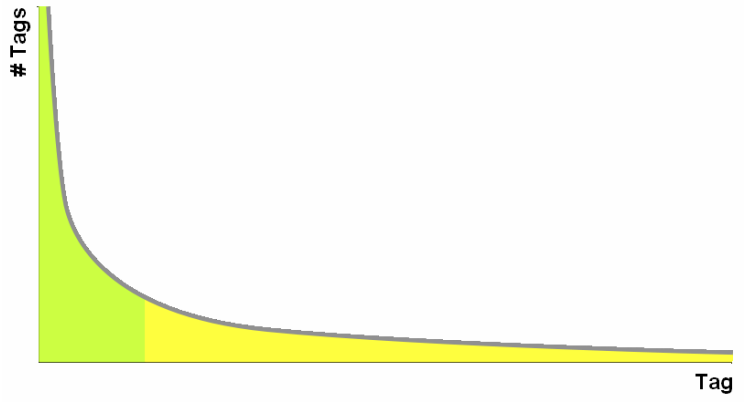

Figure 3: The power curve with the long tail effect.

1.6.3.2 Narrow Folksonomy

A narrow folksonomy restricts the tagging of a resource only to one user. In most of the times it is the creator of the resource who will tag the resource. Narrow folksonomies loose the power of the mass but they provide an efficient way in making resources findable which are hard to find with traditional search tools (like full-text search or other text-related tools). An example of a narrow folksonomy is YouTube[16]. YouTube allows the user to store his videos and to tag these videos. Based on these tags, extra means can be included which help with the search and retrieval of the videos.

1.6.4 Problems

At first sight folksonomies seem to be the best alternative for taxonomies and ontologies, eliminating some of the disadvantages in these classification schemes. But because of its uncontrolled nature and simplified structure, it leads to some problems [15, 30]. We can find the first problems in the context of the vocabulary arising from the tags:

- *Ambiguity*, *polysemy* and *homonymy*: Ambiguity appears when words have an undefined or unclear meaning. Polymesy appears when one word can have multiple related meanings while multiple unrelated meanings imply homonymy. So in all these cases the actual semantic meaning of a word is uncertain. For example "apple" can refer to a fruit or to a computer hardware and software manufacturer. The result of a search operation with "apple" can therefore return a list of results with different semantic meanings. In case of ambiguity and homonymy one can clear out part of the unwanted semantic meaning by adding extra keywords in the search operation.

- *Synonymy*: When different words all have a similar semantic meaning then we call this synonymy. For example products related to the Apple incorporation can lead to tags like "Mac", "Apple" and "Macintosh". Another problem occurs with singulars and plurals. The tags "computer" and "computers" are two different words representing semantically equal object.

Another problem occurring in a folksonomy is related to the structure resulting from a folksonomy. Folksonomies are build up by a list of annotations in a flat namespace [30], unlike taxonomies and ontologies where there exist strong and direct relations forming a well defined structure. This implies that in a folksonomy we loose a lot of information about the resources that was available in taxonomies and ontologies.

[16] http://www.youtube.com/

1.6.5 Folksonomy Presentation Techniques

In folksonomies, the amount of available tags can be extremely large. It can therefore happen that for certain resources, a larger amount of tags need to be displayed. We will introduce some techniques for efficiently presenting a list of tags.

For presenting a set of tags some Web systems use simple lists. Another approach for presenting tags and which is very popular among a lot of Web systems is the use of tag clouds (Figure 4). Tag clouds are visual presentations of a set of tags in which properties of the text such as size, weight or color can be used to represent the frequency of the associated tags. Furthermore a number of variations on tag clouds have emerged. Hassan-Montero, Y. and Herrero-Solana, V. [20] group N-most relevant tags together in the tag cloud which are selected from clusters based on semantically related tags. Shaw [39] introduces a tag cloud mapped like a graph where tags are represented as nodes and similarity relationships between the nodes (the tags) as edges. Bielenberg, K. and Zacher, M. [3] present their tag cloud in a circular form, where font size and distance to the center represent the importance of a tag.

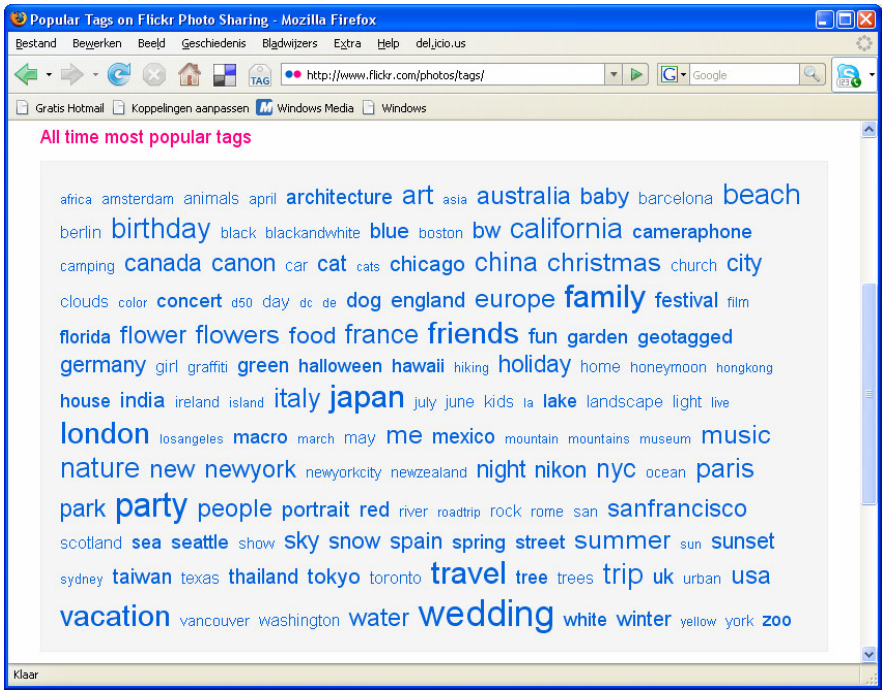

Figure 4: Tag cloud in Flickr showing the most popular tags.

Despite the popularity of tag clouds not much research has been done measuring the effectiveness of tag clouds. Halvey, M. and Keane, M. [19] tried to measure the effectiveness of tag clouds in finding certain tags and compares it with other tag presentations like horizontal and vertical tag lists. Their research produced some expected and unexpected results e.g. sorting the tag clouds and tag lists alphabetically improves the findability of tags,

position of tags is important and the font size, where larger tags are more easily found, is also very important in finding tags.

1.7 Web Application Design Methodologies

Since years, and still even today, Web applications have mostly been developed and created ad-hoc, starting from small applications and eventually evolving to large applications and soon becoming unmaintainable [24]. At first some guidelines and tools were proposed to assist developers with the creation of Web applications. However when designing large-scale applications, these means seem to fail. Reasons for failure seemed to be the lack of planning and process management and inappropriate models and application architectures.

Just as there was a need for the proper engineering of large-scale software applications during the early 70's, there became a need for engineering Web applications. With software engineering it is possible to model Web applications to a certain degree. For example, designing the conceptual model and a conceptual navigation structure is possible with existing software engineering approaches.

Fundamental differences, however, between Web applications and software applications, make a pure transposition of techniques both difficult and inadequate [31]. First of all, an important part of Web design involves aesthetic and cognitive aspects. Furthermore, Web applications are subject to their growth of the requirements and their continual change in the information content. These attributes require the design and creation of easily scalable and maintainable systems, features which can't be added later on. Thus whatever the development methodology we are going to use, it needs to have specific activities that will take into account the scalability and maintainability requirements, space for creativity that will enhance the aesthetic appeal of the user interface and the ability to respond to continuously changing user requirements [14].

During the last years, many development methods have been defined and introduced. In this section, we will discuss a few of them in brief. Since we are interested in designing and building Web applications supporting tagging, we will later on look at possible methods, tools or functionalities that are included in these Web application design models and that are specially designed for the support of tagging.

1.7.1 WebML

WebML enables designers to express the core features of a site at a high level, without committing to detailed architectural details [6]. During the WebML process (Figure 5) different phases are applied in an iterative and incremental manner where each cycle produces a partial version (model) of the Web system. WebML models are associated with an intuitive graphic representation (Figure 6), which can be easily supported by CASE and a

corresponding XML syntax which allows software generators to automatically implement the Web system.

The specification of a Web system by using WebML consists out of subsequently modeling five different aspects of the Web system by using the models provided by WebML: the data content (structural model), the Web pages that present the data (composition model), the links between Web pages (navigation model), the layout and graphic requirements for presenting the Web pages (presentation model), and the customization features for one-to-one content delivery (personalization model). For the remainder of this section we will briefly describe the five models.

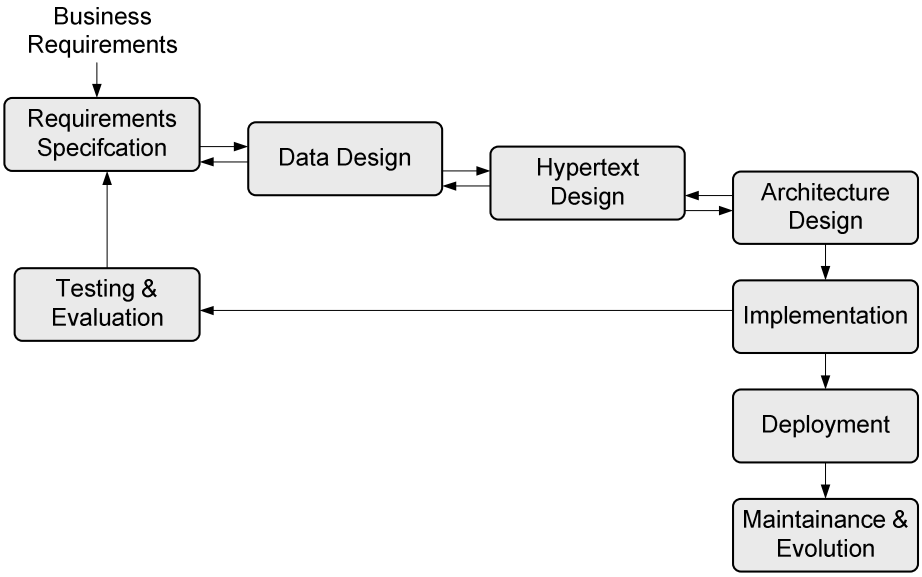

Figure 5: WebML design process.

1.7.1.1 Structural Model

The structural model is the result of the data design phase. It expresses the data content of the Web system in terms of entities and relations between them. Entities are a set of data elements, each with there associated type. Relations can have cardinalities. For describing the entities and relations, WebML has not proposed any specific data modeling language. This makes room for the designer to use classical data modeling tools like ER modeling or UML class diagrams.

1.7.1.2 Composition Model

The composition model defines the different pages of the Web system and the components they are composed of. For describing the components that are clustered and presented by the Web pages, WebML provides a set of component units where each component unit describes a piece of information.

- Data units: show information about a single object, usually an instance of an entity.
- Multidata units: show information about a set of objects, e.g. all instances of an entity.
- Index units: show a list of objects without presenting any detailed information.
- Scroller units: show commands (first, next,…) for scrolling through a set of objects.
- Filter units: show input fields where users can enter values which are required for the resulting objects.
- Direct units: denote the connection between two objects which are semantically related.

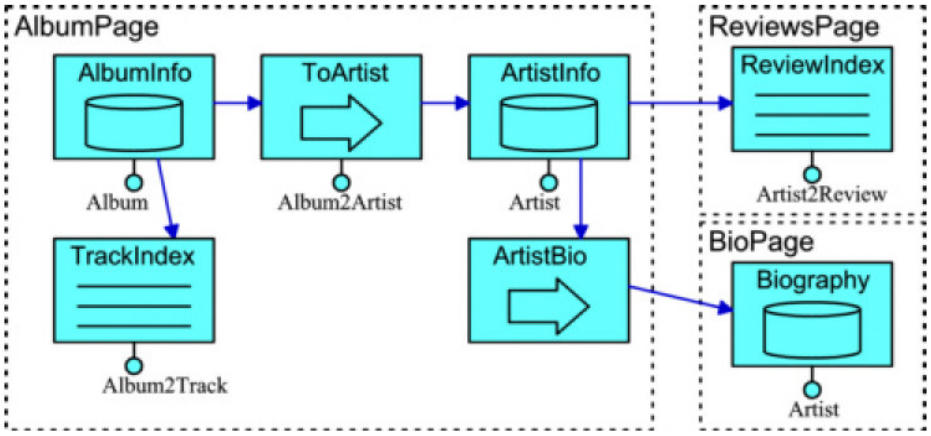

Figure 6: A WebML example of the composition and navigation specifications.

1.7.1.3 Navigation Model

The navigation model together with the composition model are the result of the hypertext design phase. During the navigation model we define the structure of the Web system. This is done by linking the different pages and component units. WebML provides two different kind of links:

- Contextual links: connect semantically related component units based on the semantics from the structural model. A contextual link gives information to a destination component from a source component, which defines the information that needs to be shown in the destination component.
- Non-contextual links: connect pages with each other independently from the components and relations they contain.

1.7.1.4 Presentation Model

The presentation model defines the layout and the graphical appearance of a page independently from any output device or any rendering language. Instead, WebML provides for this purpose its own abstract XML syntax.

1.7.1.5 Personalization Model

The personalization model provides two predefined entities: User and Group. These entities allow to specify user specific or group specific information like for example shopping suggestions and list of favorites.

1.7.2 Object-Oriented Hypermedia Design Model

The Object-Oriented Hypermedia Design Method (OOHDM) [38] uses abstraction and composition mechanisms in an object-oriented framework to, on one hand, allow a concise description of complex information items, and on the other hand, allow the specification of complex navigation patterns and interface transformations [37]. Building a Web application using OOHDM follows a process of four phases, based on an incremental or prototype process model.

1.7.2.1 Requirements Gathering

During the requirements gathering, all information about the different actors (stakeholders) is collected, together with the information about the tasks they need to perform. Afterwards scenarios are defined for each task and actor and are represented by User Interaction Diagrams which are collected to form a Use Case.

1.7.2.2 Conceptual Design

The starting point of the OOHDM design method is the conceptual design phase. During this phase the universe of discourse will be defined. This is done by using object-oriented modeling principles [35] augmented with some primitives such as attribute perspectives and sub-systems. The outcome of this phase, the conceptual schema, consists first of all out of conceptual classes, which may be build using aggregation and generalization/specialization hierarchies, together with relationships and sub-systems.

1.7.2.3 Navigational Design

The navigation design defines the navigation structure of the Web application in terms of navigation objects such as nodes and links. The nodes in the navigational design represent views on the conceptual classes defined in the earlier conceptual design and are customized for the user's profiles and tasks.

We derive the links from the relations between the conceptual relationships. Collecting nodes and defining links between the nodes results in meaningful sets called Navigational Contexts. These Navigational Contexts describe how the navigation will proceed in the navigational space.

1.7.2.4 Abstract Interface Design

The navigation objects are not directly perceived by the user. They are rather accessed by interface objects. The abstract Interface Design specifies what objects are responsible for mediating user interaction with navigation objects, by means of Interface Classes. These interface classes are defined as aggregations of primitive classes (such as test fields or buttons) and recursively of interface classes.

1.7.2.5 Implementation

During the final phase, the Implementation, a mapping between the conceptual object, the navigational objects and the interface objects onto a particular runtime environment need to be specified. When the target runtime environment is not fully object-oriented we have to map the OOHDM objects onto concrete objects from the chosen runtime environment.

1.7.3 Semantic Hypermedia Design Method

The Semantic Web [2] is an extension to the current Web with the goal to give a meaning to data for achieving a better cooperation between humans and computers. With the Semantic Web in prospect, the model-driven design approach for Web applications SHDM [26] was introduced. The idea here is not only to provide metadata about the data itself, but also about the structure of the Web applications.

SHDM is based on the OOHDM Web application design method. Just like OOHDM, the design process is build up out of set of incremental and sequential steps, each producing a certain aspect of the Web application: Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. As we can see the steps of SHDM resemble the steps from OOHDM.

SHDM keeps the separation between the conceptual design and the navigational design. This was an important cornerstone of the OOHDM design method. The models of both design phases in SHDM are however enriched with several new mechanisms, inspired by the languages being introduced for the Semantic Web.

The objects in both the conceptual model and the navigation class schema are now being defined and manipulated by using Semantic Web languages, such as the W3C Resource Description Framework (RDF). SHDM further characterizes objects by using ontology definition languages as DAML+OIL and OWL, which allow modeling advanced aspects such

as constraints, enumerations and XML Schema data types. Furthermore, in order to represent the navigational model, an XML capable query language is introduced.

1.7.4 Other Web Modeling Approaches

Next to the earlier discussed Web application modeling approaches, there are also others available. An other approach is the OO-H method [16]. OO-H is a generic approach that provides the designer with the necessary semantics and concepts for designing Web applications. The OO-H method is an extension on the OO-Method [32], a conceptual modeling approach which will be used in OO-H for the conceptual design of Web applications. The OO-Method is complemented with additional diagrams: the Navigational Access Diagram (NAD), which defines the navigation view, and the Abstract Presentation Diagram (APD), which defines all the concepts related to the presentation. Both the NAD and APD describe the interface related information with the aid of design patterns, which are defined in the Interface Pattern Catalog.

Just like all other methodologies, UML-based Web Engineering (UWE) [25] supports the development of Web applications. UWE aims for a systematic design, personalization and semi-automatic generation. The design of the Web application is based on the Unified Modeling Language (UML) [13]. For designing most of the aspects of a Web application, UML can be used. However, to cover the special aspects of a Web application, UML has been extended with new graphical representations that allow modeling Web application aspects such as navigation and presentation.

1.7.5 Web Modeling and Tagging

In the previous chapter we introduced folksonomies and the principal of tagging. What we want to achieve now is building Web applications that support the principal of tagging. As we have seen in this chapter, Web application design methods have been developed and introduced for the purpose of Web engineering.

What we want to achieve are ways (e.g., tools, concepts or methods) that will aid the designer of a Web application that should allow tagging, with the design of this Web application.

What we have analyzed next is in what way the previously mentioned Web design methods support tools, concepts or methods that will aid the designer with specifying the tagging functionality when designing a Web application. However after analyzing the specifications of the Web design methods, no such tagging aids were discovered.

1.8 Conclusion

In this chapter we presented the fact that the amount of information on the Web is expanding in a high degree. This information needs to be organized in some way, if we ever want to be able to look up and recall available information. By adding metadata to the information we are able to give a certain meaning to the information, which aids further organization of the information and improves browsing navigation and searching.

Two classification schemes, taxonomy and ontology, were introduced to help us organize information. When we looked at the Web, we found out that these classification schemes are subject to some disadvantages. Therefore, it is necessary to find other means for organizing information on the Web. We found an alternative way of creating metadata: folksonomies. In folksonomies, metadata is created by annotating resources on the Web with tags in a social platform.

Next to metadata schemes, we have considered Web design methods, which allow a Web application designer to systematically build a Web application. These Web application design methods improve the quality of the design and the maintainability of the Web sites. We have summarized a few Web application design methods.

We have analyzed the Web application design methods and looked for means that will facilitate the design of a Web application that need to support tagging. However, no such means were discovered in the analyzed Web application design methods.

Since no current research has yet been conducted towards Web application design methods and means that will facilitate the design of a tagging Web system, it seams interesting to investigate the possibilities involving primitives, methods or tools, which will aid the designer with the design of the functionality of tagging in his Web application.

In the next chapter, we will introduce yet another Web design method, WSDM. Also, this method lacks the necessary tools, methods or primitives which aim for the simplification of designing and adding tagging support to a Web application. Our solution to this problem will be extending the design method with tagging primitives, facilitating the adding of tagging support to a Web application.

2 Web Semantics Design Method

In this section, we will take a look at the Web Semantics Design Method (WSDM). WSDM, developed in 1998 [11], is an audience driven design method for Web sites. We first start by giving an overview of the design method. Afterwards, we will clarify and go deeper into the design method and its different design phases by introducing a Web system example, which we will design with the WSDM design method.

2.1 Overview

Since WSDM is an audience driven design method, this design method takes as starting point the different user groups and there requirements. Afterwards, a hierarchy between the different user groups will be formed, based on their requirements. This will lead to a basic structure from which the whole Web site structure will be derived.

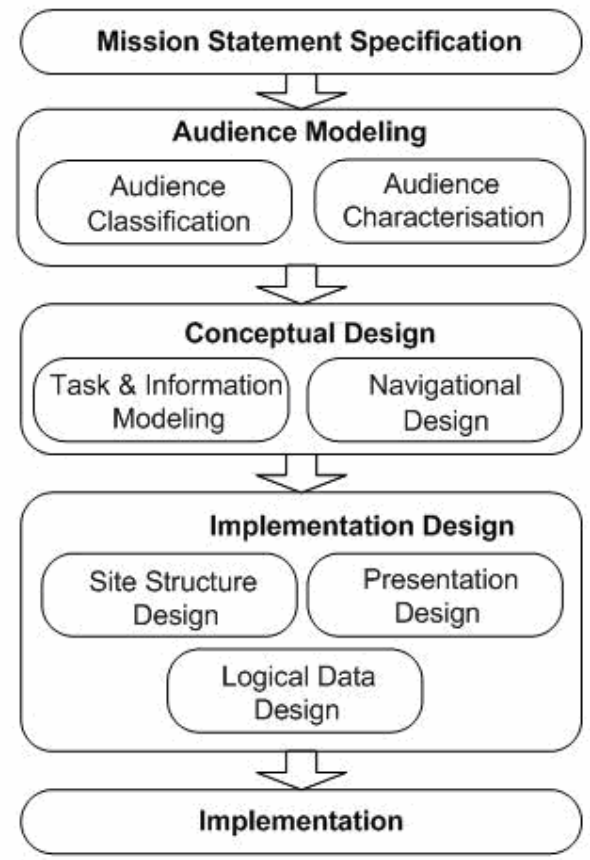

Figure 7: WSDM phases.

The WSDM design process is build up of different phases. The first phase is the *Mission Statement*. During this phase we specify the goal, the subjects and the target audiences of the Web site. This way we specify the boundaries for the design process.

The second phase is the *Audience Modeling* phase. Based on the previous phase, all the different target audiences will be identified and classified into *Audience Classes*. The classification is based on the user requirements (information, functional and usability requirements). End-users sharing the same information and functional requirements will be classified into one audience class. End-users containing extra information and functional requirements will form audience subclasses. This way a hierarchy will be formed between the different audience classes.

Next we have the *Conceptual Design* phase. The purpose of this phase is to give a conceptual description of the content, functionality and structure of the Web site. This is done in two sub phases: *Task & Information Modeling* (defining the information and functionality) and *Navigational Design* (defining the structure). During task modeling all tasks will be defined for the different audience classes based on the their requirements. For each requirement a *Task Model* will be defined which will form a hierarchical structure of relations between elementary tasks. Afterwards an *Object Chunk* will be defined for each elementary task which models the elementary task's information or functionality.

During *Navigational Design* the conceptual structure of the Web site will be modeled and it will define how the end-users will navigate through the Web site. For each audience class a *Navigation Class* will be defined based on the task models. This results in a navigation class diagram containing nodes which represent the different elementary tasks and links connecting the nodes.

The following phase, the *Implementation Design*, completes the conceptual model which makes it possible to create the Web site. This phase defines the different page structures and the look and feel for the pages. Finally, for data intensive Web sites, mappings between the conceptual data model and the actual data resource will be defined.

The final phase, the implementation, generates the Web site. Based on all previously defined models, the Web site can be generated automatically.

2.2 Running Example

In the previous section we gave a short overview of WSDM and its different phases. Here we will present an example to clarify the different WSDM steps. In our example, we will design a system that will be able to track problems and bugs for different software projects. A full description will now follow.

Users of the Web site should be registered for one or more projects. A registered user can report bugs and problems for the systems he is registered for. For each problem the user should give a description of the problem and the context in which the problem appeared. These users also have the ability to browse existing problems. When browsing existing problems, the users can see the state of a certain problem. When the problem has been solved they can see the date it has been solved.

Each project has a project manager who can adapt all information about the project. Project managers can schedule problems and change the status of certain problems. Different problems that have been reported may actually indicate the same problem. Therefore, the project manager can cluster problems, in which case the status, planning, solution,… must only be given once.

A project can also have one or more supervisors. Supervisors can see all information about the projects for which they are registered. A supervisor can also enter remarks to a problem (like a possible solution) or to a solution (like the fact that the solution didn't work). This information is only visible for supervisors of the project and for the project manager.

Project managers as well as supervisors can also report bugs and problems just like normal users can.

2.3 Mission Statement

The mission statement is the first step of the whole WSDM development process. During the mission statement we first of all define the *purpose* of the Web site. Without defining any clear purpose for the Web site there will be no proper basis for making design decisions or for evaluating the effectiveness of the Web site [10]. After stating the purpose we will describe the subjects of the system. Even if the purpose of the system is already clear we will add the subjects. This will prevent any misunderstandings. Finally the target users are defined. It is impossible for a designer to design a Web site for all the users visiting a Web site. Therefore we define the relevant end users and design the system based on there needs. Of course these end users can also be derived from the purpose, just like the subjects, but again to avoid misunderstandings we explicitly add the target users.

When we analyze our example we can define the following mission statement:

> *To build a system to track bugs and problems for software projects by allowing registered users to browse through the projects and problems and to report problems for projects. The management of the projects and all information which is included with these projects (problems and solutions) will be handled by the project manager. Furthermore supervisors can observe projects and add remarks which adds some quality control to the projects.*

As we saw before the mission statement describes the purpose, the subjects and the target users. These three aspect have to be explicitly defined::

- **Purpose**: to build a bug/problem tracking system for software projects by:
  - Managing projects and all there information.
  - Letting supervisors control projects.
- **Subjects**:
  - Projects
  - Problems
  - Solutions
  - Remarks
- **Target Users**:
  - Software Users
  - Project Managers
  - Project Supervisors

2.4 Audience Modeling

During the mission statement, we have given an informal and far from, complete description of the Web system. Since WSDM is an audience driven design method, our first concern here is the set of target users. We will therefore refine the target users identified in the mission statement into *audience classes*. This is done into two sub-phases: *Audience Classification* and *Audience Characterization*. During audience classification, we identify the different user types and classify them into audience classes. Afterwards, we will specify relevant characteristics for the audience classes during the audience characterization sub-phase.

2.4.1 Audience Classification

The goal of the audience classification is to refine and classify the target users which were identified during the mission statement into audience classes. When classifying the target users we will group all the users together sharing the same information and functional requirements.

In order to discover the different audience classes we use a method described in [10]. This method uses the activities of the organization which fulfill the Web system's purpose and the role people play in these activities. The following description describes the different steps which build up the method for discovering the audience classes:

Step 1: Consider the activities of the organization related to the purpose of Web system.

Step 2: For each activity:

1. Identify people involved.
2. Restrict them to the target users.
3. Identify their requirements.
4. Divide them into audience classes based on different information or functional requirements.
5. Decompose the activity if possible and repeat Step 2.

Given the description of the method we will now apply this method to our example. In Figure 8 we present the different activities which are involved in our example. We see that we have three different activities and for all these activities we identified the people who are involved. Apparently these are also exactly the people who represent our target users. For all these target users we identify the requirements. For the 'Software Users' we have the following requirements:

- Functional requirements:
    1. Browse/search projects.
    2. Browse/search problems.
    3. Report problem.
    4. Register for project.
- Informational requirements:
    1. Information about projects, problems and solutions.

Both the Project Managers and Project Supervisors share the same functional and informational requirements as the Software Users. On top of these requirements they have some additional requirements. We will now give for both target users some additional requirements:

Project Manager:

- Functional requirements:
    1. Maintain projects, problems and solutions.
    2. Define supervisors.
    3. Group problems.

- Informational requirements:
    1. Information about projects, problems, solutions and remarks.

Project Supervisor:

- Functional requirements:
    1. Add remarks.
- Informational requirements:
    1. Information about projects, problems, solutions and remarks.

Based on the functional and informational requirements we can now divide the target users into different audience classes. All the target users sharing the same information and functional requirements are grouped into one audience class. Target users having additional requirements will be grouped together in a so called audience sub-class. This results into the audience class hierarchy which is shown in Figure 9.

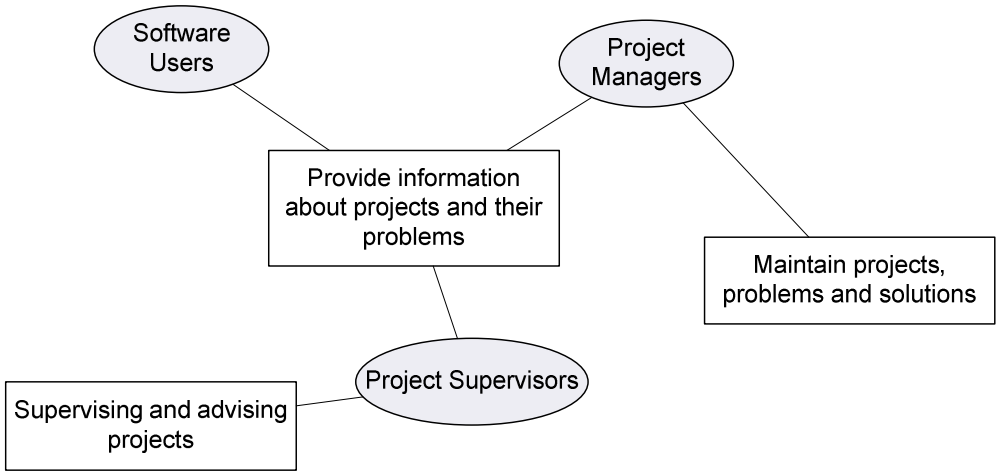

Figure 8: Bug Tracking Activities.

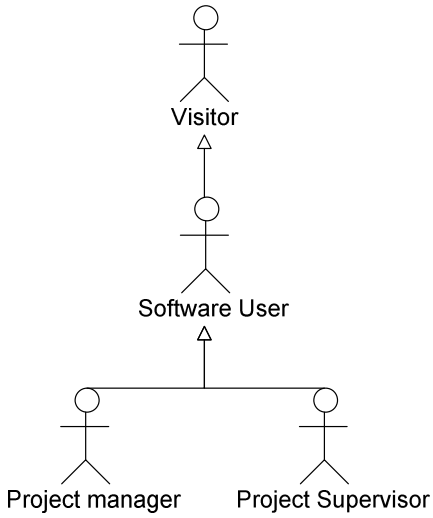

Figure 9: Audience Class Hierarchy.

2.4.2 Audience Characterization

In the second sub-phase of the Audience Modeling, the Audience Characterization, we specify relevant characteristics for the different audience classes. Examples of characteristics are experience related to the use of internet, age, language, education, lifestyle,… These characteristics can later on be translated into usability requirements or will affect the design choices regarding the "look and feel" of the different Web pages, e.g., choice of colors and fonts. Since there are no remarkable characterizations for our example we will now continue with the next phase of the WSDM design process.

2.5 Conceptual Design

At this point we have identified all the requirements and characteristics of the different target users are identified and different audience classes are defined. The goal of the conceptual design is to turn these informal requirements into high level and formal descriptions from which later on the Web system can be generated.

During Conceptual Design we concentrate on the conceptual "what and how" of the Web system. The conceptual "what" is covered by the Task Modeling sub-phase which models the information and functionality. The conceptual "how" is covered by the Navigational Design sub-phase which models the navigational structure of the Web system.

2.5.1 Task & Information Modeling

In the Task & Information Modeling phase we will define all the tasks all the users of the different audience classes need to be able to perform together with the information and the functionality which is involved with each task. Task & Information Modeling is based on the requirements we found during Audience Classification. Our first step here is to define tasks

that will allow us to satisfy the requirements. We model the different tasks by using an extended version of Concurrent Task Trees [33]. A full description of this extended CTT modeling technique is available at [10]. The main idea here is that we decompose a task into several elementary tasks. This results in a hierarchy of related elementary task together forming a *task model*.

In our example we have the following requirements:

1. Browse projects
2. Register for project
3. Report problem

Since the last two requirements usually follow after browsing through the projects, we design all three requirements into one task model. The result of this task model we show in Figure 10.

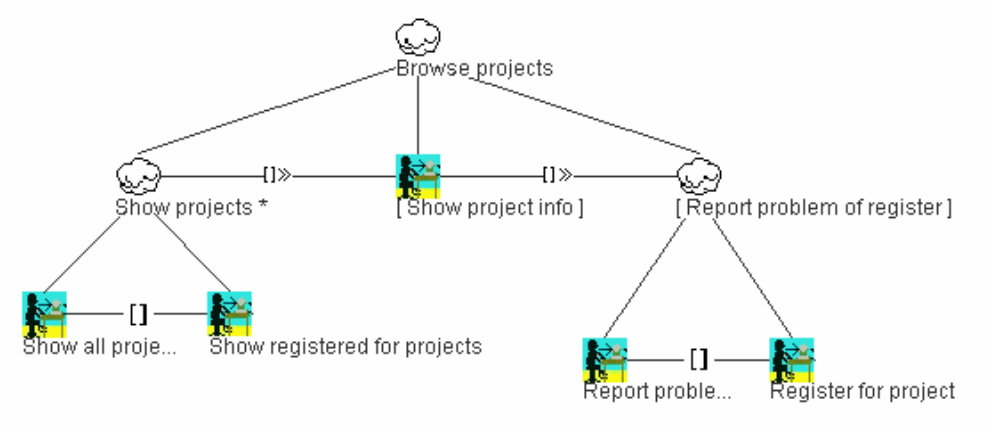

Figure 10: Task Model for Projects Browsing.

When all requirements are modeled by task models, we create *Object Chunks* for each elementary task in the task models. These object chunks model the information or functionality which is needed by the user when performing the elementary task. When we model the information, a standard conceptual modeling language can be used. For this purpose WSDM uses OWL. OWL is becoming the standard for the Semantic Web. The use of OWL for modeling the information in the object chunks makes an integration of the object chunks with existing domain ontologies possible and in case no relevant ontology exist the object chunks can be made available as ontology. Since there isn't yet any generally used graphical notation available for OWL we use the ORM's [18] graphical notation as a graphical representation for OWL.

However, when we want to model functionality we need a language which can model data manipulation. Since ORM is specifically developed for modeling data, ORM's graphical notation has been extended with some extra primitives to model the most commonly used functionality for Web systems. A full description of this extended version of ORM for the use of WSDM can be found at [9].

Let us now continue with our example and consider the following two elementary tasks:

1. Show all projects.
2. Report problem

The first elementary task is associated with an informational requirement. For the requirement we model the information that we show the users for all the listed projects. The result is shown in Figure 11.

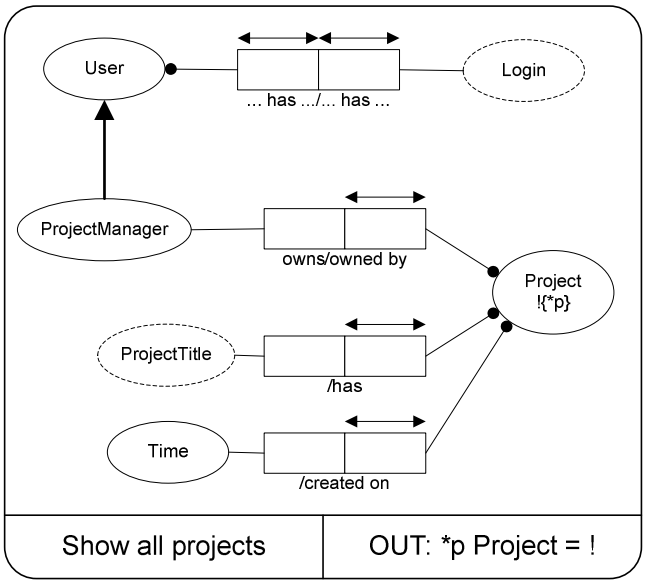

Figure 11: Object chunk for 'Show all projects' task.

The second elementary task on the other hand is associated with a functional requirement. Here, we model that a new problem must be created for which the users needs to specify some parameters. The result is presented in Figure 12.

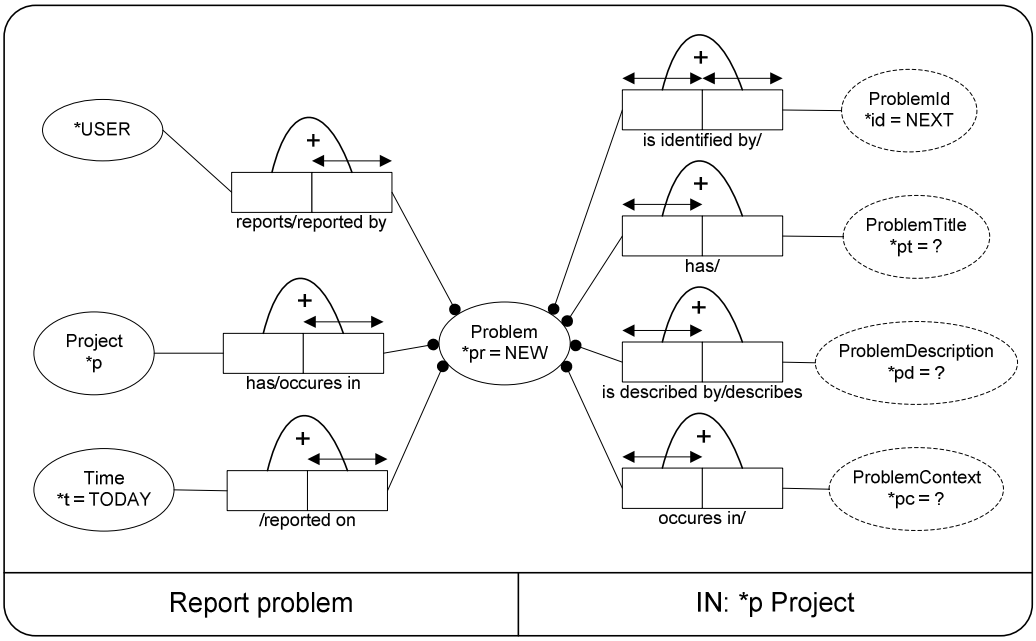

Figure 12: Object chunk for 'Report Problem' task.

2.5.2 Navigation Design

The goal of the Navigation Design is to define the conceptual structure of the Web system and to model how the users from the different audience classes can navigate through the Web system and perform their tasks [10]. For each audience class we define a *Navigation Track*. Such a track contains and defines all the information and functionality available for all the users of a certain audience track. All the audience tracks together (which are connected by *structural links* [8]) form the conceptual navigation structure. The outcome of this design phase is the *navigational model* which is expressed in terms of components and links connecting these components. For our example the conceptual navigation structure is represented by Figure 13.

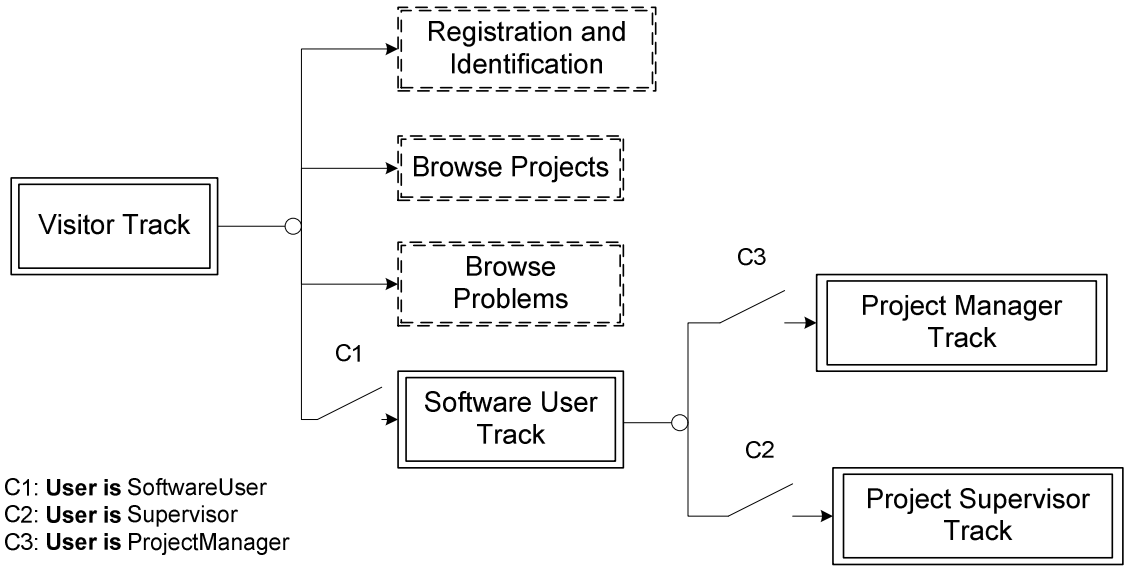

Figure 13: Conceptual Navigation Structure.

For constructing the navigational tracks for each audience class, we create a *task navigational model* for each task available for the audience class. The creation of the task navigational models are actually transformations from the task models (define in the Task Modeling phase) into task navigational models. Every elementary task is represented by a node. For each node, links define the following nodes and the object chunks are connected which define the information or functionality necessary for succeeding the elementary task. Figure 14 defines the navigation track following from the task model we defined earlier in this section.

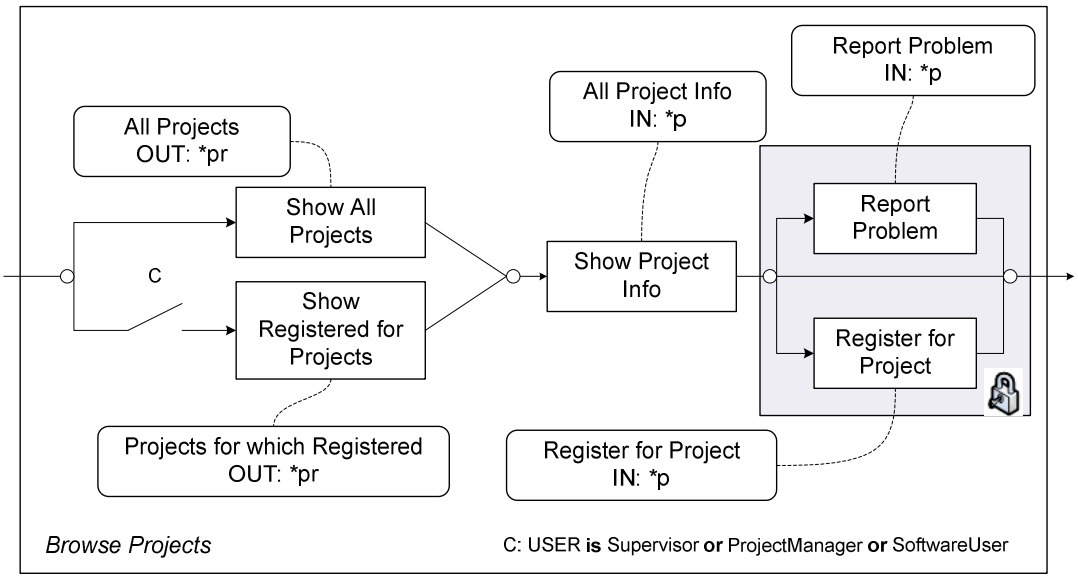

Figure 14: Navigation Track for browsing projects.

During the navigational design phase, it is possible to enhance the navigability by introducing semantic links. Semantic links are based on the semantic relations between entities. Lets consider in our example the projects concept. When showing the information of a project we also get a short list of its problems. It would be nice if we could browse to the information of these problems. We can obtain this by adding semantic links between the object chunk that shows the information of a project and the one that shows information of a problem where both object chunks contain the semantic relation between projects and problems. It is of course possible to further extend our semantic links by adding semantic links between projects and solutions and so on. Figure 15 shows a part of the semantic links between the object chunks for our Bug Tracking Web system.

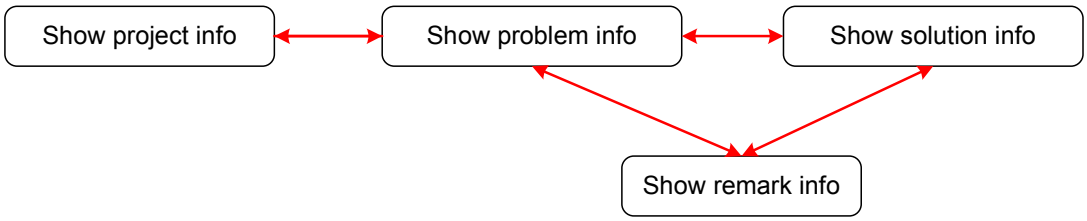

Figure 15: Semantic links between semantically related entities enhance the navigability.

2.6 Implementation Design

The goal of the implementation design is to complete the conceptual design with the necessary details in order to be able to implement the Web system. These necessary details are modeled by using three sub-phases: *Site Structure Design*, *Presentations Design* and *Logical Data Design*.

During the Site Structure Design we will define which information will be grouped into pages by grouping the components from the navigational model.

Next, during Presentation Design, the "look and feel" of these pages will be defined. In order to maintain a consistent look and feel among the pages, templates will be created. These templates contain presentation concepts for setting the structure of a page and which kind of data the page contains.

Finally, during the logical data design, we have to define a mapping between the *reference ontology* and an actual *logical data source*. The reference ontology was incrementally created during the task modeling and is formed by grouping all the object chunks into one ontology. This reference ontology will serve as the conceptual schema for the data provided by the data store. The object chunks, which are basically conceptual queries, need to be transformed into queries for the logical data source. When there is no need for a database, logical data schemas for XML, RDF definitions, etc. can be used as well.

2.7 Implementation

The final phase of WSDM, the implementation, takes care of the actual realization of the Web system by using a certain implementation environment (HTML for example). Based on all the information coming from the models resulting from the previous design phases and based on the available tools, the Web system can be generated automatically.

2.8 WSDM Ontology

With the emergence of the semantic Web, WSDM has been adapted to the support the development of semantically annotated Web sites [5]. WSDM supports semantic annotation not only of the content but also of the structure of the Web site. In order to achieve this, an ontology for WSDM was created, which is called the WSDM ontology[17]. This ontology formally describes the concepts used in the different phases of WSDM by using the ontology modeling language OWL. Furthermore, OWL is used as the conceptual modeling language for the content and the functionality of the Web site. This ontology serves as the meta-model for WSDM. Designers need to fill the meta-model, and based on these meta-model instances, new Web sites can be created during the implementation phase..

2.9 Conclusion

In this chapter we presented the WSDM Web application design method, a user-centered design method for designing Web applications with six different sub sequential phases: the Mission Statement, Audience Modeling, Conceptual Design, Implementation Design and finally the implementation. All phases were introduced and explained using a running example, the Bug Tracking Web application.

As we have seen during the explanation of the different phases of WSDM, no specific means for adding tagging support to a Web application are available. Our goal now is to introduce these means, by creating and adding tagging modeling primitives into WSDM, which will aid the designer with the design of a tagging Web system.

[17] The latest version of the WSDM ontology is available at:
http://wise.vub.ac.be/ontologies/WSDMOntology.owl

3 Extending WSDM With Support for Tagging

Our goal is to introduce a framework that helps designers and developers designing Web systems that support tagging. In the previous section, we have introduced WSDM. WSDM is a Web site design methodology that starts from the specification of the requirements for the target audiences. Afterwards these requirements will form the basis for the creation of the content, the functionality and the structure of the Web system.

We will further on use WSDM and extend this design method with the necessary modeling primitives that will allow Web system designers and developers to easily build Web systems that support tagging. In order to find these primitives it is first of all important to know all possible requirements that can come along when building a Web system that supports tagging. For this purpose, we will first analyze a representative set of Web sites and analyze the requirements which involve tagging support. Based on these requirements we will afterwards create the necessary tagging modeling primitives.

3.1 Tagging Web Systems Requirements

We first start with an analysis of a variety of tagging systems on the Web. With this analysis we can then construct a list of properties and requirements which are typical for tagging Web systems.

3.1.1 Tagging Web Systems Analysis

In choosing the Web sites we first of all based ourselves on those Web sites which took an active role in past research related to folksonomies and tagging, e.g., Delicious[15, 30, 40, 47], Flickr[29, 30, 45], Technorati[4], and those Web sites which are commonly referred to by this kind of research, e.g., 43Things[21, 34], BibSonomy[21, 36], CiteULike[15, 28, 29], MyWeb [15, 48] and YouTube[15, 29]. The other Web sites were chosen according to our own research and chosen based on differences they showed in supporting tagging. This way, we have tried to take as much differences regarding possible requirements for Web applications supporting tagging as possible into account. We now present a list of the analyzed Web systems together with a short description of each Web system.

- http://www.43things.com: allows users to share goals in life, whether you already achieved them or still want to reach them.

- http://www.bibsonomy.org: a Web system for sharing social bookmarks together with publications.

- http://www.citeulike.org: a publications sharing Web site.
- http://del.icio.us: allows users to store and share their bookmarks. This way their bookmarks are accessible at every system connected on the internet.
- http://www.flickr.com: a photo sharing Web site.
- http://www.furl.net: allows users to store their bookmarks. Furl.net stores a copy of the original Web page instead of only the bookmark like Delicious does.
- http://www.last.fm: builds up a profile of users' musical tastes based on their listening pattern. Next to that, it provides information about artists, albums and songs, accessible for all the Web site's visitors.
- http://myweb.yahoo.com: Social bookmark Web system created by Yahoo!.
- http://www.plazes.com: allows users to share specific locations based on coordinates from Google maps.
- http://www.technorati.com: a search engine for blogs.
- http://wise.vub.ac.be/wiki: a WIKI aimed for students who are subscribed for the Web Information Systems course at the Vrije Universiteit Brussel, based on the TWiki engine.
- http://www.youtube.com: a video sharing Web site where users can upload, view and share their videos.

3.1.2 Tagging Requirements

After analyzing the different Web systems, we can create a list of requirements relevant for such tagging Web systems. We will present a list of requirements, collected from all the Web systems' analysis's. This list of requirements will not contain all the requirements for each and every Web system but we will give a summary of the most important and general requirements, which will give a broad view of all possible requirements for a tagging Web system. Furthermore we will divide the requirements into three different categories [7]. First we have the information requirements. These requirements concern the information that will be provided to the end-users and the information the end-users will be working with. Furthermore, we have the functional requirements that will describe the functionalities provided by the Web system. Finally we have the navigational requirements that will describe the navigation in the Web system.

Before we start with describing the requirements regarding the tagging support of a Web system we will recall the structure of a folksonomy as we defined it in our background

chapter. A folksonomy is build up by a flat namespace of *annotations* where each annotation relates a *user* annotating a resource, the *resource* being annotated and the *tag* describing the resource. This way we will try to recall and clarify the differences between the use of the terms annotation and tag in the following requirement sections.

Informational Requirements

When we analyze the different Web sites, the first thing we notice regarding informational requirements is that it we will be necessary to handle a large variety of resources. On some Web sites we encounter only one kind of resource (for example: photographs on Flickr, videos on YouTube, Google-Maps locations on Plazes) while other Web sites provide a variety of resources (for example: artists, songs and albums on Last.fm, bookmarks and citations on BibSonomy).

A lot of Web sites show a list of tags based on all the annotations in the Web site. Displaying a set of tags based on all the annotations is usually done by displaying the set of most popular tags occurring in the Web system (for example in Flickr, Delicious, Technorati), i.e. those tags which are most commonly used in all the annotations.

When displaying a resource, information describing the resource is provided. Together with the information, a set of tags are shown annotating that particular resource. Instead of showing all the existing tags from the Web system, we only want to show those tags that annotate the resource. We therefore restrict the set of annotations to those annotations annotating that particular resource and we display the tags occurring in the resulting annotations.

Multiple people can annotate the same resources. It is therefore important that we take into account whose tags we want to display. It is for example possible that we want all the tags of a certain user to be displayed, e.g., a personal user page in Delicious (see section 1.6.1) presents an overview of only those tags which are used by the user. In another example, we will want to display the tags of a specific group of users, e.g., CiteULike provides the opportunity to create groups and join groups. On the group Web page, all the tags used by any group member are shown. In all these cases we restrict the set of annotations by those annotations being made by specific users.

It is furthermore possible we want to show the tags of a certain resource given by specific users, e.g., Web systems like 43Things, MyWeb, YouTube, Plazes show personal resources with the tags the user personally made for that particular resource. We restrict therefore the resources and the users for the returning annotations and show the tags of these annotations.

As pointed out before, tags and annotations are two different kinds of information; tags are related to users and resources by these annotations. We already required tags to be displayed based on annotation sets restricted by users or/and resources. We will now add the

requirement that we can show resources being annotated by a tag and/or users using the tag. This results in most Web sites as a sort of tag Web page where all relevant resources are displayed (CiteULike, Flickr, Last.fm). These relevant resources are the resources being annotated by the tag. Furthermore, a list of users who use the tag can be displayed (Delicious).

In the TWiki Web system, we encountered the possibility to display a list of tags, not only those tags annotating resources but also tags possibly not being used, meaning not annotating any resources. The reason for this is that users first need to enter tags into the Web system. When a user wants to annotate a resource afterwards, he can only pick tags from the list of earlier added tags that is available for each user.

As we can conclude from this section, we don't need to display annotations themselves. We should allow displaying the tags, the resources and the users of the annotations we retrieved and display the necessary tags with the necessary resources and/or necessary users based on the annotations.

Functional Requirements

When we examine the functionalities that apply to tagging, we can see that we need to be able to annotate resources by annotating these resources with tags, we need to be able to delete annotations or we want to be able to edit annotations by changing the tags.

We want to be able to add new annotations or delete existing annotations (both functionalities occur in all analyzed Web systems, except for Last.fm where deleting annotations wasn't possible by the end-users). In order to be able to add a new annotation we require three parameters: the user, the resource and the tag. The same applies for deleting annotations. Furthermore, for an annotation it might be possible we want to change the tag into another tag (e.g., as in Delicious, Flickr). In this case we need an additional parameter describing the new tag next to the three earlier mentioned parameters (resource, user and tag) defining the particular annotation to be edited. Note that when adding, deleting or editing annotations, not all users can add, delete or edit just any annotation. In some Web sites, every member is allowed to tag any resource (Technorati, CiteULike, Plazes), in other Web sites, only creators of resources are allowed to tag their own resources (YouTube) and/or other Web sites allow members of certain groups to tag resources (Flickr). In all analyzed Web sites, users need to become members at least, in order to annotate resources and only the creators of annotations can delete or edit their annotations.

In cases where we want to add, delete, edit annotations or find resources annotated by certain tags, we need ways for the user to specify tags. First of all we can request a tag by letting the user enter a tag. Another way of specifying a tag is by first listing a set of tags, e.g., the most popular tags or all tags that annotate a certain resource, and then let the user select one of the tags. After specifying a tag, two options follow. The tag will or will not be stored by the Web

system. The first option is necessary when we add a new annotation. We, of course, want to keep track of the annotating tag. But, when for example, we want to lookup all the resources which are annotated with a certain tag, we can let the user specify relevant tags but we don't require these specified tags (being part from the query) to be stored by the system for later use. So how we specify tags together with adding tags, will basically form the functionalities regarding tags.

Navigational Requirements

As pointed out earlier during this document, tags enhance the browsing possibilities in a Web system. This may improve discovery of (related) resources in the Web system. Tags also make it easier to find resources which are difficult to find with classic text searches, e.g. pictures (Flickr) or videos (YouTube). It is therefore necessary to include ways to browse from a tag to a set of related resources.

3.1.3 Requirements Overview

In the previous sections we described all the requirements that could appear in a tagging Web system. We will now give a full list of all these requirements to get a clear overview. Afterwards, we will specify which of these requirements we can find back in the Web sites we analyzed.

Informational Requirements:

1. Show all annotations.
2. Show a subset of all annotations restricted by specific user(s).
    a. Show the tags of the active user of a session.
    b. Show the tags of a certain user.
    c. Show the tags of a certain group of members. Usually a group defined by a user specifying his/her friends, family, etc.
3. Show a subset of all annotations restricted by specific resource(s).
    a. Show the list of tags annotating a certain resource.
    b. Show the list of tags of a certain group of resources. This is commonly used in systems where only public resources and there tags are visible for the public.
4. Show a subset of all annotations restricted by specific tag(s).
    a. Show the set of resources which are annotated by a certain tag.
    b. Show the users who used a certain tag.
5. Show all tags.

Functional Requirements:

6. Add a new annotation.
    a. Each visitor can annotate resources.
    b. A member of the Web application can annotate all resources.
    c. The creators of a resource can annotate a resource.
    d. Members of certain groups can annotate a resource.
7. Delete an existing annotation (only by the creators of an annotation).
8. Edit an annotation by changing its tag (only by the creators of an annotation).
9. Add a new tag into the Web system.
10. Specify a tag by selecting a tag from a list of tags.
11. Specify a tag by direct user input.

Navigational Requirements:

12. Browse from a tag to relevant resources or users.
    a. Possible to browse from each displayed tag.
    b. Not always possible to browse from a tag.
    c. Never possible to browse from a tag.

We will now summarize the analysis of the tagging Web sites by specifying which requirements apply to which Web sites. For this summarization we will use a requirements matrix (Table 1). Note that we didn't always use the full description of each requirement but instead we used the numbers of requirements. A full description of the requirement can be found in the previous list.

3.1.4 Conclusion

We presented a list of possible requirements for a tagging system. We divided the requirements into three different categories: informational, functional and navigational requirements. This division will help us find out in which steps of WSDM we should provide tagging modeling primitives for the corresponding requirements. In the next section, we will repeat the WSDM design phases and describe the modeling primitives we added in each step in order to design a tagging Web system.

Web site	Resources	Informational requirements									Functional Requirements											Navigational Requirements		
		Annotations								Tags	Annotations										Tags			
		1	2			3		4		5	6						7		8		9	12		
			a	b	c	a	b	a	b		a	b	c	d	10	11	10	11	10	11		a	b	c
43Things	Goals in life	X	X	X		X		X				X	X			X	X				X	X		
BibSonomy	Bookmarks, Publications		X	X		X	X	X				X	X			X		X		X	X	X		
CiteULike	Bookmarks	X	X	X	X	X		X				X	X	X		X		X			X	X		
Delicious	Bookmarks		X	X		X	X	X	X			X	X		X	X	X	X		X	X	X		
Flickr	Pictures		X	X	X	X	X	X				X	X	X	X	X	X	X		X	X	X		
Furl	Bookmarks		X			X	X	X				X	X		X	X	X	X			X		X	
Last.fm	Artists, Albums, Songs	X	X			X		X				X	X		X	X					X	X		
MyWeb	Bookmarks		X	X	X	X	X	X				X	X	X		X		X			X	X		
Plazes	GoogleMaps coordinates		X		X	X	X	X				X	X	X		X	X				X	X		
Technorati	Blogs	X	X			X		X				X	X			X	X				X	X		
WIS-Wiki	Pages	X	X			X		X		X		X	X		X		X				X	X		
YouTube	Videos		X			X		X					X			X		X			X	X		

Table 1: Web sites analysis: which requirements apply to which Web site.

3.2 Tagging Modeling Primitives in WSDM

In chapter 2, we discussed how to use WSDM as a method for designing and implementing a Web system by means of different design phases. When we want to design a Web system that supports tagging, we will encounter additional requirements concerned with the tagging functionality. Currently these requirements should be modeled by the designer manually, just like any other requirement, by using the WSDM design method as described in chapter 2.

Based on the requirements that we gathered, we will devise a set of modeling primitives that allows us to model these requirements. These modeling primitives, i.e. the *tagging modeling primitives*, will be added to the set of existing modeling primitives of WSDM. When using these tagging modeling primitives, WSDM will automatically provide tagging support in the generated Web system.

In this chapter we will review the WSDM design process. For each WSDM phase we will describe the changes necessary to add tagging support. The changes will be explained by using examples occurring in existing Web systems.

3.2.1 Mission Statement

The starting point of the whole WSDM design process is the mission statement. The goal of the mission statement is to describe three aspects of the Web system: the purpose, the subjects and target audiences. By stating the mission statement, which is done in an informal way, we clearly define the boarders of the Web system.

The level of the mission statement describing a Web system is higher than the level where the specifications for a tagging system need to be specified. For this reason no adaptations are necessary for the mission statement phase for describing tagging support for a Web system.

3.2.2 Audience Modeling

During audience modeling, the target users defined in the mission statement will be further refined into audience classes. This classification is based on the requirements from all the audience classes: target users with similar functional and informational requirements are grouped together into one audience class and target users sharing additional functional and/or informational requirements are grouped into audience sub-classes.

When adding support for tagging to a Web system, we will encounter an additional set of requirements which are aimed for the use of tagging in the Web system, i.e. the tagging requirements. In the previous section we analyzed different existing tagging Web systems in order to discover what kind of requirements come along with such tagging Web systems.

During the audience modeling phase, we separate the tagging requirements from the rest of the requirements of the Web system. For each audience class we group the tagging requirements and keep them separate from the other requirements for that audience class. This way we get a clear overview on the tagging requirements for each member of each audience class. Furthermore, by looking at the complete list of the tagging requirements of all audience classes we get a complete overview of the whole tagging support in the Web system.

As an example we will present a list of possible requirements that are involved with the tagging support in a Web system. The examples represent requirements coming from existing Web systems and we will show with which requirements, from the ones we discovered, these example requirements correspond. In this example we don't take different audience classes into account, we just present an example of possible tagging requirements.

- Functional requirements:
    - YouTube: Video creator can add (6c) and delete (7) tags from his videos.
    - Furl, Delicious: User can add (6c) and delete (7) tags from his personal bookmarks.
    - Last.fm: Users can tag (6c) artists, tracks and albums.

  - Plazes: Users can look up places by specifying tags (4a).
  - Flickr: Members of the friends group can tag pictures belonging to this group (6d).

- Informational requirements:
  - Delicious, Furl: A bookmark is shown with all the personally made annotations (2a).
  - Last.fm: The information about an artist comes along with the all tags given by every user (1).
  - Flickr: Pictures belonging to the friends group show all the tags made by the members of the group (2c).

- Navigational requirements:
  - Flickr: Clicking on a personally made tag will show all personal pictures annotated with that tag (12).
  - Delicious: Clicking on a personally made tag will show all bookmarks annotated with that tag (12).
  - Technorati: Clicking on a tag will list related blogs, video, pictures and tags (12).

After listing all the Web system's requirements, which include all the requirements concerning tagging support, the audience class hierarchy needs to be constructed. The construction of the hierarchy of the different audience classes is based on the requirements of the Web system without considering the tagging requirements. These tagging requirement don't influence the hierarchy. They only describe the tagging requirements of the different audience classes.

By classifying all the target users into audience classes based on their regular requirements and by creating a hierarchy between these audience classes, we clearly state the functionality, information and navigability for the different target users on the Web system. Next to these regular requirements, we create a separate list with the tagging requirements. By adding some tagging requirements to one audience class and not to other audience classes (not inheriting requirements from that audience class), we can describe tagging requirements being applicable to only that audience class. This is necessary for the following situation: When we look at the "adding annotation requirement" (requirement nr.6) we see that we have some sub requirements. These sub requirements define for example which groups or users can add annotations. When only users from certain audience classes can annotate certain information we define this requirement for these audience classes only. This way the audience modeling phase will tell us what information can be tagged and by whom.

3.2.3 Conceptual Design

At this point all the necessary requirements are identified and described in an informal manner. In the next phase, the conceptual design phase, we will formalize these informal requirements. In this section, we will only focus on those requirements that are involved with the tagging support of the Web system. We will now present a set of *tagging modeling primitives*, the modeling primitives that allow representing the information, functionality or navigability, needed to fulfill the requirements we introduced earlier. These modeling primitives will allow us to model the tagging requirement needs of a particular Web system.

The conceptual design phase is build up of two sub-phases: the task & information modeling and the navigational design. We will cover both sub-phases and describe the changes made and primitives added for modeling tagging support in a Web system.

3.2.3.1 Task & Information Modeling

During the task & informational modeling phase we give a conceptual description of the information and functionality of the Web system. The information and functionality of the Web system is covered by the information and functionality requirements which were identified and informally described in the previous design phase. In this phase we formally model the information and functionality required to fulfill these requirements.

First, the different tasks will be defined based on the functional and informational requirements that were defined during the audience modeling phase. Each task will be decomposed into elementary sub task. Afterwards, for each and every elementary task an object chunk will be created. The object chunks define the necessary pieces of information or functionality, required for the corresponding tasks. These pieces of information and functionality are modeled by the conceptual modeling language OWL. For modeling the information graphically, ORM is used which will be extended with an additional set of data manipulation concepts in order to model the commonly used functionality on Web sites.

We will now extend WSDM with a first set of tagging modeling primitives. The tagging modeling primitives we present in this section have a graphical representation. Afterwards we will describe how we have updated the WSDM ontology in order to implement these tagging modeling primitives into WSDM.

Since we are currently working in the task & information modeling sub-phase, the tagging modeling primitives we will introduce here will represent the information and functionality required to fulfill the informational and functional tagging requirements discovered during the audience modeling phase. We will first start with the informational tagging requirements and afterwards we will handle the functional tagging requirements.

3.2.3.1.1 Informational Tagging Primitives

As noted before, a folksonomy behind a Web system with tagging support is build up by a set of annotations which are typically build up of three parts: a user making the annotation, the resource being annotated and the tag annotating the resource. When modeling the information of a Web system we need to model all this information: the annotations, the users, the resources and the tags.

In our approach for modeling a tagging Web system we will provide a set of tagging modeling primitives which will represent the tags and the annotations. This avoids the manual modeling of the tags and the annotations by using standard modeling concepts available in WSDM by the designer.

- `T` We will use the character "T" for representing the list of all the existing tags/keywords in the Web system.

- Ⓣ An encircled character "T" will represent the list of all annotations. By encircling the tags primitive "T" we state that we return all the annotations based on all the tags.

Regarding the informational tagging modeling primitives we just presented, we have two remarks. First of all it can be possible that we want to refer to a specific tag or annotation. For this purpose we can use the current WSDM notation for referring instances (the referent notation). For referring to a specific tag we thus get the following notation `*T` and for pointing to a specific annotation we have *Ⓣ.

For our second remark, we would like to point to the tagging requirements 2, 3 and 4 where we display tags, resources or users, based on a set of annotations with restrictions towards tags, resources or users in these annotations. We introduced Ⓣ as being the primitive which refers to all existing annotations in the Web system. We now need to find ways to model and to represent a set of annotations being restricted by tags, resources and/or users.

Let us start by restricting resources. When we want to restrict the resource type of all returning annotations (e.g., in Last.fm with resource types Artist, Album and Song, we for example only want the annotations on Artists), we insert Ⓣ into the concept representing the resource type. We can continue the restriction by not only specifying the resource type but also by specifying certain specific resources. This is done by specifying a WSDM referent in the concept defining the type of the resources (Figure 16). We thus return all the annotations of the resource pointed out by the referent. This is a commonly used mechanism among

tagging Web systems. We can for example return all the tags of a certain picture (Flickr), bookmark (Delicious), blog (Technorati), etc.

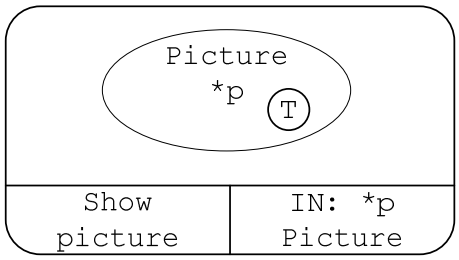

Figure 16: Returns all the annotations of a certain picture referred by the referent *p.

It should be possible not only specifying the resources of annotations but also the users that made the annotations. This makes it possible to return all the annotations a certain user made (for example the user using the Web system). In Web systems like Delicious or Flickr a user can get an overview of all his personal annotations. We can limit the annotations to only the annotations given by a certain user (Figure 17) or group of users (e.g., in Flickr it is possible to allow only friends to tag pictures and to show only their tags Figure 18) by specifying the users between the brackets of T() .

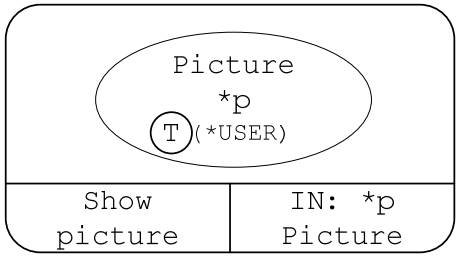

Figure 17: Returns all the annotations for picture *p given by the user.

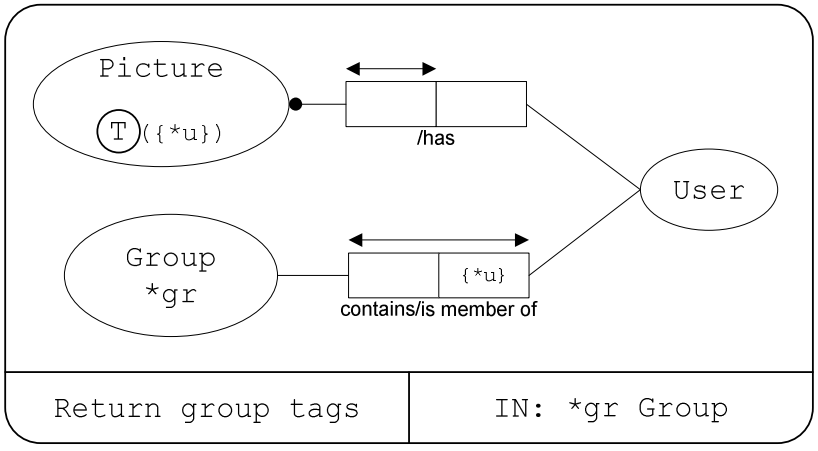

Figure 18: Returns all annotations given by the members of group *gr.

Since annotations contain three parameters (resource, user and tag) and since we can already restrict the resource and the user parameters up till now, we still need to be able to restrict according to the last parameter: the tag. Restricting by means of the tag is a common way for searching specific resources which are annotated with certain tags. By specifying tags we can find Google-Maps locations in Plazes, articles in CiteULike, pictures in Flickr and this list of examples can go on.

Our presentation of all annotations (T) returns a list of all annotations based on all tags `T`. When we want to restrict the set of tags occurring in our annotations, we specify the desirable tags within the circle of the annotations primitive (T) (Figure 19). For example, we can specify the tags occurring in the annotations by user input (T?) or by selecting a tag from the list of tags (T!).

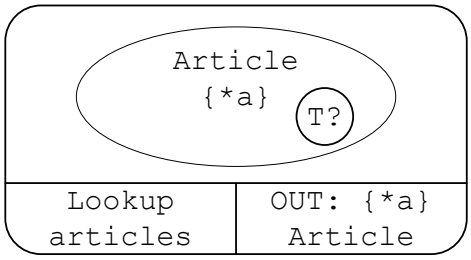

Figure 19: We request all the articles annotated with a specific tag.

So far we have presentations for the tags and the annotations. But we not only want to represent these pieces of information. We also want to be able to model functionality based on tags and annotations. We will now present a list of possible functionalities for tags and annotations together with their graphical representations.

3.2.3.1.2 Tag Functionalities

From our list of tag requirements we will look at the functional requirements and we will present a list with the tagging modeling primitives needed for modeling functionality related to tags.

- `T?`: Specifies and returns a tag by letting the user enter one.
- `T??`: Specifies and returns a set of tags by letting the user enter the tags.
- `T!`: Specifies and returns a tag by letting a user select one from a list of tags.
- `T!!`: Specifies and returns a set of tags by letting a user select multiple tags.

- `T+`: Adds a tag to the list of tags in the Web system. The tag that needs to be added needs to be specified first.

- `T−`: Deletes a tag from a list of tags in the Web system. The tag to be deleted needs to be specified first.

Combinations between certain concepts are possible and even required. The two tag functionality concepts add and delete, lets call these the *manipulation functionalities*, require that we specify a certain tag first. Specifying a tag is possible with the rest of the presented tag functionality concepts, lets call these *specifying functionalities*. What we need to do is combining tag manipulation functionalities together with tag specifying functionalities. For example, when we want to be able to add a new tag which will be specified by single user input, we obtain the following combination: `T+T?` for which we will be using the shortcut notation `T+?`.

3.2.3.1.3 Annotation Functionalities

The annotation functionalities, we can find from the list of functional tagging requirements. When we look back at this list of annotation requirements, we notice that we need to be able to add (requirement 6), delete (requirement 7) and edit (requirement 8) annotations. We will now introduce concepts for adding, deleting and editing annotations.

- Ⓣ+ Adds a new annotation.

- Ⓣ− Deletes an existing annotation.

- Ⓣ+/- A combination of adding and deleting annotations. A handy mechanism for specifying a list of tags. Tags appearing in the list will be added, annotations with all other tags will be deleted.

- Ⓣ->T′ Changes the tag `T` into the new tag `T′`.

When adding new annotations (Figure 20) we need to know the specific resources we add the annotations to and the specific tags we want to describe the resources with. The user adding the annotation is known by the default referent `*USER` which points to the user of a session. Specifying another user or set of users, is also possible by specifying the users between user brackets, e.g., Ⓣ+?(*u)+ .

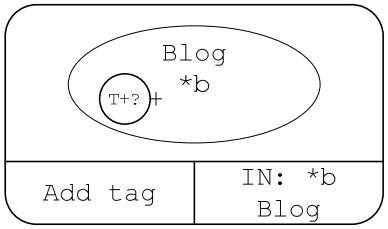

Figure 20: Adds a new annotation to the specified blog.

Just as with adding new annotations, we need to specify the resources when we want to delete annotations and the tags that are used for these annotations. With the combinatorial functionality for adding and deleting (Figure 21) we usually specify a list of tags. These tags will be added, when they aren't already annotating the resource. The annotations with tags that not appear in the list of specified tags will be deleted.

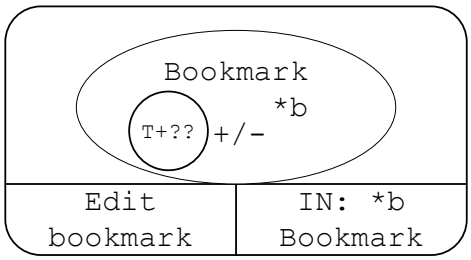

Figure 21: A bookmark is being annotated by a series of tags.

When we want to edit a tag we not only need to specify the resources and a tag. We also need to specify a second tag which will replace the first tag for all annotations annotating the resources with the first specified tag. Figure 22 shows how a specific tag of all annotations annotating the pictures of the active user of a session is changed into a new tag.

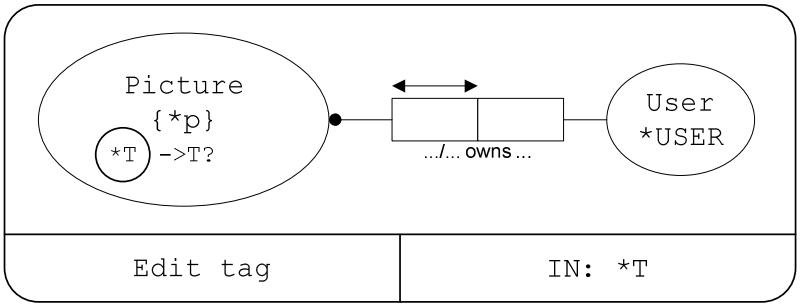

Figure 22: Changes the annotations of a specific set of pictures annotated with tag *T.

3.2.3.2 Navigation Design

During navigation design we define the conceptual structure of the Web system and model how audience class members will navigate through the Web system. For each audience class a audience track will be created which defines the functionality and information available for

the audience class members. The structure of the audience tracks will be derived from the task models created in the previous task modeling phase. All audience tracks combined form the complete navigation structure of the Web system and is called the navigation model.

When modeling the navigation model it is possible we want the ability for a certain or all audience classes to display tags based on all annotations, i.e. the set of annotations without any restrictions towards resources, users or tags. A common used application for this principal we can find in a lot of Web systems, e.g., Last.fm, Flickr, CiteULike, BibSonomy, Delicious and Technorati, as a way for showing their most popular tags. When we want to return all annotations without any restrictions and display their tags at a certain point in the Web system, we can create an empty object chunk embedding the annotations primitive. Afterwards, we add the object chunk to the corresponding node in the navigation model. In Figure 23 we illustrate such an example. Note that we used an abbreviated version of an object chunk in this example.

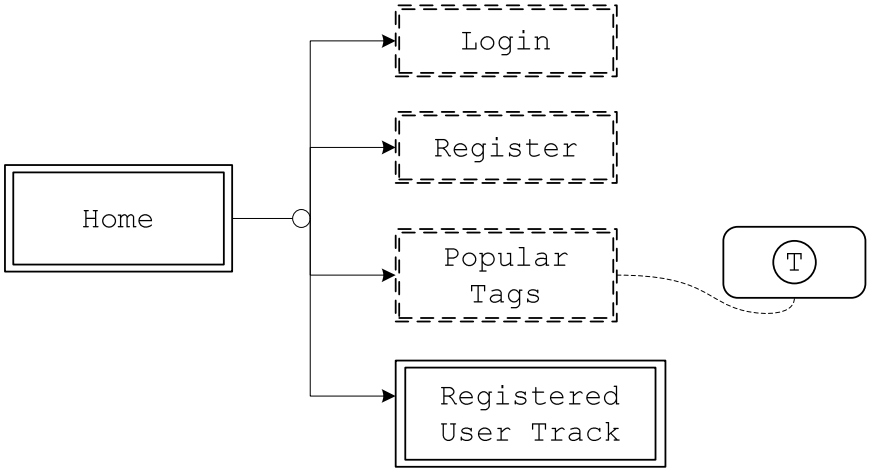

Figure 23: Shows the popular tags at a certain point in the navigation structure.

As we saw before, an advantage of tagging is the improvement of browsing and therefore navigability. By browsing through tags we get a whole list of resources which are annotated by the tags or even users which use these tags. When adding tagging primitives to WSDM we also require the ability to model this navigation behavior. To model this, we require an object chunk in which tags are displayed and are able to be selected. In the navigation model a node for this object chunk needs to be provided. Afterwards, we need to link the node to following nodes which will show resources based on the selected tag. This will allow the user to browse to a page with resources annotated by a tag, after selecting that tag.

As an example, let us consider the situation in Flickr where a user views a personal picture. Each picture is shown with a list of tags. There are two possible links for each tag. One link will show all personal pictures annotated with the tag. The other link will show all pictures in Flickr annotated with the tag. To model this we need three object chunks: one which shows

all information about a picture (Figure 24), a second which shows all personal pictures (Figure 25) and the third which shows a list of all pictures (Figure 26). In the figures we present, we only include one attribute for the picture concepts as an example. We left out the rest of the attributes for the reason of saving space.

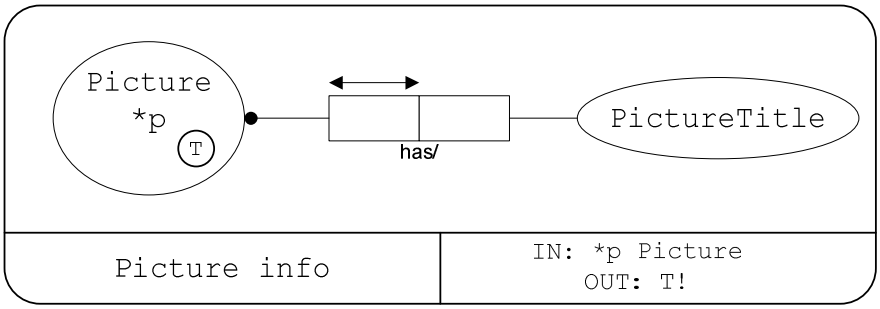

Figure 24: A picture and all its details including the tags it has been annotated with.

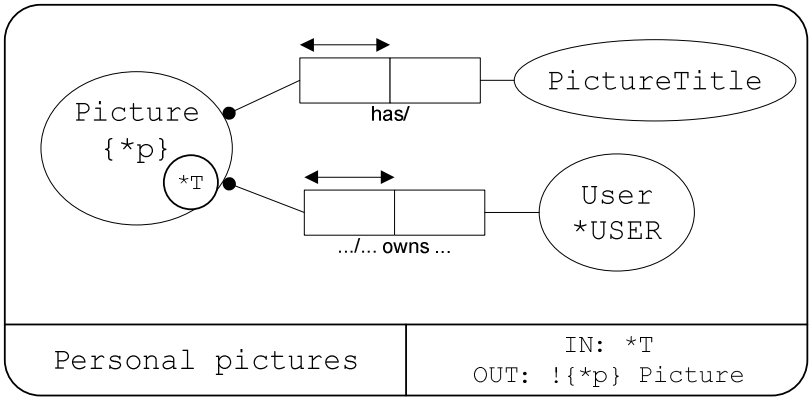

Figure 25: A list of all personal pictures which are tagged with *T.

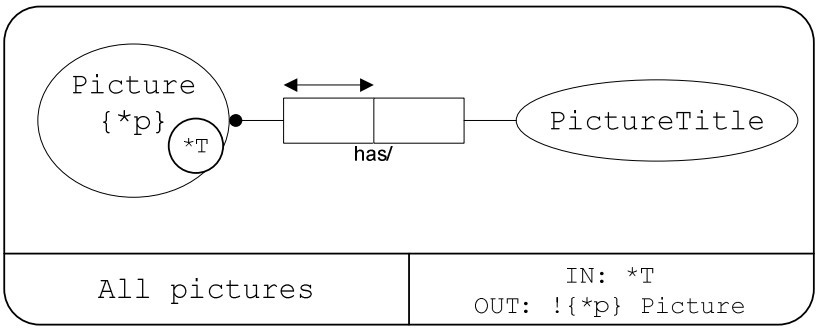

Figure 26: A list of all pictures which are tagged with *T.

Having these three object chunks, we will now connect them with corresponding nodes in the navigational model and provide the necessary links between these nodes. By connecting the nodes with the links, it is possible to browse from a tag to a list of related pictures (personal or all pictures, depending on the selected link). Figure 27 illustrates how we linked the nodes.

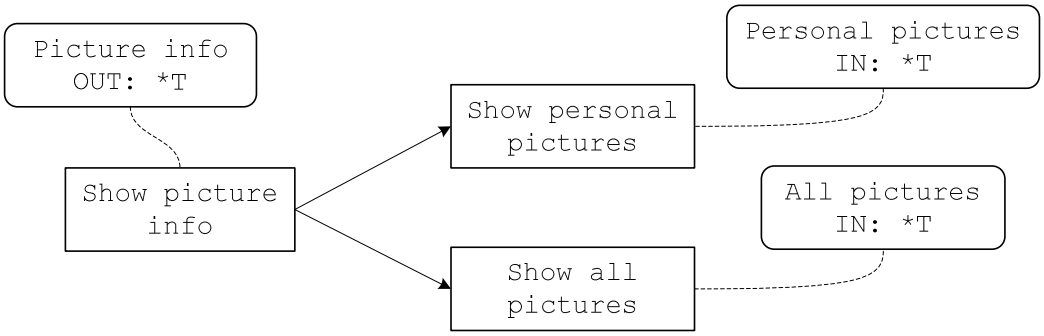

Figure 27: Navigation structure for browsing through tags.

3.2.4 Implementation Design

During the implementation design, we complement the conceptual design with the necessary details in order to be able to implement the Web site. The complementary details allow us to define a structured presentation and uniform look and feel for the Web pages and in case of a data intensive Web site a mapping between the conceptual data and the actual data store.

3.2.4.1 Presentation Design

The presentation design phase is one of the sub phases of the implementation design. During the presentation design, we define the look and feel of the different Web pages. In order to keep consistency between the different Web pages we define page templates. Afterwards, for each subsequent page we define the layout. For defining both the templates and the pages, a number of presentational and control concepts are available.

In tagging Web systems, we saw that the amount of available tags can be extremely large. It can therefore happen that for certain resources, a large amount of tags needs to be displayed. In our background chapter, we introduced some techniques for efficiently presenting a set of tags and for which we will create new concepts that we can add to the current list of WSDM presentation concepts.

In order to present a set of tags, we saw two ways to organize this:

1. By listing the tags in a (horizontal/vertical) list.
2. By creating a tag cloud for a set of tags.

For the first option we can rely on currently existing WSDM presentation concepts for creating lists. We only need to adapt these presentation concepts so that they can take a set of tags as input for which a list will be created.

The second option, the tag cloud, requires a new presentation concept to be created and added to the list of WSDM presentation concepts. As we saw, tag clouds use text properties such as size, weight or color to represent the frequency of the associated tags in a set of annotations.

Our tag cloud presentation concept contains some input parameters, defining how the tag cloud will be constructed:

- **MinTagFrequency**: The minimum frequency for a tag to be able to taken into account by the tag cloud.
- **MaxTagFrequency**: The maximum frequency for a tag to be able to taken into account by the tag cloud.
- **MaxTagsToDisplay**: The maximum amount for tags to be displayed by the tag cloud.
- **MaxTagCloudCategory**: Depending on the frequency of a tag, the text properties of the tag will have certain values, i.e. the tag frequency will define the visual appearance of the tag. Since the high amount of tags and the high variation of frequencies, not every single frequency will result in totally different appearing text with different text properties. We will therefore create different categories and depending on the tag frequency, a tag will end up in a certain category, defining the tag properties for this tag. This **MaxTagCloudCategory**-parameter will define the maximum amount of categories there will be instantiated. Afterwards, the designer will need to define which text style belongs to which tag presentation category. This can be done by assigning a certain text style to certain category by means of a Cascading Style Sheet[18].
- **SortStyle**: There are two ways to sort the tags: alphabetically or by frequency.
- **SortOrder**: The tags that can be sorted in ascending order or in descending order.
- **Annotations**: The set of annotations defining the tags to be displayed and from which the frequencies of each tag can be derived.

Besides presenting tags, we need ways to specify tags. One solution is already included in the TagCloud concept. A tag cloud shows tags and makes them selectable. We only need to specify the link to be followed after a tag selection. Furthermore, specifying a navigation reference for tags in a list with tags makes these tags also selectable.

[18] Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents (http://www.w3.org/Style/CSS/).

An other way of specifying tags is by user input. For this purpose, WSDM already provides the necessary concept, the TextBox concept. We will continue using this concept for entering tags.

3.2.4.2 Logical Data Design

During the conceptual design phase we described all the information the Web system will provide. This is done by defining object chunks for each piece of information. All these object chunks are somehow related to each other and together they form the reference ontology. This reference ontology will be considered as the conceptual data schema for the data to be provided by the Web system.

In order to store data we will need an actual data store. For the data store we need a logical data schema which describes the data in the data store. To define a logical data schema we can use, for example, a relational database schema, an XML schema, a RDF schema or even the OWL schema of our reference ontology. When constructing the logical data schema we need to keep the mapping between the reference ontology and the logical data schema in mind. Afterwards we will need to build queries and updates in the language of the logical data schema based on the conceptual queries and updates represented by the object chunks.

As we saw, for modeling tagging support, we introduced a set of tagging modeling primitives. In order to be able to define a logical database schema we need to be aware of the semantics behind the information and functionality that is represented by these tagging modeling primitives.

As all the object chunks together form the reference ontology, so will all the information represented by the tagging modeling primitives together form a certain *tagging ontology*. The tagging ontology is illustrated in Figure 28 and represents the flat namespace that builds up a folksonomy. This flat name space is constructed by a set of annotations, where each annotation relates three concepts: a user, a resource and a tag. The Tag and the Annotation concepts will be automatically provided by the extended WSDM methodology. As for the resources, we saw that it was possible to annotate each resource that we model in an object chuck. We model this by embedding the concept that models the resource, with an annotation modeling primitive (e.g., Figure 20). This means that in the tagging ontology we require a general concept which defines all the possible concepts that can be modeled in an object chunk. As we saw before, we use OWL for the modeling of the information in object chunks. The concepts we define and model in object chunks are therefore defined as OWL concepts. This means that all these concepts are actually instances of the owl:Class concept, i.e. the concept that defines all the concepts being defined by OWL. It is this concept that we require to form the general set of resources that can be annotated with tags. For the users we create a User concept which defines the users that can annotate resources.

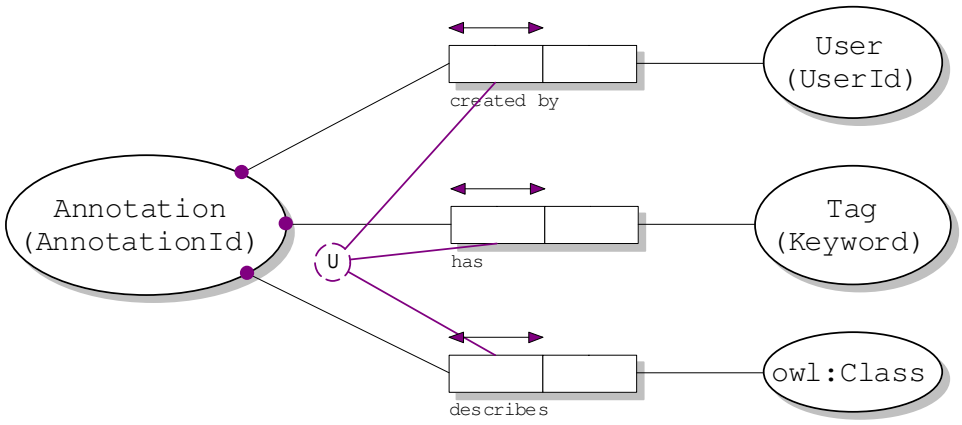

Figure 28: Tagging ontology for WSDM.

For the conceptual design phase we introduced a set of tagging modeling primitives. These tagging modeling primitives represent information or functionality for designing tagging support. With each and every tagging modeling primitive we introduced in this Master Thesis, a certain semantic corresponds, defining the information or functionality. Designing a tagging Web system with these tagging modeling primitives, will automatically provide the designer with the corresponding built-in semantics necessary for the design of the tagging support. The semantics of these tagging primitives will rely on the tagging ontology we just defined. During the rest of this section we will give a representation for the semantics behind the tagging primitives based on object chunks and object chunk templates.

In order to represent the semantics behind the tagging primitives, we need a way to represent these semantics. We will define each tagging modeling primitive as an object chunk template. These object chunk templates are separate pieces of an object chunk which can be connected with each other, depending on their parameters, and form a complete object chunk. Templates can have input parameters or output parameters. Information for input parameters can come from input parameters of the complete object chunk or from output parameters of other object chunk templates. These input parameters are noted between brackets, e.g., <...>. Output parameters of an object chunk template can serve as output parameters for the complete object chunk or as noted already, as input for input parameters of other object chunk templates. The output parameters are noted in bold text.

Let us, as an example, present the object chunk templates for both information tagging primitives, i.e. the Tag primitive (T) and the Annotation primitive (Ⓣ). The tag primitive returns a set of all the existing tags. We can see that the object chunk template representing the tag primitive (Figure 29), models the return of the set of tags, which is defined by its output parameter. The annotation primitive returns a set of all the annotations. The object chunk template representing the annotations primitive (Figure 30), models the return of the annotations, returning them by its output parameter. Furthermore we made it possible to

specify restrictions on the set of annotations being returned: the restriction on resources is defined by the concept embedding the annotation primitive (e.g., Figure 16), the restriction on the users is defined by the users specified within the brackets of the annotation primitive (e.g., Figure 17) and finally the restricting tags are specified by the tags encircled by the annotation primitive (e.g., Figure 19). Each restriction is defined as an input parameter for the object chunk template representing the annotation primitive.

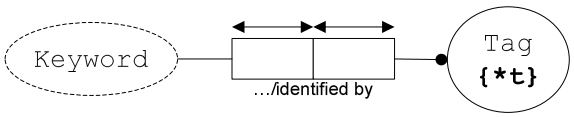

Figure 29: Object chunk template for the Tag primitive (T).

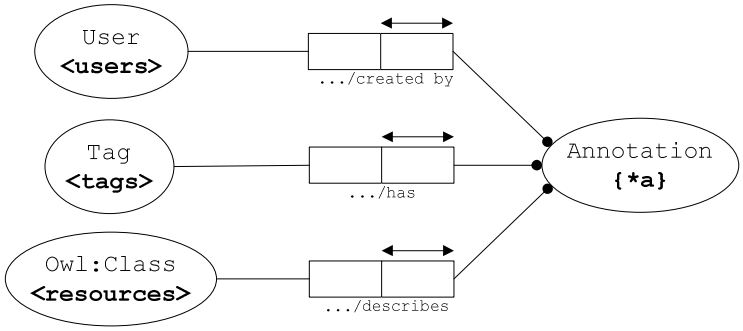

Figure 30: Object chunk template for the Annotation primitive (Ⓣ).

Next to the informational tagging primitives which represent the tags and the annotations, we presented functional tagging primitives which represent functionality related to these tags and annotations. Like the informational tagging primitives, the functional tagging primitives can be expressed by object chunk templates defining their semantics. As an example we will show two object chunk templates defining the semantics of functional tagging primitives. The first object chunk template (Figure 31) defines how tags are selected. The set of tags can be restricted to only those tags tagging a set of annotations. The second object chunk template (Figure 32) defines how annotations are added.

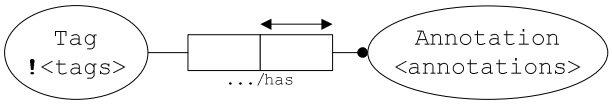

Figure 31: The object chunk template for the tag selection primitive (T!).

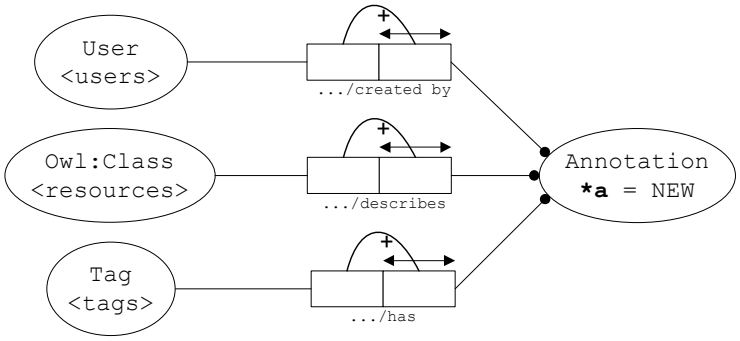

Figure 32: The object chunk template for the annotation adding primitive (Ⓣ+).

We just introduced the object chunk templates that express the semantics of the informational and functional tagging modeling primitives. What we want to show now is how we can represent the semantics of a set of tagging modeling primitives when they are used in an object chunk, defining some information or functionality for the Web system. For this purpose we will transform an object chunk containing tagging primitives into its corresponding semantic representation, based on the object chunk templates we introduced.

The transformation of an object chunk with tagging primitives into its semantic representation results in an extended object chunk where the tagging modeling primitives are replaced by the object chunk templates modeling these tagging modeling primitives. An exact description of how we extend the object chunk will now follow:

1. We clear out the tagging modeling primitives from the original object chunk that will be transformed into its semantic representation.

2. It is possible that some of the tagging modeling primitives that we cleared out the object chunk, were appearing in the list of input parameters of the original object chunk. We therefore start concatenating the object chunk templates which model these tagging modeling primitives. We start with concatenating these object chunk templates since they might specify tags which are required for input parameters of later object chunk templates.

3. We concatenate the object chunk templates, based on their input and output parameters, describing information and functionality related to tags.

4. We continue concatenating object chunk templates. Now we add the object chunk templates describing information and functionality related to annotations. Input parameters for these object chunk templates are based on the input parameters coming from the original object chunk that specify certain users or resources or are based on the output parameters of the object chunk templates related to tags. This is why we start concatenating the object chunk templates modeling the modeling primitives

related to tags and afterwards the object chunk templates modeling the modeling primitives related to annotations.

5. It is possible that some of the tagging modeling primitives which we cleared out from the object chunk, were appearing in the list of output parameters of the original object chunk, e.g., when a user selects a tag from a set of annotations. We therefore end the concatenation with the object chunk templates that model these tagging modeling primitives and add the output parameter of these object chunk templates to the list of output parameters of the original object chunk..

6. Finally, we match remaining output parameters from the original object chunk with remaining output parameters from the object chunk templates.

After the transformation we get an object chunk which models the semantic of an object chunk with tagging primitives.

To illustrate the process to transform an object chunk with tagging primitives into its corresponding representation defining the semantics of that object chunk, we will now present two examples.

For the first example we choose an object chunk (Figure 45) from our case study showing the information of a certain project. For this object chunk we will now present an abbreviated version, presented by Figure 33, which will be further used during this example. Note that there are two sets of tagging primitives in this object chunk. The ones for which we will present the semantic representations are marked in boldface (not to confuse with output parameters from object chunk templates).

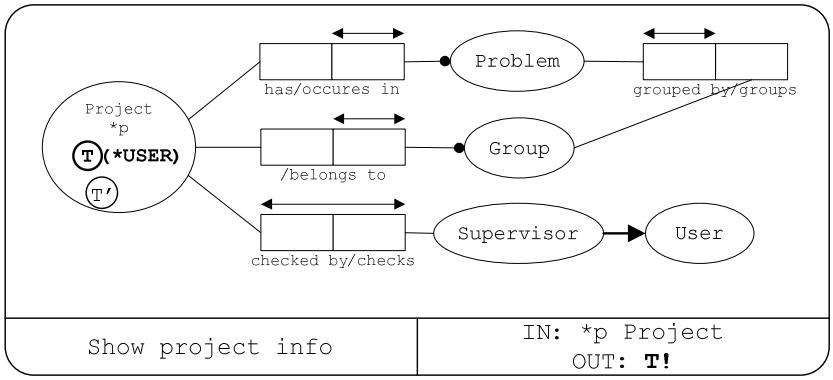

Figure 33: Abbreviated version of the object chunk presented by Figure 45.

The first step of the transformation process consists of eliminating the tagging primitives from the original object chunk (Figure 34).

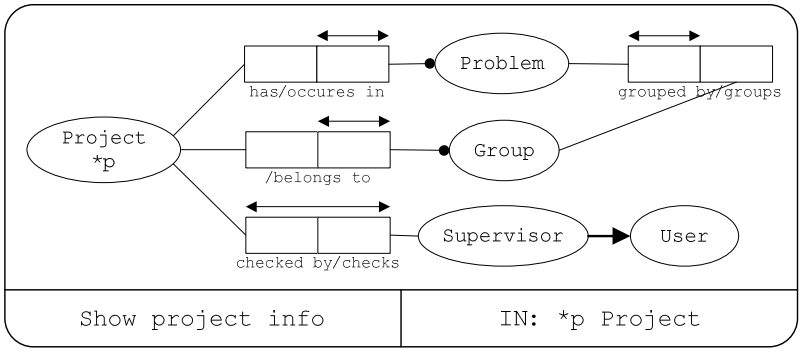

Figure 34: The first step of our transformation example.

Since there are no tagging modeling primitives appearing in the original object chunk's input parameter list, we start concatenating object chunk templates by first adding those object chunk templates related to the tags. We have one tagging modeling primitive which is only related to tags: `T`. This primitive returns all the tags. The object chunk template is presented by Figure 29. The result we obtain so far in our transformation process after this step is shown by Figure 35. Notice that we are still in the process of transforming the object chunk with tagging primitives into its semantic representation. Therefore, the following object chunks may look awkward and incomplete.

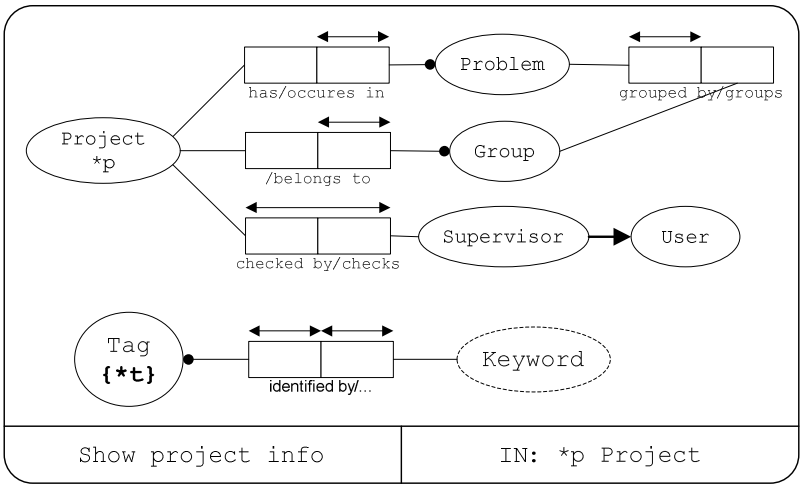

Figure 35: The result after the third step of the transformation.

In the next step we continue concatenating object chunk templates. We now add the object chunk templates representing tagging primitives related to annotations (Figure 36). In this example we only have one such tagging primitive: Ⓣ. The object chunk template for this tagging primitive has been presented earlier by Figure 30 and returns a set of annotations, possibly restricted by certain users, resources and/or tags. We can see that we have made restrictions towards the users and the resources.

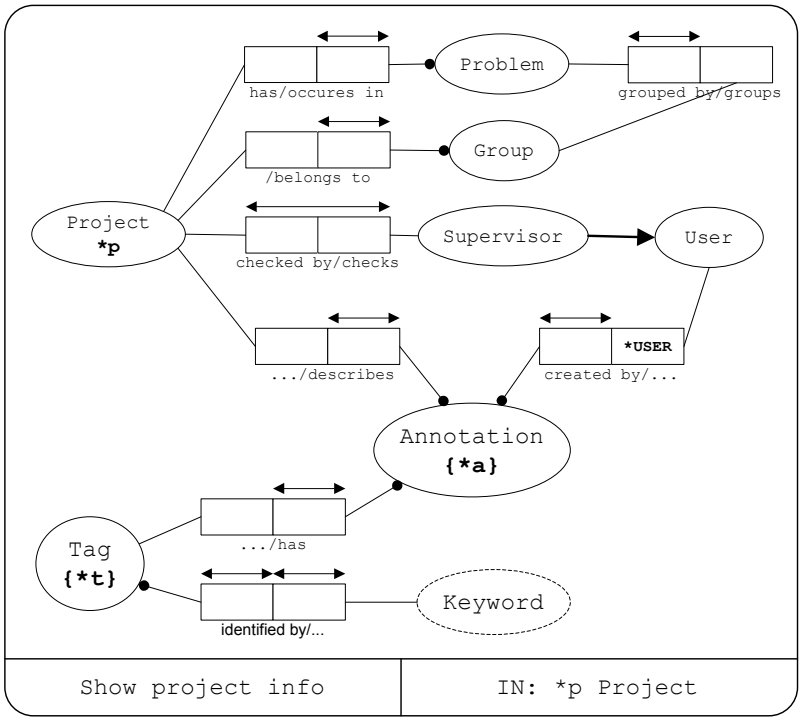

Figure 36: The result after the fourth step of the transformation.

One of the tagging modeling primitives appeared in the output parameter list of the object chunk. We perform one last concatenation, i.e. the object chunk template modeling this tagging modeling primitive. This tagging modeling primitive models the selection of a tag from a list of annotations (Figure 31). We add the output parameter of this object chunk template to the object chunk.

Afterwards, as a final step of the transformation, we match remaining output parameters of the object chunk with the remaining output parameters of the object chunk templates. No such parameter matching needs to be done. The complete result of the transformation is shown by Figure 37 and represents the semantics of the original object chunk with tagging modeling primitives.

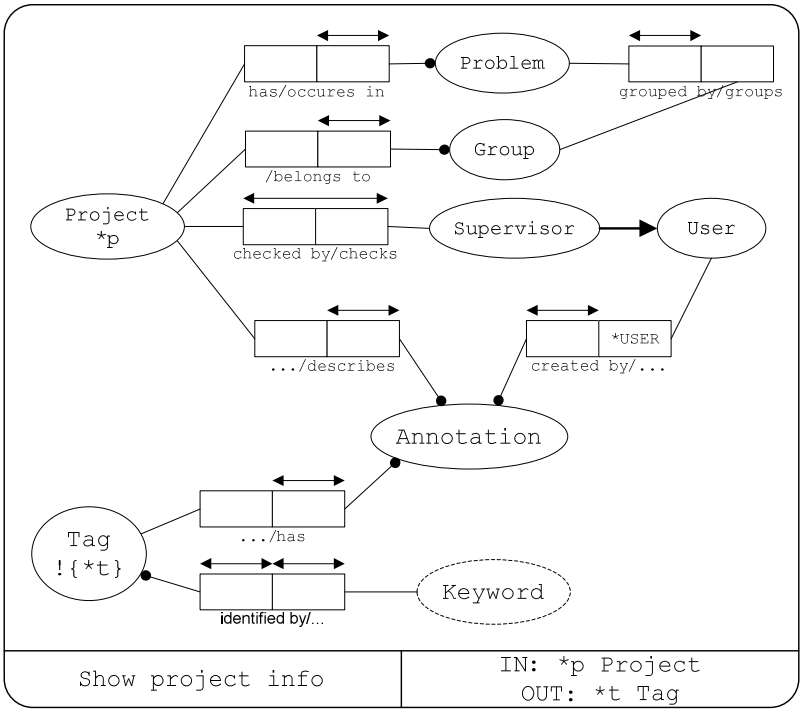

Figure 37: The final result of the transformation represents the semantics of an object chunk with tagging modeling primitives.

The example we just used, for explaining the semantics behind the tagging primitives, was based on informational tagging primitives. We will now present the second example more briefly and based on some functional tagging primitives. For this purpose, again we will use an object chunk from our case study. The object chunk describes how a problem will be reported for a project (Figure 46). When reporting a problem, tags can be entered annotating the problem. We will give an abbreviated version of this object chunk (Figure 38) which will be used for the remainder of this second example.

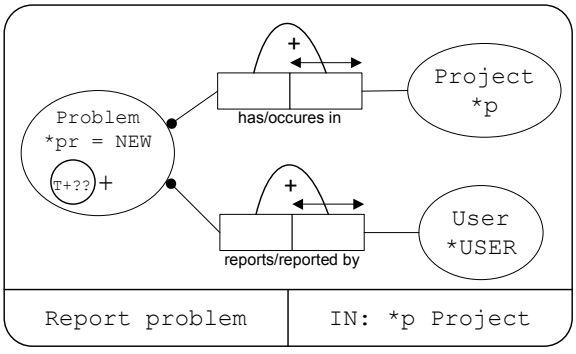

Figure 38: Abbreviated version of object chunk in Figure 46.

To represent the semantics of the object chunk we start the transformation we defined earlier. The first step of the transformation (Figure 39) consists of eliminating the tagging modeling primitives from the object chunk.

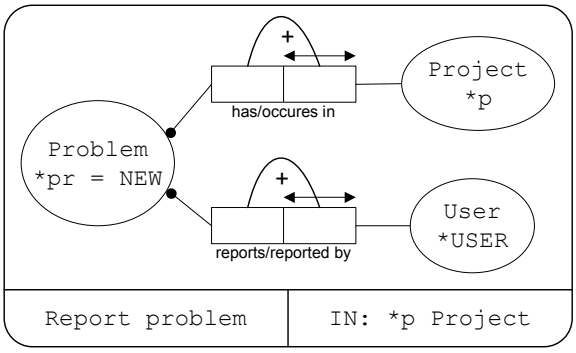

Figure 39: The first step of the transformation produces two object chunks.

During the following steps we concatenate the object chunk templates modeling the tagging primitives and add them to object chunk. We start with the object chunk templates modeling the information or functionality related to tags only. We see that we need to enter a set of tags (Figure 40) and that these tags will be added to the Web system (Figure 41). Note that when a tag already exists in the Web system, this tag will only be returned instead of creating a new record. This prevents the existence of multiple tags with the same keyword. Afterwards, we concatenate the object chunk templates modeling the information and functionality related to annotations. There is one tagging primitive describing functionality related to annotations, namely adding of new annotations. The object chunk template is presented by Figure 32. In this example, multiple tags are being specified. This results in the creation and adding of multiple annotations, i.e. one for each tag.

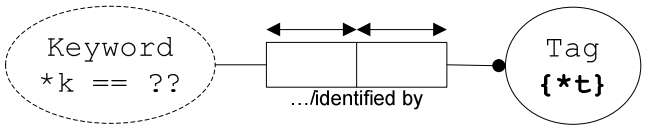

Figure 40: Object chunk template for entering multiple tags (`T??`).

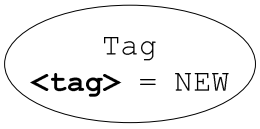

Figure 41: Object chunk template for adding new tags (`T+`).

As a final step, after the concatenation of the object chunk templates, we need to set the output parameters for the object chunk. As we can see from the original object chunk, no output parameters are returned and therefore no output parameters need to be defined. The final result of this transformation process results in the representation of the semantics of the object chunk with tagging primitives (Figure 38) and is illustrated by Figure 42.

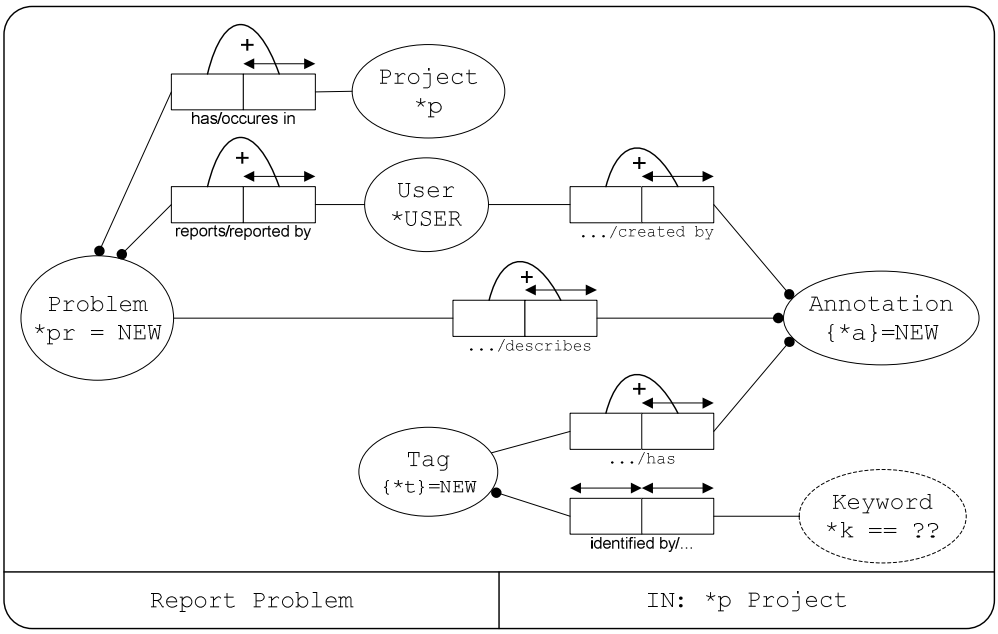

Figure 42: The final result of the transformation process for the second example.

In the beginning of this section we stated that combining all the object chunks together will result in the reference ontology for our Web system. Based on this reference ontology and the tagging ontology, we will afterwards design and build our data store which uses a logical data schema to describe its data. To illustrate this principal we will design the logical data schema for our first example, based on the object chunk that returns the annotations for a certain project (Figure 45). This object chunk designs a small part of the Bug Tracking Web system which will be further extend in our case study. The resulting logical data schema may therefore look incomplete.

For the design of our logical data schema, based on object chunks with tagging modeling primitives, we use the semantic representations for these object chunks. These semantic representations are constructed by applying the transformation process we described earlier. This process presents the semantic of an object chunk with tagging modeling primitives by an object chunk where the tagging primitives are replaces by their corresponding object chunk templates. These object chunk templates are parts of the tagging ontology.

There are various ways to design the logical data schema for a Web system with support for tagging. Keller P. [23] presented a few possible schema's for tagging Web systems. The

proposed database schema's are Delicious orientated since he gathered his information from a Delicious mailing list. The schema's however can be applied on any other Web system supporting tagging. The first proposed schema is a denormalized schema where all tags are stored in one attribute. The is the least normalized form but can improve query speed when using full text search. The second one uses a separate table for storing the tags. This results in a better normalization and the unlimited amount of tags that can be stored is resolved (in the previous solution, tag amount is limited by the maximum capacity of a attribute, e.g., VARCHAR). The final solution is a normalized database schema (3NF) where different tables are used to store the annotations and the tags.

For creating the logical data schema for the object chunk illustrated by Figure 45, we used the last solution, being the most normalized solution. We created a table for the annotations, defining each annotation by a user, resource, tag and label. The label defines what resource type is being annotated, e.g., a project or a problem. Another solution for this was leaving out the label and using a separate annotation table for each resource type. We chose the first solution because when we want more resource types to be annotated we need to construct more annotation tables. This will result in more difficult queries. The users are identified by a login. The tags are identified by an identification number. The resulting logical data schema is shown by the next code-snippet:

Logical data schema for the object chunk illustrated in Figure 45.

```
Group(group-id, group-title, project-id)
Project(resource-id, project-title, project-description, report-time)
Problem(resource-id, problem-title, problem-description, problem-state,
project-id, group-id)
SupervisorProject(login, project-id)
User(login)
Tag(tag-id, keyword)
Annotation(login, resource-id, tag-id, resource-label)
```

3.3 Updated WSDM Ontology

The complete WSDM design methodology is described by an ontology. This WSDM ontology gives a conceptual definition for the modeling concepts being used in the different WSDM design phases. For the description of the modeling primitives, the conceptual design language OWL has been used.

When extending WSDM with new modeling primitives, like what we did by adding the tagging modeling primitives, we need to update the WSDM ontology. The ontology needs to be updated with new concepts which formally describe the new tagging modeling primitives. This allows the generation of a Web system that has been modeled with these tagging primitives and allows the development of tools that provide the designer with the ability for using these tagging modeling primitives.

In our previous section, we introduced all the tagging modeling primitives and their corresponding semantics. First we introduced a set of tagging modeling primitives, representing information and functionality which is necessary to model a tagging Web system. For this set of tagging modeling primitives we will create a new set of OWL concepts, i.e. the *tagging concepts*.

Afterwards, we added a new special purpose link that allows the navigation between nodes together with passing on a certain tag or set of tags.

Furthermore, we added new reference types. This set of references contains two new references: one for referring to a set of tags and one for referring to a set of annotations.

Finally, we updated the Presentation Model. We added a new presentation concept, i.e. the *TagCloud* concept, which is designed for presenting a set of tags based on a set of annotations.

We will now give a more detailed description of the changes we made in the ontology. A full definition for each OWL concept resulting from these changes can be found in Appendix A.

3.3.1 Tagging Concepts

The WSDM ontology has been updated with a new set of concepts, i.e. the tagging concepts. These tagging concepts represent the information and the functionality related to tagging, as we introduced and described them during the previous section 3.2.

First we added concepts that describe the tags and the annotations. The *Tag* concept contains an attribute that defines a keyword. The *Annotation* concept contains three attributes: one defines a user, the second one a resource and the last one a tag. For both concepts we created a reference concept. The first reference concept refers to a set of tags, corresponding to the tagging modeling primitive `T`, and the second reference concept refers to a set of annotations, corresponding to the tagging modeling primitive Ⓣ. We implemented the annotation reference with three additional attributes. These attributes put restrictions on the users, resources or tags that appear in the set of annotations.

Together with the concepts representing the informational tagging primitives, we added concepts representing functionality related to these informational tagging primitives. For all the annotation functionalities and tag functionalities, we created a concept representing a particular functionality. Each functionality concept contains an attribute referring to the set of tags or annotations the functionality applies to.

An overview of all the tagging concepts is illustrated by Figure 43. This figure only illustrates the tagging concept classes and how they are mutually related. For a detailed description of each concept we refer to Appendix A, where a full description of the tagging concepts and the tagging reference concepts are presented in OWL.

To clarify the structure of our tagging concepts, we will present an example of an OWL instance based on a graphical representation we used earlier in this Master Thesis. For this example, we will create the instance for editing an annotation which is graphically represented by Figure 22. The object chunk in this figure models the change of tags into other specific tags, for those annotations that annotate the resources of the user of a session. The formal description of this graphical representation is given by the following OWL code.

OWL description for the tagging modeling primitive illustrated by Figure 22.

```
<wsdm:Tags rdf:ID="tags1"/>

<wsdm:TagReference rdf:ID="tag_ref1">
      <wsdm:referringToTag rdf:resource="#tags1"/>
</wsdm:TagReference>

<wsdm:Tags rdf:ID="tags2"/>

<wsdm:FillOutTagsFunction rdf:ID="tag_input1"/>
      <wsdm:ontoTags rdf:resource="#tags2"/>
      <wsdm:hasCardinality rdf:datatype="&xsd;int">
            1
      </wsdm:hasCardinality>
</wsdm:FillOutTagsFunction>

<wsdm:AnnotationsEditFunction rdf:ID="edit1">
      <wsdm:ontoAnnotations rdf:resource="#annotations1"/>
      <wsdm:hasNewTag rdf:resource="#tags2"/>
</wsdm:AnnotationsEditFunction>

<wsdm:AnnotationReference rdf:ID="annotations1">
      <wsdm:hasAnnotationsFunction rdf:resource="#edit1"/>
      <wsdm:hasRestrictionByResources  rdf:resource="#p"/>
      <wsdm:hasRestrictionByUsers  rdf:resource="#USER"/>
      <wsdm:hasRestrictionByTags  rdf:resource="#tag_ref1"/>
</wsdm:AnnotationReference>
```

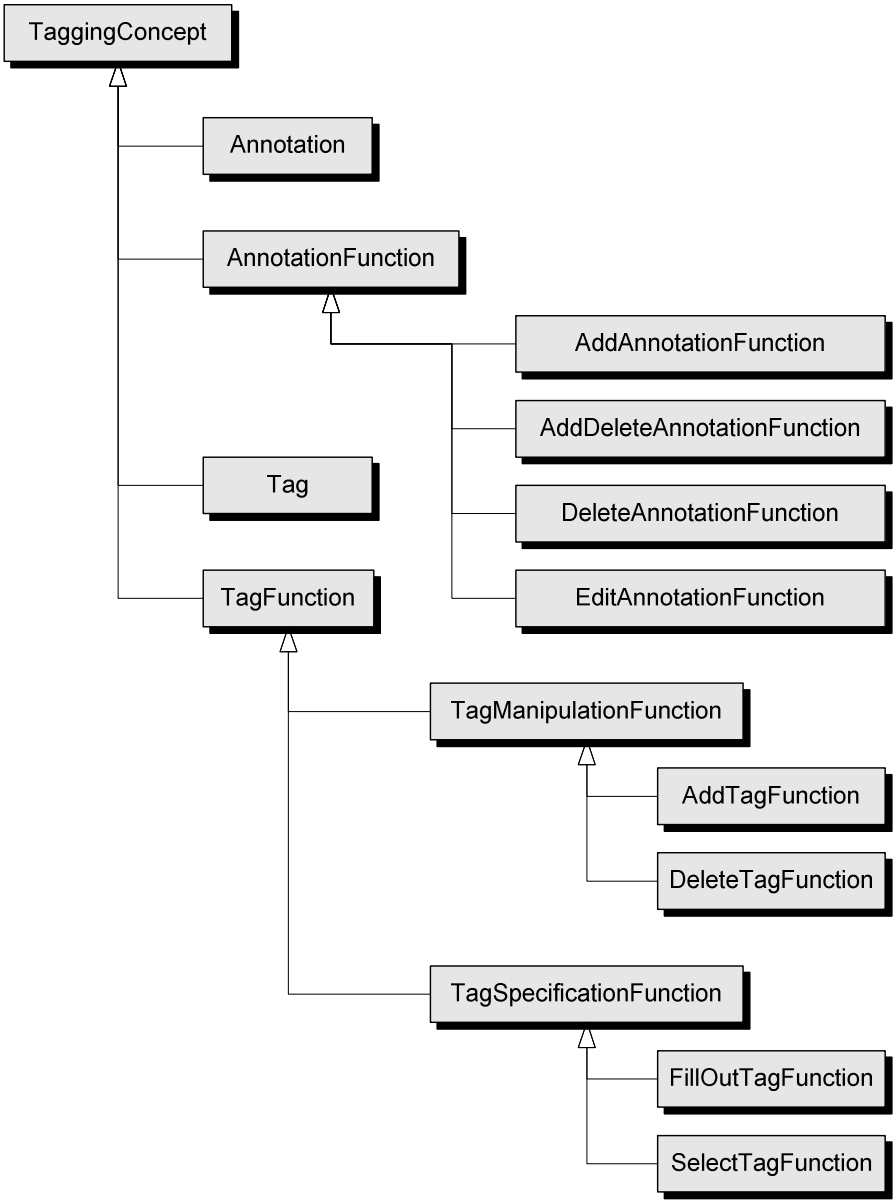

Figure 43: Tagging concepts structure.

3.3.2 Tagging Presentation Concepts

In WSDM, there are cell and item concepts available that can contain single data concepts, in order to structure these data concepts on a Web page in between a bigger set of data. A various set of these cell and item concepts is available, e.g., Cell, ListItem or TableCell. These cell and item concepts are chosen depending on their context, e.g., list, menu or table. Possible content for these concepts is provided by object chunk references, multimedia concepts or even complex presentation concepts. The multimedia concepts define a variety of concepts, e.g., Image, Email or Video. All these multimedia concepts contain an attribute that defines the text that needs to be displayed when this multimedia concept is presented. We

created the Tag concept as a sub-concept of this MultiMedia concept, allowing a tag to be positioned by these cell and item concepts. The value attribute of the Tag concept will contain the keyword of the tag and will therefore be displayed when a tag is presented.

For presenting a set of tags, based on a set of annotations, we introduced a new presentation concept in the presentation design section (3.2.4.1), i.e. a TagCloud concept. We will create and add this TagCloud concept to the list of complex presentation concepts in WSDM. The complete specifications for the TagCloud concept can be found in the presentation design.

3.4 Conclusion

In the previous chapter we introduced WSDM. In this chapter we looked for new means to facilitate the design of a tagging Web system based on WSDM. For this purpose we have extended WSDM with new modeling primitives, explicitly designed for the design of a tagging Web system.

In order to discover what exact modeling primitives are required for designing and modeling a tagging Web system, we analyzed existing Web systems which support tagging. During our analysis we searched for all possible requirements involved with such a tagging Web system. Afterwards, these tagging requirements formed the basis for the new modeling primitives.

The tagging requirements we discovered could be divided into three requirement categories: informational requirements, functional requirements and navigational requirements. Based on these categories we knew exactly in which design phase of WSDM we needed to provide modeling primitives for certain tagging requirements.

Afterwards, we defined the semantics for our introduced modeling primitives, i.e. the tagging modeling primitives. We represented the semantics of tagging primitives by means of object chunks and object chunk templates.

Finally we extended the WSDM ontology. We defined and added the necessary concepts to this ontology in order to support the tagging primitives we introduced.

4 Case Study

In chapter 2 we introduced the Web design methodology WSDM. In order to clarify WSDM and its different design phases we used a running example: the Bug Tracking Web system. In chapter 3 we afterwards introduced an extension for WSDM for adding tagging support to a Web system by using a set of tagging primitives. We will now set up a case study as a proof of concept for our introduced tagging primitives as an extension to WSDM. In this case study we will extend our running example from chapter 2, the Bug Tracking Web system, with support for tagging.

4.1 Mission Statement

When defining the mission statement, we define higher leveled aspects of the Web system then tagging aspects and its corresponding requirements. Therefore, we do not need to change our current mission statement as we defined it in section 2.3 when adding tagging support to our Bug Tracking Web system.

4.2 Audience Modeling

During the audience modeling phase, we classify and characterize the different target audiences. The classification of these target audiences is based on the informational and functional requirements of the target audiences. We already defined a list of requirements for the different target audiences and created a set of audience classes based on these requirements. We now need to add an additional set of requirements which will describe the tagging support in our Bug Tracking Web system, i.e. the tagging requirements.

For the Bug Tracking Web system, all the different audience classes have already been defined and their hierarchy has been created. Each audience class contains a set of requirements, necessary for that audience class. For each audience class, we will now define a separate list of requirements, the tagging requirements. These tagging requirements well describe the tagging behavior for the audience class. Of course, the tagging requirements we add will resemble those tagging requirements we discovered in section 3.1.

We will now present the different audience classes together with their tagging requirements. For each tagging requirement, we will point out which tagging requirement they resemble from our Web systems analysis.

Software User:

- Functional tagging requirements:
    - Can annotate (add/delete tags) projects for which the software user is registered (reqs. 6 and 7).
    - Can annotate (add/delete tags) problems belonging to the projects for which the software user is registered (reqs. 6 and 7).
- Informational tagging requirements:
    - Projects and problems information together with their annotations given by the entire Web system's community (req. 3a).
    - Projects and problems information together with their annotations given by the software user (reqs. 2a and 3a).
    - Most popular annotations based on all annotations (on projects and problems given by any user) in the system (req. 1).
    - A tag page showing projects and problems annotated with that tag (req. 4a).
- Navigational tagging requirements:
    - Clicking on a tag will list all the projects and problems which are related to that tag, meaning that they are also annotated with the same tag. When a tag is clicked from the list of annotations given by any user all the projects and problems annotated by the tag are listed. On the other hand, when a tag is clicked from the list of personally made annotations, only those projects and problems are listed for which the software user is registered (req. 12).

Project Manager:

- Functional tagging requirements:
    - Can annotate managed projected (reqs. 6 and 7).
    - Can annotate the problems belonging to managed projects (reqs. 6 and 7).

Project Supervisor:

- Functional tagging requirements:
    - Can annotate projects being supervised (reqs. 6 and 7).
    - Can annotate the problems belonging to the projects that need to be supervised (reqs. 6 and 7).

4.3 Conceptual Design

So far we described all the requirements, including the tagging requirements, for the different audience classes in an informal way. The goal of the conceptual design phase is to translate these informal descriptions of the requirements into formal modeling concepts describing the conceptual information, functionality and structure of our Web system. As we saw, this happens in two sub-phases: the task & information modeling sub-phase where we define the information and functionality of the Web system and the navigational design sub-phase where we define the structure of the Web system.

A large part of the information, functionality and structure that is needed for our Bug Tracking Web system is already modeled and available for further use. What rests now is the modeling of the information, functionality and navigability which is required to satisfy the tagging requirements. For these requirement needs we introduced a set of tagging primitives which will help us out with modeling the information, functionality of navigability required for these requirements and with designing the support for tagging in our Bug Tracking Web system.

We will now repeat both sub-phases of the conceptual design and where needed we will model the necessary tagging requirements we presented in our audience modeling phase.

4.3.1 Task & Information Modeling

During the task & information modeling phase we model the different tasks each audience class member needs to perform, together with the information and/or functionality which is required to perform these tasks. For formally describing the tasks together with their information and/or functionality, WSDM uses certain modeling techniques: we model the tasks by using task models and we model the information and functionality by using object chunks.

Since we already designed our Bug Tracking Web system, most of the task models and object chunks are already available. What remains to be modeled during this phase are the information and functionality concerning the support for tagging in our Bug Tracking Web system. For this purpose we will continue from the existing task models and object chunks and we will extend and update them where necessary.

First of all we start with the task modeling. It is possible that certain task models need to be updated with additional elementary tasks. The elementary tasks we add here are tasks related to the tagging support that needs to be implemented. Let us for example continue from the task model we described during our earlier modeling in chapter 2 which models the browsing through projects and showing these projects (Figure 10). Originally, when we show the information of a project we afterwards provide the possibility to register for the project or report a problem. We will now add the possibility to annotate the project with a new tag or

delete an existing annotation from the project. This results in the updated task model shown in Figure 44.

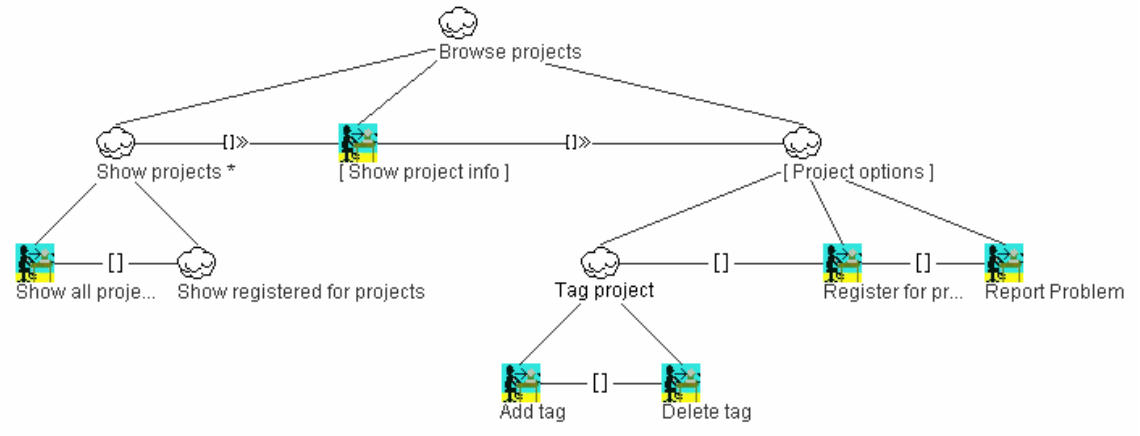

Figure 44: Updated task model which allows us to add or delete tags to projects.

After updating the task models we will now define the information and functionality that is related to the tagging support we want to add to our Bug Tracking Web system. For this purpose we will update the object chunks belonging to some elementary tasks. Besides that we will have to model additional object chunks for those elementary task we just added to our task model. The object chunks that need to be updated are those object chunks that somehow describe any kind of information or functionality which is related to the tagging support. Let us for example consider the elementary task “Show project info” belonging to the task model shown by Figure 44. With this elementary task corresponds an object chunk that will display all the information of a certain project. In our updated version of the Bug Tracking Web system we want this object chunk to also show the tags that annotate the project. For this reason we will have to update the object chunk which will result in the object chunk shown by Figure 45. As we can see from this object chunk, two lists of annotations are returned: the annotations made by the user of a session of the Web system and those annotations that are made by any user.

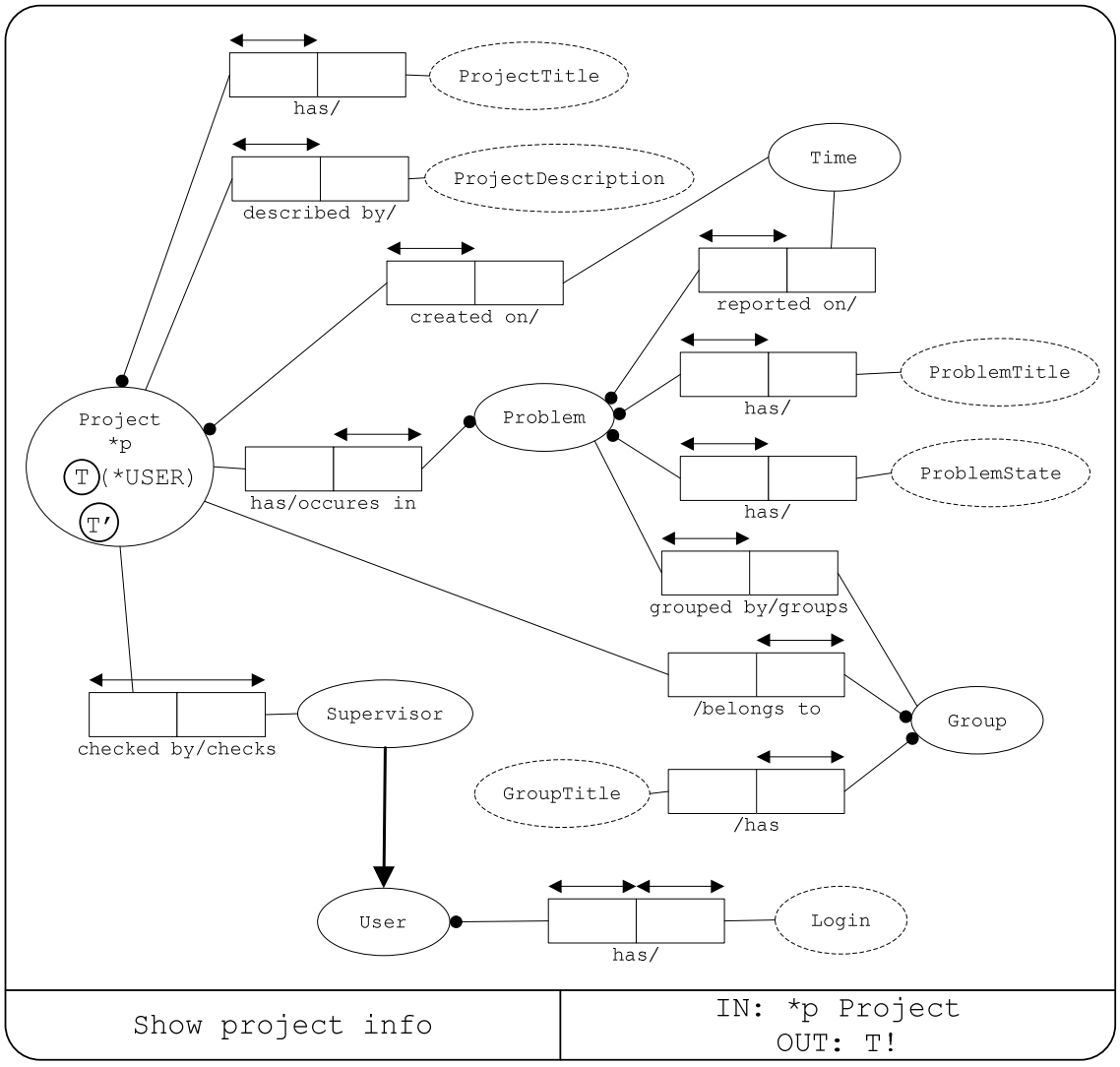

Figure 45: Updated object chunk showing the information and tags of a project.

The previous updated object chunk showed additional information concerning the tagging for our tagging support. We will now update a second object chunk. This object chunk (Figure 46) will describe how we can add new annotations to a problem when we report a new problem for our project. The object chunk will show that a list of multiple tags can be entered by the user. These tags will afterwards annotate the problem.

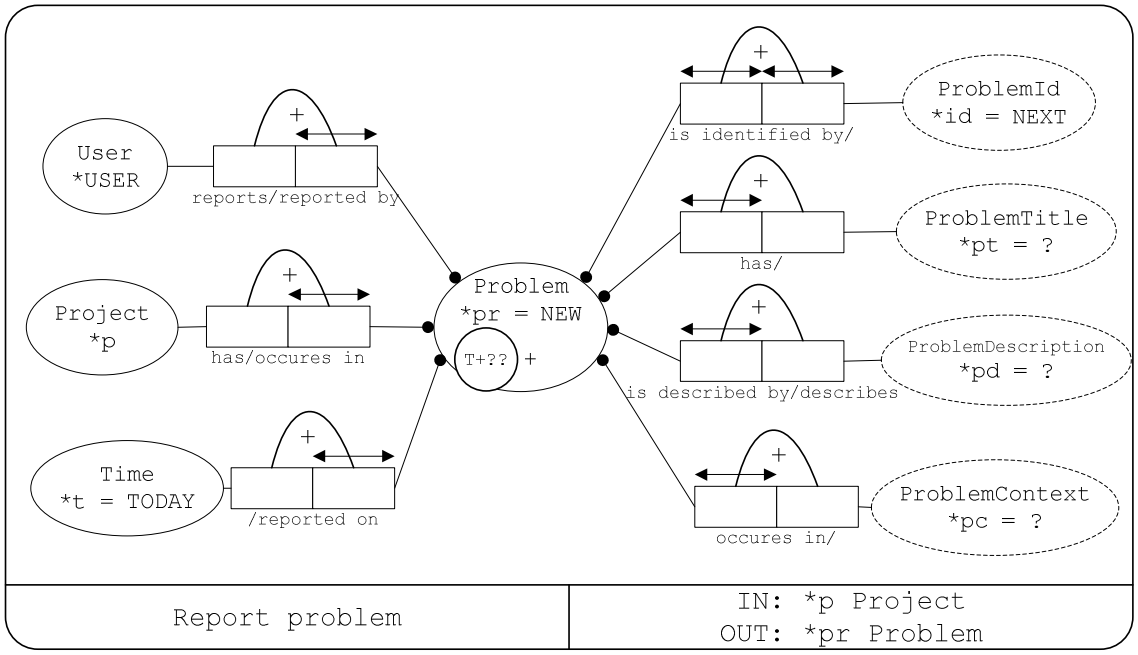

Figure 46: When reporting a problem, a list of tags can be entered.

Apart from updating certain object chunks we will create new object chunks for the additional elementary tasks concerning the support for tagging. In our task model (Figure 44) we added two elementary task. These two new elementary tasks result in the following new object chunks: the first describes how new annotations will be added to a project (Figure 47) and the second how existing annotations will be deleted from a project (Figure 48).

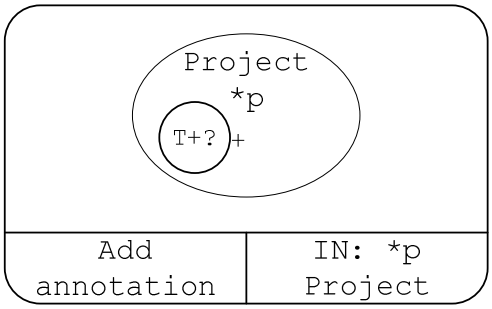

Figure 47: Requests a tag to be entered by the user.

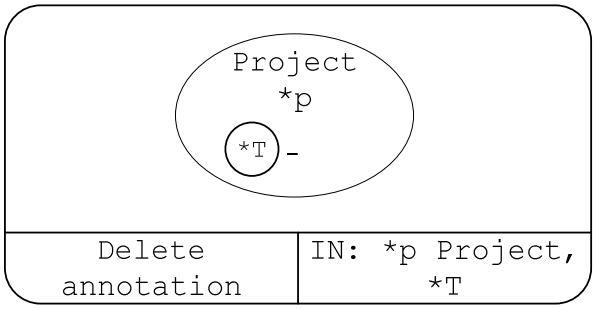

Figure 48: Deletes an annotation from the specified project.

For our last object chunk, we will show how we can look up a list of projects which are annotated by a certain tag. Our example describes how we can look up projects based on their title, description and/or tags. These three concepts will be matched with the input a user entered. We can see here that the returning projects are matched and restricted by the annotations containing the tags the user entered.

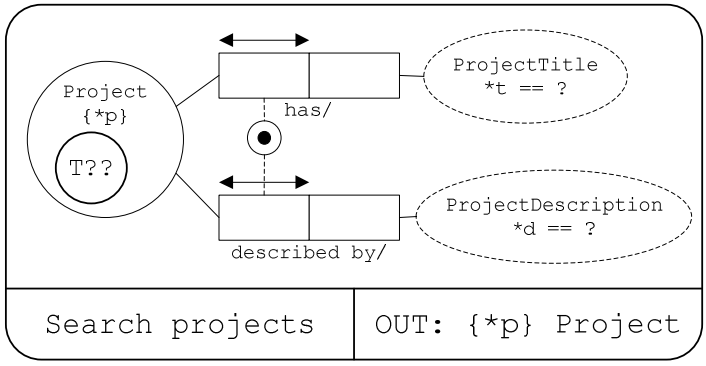

Figure 49: Projects are looked up based on their title, description and/or tags.

4.3.2 Navigational Design

The navigational design phase defines the conceptual structure of the Bug Tracking Web system. It describes how de audience class members navigate through the Web system and how they perform their tasks. The result of the navigational design phase is the navigational model which is composed out of different navigation tracks. The creation of these navigation tracks are actually transformations from the earlier defined task models into their corresponding navigational models.

Since we base the creation of the navigation tracks on the task models and since we updated some task models for adding tagging support to our Bug Tracking Web system, we of course need to review our existing set of navigation tracks and update them where necessary. Let us for example consider the navigation track we defined in Figure 14. This navigation track was based on the task model of Figure 10. But since we edited this task model we will review the navigation track. The result of the updated navigation track is shown by Figure 50. As we can

see we added an extra condition for being able to add or delete annotations. From the requirements we can see that it was only possible for managers, supervisors and users registered for a project to add or delete annotations from the project.

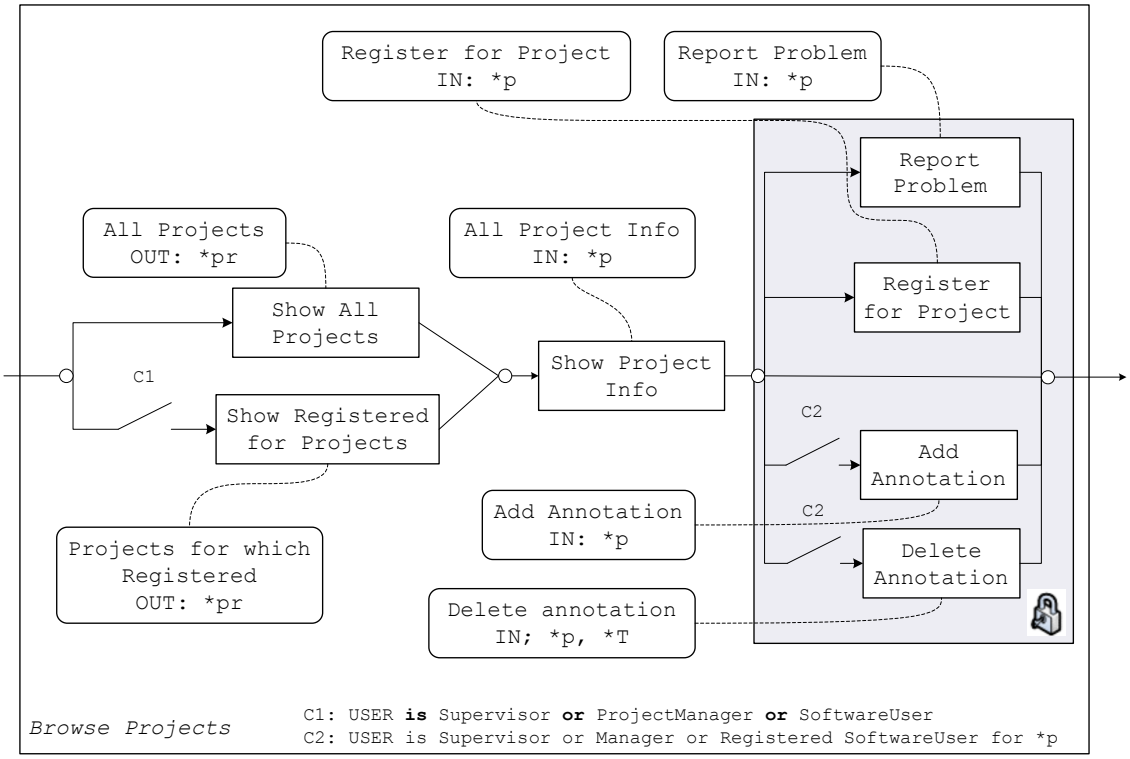

Figure 50: Updated navigation track for browsing projects.

From our list of tagging requirements we see that we want an additional Web page where we have an overview of the most popular tags. To design this we will add an additional navigation track (Figure 51) into our main conceptual navigation structure (Figure 52). This navigation track starts with a node connected with by an object chunk which returns all the annotations without any restrictions towards users, resources or tags. Of course it must be possible to show a list of projects and problems which are annotated with a certain tag after selecting that tag. Therefore we will pass the selected tag through to two following nodes. The first node is connected by an object chunk that shows a list of related projects. The second node is connected by an object chunk that lists related problems.

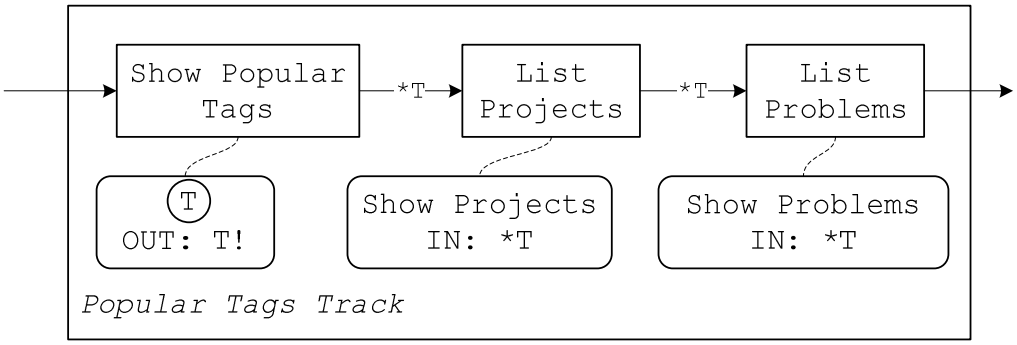

Figure 51: Navigation track for showing popular tags.

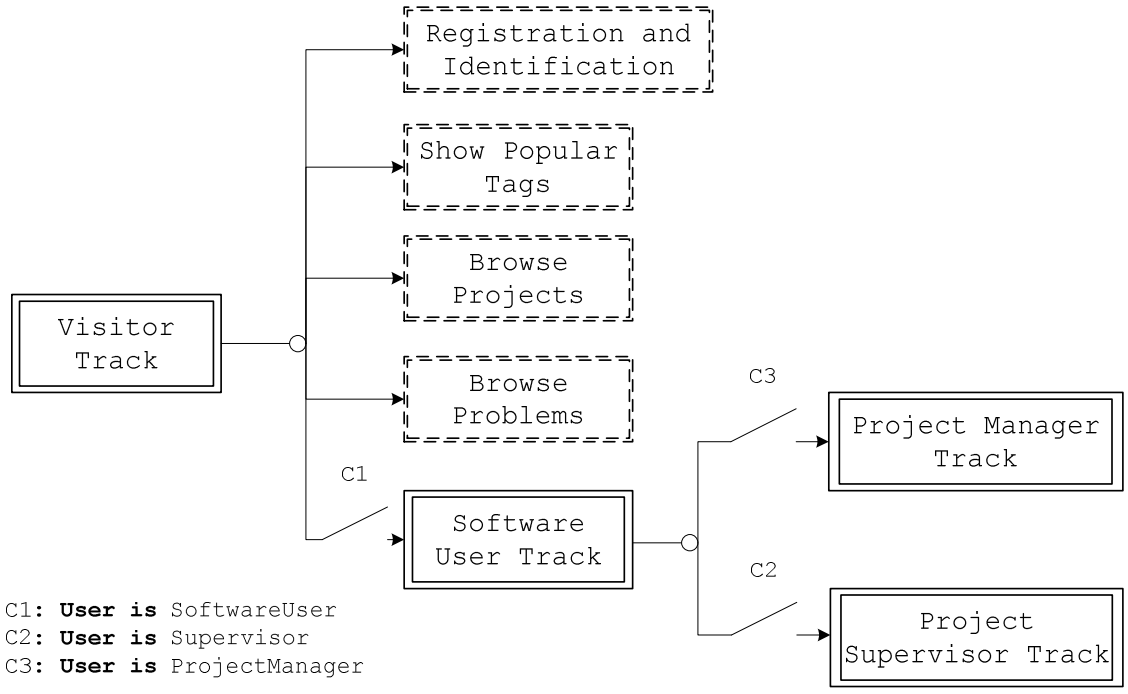

Figure 52: Updated conceptual navigation structure.

Finally, we have to model the additional navigational requirement which comes along with our tagging support. This requirement allows us to browse from a tag, appearing in a list of tags annotating a certain project or problem, to a list of projects or problems which are annotated by that tag. This requirement matches requirement nr. 12 from our list of requirements we discovered in section 3.1.

Our solution for this navigational tagging requirements is presented in section 3.2.3.2. As we can see, we model this requirement by means of a link which connects two nodes: The first node displays the resources (e.g. project or problem) including the tags annotating the resource or any other Web page for that matter where tags are shown annotating some resource. The second node displays the list of resources being annotated by a tag. By

specifying the tag in the first node, the tag will define the resources to be displayed in the second node.

As an example, let us consider the node showing the information of a project. By adding the tagging support to the Bug Tracking Web system, this node will also display the tags annotating the project. For each tag two links will be available: selecting the first one will show all the personal projects and selecting the other one will show all the projects from the entire Bug Tracking Web system (similar to our earlier semantic links example from section 3.2.3.2). The result of our updated set of semantic links is shown by Figure 53.

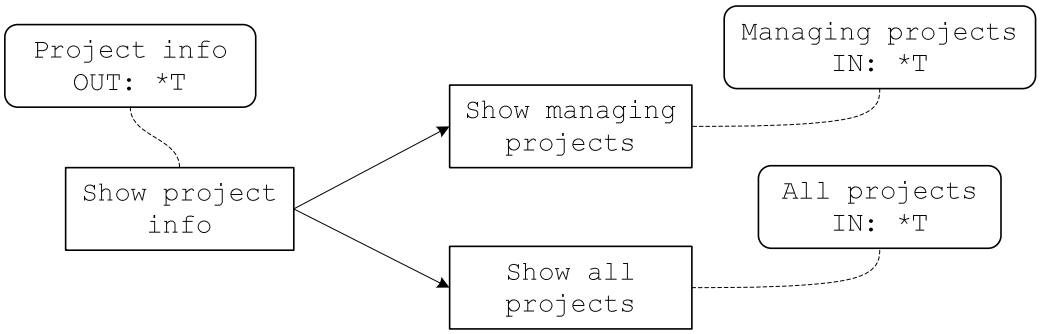

Figure 53: Updated set of semantic links.

4.4 Implementation Design

During the implementation design, we updated the presentation design model with additional presentation concepts in order to display tags, to display sets of tags, by means of tag clouds, and to enter tags.

The data store has been updated. We created new tables for storing the tags and annotations. Furthermore, queries and updates were created in order to query and update these new tables.

We leave out any further details in order to save space in this Master Thesis.

4.5 Implementation

After completing the different design phases of our extended version of WSDM, tagging support has been added into the Bug Tracking Web system we designed earlier in chapter 2. With this additional design for the support of tagging, we now have all the information and all the models we need to implement a full working tagging Web system. Based on all the models we defined during the different phases of WSDM and our extended version of WSDM, we have implemented the Bug Tracking Web system included with support for tagging.

Future Work

In the Master Thesis we presented a set of tagging modeling primitives. When we examine the informational tagging primitives, which return a set of tags or annotations, we see that this information can be queried by simple queries and without any further processing. To query specific tags (based on keywords) or annotations (based on certain resources, users and/or tags) simple queries satisfy onto the set of tags or annotations, returning the required information straight from the data store.

When we further analyzed the structure of folksonomy-based Web systems, we saw that implicit relations exist between annotations in the flat namespace that builds up a folksonomy. Based on these implicit relations it is possible to obtain extra information. In folksonomy-based Web systems, it is for example interesting to present a set of related tags for a certain tag or to present a set of interesting resources for a certain user.

An interesting topic, for later research, is how to extract these implicit relations and to obtain this extra set of interesting and valuable information. With this information, one can for example improve the discovery of new and interesting resources. Research pointing to this direction has already been conducted [46-48]. The technical term for this way of looking for information is called Collaborative Filtering. Collaborative filtering is the method of making automatic predictions (filtering) about the interests of a user by collecting taste information from many users (collaborating). The main idea behind collaborative filtering is that when looking up interesting information, we should look for users sharing similar interests and then recommend other resources that those users like.

Collaborative filtering searches the complete flat namespace in order to collect users with similar interests and to make prediction based on these sets of users. This requires complicated calculations demanding high computational resources.

It would be interesting to further explore the field of collaborative tagging and analyze how we can implement this mechanism into WSDM. This way we provide the designer with an extra set of modeling primitives that models extra information related to folksonomies and extends the functional possibilities in the Web system.

Conclusion

We started this Master Thesis with the fact that the amount of information being available on the World Wide Web is growing all the time. People need to find ways to organize their information for later retrieval and to discover new and interesting information in between a big pile of junk available on the Web.

Classical classification schemes, such as Taxonomies and Ontologies, provided a solution for this problem. These schemes define the structure of extra information – Metadata – being added to the information. This improved and created ways for organizing data, where it is even possible for automated agents to process this data and automating the organization of the data.

However, since these classification schemes lacked the power of the masses: modeling classification schemes require good knowledge of the domain and professional modeling skills. Furthermore the domain needs to be stable and the classes need to have clear boundaries.

We saw that folksonomies provided an answer to these problems. End-users were able to annotate their information by means of annotation information by single keywords. A simple mechanism which reaches a broad variety of users annotating resources in a open community.

The growing success of folksonomies lead to the goal of this Master Thesis: to analyze and search for currently available means which aid the designer with the design of a folksonomy Web system and in case no such means exist, to design and build such means.

After analyzing currently available Web design methodologies, no means were found that facilitate the design of a folksonomy-based Web system.. We therefore started our research towards such design aids. For this purpose we used WSDM (Web Semantics Design Method) as the framework for this Master Thesis. WSDM is developed by the Web & Information Systems Engineering research group at the Vrije Universiteit Brussel and is a Web design methodology which takes as starting point the different target audiences and creates the Web system's structure based on their target audiences' requirements.

Since WSDM did not contain any means for designing a folksonomy, we extended WSDM with design aids for folksonomies. In order to achieve this, we analyzed currently available folksonomy-based Web systems and extracted their requirements towards their use related to the folksonomy. This resulted in the *Tagging Requirements* which we could categorize in three different categories: (1) Informational Requirements, (2) Functional Requirements and (3) Navigational Requirements.

We evaluated each design phase of WSDM and analyzed what needed to be changed in order to support the discovered tagging requirements. When modeling the tagging support for a Web system, we state these tagging requirements in the Audience Modeling phase. For each tagging requirement we created modeling primitives which were added to the Conceptual Design phase. These modeling primitives were used to model either information, functionality or navigability related to the support for tagging in a Web system. All the modeling primitives together formed the set of *Tagging Modeling Primitives*.

For each tagging modeling primitive we first presented their graphical representation. Afterwards, we defined their semantics by means of object chunk templates. Finally, in order for WSDM to support these tagging primitives, we updated the WSDM ontology.

What results from this Master Thesis, is an extension for WSDM with an extra set of modeling primitives that are specially designed for facilitating the design of a folksonomy-based Web system. These modeling primitives provide the necessary semantics for modeling information, functionality and navigability, related to folksonomies. The modeling of these issues is therefore left out for the designer.

As a proof of concept for the use of these tagging modeling primitives, we made a case study where we extended a Web system, the Bug Tracking Web system, with additional support for tagging. We repeated all the different steps of WSDM and updated each corresponding model with the necessary tagging modeling primitives in order to add support for tagging to the Bug Tracking Web system.

Bibliography

1. Bergman, M. The Deep Web: Surfacing Hidden Value.

2. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American*.

3. Bielenberg, K. and Zacher, M. Groups in Social Software: Utilizing Tagging to Integrate Individual Contexts for Social Navigation, 2005.

4. Brooks, C. and Montanez, N., Improved annotation of the blogosphere via autotagging and hierarchical clustering. in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (2006), ACM, 625-632.

5. Casteleyn, S., Plessers, P. and De Troyer, O. On Generating Content and Structural Annotated Websites Using Conceptual Modeling. in *Conceptual Modeling - ER 2006*, 2006, 267-280.

6. Ceri, S., Fraternali, P. and Bongio, A., Web Modeling Language (WebML): a modeling language for designing Web sites. in *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, (2000), North-Holland, 137-157.

7. De Troyer, O. and Casteleyn, S. The Conference Review System with WSDM.

8. De Troyer, O. and Casteleyn, S. Exploiting link types during the conceptual design of websites. *International journal of Web engineering and technology*, *1* (1). 17-40.

9. De Troyer, O., Casteleyn, S. and Plessers, P. Using ORM to Model Web Systems. in *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, 2005, 700-709.

10. De Troyer, O., Casteleyn, S., Plessers, P., Rossi, G., Pastor, O., Schwabe, D. and Olsina, L. WSDM: Web Semantics Design Method. in *Web Engineering: Modelling and Implementing Web Applications*, 2007.

11. De Troyer, O. and Leune, C.J. WSDM: a user centered design method for Web sites. *Computer Networks and ISDN Systems*, *30* (1--7). 85-94.

12. Dewey, M. and Amherst, U.S.A. *A Classification and Subject Index for Cataloguing and Arranging the Books and Pamphlets of a Library*, 1876.

13. Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Addison-Wesley, 2003.

14. Ginige, A., Web engineering: managing the complexity of web systems development. in *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, (2002), ACM, 721-729.

15. Golder, S. and Huberman, B. Usage patterns of collaborative tagging systems. *J. Inf. Sci.*, *32* (2). 198-208.

16. Gomez, J., Cachero, C. and Pastor, O. Conceptual modeling of device-independent Web applications. *Multimedia, IEEE*, *8* (2). 26-39.

17. Gulli, A. and Signorini, A., The indexable web is more than 11.5 billion pages. in *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, (2005), ACM, 902-903.

18. Halpin, T. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2001.

19. Halvey, M. and Keane, M., An assessment of tag presentation techniques. in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (2007), ACM, 1313-1314.

20. Hassan-Montero, Y. and Herrero-Solana, V.c. Improving Tag-Clouds as Visual Information Retrieval Interfaces. *I International Conference on Multidisciplinary Information Sciences and Technologies*.

21. Hotho, A., JÃ¤schke, R., Schmitz, C. and Stumme, G. *Information Retrieval in Folksonomies: Search and Ranking*, 2006.

22. Jacob, E. Classification and categorization: a difference that makes a difference. *Library Trends*, *52* (3). 515-540.

23. Keller, P. Tags: Database schemas, 2005; (http://www.pui.ch/phred/archives/2005/04/tags-database-schemas.html).

24. Koch, N. A Comparative Study of Methods for Hypermedia Development, 1999.

25. Koch, N. and Kraus, A. The expressive Power of UML-based Web Engineering, 2002.

26. Lima, F. and Schwabe, D. Modeling Applications for the Semantic Web. in *Web Engineering*, 2003, 425-441.

27. Lyman, P. and Varian, H. How Much Information?, 2003.

28. Macgregor, G. and McCulloch, E. Collaborative Tagging as a Knowledge Organisation and Resource Discovery Tool *Library Review*, 2006, 291-300.

29. Marlow, C., Naaman, M., Boyd, D. and Davis, M., HT06, tagging paper, taxonomy, Flickr, academic article, to read. in *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, (2006), ACM, 31-40.

30. Mathes, A. Folksonomies - Cooperative Classification and Communication Through Shared Metadata, 2004.

31. Nanard, J. and Nanard, M. Hypertext design environments and the hypertext design process. *Commun. ACM*, *38* (8). 49-56.

32. Pastor, O., Insfran, E., Pelechano, V., Romero, J. and Merseguer, J., OO-{METHOD}: An {OO} Software Production Environment Combining Conventional and Formal Methods. in *Conference on Advanced Information Systems Engineering*, (1997), 145-158.

33. Paterno, F., Mancini, C. and Meniconi, S., ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. in *INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, (1997), Chapman, 362-369.

34. Quintarelli, E. Folksonomies: power to the people, 2005; (http://www-dimat.unipv.it/biblio/isko/doc/folksonomies.htm).

35. Rumbaugh, J., Blaha, M., Lorensen, W., Eddy, F. and Premerlani, W. *Object-Oriented Modeling and Design*. Prentice Hall, 1990.

36. Schmitz, C., Hotho, A., JÃ¤schke, R. and Stumme, G. Mining Association Rules in Folksonomies. in *Data Science and Classification*, 2006, 261-270.

37. Schwabe, D. and Rossi, G. The object-oriented hypermedia design model. *Commun. ACM*, *38* (8). 45-46.

38. Schwabe, D. and Rossi, G. An object oriented approach to Web-based applications design. *Theor. Pract. Object Syst.*, *4* (4). 207-225.

39. Shaw, B. Utilizing Folksonomy: Similarity Metadata from the Del.icio.us System, 2005.

40. Shirky, C. Ontology is Overrated -- Categories, Links, and Tags; (http://www.shirky.com/writings/ontology_overrated.html).

41. Udell, J. Collaborative knowledge gardening, 2004; (http://www.infoworld.com/article/04/08/20/34OPstrategic_1.html).

42. Wal, T. Explaining and Showing Broad and Narrow Folksonomies; (http://personalinfocloud.com/2005/02/explaining_and_.html).

43. Wal, T. Folksonomy definition and wikipedia; (http://www.vanderwal.net/random/entrysel.php?blog=1750).

44. Wal, T. Folksonomy in New York Times Magazine Year in Ideas, 2005; (http://www.vanderwal.net/random/entrysel.php?blog=1761).

45. Walker, J., Feral hypertext: when hypertext literature escapes control. in *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, (2005), ACM, 46-53.

46. Wu, H., Zubair, M. and Maly, K., Harvesting social knowledge from folksonomies. in *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, (2006), ACM, 111-114.

47. Wu, X., Zhang, L. and Yu, Y., Exploring social annotations for the semantic web. in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (2006), ACM, 417-426.

48. Xu, Z., Fu, Y., Mao, J. and Su, D., Towards the Semantic Web: Collaborative Tag Suggestions. in *Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006*, (2006).

Appendix A

In this appendix we will present the full OWL descriptions for the tagging primitives we introduced in chapter 3.

A.1 Tagging Concepts

The tagging concepts defines the structure of the tagging concepts regarding the information and functionality related to tagging.

```
<owl:Class rdf:ID="Annotation">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:subClassOf rdf:resource="#TaggingConcept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasResource"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasTag"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasUser"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#AnnotationFunction">
  <rdfs:subClassOf rdf:resource="#TaggingConcept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="ontoAnnotations"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="AddAnnotationFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#AnnotationFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="AddDeleteAnnotationFunction">
  <rdfs:subClassOf rdf:resource="#AnnotationFunction"/>
</owl:Class>

<owl:Class rdf:ID="DeleteAnnotationFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#AnnotationFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="EditAnnotationFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="AnnotationFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Tag">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#MultimediaConcept"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TaggingConcept"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#TagFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TaggingConcept"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="ontoTags"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#TagManipulationFunction">
  <rdfs:subClassOf rdf:resource="#TagFunction"/>
</owl:Class>

<owl:Class rdf:ID="AddTagFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TagManipulationFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="DeleteTagFunction">
  <rdfs:subClassOf>
```

```
    <owl:Class rdf:about="#TagManipulationFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="TagSpecificationFunction">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TagFunction"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="FillOutTagFunction">
  <rdfs:subClassOf rdf:resource="#TagSpecificationFunction"/>
</owl:Class>

<owl:Class rdf:ID="SelectTagFunction">
  <rdfs:subClassOf rdf:resource="#TagSpecificationFunction"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasTaggingParameter">
  <rdfs:domain rdf:resource="#TaggingLink"/>
  <rdfs:range rdf:resource="#TaggingReference"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasTagNavigationReference">
  <rdfs:range rdf:resource="#TagReference"/>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#ontoTags">
  <rdfs:domain rdf:resource="#TagFunction"/>
  <rdfs:range rdf:resource="#TagReference"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#ontoAnnotations">
  <rdfs:domain rdf:resource="#AnnotationFunction"/>
  <rdfs:range rdf:resource="#AnnotationReference"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#fromAnnotations">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TagCloud"/>
        <owl:Class rdf:about="#SelectTagFunction"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#AnnotationReference"/>
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:ID="hasAnnotationFunction">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Annotation"/>
        <owl:Class rdf:about="#AnnotationReference"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
```

```
  <rdfs:range rdf:resource="#AnnotationFunction"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasNewTag">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#EditAnnotationFunction"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#hasObjectChunk">
  <rdfs:domain rdf:resource="#TaggingReference"/>
  <rdfs:range rdf:resource="#ObjectChunk"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#hasUser">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Annotation"/>
  <rdfs:range rdf:resource="#ObjectChunkReference"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#hasResource">
  <rdfs:domain rdf:resource="#Annotation"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#ObjectChunkReference"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasNavigationReference">
  <rdfs:range rdf:resource="#NavigationReference"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#PresentationConcept"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasKeyword">
  <rdfs:domain rdf:resource="#Tag"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasCardinality">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#FillOutTagFunction"/>
        <owl:Class rdf:about="#SelectTagFunction"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#hasTag">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Tag"/>
  <rdfs:domain rdf:resource="#Annotation"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasTagFunction">
  <rdfs:range rdf:resource="#TagFunction"/>
```

```
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Tag"/>
        <owl:Class rdf:about="#TagReference"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>
```

A.2 Tagging Reference

We added an extra reference by which we can refer to a specific tag or annotation.

```
<owl:Class rdf:about="#TaggingReference">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasObjectChunk"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Reference"/>
</owl:Class>

<owl:Class rdf:ID="AnnotationReference">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="referringToAnnotation"/>
      </owl:onProperty>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasRestrictionByUsers"/>
      </owl:onProperty>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasRestrictionByTags"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasRestrictionByResources"/>
      </owl:onProperty>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TaggingReference"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="TagReference">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="referringToTag"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#TaggingReference"/>
</owl:Class>

<owl:ObjectProperty rdf:about="#referringToAnnotation">
  <rdfs:domain rdf:resource="#AnnotationReference"/>
  <rdfs:range rdf:resource="#Annotation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasRestrictionByTags">
  <rdfs:domain rdf:resource="#AnnotationReference"/>
  <rdfs:range rdf:resource="#TagReference"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasRestrictionByResources">
  <rdfs:domain rdf:resource="#AnnotationReference"/>
  <rdfs:range rdf:resource="#ObjectChunkReference"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasRestrictionByUsers">
  <rdfs:domain rdf:resource="#AnnotationReference"/>
  <rdfs:range rdf:resource="#ObjectChunkReference"/>
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:ID="hasAnnotationFunction">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Annotation"/>
        <owl:Class rdf:about="#AnnotationReference"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#AnnotationFunction"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
```

```
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasTagFunction">
  <rdfs:range rdf:resource="#TagFunction"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Tag"/>
        <owl:Class rdf:about="#TagReference"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:about="#referringToTag">
  <rdfs:domain rdf:resource="#TagReference"/>
  <rdfs:range rdf:resource="#Tag"/>
</owl:ObjectProperty>
```

A.3 Tagging Link

A new link type has been added to the ontology for browsing through the Web system based on tags.

```
<owl:Class rdf:ID="TaggingLink">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasTaggingParameter"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Link"/>
  </rdfs:subClassOf>
</owl:Class>
```

A.4 Tagging Presentation Concepts

In this section we will define the formal OWL description for the TagCloud concept.. We added this concept to the list of complex presentation concepts in the WSDM presentation model.

```
<owl:Class rdf:ID="TagCloud">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="fromAnnotations"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
  <rdfs:subClassOf>
    <owl:Class rdf:ID="IndependentComplexPresentationConcept"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="maxTagCloudCategory"/>
      </owl:onProperty>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="maxTagFrequency"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="maxTagsToDisplay"/>
      </owl:onProperty>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
    <owl:onProperty>
      <owl:FunctionalProperty rdf:ID="minTagFrequency"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:FunctionalProperty rdf:about="#maxTagFrequency">
  <rdfs:domain rdf:resource="#TagCloud"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#maxTagCloudCategory">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#TagCloud"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#maxTagsToDisplay">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#TagCloud"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
```

```
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="#minTagFrequency">
  <rdfs:domain rdf:resource="#TagCloud"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:about="#fromAnnotations">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TagCloud"/>
        <owl:Class rdf:about="#SelectTagFunction"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#AnnotationReference"/>
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:ID="hasSortStyle">
  <rdfs:domain rdf:resource="#TagCloud"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil"/>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >freq</rdf:first>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >alpha</rdf:first>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="hasSortOrder">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#TagCloud"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >asc</rdf:first>
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil"/>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >desc</rdf:first>
        </rdf:rest>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:FunctionalProperty>
```