User modeling for exploratory search on the Social Web

Exploiting social bookmarking systems for user model extraction, evaluation and integration

Dissertation

zur Erlangung des Doktorgrades
der Philosophischen Fakultät
der Universität zu Köln
im Fach Informationsverarbeitung

von
Mirko Gontek

Köln, den 10. Juni 2011

Abstract

Exploratory search is an information seeking strategy that extends beyond the query-and-response paradigm of traditional Information Retrieval models. Users browse through information to discover novel content and to learn more about the newly discovered things. Social bookmarking systems integrate well with exploratory search, because they allow one to search, browse, and filter social bookmarks.

Our contribution is an exploratory tag search engine that merges social bookmarking with exploratory search. For this purpose, we have applied collaborative filtering to recommend tags to users. User models are an important prerequisite for recommender systems. We have produced a method to algorithmically extract user models from folksonomies, and an evaluation method to measure the viability of these user models for exploratory search. According to our evaluation web-scale user modeling, which integrates user models from various services across the Social Web, can improve exploratory search. Within this thesis we also provide a method for user model integration.

Our exploratory tag search engine implements the findings of our user model extraction, evaluation, and integration methods. It facilitates exploratory search on social bookmarks from Delicious and Connotea and publishes extracted user models as Linked Data.

Contents

1 Introduction **3**
- 1.1 The problem . . . 4
- 1.2 Scope of this thesis . . . 15
- 1.3 Thesis outline . . . 17
- 1.4 Summary of contributions . . . 18
  - 1.4.1 A user model extraction method . . . 18
  - 1.4.2 A user model evaluation method . . . 20
  - 1.4.3 A user model integration method . . . 21
  - 1.4.4 An exploratory tag search engine . . . 22

2 State of the art **25**
- 2.1 Exploratory search . . . 29
  - 2.1.1 Exploratory search context . . . 34
  - 2.1.2 Exploratory search activities . . . 37
  - 2.1.3 Exploratory search behavior . . . 39
  - 2.1.4 Attributes of exploratory search . . . 45
  - 2.1.5 Challenges . . . 47
- 2.2 User Modeling . . . 51
  - 2.2.1 Personalization process . . . 54
  - 2.2.2 User model content . . . 74
  - 2.2.3 User model representation . . . 86
- 2.3 User modeling on the Web . . . 95
  - 2.3.1 Challenges . . . 98
  - 2.3.2 Approaches . . . 104
  - 2.3.3 Conclusion . . . 139
- 2.4 The Social Web . . . 143
  - 2.4.1 Characteristics . . . 145
  - 2.4.2 Social bookmarking . . . 151
  - 2.4.3 Potentials for exploratory search . . . 157
- 2.5 Summary . . . 167

3 Contribution 1: A user model extraction method 171
3.1 Knowledge representation on the Web . . . 179
3.1.1 Ontologies versus folksonomies . . . 183
3.1.2 Connect ontologies and folksonomies . . . 193
3.2 Ontology modeling methods . . . 195
3.2.1 Ontology engineering . . . 196
3.2.2 Ontology learning . . . 200
3.2.3 Crowd-sourced ontology engineering . . . 204
3.2.4 Ontology extraction . . . 211
3.3 An abstract model for semantics extraction from folksonomies 212
3.3.1 A tripartite graph . . . 214
3.3.2 Affiliation graphs . . . 215
3.3.3 Similarity . . . 216
3.3.4 Statistical Analysis . . . 217
3.3.5 Semantic interpretation . . . 218
3.4 Collaborative Filtering . . . 219
3.5 Output . . . 231
3.6 Extraction algorithms . . . 233
3.6.1 User interest prediction . . . 234
3.6.2 Similar users prediction . . . 236
3.6.3 User knowledge extraction . . . 238
3.7 Conclusion . . . 244

4 Contribution 2: A user model evaluation method 251
4.1 Evaluation strategies . . . 258
4.1.1 Predictive Accuracy . . . 259
4.1.2 Novelty . . . 263
4.1.3 Coverage . . . 265
4.1.4 User-perceived quality . . . 266
4.2 Our evaluation method . . . 268
4.3 Input dataset . . . 269
4.3.1 Tag popularity . . . 271
4.4 Recommender algorithms . . . 273
4.4.1 Algorithm 1: Unbiased . . . 273
4.4.2 Algorithm 2: Biased towards popular tags . . . 274
4.4.3 Algorithm 3: Biased towards unpopular tags . . . 274
4.5 Evaluation datasets . . . 275
4.5.1 Recommendation networks . . . 277
4.5.2 Query result matrix . . . 279
4.6 Experiments . . . 281
4.6.1 Domain discovery . . . 281

4.6.2 Domain learning . . . 309
4.7 Conclusion . . . 326

5 Contribution 3: A user model integration method 335
5.1 A user modeling architecture . . . 339
5.1.1 Activity Providers . . . 343
5.1.2 Service User Interfaces . . . 345
5.1.3 User Modeling Services . . . 346
5.1.4 Personalization Service . . . 355
5.1.5 User Property Statements . . . 356
5.2 A user model scheme . . . 360
5.2.1 User interest . . . 364
5.2.2 User knowledge . . . 365
5.2.3 User similarity . . . 366
5.2.4 Metadata . . . 367
5.3 A user model representation method . . . 372
5.3.1 Representation problems . . . 373
5.3.2 Representation patterns . . . 377
5.3.3 User Property Statements . . . 391
5.3.4 Metadata . . . 405
5.4 Conclusion . . . 407

6 Contribution 4: Earlybird – an exploratory tag search engine 413
6.1 Functionality . . . 420
6.2 Software architecture . . . 427
6.3 Data Maintenance . . . 431
6.4 Presentation . . . 436
6.5 Application . . . 443
6.6 Storage . . . 458
6.6.1 Bookmark management . . . 465
6.6.2 Account management . . . 474
6.6.3 RDF user model publishing . . . 476
6.6.4 Usage logging . . . 489
6.7 Conclusion . . . 492

7 Conclusions 499
7.1 Summary of contributions . . . 501
7.1.1 User model extraction . . . 501
7.1.2 User model evaluation . . . 503
7.1.3 User model integration . . . 504
7.1.4 An exploratory tag search engine . . . 505

7.2 Limitations and further research 507
7.2.1 User model extraction 508
7.2.2 User model evaluation 510
7.2.3 User model integration 511
7.2.4 Theory and practice 513

Bibliography **515**

List of Figures **583**

List of Tables **587**

Chapter 1

Introduction

The goal of this thesis is to improve information retrieval on the Web. Query-and-response Web search engines, where users can type in search queries and get a list of results, have been the predominant strategy to retrieve information on the Web for the last decade. But the query-and-response paradigm does not fully cover the strategies involved in finding and accessing information on the Web. Everyday information retrieval strategies on the Web are not limited to querying for keywords and selecting the best search result. In this thesis, we take a broader perspective on information retrieval on the Web.

1.1 The problem

We will start with an example to illustrate the problem addressed in this thesis and to sketch our vision of information retrieval on the Web.

Meet Tom, a social media expert who works in social media marketing and has reasonable knowledge on the topic. Tom keeps himself updated on what happens in his field of work with a social search engine: Earlybird. Tom uses Earlybird to keep up-to-date on the topic *social media*. Earlybird helps him to regularly check for new webpages, blog articles, tutorials, and companies, and to

stay up-to-date with news and recent trends. Earlybird retrieves webpages that have been recently tagged in social bookmarking services. Social bookmarking services are Web services that enable users to share newly discovered webpages by bookmarking and annotating webpages with tags – keywords which describe the webpages. Earlybird helps Tom to stay up-to-date by retrieving the latest content on *social media*.

Tom also has a more general interest in the *mobile web*, a subfield of *social media*. But Tom is not an expert in *mobile web*, his knowledge here is rather superficial. For example, Tom has no specific knowledge of *geofencing*. He has seen the term on a few occasions, but he does not know that it describes a technique to localize mobile web devices commonly used by location-based services. Earlybird records the search queries of Tom and all other users. It assumes that Tom has interest and some knowledge on *social media*, because he regularly checks for social media-related webpages. Earlybird knows that many users that have searched for *social media* have also searched for *geofencing*. Through further analysis, Earlybird derives that the topic *geofencing* is a subfield of *social media*. Tom has not searched for *geofencing* yet, although the topic is related to *social media*. Therefore, Earlybird informs Tom that *geofencing* is a subtopic of *social media* when he searches for *social media* the next time. Earlybird recommends to search for the topic and Tom follows the recommendation. A quick scan of the search results for *geofencing* shows what the term means. Earlybird also informs Tom that other topics are related to *geofencing*: *location-based services, gps, maps* – some of these topic are topics in which Tom has some interest and knowledge. Tom gets an idea of how *geofencing* is related to topics which he knows more about. Earlybird helped Tom to discover that *geofencing* is indeed a highly relevant topic for his work in social media marketing.

Tom becomes aware that he really should dig deeper into the *geofencing* topic. He wants to find out what the important trends

in *geofencing* are. He reads through the search results for the topic. Over time, Tom acquires a deeper knowledge of *geofencing*. He learns about current trends in the topic and gets to know the important blogs that cover the topic. Earlybird suggests a subtopic of *geofencing*: *foursquare*. Tom scans through the results for the suggestions and learns that the service is one of the most popular *geofencing* applications. After Tom has gained reasonable knowledge on *geofencing*, he wonders how *geofencing* is used in the context of social media. Earlybird informs Tom, as he searches for *geofencing*, that *social media* and *geofencing* are related. Tom adds *social media* – suggested as a related topic – to his initial search, thus narrowing down the search results to webpages that are related to both *geofencing* and *social media*. He reads through the results and gains new insight into geofencing in social media services. Later, he wants to get even more specific results. He adds a suggested subtopic of social media – *foursquare* – to the search query. The results are now narrowed down to *geofencing* and *foursquare*. Earlybird helped Tom to deepen his knowledge on *geofencing*.

The example illustrates common search strategies in the everyday lives of today‘s Web users. It shows that search is a complex process that goes far beyond pure fact retrieval to answer well-defined questions. Questions and answers are often fuzzy and ill-defined – users „submit a tentative query and take things from there, exploring the retrieved information, selectively seeking and passively obtaining cues about where the next steps lie“ [222]. Information seeking activity is exploratory in nature and leads to discovering novel topics, learning more about known topics, and ultimately to the acquisition of more knowledge.

The example shows three strategies that we apply when we search the Web. First, it shows that we manually filter information for certain criteria to identify relevant results from irrelevant in a set of webpages. We scan

through search results and filter them to identify the best result. This usually happens intuitively; we may know from our experience that results from certain Web domains are more useful for our personal goals – results from the Web domain *wikipedia.org* often helped to retrieve key facts on an unfamiliar topic, results from *acm.org* often retrieved scientific articles. The Social Web enables new forms of information filtering – social filters. Social Web services allow us to filter information according to social criteria. For example, social bookmarking services allow us to filter the Web for webpages that have been bookmarked by certain people we trust. Second, the example illustrates that we do not always search explicitly, but discover things more or less accidentally. In many cases, we don‘t know what to search for. We accidentally discover topics by browsing or by getting hints in some form. For example, we may scan through popular tags that have been used in a social bookmarking service to discover new tags we were not aware of. Third, the example illustrates that we use search to dig deeper into topics that interest us – we interact with search results and adapt search queries to gain more insight. As we learn more about a topic, our goals change. Novices may first want key facts on the topic. After they have gained some insight on the topic, they may look for more expert results. The combination of these three search strategies – filtering, discovery, and learning – frame what has become known as *explorative search*.

In our introductory example, we defined a vision of how to support exploratory search in a search engine. Traditional query-response search engines aim at supporting *fact-retrieval*. Our vision is a search engine that supports exploratory search. We believe that much information retrieval activity on the Social Web is exploratory in nature. Therefore, we think that exploratory search could greatly improve information retrieval on the Social Web. Our envisioned search engine – Earlybird – supports the three key tasks of explorative search: *filtering*, *discovery*, and *learning*.

Earlybird involves two filtering strategies. First, it filters results by time. Earlybird exploits data from existing social bookmarking services to obtain search results. One strength of social bookmarks for search is that users bookmark webpages as soon as they discover them. This allows them to

retrieve current information. Earlybird ranks retrieved results by date to promote the very latest webpages. Second, Earlybird filters through *social* similarity of users. Earlybird suggests related topics to users considering earlier queries of similar users. Earlybird filters through the information of similar past users. It recommends *geofencing* to Tom because similar users interested in *social media* also expressed interest in *geofencing*.

The recommendations based on this social filtering strategy enable discovery and learning. The recommendations are contextual to the search queries, i.e., Earlybird suggests queries that are related to the current search query. It enables users to explore the context of their current search query. Recommendations act as navigational cues to guide users through an information space. Recommendations can guide users in two directions. First, they can support the discovery of related topics. This allows users to discover information that they did not actively search for. In this case, the recommendations act as navigational cues to promote *accidental* discovery of unknown – novel – topics. In our example, Earlybird recommends *geofencing* to Tom. He discovers the topic, which has been more or less novel to him before. Second, the navigational cues can support digging deeper into a topic. They can guide the user to more specific queries on a topic or provide context by guiding him to more general queries. In this case, recommendations act as navigational cues to promote topic learning. In our example, Earlybird suggested *foursquare* as a more specific query related to *geofencing*. This allows Tom to learn about more specific aspects of the topic domain. In the opposite direction, Earlybird suggests *social media* as a more generic query related to *geofencing*. This allows Tom to learn about the context in which geofencing is used.

The recommendations rely on a technique called *collaborative filtering* – Earlybird suggests topics to be related because other users consider them as related. Therefore, Earlybird maintains user models which model the users interest and background knowledge. Earlybird analyzes which tags users use to annotate their shared bookmarks. User models are the key prerequisite for collaborative filtering. They allow the prediction of similarities between queries and make user-specific recommendations. They can bias

recommendations towards known or novel topics. In our example, Earlybird recommended *geofencing* to Tom because the topic is closely related to *social media*, yet was, according to his user model, still a novel topic. In this case, biasing towards an unknown topic aims at promoting the discovery of a novel topic.

We illustrated the problem that underlies this thesis – exploratory search on the Web. We envisioned a search engine that addresses the key challenges of the problem: filtering, discovery, and learning. We consider the Social Web – and particularly social bookmarking services – to have great potentials for user modeling to improve exploratory search. Our envisioned search engine exploits social bookmarks to extract user models for exploratory search. In this thesis, we examine how to enable exploratory search on the Social Web.

1.2 Scope of this thesis

This thesis is strongly interdisciplinary. It comprises several research areas of the information and computer sciences to address the problem of exploratory search on the Social Web. We think that our perspective, which combines techniques and models from various research areas, yields new insights into the problem.

The starting point of our thesis is exploratory search, a sub-discipline of **Information Retrieval**. User modeling is an important element of exploratory search that has been investigated predominantly in the context of adaptive learning and **adaptive Web systems**. The techniques for automated user model extraction on the Web use **machine learning**. We apply exploratory search to the Social Web. We bridge the gap to the **Semantic Web** with a pragmatic solution that applies **Linked Data** solutions.

We think that this thesis provides new insights to readers from all of these research backgrounds. We are aware that extant knowledge on techniques, tools, and theory may vary greatly between readers. In the introductory chapter of this thesis we do not extensively introduce all broached areas. Depending on the background of the reader, some parts of the introduction may be too dense or too shallow. We decided to compromise: a presentation

that is reader-friendly, but that includes references to existing literature in the respective research domains, should anyone wish to read a more detailed introduction on the subject.

1.3 Thesis outline

The thesis is structured as follows: **Chapter 2** provides background information and summarizes the current, state-of-the-art research research in this field. In **Chapter 3**, we contribute a method to extract user models from social bookmarking services. In **Chapter 4**, we contribute a method to evaluate the extracted user models. In **Chapter 5**, we contribute a method for user model integration on the Social Web. In **Chapter 6**, we contribute Earlybird, a real-world exploratory tag search engine that was implemented as the 'proof-of-concept' to our findings. Finally, in **Chapter 9**, we conclude with a discussion of our results and how further research on the topic could be conducted.

1.4 Summary of contributions

The main contributions of this thesis are: (1) a user model extraction method, (2) a user model evaluation method, (3) a user model integration method, and (4) an exploratory tag search engine.

1.4.1 A user model extraction method

First, we contribute a user model extraction method. Our method applies collaborative filtering to algorithmically extract user models from folksonomies. The method has the following properties:

(a) it extracts user models that are important to facilitate exploratory search.

(b) it transfers implicit knowledge inherent in folksonomies into more formal ontologies.

(c) it is fully automized, hence does not require manual intervention in the extraction process.

(d) it applies collaborative filtering algorithms which are content-agnostic.

(e) it is computational cheap, because it considers only the structural features of folksonomies.

(f) it extracts user models comprised of user interest, user similarity, and user knowledge.

(g) it extracts associative and hierarchical semantic relations between tags.

1.4.2 A user model evaluation method

Second, we contribute a user model evaluation method. Our method consists of a set of experiments to evaluate user model extraction algorithms for exploratory search. The method has the following properties:

(a) it evaluates how the extracted user models facilitate the key tasks of exploratory search: domain discovery and domain learning.

(b) it evaluates non-accuracy measures, which are important in the context of exploratory search: novelty and coverage.

(c) it relies on the structural analysis of networks that result from precompiling recommendations. It measures the navigability of the recommendation networks.

(d) it considers the impact of the recommendations on the search results.

(e) it does not require user feedback, hence allowing the algorithms to be evaluated before they have been deployed in a system.

1.4.3 A user model integration method

Third, we contribute a method for user model integration on the Social Web. Our method allows for web-scale user modeling to improve exploratory search engines. The method has the following properties:

(a) it defines an abstract architecture for web-scale user modeling on the Social Web.

(b) it defines a user model content scheme for the Social Web.

(c) it defines a user model representation format that uses Semantic Web ontologies. The representation format ensures semantic integration of the user models with popular Semantic Web vocabulary.

(d) it provides a strategy to semantically align complex user models with generic Semantic Web vocabulary.

(e) it provides a practical approach for user model integration, which can be adopted easily by existing Social Web services.

(f) it is an evolutionary approach that bridges the gap between Social Web and Semantic Web technologies.

1.4.4 An exploratory tag search engine

Fourth, we contribute a real-world exploratory tag search engine named *Earlybird*. Earlybird is the proof-of-concept of our findings. It implements the key ideas of exploratory search and allows the search of data collected from two social bookmarking services, Delicious and Connotea. The contribution has the following properties:

(a) it is a real-world exploratory Web search engine that allows the search for tags.

(b) it has a *live* connection to two running social bookmarking services, Delicious and Connotea.

(c) it implements exploratory search facilities for domain learning and domain discovery.

(d) it implements our user model extraction and integration methods.

(e) it shows that our user model extraction and integration methods work in the real-world.

(f) it publishes extracted user models as RDF over a SPARQL endpoint.

(g) it acts as a test framework that allows the implementation, testing, and comparison of various recommender algorithms.

(h) it collects usage logs for future user-centric evaluations.

Chapter 2

State of the art

Exploratory search is a form of information seeking activity that has gained increasing importance in the context of the Social Web. Exploratory search focuses on how to support users to explore information on the Web. The idea of exploratory search is to combine query-and-response search with browsing. Exploratory search supports users in discovering topics that are still unfamiliar and in deepening their knowledge of the topics that they are already familiar with. For the users, the two key goals of the exploratory search are to discover unknown – novel – topics and to gain deeper knowledge of familiar topics.

The process of exploratory search is fuzzy and highly interactive. The main challenge faced by exploratory search engines is guiding users through the interactive search process and to the correct information in a huge information space, the Web. To guide users, exploratory search engines give navigation cues that help users to adapt their search, thus pointing it into the right direction. To suggest helpful navigation trails through the information space, search engines must understand the needs and goals of users.

User-specific information – preferences, opinions, skills, and existing knowledge of users – help exploratory search engines to estimate the needs and goals of users. This information is usually referred to as *user models*. Exploratory search can greatly profit from user models. For exploratory search engines, user models represent empirical knowledge about users and past

search activity which can be used to predict the needs and goals of future users. Additionally, user models allow the search process to be individually adapted for each user, based on the user attributes. Hence, *user modeling* – the creation, maintenance, and exploitation of user models – is an important element of exploratory search.

The Social Web creates new potential for user modeling. Social bookmarking services allow new strategies for user model extraction to be developed. Integration of user models across Social Web services enhances the quality and coverage of user models. Our goal is to leverage the Social Web for exploratory search through web-scale user modeling.

The remainder of this chapter is organized as follows. In Section 2.1, we introduce exploratory search and characterize its goals and challenges. In Section 2.2, we give a brief introduction to the user modeling process. We describe how user modeling is involved in the personalization process, how information required for user modeling can be collected on the Web, and how user models are constructed from this information. Furthermore, we describe the information contained within user models and how it is represented. In Section 2.3, we address user modeling on the Web. We discuss the challenges of distributed user modeling. Our goal is to present user modeling on a web-scale in order to enable exploratory Web search. We describe existing solutions for distributed user modeling and their limitations for our needs. In Section 2.4, we introduce key concepts of the Social Web and social bookmarking services. Finally, we summarize this chapter in Section 2.5

2.1 Exploratory search

Exploratory search can be described as a form of interactive information seeking activity that extends the notion of information seeking beyond the traditional lookup-based concept of information seeking most predominant in Information Retrieval. Most successful search engines available on the Web today have been developed with Information Retrieval models, which researches how to electronically store, manipulate, retrieve and disseminate information, and how to retrieve these information efficiently [214, 180, 14, 137].

Information Retrieval has strongly influenced how we understand information seeking on the Web [17]. The predominant model for information seeking in Information Retrieval research is a query-and-response pattern which is applied to the lookup of information. The lookup-based Information Retrieval model involves four components (1) a set of documents that are to be searched – e.g. webpages, (2) a representation of the documents that simplify or speed up lookup – indexes or keywords, (3) the information need of the user, and (4) a search query that represents this information need as a text string (cf. [225]). The model assumes that any search process begins with the user's well-defined information needs. Users translate their information needs into a textual query – a conception of the user's information need. The search engine yields results that match the information need with minimal need for examination (cf. [225]). The lookup model from Information Retrieval has been successful developing search engines that support lookup search scenarios like *fact-finding* or *question-answering*. However, real-life Web search often extends beyond lookup-based information seeking [19, 156]. Usually, it involves several query iterations, browsing through results, and a closer examination of results. The lookup-based retrieval model is not capable of these activities. Due to this shortcoming, new sub-disciplines of Information Retrieval have emerged to improve our understanding of how users retrieve information, leading to a new, broader perspective of the information retrieval process that focuses on how users are involved in the retrieval process and how they interact with search engines: Interactive and Cognitive Information Retrieval [178, 195], Information Visualization [237, 72, 133], information foraging [168, 184, 54], and exploratory search [225]. The user's role in the search process is of central importance to the models of the above-mentioned areas of Information Retrieval. The extended perspective has resulted in new information retrieval systems that consider the user‘s task, domain knowledge, experience, and context as determinants for successful information retrieval [225, 138, 175, 58, 59].

Exploratory search considers information systems from a user-centric viewpoint. It actively engages the user in the search process. Exploratory search assumes that information seeking activity involves mental activities

that require more than the pure retrieval of information, e.g., learning and decision-making [146, 222, 224, 225]. Exploratory search aims at fulfilling the information-seeking requirements necessary for these activities. In this thesis, we adopt the following definition of exploratory search:

> Exploratory search can be used to describe an information-seeking problem context that is open-ended, persistent, and multi-faceted; and to describe information-seeking processes that are opportunistic, iterative, and multi-tactical. In the first sense, exploratory search is commonly used in scientific discovery, learning, and decision making contexts. In the second sense, exploratory tactics are used in all manner of information seeking and reflect seeker preferences and experience as much as the goal. [222]

Our introductory example illustrates the idea of exploratory search. It involves many aspects that illustrate what makes a search *exploratory*. Exploratory search takes a broader perspective on the information retrieval process – it involves a broader range of search activity than Information Retrieval traditional does. It emphasizes the idea of exploration within the search process. Additionally, exploratory search emphasizes the cognitive activities of the users involved in the search process and the aspect of knowledge acquisition during the search process. Exploratory search aims at knowledge acquisition during the search process by providing the users with helpful information. The purpose of exploratory search is to deepen the understanding of a topic, to enhance the user‘s capabilities or skills, to support decision-making, and ultimately to support developing intellectual capabilities within a topic domain.

2.1.1 Exploratory search context

Exploratory search differs from lookup-based search by the *context* in which users search – the motivating factors that yield the search. The search context of exploratory search is usually more ill-defined than a lookup-based search. All kinds of information seeking are motivated by some kind of desire

for information that motivates the search process. Lookup-based information retrieval usually assumes that users have an information need that they want to satisfy. In exploratory search, however, the motivation is not necessarily a specific *need* for information. The motivation can also be *curiosity* or a desire to increase knowledge of a particular topic. The goals of a search and the actions necessary to achieve helpful results may be unclear. Hence, the context in exploratory search is usually ill-structured. Users usually do not have any existing domain knowledge or systematic routines that lead to a better understanding of the domain. A viable strategy to address a complex information seeking task is to decompose the task into smaller and less complex tasks. However, for an ill-defined problem, it is difficult for the user to identify sub-tasks in advance. Users can only learn about the structure of a problem during the search process. They construct a solution to the problem by accumulating information from the information space during the search process. The search context is also ill-defined because the information need often shifts during the search process. As users gain knowledge of a topic and gain new insight into a topic, they may redefine their motivation. The information need is in constant flux. An important aspect of exploratory search is therefore to support ill-defined information seeking. Exploratory search engines require facilities that can guide users through an information space, helping them to accumulate the information that will assist them in understanding, formulating, structuring, and ultimately solving the ill-defined problem that underlies the search (cf. [225]).

Our introductory example illustrates the ill-defined nature of the information need that underlies exploratory search. Initially, Tom‘s motivation was to keep himself up-to-date with the topic *social media*. This information need had no clearly defined goals. Tom assumed that new trends were always emerging in the domain, but he had no clear idea how these trends were manifested or how to detect them. Out of curiosity, he regularly checks webpages on the topic for any information that had recently emerged. During the search process, his information need changed as he discovered a sub-topic which is novel to him: *geofencing*. This caused him to want to gain initial information on this novel topic. After becoming familiar with the domain,

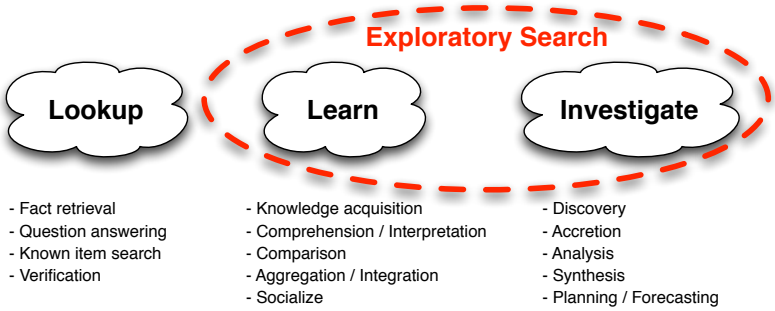

Figure 2.1: Search activities (based on [146])

he decided he wanted to gain deeper insight into *geofencing*. Throughout the search process, Tom's information need changed several times.

2.1.2 Exploratory search activities

[146] propose three groups of information seeking activities involved in information retrieval: *Lookup*, *Learn*, and *Investigate*, see Figure 2.1. The activity groups categorize common search activities, such as fact retrieval, knowledge acquisition or discovery of novel information. The groups overlap and are usually interconnected – a search may well involve search activities from more than one activity group. The categorization frames the scope of exploratory search: the main elements of exploratory search are *Learn* and *Investigate* activities.

The categorization helps to differentiate exploratory search activities from *Lookup* activities. *Lookup* aims at retrieving facts and answering discrete questions. It returns well-structured objects that answer or verify well-defined questions. *Learning*, in contrast, is more complex and involves multiple iterations. Users usually spend some time scanning information; they accumulate and compare resources and make qualitative judgments. The goal of the *Learning* is knowledge acquisition; it implies comprehension and the interpretation of data. In the third category, *Investigation*, users analyze, evaluate, and assess results to gain novel information. They integrate the novel information into existing knowledge. *Investigation* also involves multiple iterations and can take place over longer time periods. The goal of

Investigation is the discovery of information. Discovery of novel information usually emanates from the substantial existent knowledge. Besides discovering novel information, *Investigation* may also aim at discovering gaps in existing knowledge (cf. [225]).

2.1.3 Exploratory search behavior

The activities involved in the explorative search process – Investigating and Learning activities – imply two tasks that must be facilitated by exploratory search engines: *exploratory browsing* and *focused searching*.

Domain discovery

Investigation implies *domain discovery*. Domain discovery describes gaining *novel* knowledge based on existent knowledge, i.e., broadening existing knowledge. The central idea of domain discovery is the discovery of knowledge of novel topic domains, i.e., exploration of novel domains that are potentially interesting for a user. Users extend knowledge beyond already known topic domains – they broaden domain knowledge. For example, users may want to discover novel topic domains that are related to their current information needs and that meet their general interests.

Domain discovery implies *exploratory browsing*. Exploratory browsing facilities support users in discovering novel information. They expose users to information, with the goal to discover novel domains which are relevant and appropriate to broaden the users' domain knowledge. An important element of exploratory browsing is navigation. Exploratory browsing facilities provide navigation cues through the information space to discover novel information and relate it to existent knowledge. For example, exploratory browsing facilities could suggest topic domains that are related to the current search query. These suggestions help users to discover knowledge from novel domains and relate it to existent domain knowledge. Navigation cues help to mentally relate individual domains to each other. It is an important aspect of discovery, because it allows one to make sense of new knowledge. Navigation between known and unknown domains can also serve to identify knowledge gaps.

Our introductory example illustrates how exploratory browsing can be implemented. The envisioned exploratory search engine suggests similar tags for the current search query tag. It suggests that *mobile web* and *geofencing* are related to *social media*. Tom can follow recommendations to adapt the search query. Thus, he can browse through the recommendations to discover novel domains related to his initial search query. The recommendations are navigation cues that enable exploratory browsing.

Domain learning

Learning allows for *domain learning*. The central idea of domain learning is to extend existing knowledge while inside a specific domain, i.e., to deepen knowledge. Domain learning describes the process of acquiring more knowledge inside a specific topic domain. Users gain new knowledge about an already known topic – they learn more inside a topic domain. Users dig deeper into a domain to retrieve newly available information related to a topic domain and to reach more specific information from general information.

Domain discovery is facilitated by *focused search*. Focused search facilities help users to gain more specific knowledge of an already known topic domain. Navigation is an important element for focused search. Focused search facilities guide users through information inside a domain, as opposed to exploratory browsing, which supports navigation between domains. Focused search usually provides the navigation cues, which help one to reach specific information from more general information. It supports mental analysis of a domain. In our introductory example, focused searching is facilitated by tag recommendations, which enable one to navigate to more specific topics. Our envisioned search engine suggests hierarchical relations between query and recommendation tags. For example, *foursquare* is considered a sub-topic of *geofencing*. This information guides Tom to more specific information on the general domain *geofencing*.

A further aspect of focused search is long-term learning. Users may regularly search for a known domain to analyze a domain – to stay up-to-date

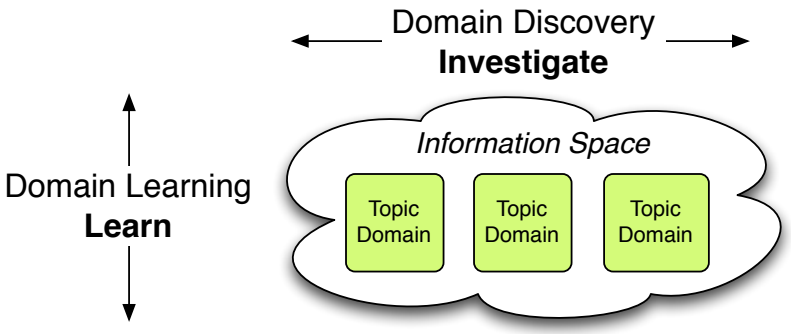

Figure 2.2: Exploration dimensions

with a topic, to see how it develops, or to detect emerging trends. Focused search facilities may expose users to novel information in topic domains that they are already familiar with. In our introductory example, focused search for long-term learning is enabled by the ranking of the search results. Our envisioned search engine ranks results according to the date that the webpages have been tagged at. Tom can regularly perform the same query, *social media*, to retrieve the latest tagged information on the topic, which we assume to be up-to-date information.

Exploratory search relies on a balanced combination of exploratory browsing and focused search. Exploratory search engines must provide facilities for both activity types. Exploratory browsing and focused search allow one to explore an information space in two dimensions, see Figure 2.2. On the horizontal axis, users investigate – they discover novel domains and contextualize it with existent domain knowledge. Usually, domain discovery is enabled by facilities that support navigation between domains. On the vertical axis, users explore one domain in depth – they analyze a familiar domain deepen their knowledge of that topic. This is usually enabled by facilities that allow search queries to be specified in order to retrieve more detailed information; it may also involve facilities that support long-term learning.

2.1.4 Attributes of exploratory search

To conclude the characterization of exploratory search, we have adopted the summary of attributes of exploratory search by [225]:

(1) Exploratory search involves multiple query iterations and usually multiple search sessions as well. The search process lasts longer insofar as it can extend over days or months. Exploratory search engines should support search activity that extends over long periods of time. User models that log searches, interests and preferences of users over sessions support long-term exploratory search activities.

(2) The information needs that motivate exploratory search are ill-defined as „open-ended, persistent, or multi-faceted“ [222]. Users are usually unclear as to what information is available. Furthermore, the task that one is to complete is often still indeterminate. Information needs often change during the learning and investigation process.

(3) The goal of exploratory search is not simply finding the right information in an information space. Instead, the focus is on learning and understanding. Knowledge about the context of information is important. The emphasis of exploratory search is on personal development, not on finding information.

(4) Interaction facilities during the exploratory search process involve exploratory browsing and focused search. Only a combination of both facilities produce effective exploratory search.

(5) Exploratory search may involve collaboration between users. Exploitation of communities of interest, e.g., from Social Web services [146], can engage other users in a collaborative search process that utilizes community knowledge for the exploratory search process.

(6) Evaluation of exploratory systems is complex. It must consider which insights and knowledge users gained and how well the learning process is supported by the system. In particular, how well a search engine facilitates the key activities of exploratory search, namely, domain discovery and domain learning must be evaluated [223].

2.1.5 Challenges

An important feature of exploratory search engines is their ability to facilitate and combine exploratory browsing and focused search. Navigation is a crucial issue for both activities because it guides users through the available information to help them solve their search problems:

> Exploratory searchers utilize a combination of searching and browsing behavior to navigate through (and to) information that helps them develop powerful cognitive capabilities and leverage their newly acquired skills to address open-ended, persistent and multifaceted problems [225].

Hence, navigation facilities are key elements of exploratory search engines. An important challenge when developing exploratory search engines is to provide users with the appropriate navigation facilities for exploratory browsing and focused search. Exploratory search engines provide navigation cues to guide users to appropriate information. The navigation cues suggest browsing trails to guide the user through the information space. The search engine must suggest browsing trails for both exploratory browsing and focused search, because exploratory search relies on a combination of both activities. The system must allow for navigation in both dimensions in the information space to enable domain discovery and domain learning.

The challenge of navigation facilities for domain discovery is to provide navigation cues that guide users to information that (1) is relevant for the current information need, (2) is novel so that it extends the users' knowledge, and (3) relates to familiar domains so that users can contextualize the novel information with existing knowledge.

The challenge of navigation facilities for domain learning is to provide navigation cues that guide users to information that (1) has an appropriate level of specificity for the current information need and the users‘ existent knowledge, and (2) relates to familiar information , allowing users to contextualize the information; navigation cues usually guide users from general information to more specific information on a domain.

The challenges imposed by the navigation for domain discovery and domain learning suggest browsing trails that are context- and user-specific. Deciding which information is novel to a user, which particular information need a user has, and which existent knowledge is assumed to have already, requires extensive user-specific knowledge. Therefore, user models are an important element of exploratory search. The personalization of exploratory search engines enables navigation cues for a given search context and a given user, based on the user‘s knowledge, interest, and social affiliation. Particularly, recommendations are a viable technique to guide users through the information space by suggesting queries, webpages, or similar users. Recommenders can make user-specific recommendations, which then enable the personalized navigation trails that guide users to the appropriate information.

2.2 User Modeling

Adaptive Web systems automatically adapt or *personalize* webpages and Web services to satisfy the needs and preferences of specific users by considering knowledge gained from the individual user‘s past behavior [40, 43, 42, 69]. Adaptive search engines dynamically deliver customized information through automated prediction of the user‘s needs [204, 38, 197, 207]. User models are required if systems are to adapt information based on users‘ knowledge, interests, preferences, opinions, or past activities. Personalization has been successfully incorporated into Web search [78, 150, 46, 194, 202, 142, 230] and is an important strategy to improve navigation facilities for exploratory browsing and focused search in exploratory search.

The goal of personalization is to customize information for users. Therefore, a personalized system must have knowledge of the user‘s interests and preferences. This knowledge can be expressed either explicitly by the user or acquired by the system automatically or semi-automatically. Implicit user modeling has proven to be more powerful in practice. The goal of personalized search engines is usually to customize information access without requiring the user to express preferences explicitly, but, rather, to „provide users with the information they want or need, without expecting from them to

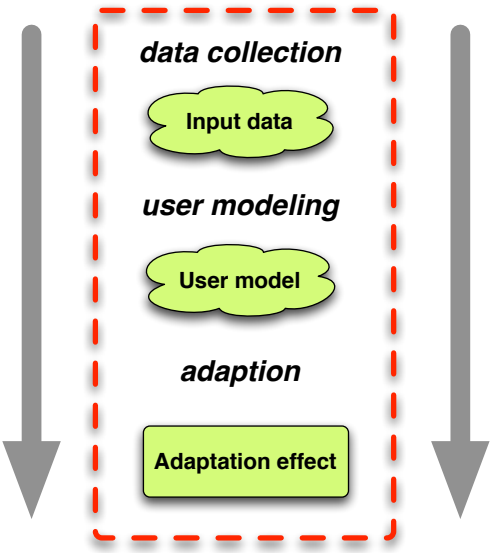

Figure 2.3: Personalization process (based on [39])

ask for it explicitly“ [158]. To provide users with useful information without being explicitly asked, the system must somehow infer what individual users might want or need. This inference is based on how the user has interacted with the system or other systems earlier, e.g., by analyzing earlier queries and clicks of the user. Thus, personalization is usually based on automatically inferred knowledge about the user, which is acquired by collecting and analyzing user behavior. The automated learning of knowledge relevant for personalization is an important component of personalized systems in which machine learning techniques are usually applied to learn about users from collected data [150, 46, 194, 202, 142, 230, 171]. The gained knowledge is user-specific in that it represents the preferences and interests of individual users. This user-specific information models individual users in the personalized system and are referred to as *user models*. User models are the key prerequisite for personalized systems.

2.2.1 Personalization process

The user model acquisition process is referred to as *user modeling*. In the following sections, we review how user modeling is involved in the personal-

ization process and discuss the user modeling in more detail. The personalization process involves three stages: (1) collecting data about the user, (2) constructing or updating the user model, and (3) applying the user model to create an adaptation effect (cf. [39, 69]). Figure 2.3 shows how user modeling relates to the personalization process.

Information collection

The first step of the personalization process is collecting information about users. In this section, we explain which information is relevant for personalization. A prerequisite for the collection of user-specific information is the identification of individual users. Therefore, techniques for user identification are described, followed by a description of the two methods for information collection: explicit and implicit information collection.

Relevant data The input data for the personalization process can be categorized into *usage data*, *content data*, *structure data*, and *user data* (cf. [87]).

(1) *Usage data* describe the navigation behavior of users on a webpage – for example querying history [194] or click-through history [118, 171, 120]. Usage data are collected, traditionally, from server access logs. The access logs record which resources the user accessed and the specific time those resources were accessed. On the Social Web, the notion of usage data becomes more complex [46, 150]. Contributions from users at Social Web services (bookmarks, comments, blog entries, tweets, private messages, etc.) are all relevant usage data but must be interpreted differently. On the Social Web, logging usage data is a more complex issue than simply logging HTTP requests on a server. Logs usually take place at application level to fully leverage the application‘s personalization potential. For example, to create usage data at an application-level, a Social Web service may log the webpages a user bookmarks, and the tags the user attaches to the webpages.

(2) *Content data* represents the actual content delivered to the user, which includes unstructured text, images, videos, or structured content such as query result lists or RSS feeds.

(2) *Structure data* describe how the content delivered to the user is structured. Markup and hyperlinks have traditionally been a relevant structuring element on the Web [81]. On the Social Web, relations between users in a system, e.g., friends and groups in a social network, are important structure data [114]. User modeling usually requires knowledge of both the content and structure of a resource to interpret the usage data. For example, to interpret a user‘s navigation on a webpage, it is necessary to combine the usage data with the structure of the page, i.e., which navigation options were available to the user [166].

(3) *User data* represent information about the user, e.g., demographics, user knowledge, skills and capabilities, interests and preferences, goals and plans. They are created either by the user, explicitly through the creation of a model or implicitly through analysis of usage data. For example, in the context of the Social Web, explicit and implicit user data can be collected from social network profiles [46]. Alternatively, usage data, like tagging behavior in a social bookmarking service, can be analyzed to derive preferences implicitly [186]. Explicit models are often subjective and thus problematic – users tend to bias user profiles intentionally or unintentionally. Implicit models created by analysis of usage data are usually more reliable and powerful [203]. However, the border between explicit and implicit modeling is blurry. Explicit user data can often be enhanced by statistical analysis to reveal further, underlying, implicit information. For example, explicitly created user data like ratings or reviews for items are clearly relevant as such. However, they are highly subjective and possibly more useful when abstracted or combined with further information. Machine learning applied to a greater amount of rating data can yield user data that are often more reliable and meaningful than individual user ratings.

User identification Users must be distinguished in order to collect information about individual users [69]. Data models of personalized systems are user-centric [5], i.e., the output of the user modeling process is information specific to individual users. Therefore, all information collected must be attributable to individual users or user groups to create user models. [78]

distinguish five methods to identify individual users in Web-based systems: logins, cookies, session IDs, software agents, and enhanced proxy servers.

(1) *Proxy servers* require users to connect to a system through a proxy server that identifies the user.

(2) *Software agents* are agent-like applications that log information from several applications on the client computer and send them to a server. Enhanced proxy servers and software agents have been used mainly for desktop applications or other systems that do not run in a Web browser. In browser-based systems, logins, cookies, and session IDs are more popular user identification methods.

(3) *Logins* are the most reliable method for user identification. Users register to a system once and sign in each time they visit the system. One drawback of the login method is that it requires intervention of the user. Users often do not want to register or sign into to a system before using it.

(4) *Cookies* are not as reliable as the login method, but don't require users to actively intervene. A user ID is created when a user visits a system for the first time. The ID is stored to a cookie in the user‘s browser. When the user returns, the system can identify the user by the cookie. Cookies are popular because they are unobtrusive. However, they do have shortcomings: when multiple users use the system from one computer, their models mingle. Also, when a single user uses a system from multiple computers, the user will be given several IDs. Finally, if users delete cookies, their models are lost.

(5) *Session IDs* can be used to identify users for a short time, usually during a single browser session. Here, users are assigned a session ID each time they visit the system. This ID is valid only during the current session. Thus, session IDs do not allow the building of a long-term user model.

Information collection Once a system has identified users, it can collect information about them. In the following, we describe various methods for information collection. We distinguish the explicit information collection method from the implicit as well as hybrids of both methods. Technically, information collection occurs either at the server or at the client. In web-based systems, the collection is mostly at server-side.

One strategy to collect information on users is to ask them about their preferences, interests, demographic information, etc. This is called *explicit modeling*. Explicit modeling makes use only of information that has been explicitly and voluntarily expressed by the user. Users are required to express interests or preferences and to deliver this information to a system, by means of registration forms, questionnaires, or other feedback mechanisms on the system‘s user interface. For example, users may rate items, like articles or products, or select topics or categories of interest from a predefined list. Explicit information collection has become more popular in Social Web services, which often involve a high level of user engagement. A popular method for explicit feedback collection on the Social Web are ratings, where users express their opinion on an item by selecting their preference on a predefined scale. In many Social Web applications, i.e., Qype[1], or Ciao[2], users contribute and share ratings voluntarily. However, explicit user information collection has shortcomings:

(1) Explicit collection depends on the user‘s willingness to provide information. Users are usually unwilling to provide information because explicit collection is obtrusive and distracts the user‘s workflow. Explicit information collection places an additional burden on the users [78], and they usually don‘t want to invest extra time when they use a service [171]. For example, some services require the user to express preferences or answer questions during the sign up process. Users tend to provide only small amounts of information here, even though they would probably save time later by getting better recommendations. This effect becomes even more of an issue with user information that updates regularly. Depending on the context, interest for specific items may change fast. For example, users may be interested in products when they plan to buy them, but this interest usually decreases after the purchase. Users do not normally update their models regularly because it is a tedious task. Systems that regularly ask users to update their profile information are perceived as obtrusive.

(2) Explicitly expressed opinions are often intentionally biased by users.

[1] http://www.qype.com (last access: 2011-05-17)
[2] http://www.ciao.com (last access: 2011-05-17)

Sometimes, users don‘t want to provide information to services on the Web. They tend to conceal information about their preferences due to privacy concerns. For example, users may not want to provide demographic information like their age, gender or address to a service. They may, thus, either refuse to provide information or provide wrong information.

(3) Explicitly expressed preferences are highly subjective and hard to compare. Users may obscure information unintentionally, even if they do not bias their preferences willingly [185, 98]. For example, users may unintentionally use different values to express the same opinion when rating items. One user might rate an item with four points on a scale of five to express that she likes it, and another user might rate the item with three or five stars to express the same preference.

Implicit information collection is an alternative to explicit profiling that overcomes these shortcomings [190]. Implicit modeling collects information unobtrusively, without explicit user interaction. This strategy has been proven to create equally good or better user models than explicit modeling [203]. Implicit collection logs the data that has been produced implicitly by users through their behavior. Depending on the application, a variety of implicit information may be relevant. For example, implicit collection may log webpages searched for [194] or bookmarked [124] and click-through rates [118, 171, 120]. Implicit modeling is generally more powerful because it produces more data and updates faster than explicit information collection. This allows for larger and more dynamic user models. Most real-world personalized systems rely on implicit information collection or a combination of explicit and implicit collection.

A problem of implicit collection is that data are more noisy than the data from explicit modeling. For example, clicking on a specific search result in a search engine does not automatically mean that the result was helpful for the user [118]. Implicit data collection usually involves data cleansing, preprocessing and modeling before the collected data can be further analyzed for personalization [69]. In this additional step data are categorized to fit a conceptual data model, which is then used in the further personalization process. Often, machine learning techniques are applied to categorization.

For example, it may be necessary to extract the relevant features with natural language processing techniques from free text collected from webpages. Even when collecting structured data, like RSS or Atom feeds from various sources, it may be necessary to extract the relevant elements and aggregate the data into a single data format.

User model construction

Once the data about the users has been collected, the personalized system can construct user models. In the model construction step, analysis techniques are applied to the collected information to create relevant knowledge – user models – for the adaptation step. This step results in the creation of user models that represent the properties of individual users relevant for the application [5]. The user model construction step usually involves machine learning to discover patterns in the data [219, 242].

Machine learning statistically analyzes large data sets to predict future trends and behaviors based on the past events represented by the input data [27]. „The goal of machine learning is to program computers to use example data or experience to solve a given problem“ [4]. Machine learning algorithms „scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations“ [3]. In the context of user model construction, all information collected in the personalization process is treated as examples that illustrate user behavior. Machine learning algorithms automatically recognize complex patterns and derive user models from the observed examples. Three machine learning techniques commonly used for automated user model construction include: *classification*, *clustering*, and *recommenders*.

(1) *Classification* algorithms automatically categorize items based on past events. They predict the pre-defined category an item belongs to based on the available examples: the training set. A classification algorithm learns to make decisions from the training set in a process referred to as training. The result of the training process is a model – a „function that can then be

[3] http://www.thearling.com/text/dmwhite/dmwhite.htm (last access 2011-04-22)

applied to new examples to produce outputs that emulate the decisions that were in the original examples. These emulated decisions are the end product of the classification system“ [162]. The goal of classification algorithms is to automatically classify items into *pre-defined* categories: „[c]lassification is a decision based on specific information (input) that gives a single selection (output) chosen from a short list of pre-determined potential responses. [...] The output of a classification system is the assignment of data to one of a small pre-determined set of categories. The usefulness of classification is usually determined by how meaningful the pre-chosen categories are“ [162]. For example, a personalized search engine may classify users as expert or non-expert to predict whether or not they will be interested in webpages from a certain Web domain.

(2) *Clustering* algorithms analyze a set of items in order to organize it into groups of similar items [20, 113]; these groups are referred to as clusters. „[C]lusters could be thought of as a set of items similar to each other in some ways but dissimilar from the items belonging to other clusters“ [162]. Clustering algorithms differ from classification algorithms in that they do not require pre-defined categories. Instead, clustering algorithms independently observe from the example data which attributes of the data may have important distinctive features, and automatically create categories from these observations. A crucial issue in clustering is automatically finding attributes to define the similarity of items. A key potential of clustering is the detection of unforeseen structures in data. For example, a personalized search engine may detect that *level-of-activity* is an important attribute to distinguish users. The system may detect three existing clusters of users with similar activity patterns. The system would then automatically filter out certain results according to the user‘s cluster affiliation.

(3) *Recommender* algorithms automatically identify items that are relevant for a specific user. They can recommend interesting items to users, hence *filtering* sets of items to choose the items that are relevant to a particular user. Filtering can be achieved with two strategies. First, *content-based filtering* algorithms rely on the attributes of items that are to be filtered [143, 164]. For example, a personalized search engine may recommend a

webpage to a user because the user liked another webpage with similar content. Second, *collaborative filtering* algorithms rely on the shared preferences of users to filter items [182]. For example, a personalized search engine could recommend a webpage to a user because another user with highly similar preferences liked the webpage.

User model construction by machine learning algorithms is usually computationally costly. Therefore, the data analysis is usually conducted offline in a live system, i.e., the user models are pre-computed to be utilized later, during runtime.

Adaptation

Once the user models have constructed, the personalized system adapts the information presented to the user according to the information that was taken from the user models. [131] distinguish three possible levels of adaptation. (1) *Content adaption* includes multiple methods to personalize the content delivered to the user according to the user‘s model. It can take several forms: a personalized webpage on a particular topic domain may provide optional explanations to users that do not have knowledge about the presented topic and omit these explanations for users with extensive domain knowledge. A personalized search engine may suggest topics that a user may be interested in based on the current search context. It may also suggest breaking news about a topic that a user regularly searches. (2) *Structure adaption* describes methods to adapt „the link structure of hypermedia documents or its presentation to users“ [131]. While content adaption changes the information which is delivered to the user, structure adaption changes how information is structured or interlinked. In the context of exploratory search, structure adaption is usually used to deliver personalized navigation structures for exploratory browsing [24]. For example, a personalized search engine may rank search results according to their relevance for the user. It may also annotate results – a rating-like annotation indicates relevant links on the result page. (3) *Presentation adaption* describes methods to adapt the format and layout of the content delivered. For example, a personalized system may adapt the

layout of a webpage according to connection speed, hardware device, or the cognitive abilities of the user. Additionally, the modality of content may be adapted. For example, audio may be delivered as text if the user‘s device does not support audio playback (cf. [131]).

We have described the process that underlies personalization systems: collection of user-specific information, user model construction thereof, and adaptation of the content and structure of the delivered information based on the user models. In the following sections, we address the question regarding what information the user models are to contain, and how this information can be represented to enable adaptive Web systems.

2.2.2 User model content

The key goal of user modeling in the context of exploratory search is to be able to predict what information a user needs in the current search context. The user model of an exploratory search engine should model all information that could possibly be relevant for the adaptation. In the following sections, we describe *what* information is stored in the user models. The content that is usually represented in user models (cf. [43]) is categorized as follows: knowledge, interest, goals and tasks, background, individual traits, context of work, and application-specific content.

Knowledge

Knowledge is the most common content type in personalized systems. Much research on user modeling originates from the context of adaptive learning systems [43, 41, 169]. For adaptive learning systems, existing user knowledge of a domain is required to produce a personalized presentation of the learning content. Knowledge is also highly relevant for exploratory search engines. For example, a search engine could filter results for already known webpages, or it could bias results towards expert content when the user already has extensive knowledge of a topic domain. Knowledge is a user feature that usually changes over time: users can gain new knowledge and forget old knowledge – knowledge of a domain can increase or decrease. Therefore,

	Tom	*Eric*	*Steve*
1. Science	0.8	0.4	0.6
2. Music	0.5	0.1	0.3
3. Politics	0.7	0.9	1.0
4. Sport	0.0	0.2	0.5

Figure 2.4: Scalar model

user models that model knowledge must be highly dynamic and regularly updated. [43] distinguish two approaches of how knowledge can be modeled: *scalar models* and *structural models*.

(1) *Scalar models* provide a simple approach to model user knowledge. They assess knowledge of a user for *pre-defined* knowledge domains on a linear scale, see Figure 2.4. For each knowledge domain, the user's knowledge state is assessed as either quantitative or qualitative. Quantitative scalar models represent the level of knowledge numerically, e.g., in a range between 0 and 1; qualitative scalar models use predefined values, e.g., poor, average, good. Because of their simplicity, scalar models are easy to implement into personalized systems. A number of early systems have used scalar knowledge models and shown that these models can provide reasonable adaptations despite their simplicity [31, 30, 18, 71]. However, a key shortcoming of scalar models is that they provide only a rather coarse granularity of knowledge domains. They are thus less suitable for fine-grained adaptations.

(2) *Structural models* are an enhancement of scalar models. They provide a more fine-grained approach and allow for more precise knowledge modeling. In the real world, user knowledge of a domain can vary in the different parts of a domain. For example, a user that has knowledge about *social media* may have advanced knowledge of the *mobile web*, yet know little about *geofencing* despite the fact that both topics are subfields of *social media*. Structural models take this into consideration. They divide the user's domain knowledge into independent fragments. These domain fragments are much more precise than the overall domains. The most popular type of a structural knowledge model is the *overlay model*, which was initially developed in the context of

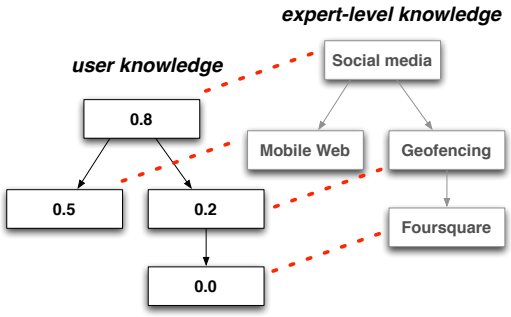

Figure 2.5: Overlay model

adaptive tutoring systems [4]. Overlay models represent user knowledge as a weighted subset of expert-level knowledge, see Figure 2.5. The expert-level domain knowledge is assumed to represent the overall domain knowledge. User knowledge is modeled as independent fragments of this domain knowledge. A quantitative or qualitative value exists for each domain fragment, representing whether or not, or to what degree the user has knowledge of this fragment.

Interest

User interest is a highly relevant content type in the context of exploratory search. Access to information on the Web is largely interest-driven [43]. As opposed to learning systems which are driven mainly by learning goals, exploratory search engines, online stores, news coverage, music and video recommendation services, etc., are driven primarily by user interest. Therefore, modeling of user interest has gained attention with the development of these services [26, 165, 11, 10, 107, 226].

Usually, interest is represented as a weighted set of keywords for each user. The keywords represent interest domains and weight values describe the level of interest in the domain. This keyword-based interest modeling approach has been criticized for oversimplifying. Concept-level models that model interests as concepts allow for a more accurate interest modeling [82]. These models represent interest as concepts which can be further divided into

[4] http://dspace.mit.edu/handle/1721.1/5772 (last access 2011-04-22)

subsets. Additionally, concept-based interest models can overlay the interest model of a specific user with a domain model. This approach is similar to the aforementioned overlay model for knowledge modeling described above. The approach requires a domain model that models all possible interests. Then, the (weighted) interest of the individual users is projected onto the domain model. Although overlay interest models can be more powerful user models, their use is still limited in real-world systems. One reason for this is their need of a complete domain model, which is problematic to create in open-world systems where a clear-cut domain can hardly be identified, cf. Chapter 3. [43] state that two groups of systems have been established. The first group, closed-corpus systems, usually uses overlay interest models. The second group, open-corpus systems, like most web-based systems, still uses keyword-based interest models. As we will show in the remainder of this thesis, these groups could converge again due to the potential of automated user model extraction from social bookmarking systems.

Goals and tasks

The goals or tasks of users define the purpose for the interaction with a personalized system. In the context of exploratory search, user goals are also referred to as *information need* [32] or *search context* [146]. For example, a goal could be to look-up information to answer a specific question or to discover a novel webpage on a topic. To know what the user aims at achieving when interacting with a system can be highly relevant to create adaptations [177]. Personalized systems usually categorize the current goal of a user into a pre-defined set of possible goals to select an appropriate adaptation strategy. Hence, goals are usually modeled with a *goal catalog* – a predefined set of possible goals that are supported by a system (cf. [43]). Similar to the above-mentioned overlay models, the goal catalog approach assumes a global set of possible user goals. For each individual user, one goal is selected from the catalog to model the user goals. Alternatively, a system can calculate the probability that a goal is one of the user‘s goals for each goal in the catalog [151]. Typically, the goal catalog is a small set of independent goals. In more

elaborate systems, relationships between goals can be used to represent a goal hierarchy that is similar to the fragments of knowledge domain models described above. In these systems, higher-level goals can be automatically decomposed into subgoals.

The identification of goals is a challenge for goal modeling. Goals can change fast – each time the user visits the system, or even during a single session. Furthermore, the goals underlying exploratory search are ill-defined. There are two strategies of goal model recognition: manual and automatic goal recognition. In the manual approach, the user manually selects her current goal from the catalog manually. Explicit manual intervention is, however, cumbersome for users and hence problematic, for reasons previously explained. Algorithmic approaches where examined to identify user goals automatically [117, 109]; however, automated goal recognition remains difficult to achieve.

Background

Background is defined as a set of features related to users that are not directly related to the core domain of a specific system, but that can still be useful for the adaptation process [43]. Background information can be demographic information, skills, language abilities, job responsibilities, etc. For example, a search engine may model the user‘s professional role. From this, it can infer implicit information from the user‘s level of knowledge in particular domains. Another relevant background feature is the user's language abilities – a search engine can adapt the content according to the languages that a user is able to understand. Background is usually modeled and handled rather simplisticly. As opposed to knowledge or interest, background does not normally change very fast. It is hardly possible to implicitly infer background. Therefore, background information is typically provided explicitly by users.

Individual traits

Individual traits are a set of features that describe the user as an individual. They contain features such as like cognitive styles and learning styles that

can be assessed through psychological tests. The cognitive style of a user defines her approach to organizing and representing information [209]. The learning style defines how users prefer to learn [48]. Although it is obvious that individual traits can play an important role for personalization, they have not been used in real-world applications yet.

Context of work

Context of work summarizes the user‘s context – time, location, physical environment, social context, user platform, etc. It is relevant, predominantly, from the perspective of mobile or ubiquitous computing [62, 67]. For example, context can be used to adapt to the user hardware – screen resolution, bandwidth, and input devices. Also, the time and location of the user are important context features to deliver appropriate content.

Application-specific content

The content categories discussed provide a generic view on the user‘s features. In real-world applications, developers often go a more pragmatic way. Often, it is not necessary to abstract user models so that it can be shared or reused by other applications. Therefore, user models in real-world personalized systems are often highly specific for a given use case [21]. In practice, user models are usually tailored to the content that the system provides. For example, a personalized book store may model application specific content like purchased books, a user‘s wish list etc. Other systems may model only content specific to music [3], movies [88], or news [57].

2.2.3 User model representation

User model representation defines *how*, and in which complexity, users are modeled. The user model representation is often specific to an application context. It depends on the techniques applied for user modeling extraction [5], on the data available, or the required granularity of the models. In the following sections, we discuss three kinds of user model representations that have been used successfully in real-world applications: keyword models,

semantic network models, and ontology models. Keyword user models model a simple list of items relevant for a user [5]; semantic network-based models and ontology models are more complex.

Keyword models

Keyword-based models are the most simple and the most common representation of users [77]. They consist of a set of keywords that represent the user‘s preferences. In some systems, keywords are grouped into predefined categories, so that a profile consists of several keyword sets, one for each category. In most applications, keywords are weighted. Weights associated with keywords represent the degree of interest, so that a user model consist of a set of keyword-weight pairs. A numerical value between 0 and 1 is usually used for the keyword weights. The preference for a keyword is higher when the weight is higher.

In the most simple case, keywords are single words which can be extracted from content collected from the Web [227]. Important words or word sequences can be extracted from the text of each webpage a user visits and stored into the user‘s keyword model. From new webpages, important words can be extracted the same way and matched against the keywords in the user model. Social bookmarking systems provide an even more simple way to retrieve keywords for modeling, since tags can already be regarded as keywords [119]. In both examples, keywords are regarded to be identical to *concepts*. This can, however, be oversimplifying. There is a distinction between keywords and concepts. For example, a user interested in *internet marketing* is probably not interested in both *internet* and *marketing*, but in the concept *internet marketing*. In this case, a tuple of words is necessary to model the concept *internet marketing*. It may be necessary or more accurate to model concepts with broader terms, represented by a tuple of words. This problem must be considered not only when creating keyword profiles from text. For example, when extracting models from tags in a social tagging system, tags are usually regarded as keywords. Tags are, however, not necessarily identical with concepts. Single tags can represent concepts already, but they

don‘t have to. A set of tags can have a different meaning than the individual tags. Users may use the two tags *internet* and *marketing* to tag a resource on *internet marketing*. The disparity between keywords and concepts is a major weakness of keyword based models. The problem is addressed by more complex representations of user models, like semantic network or ontology models.

Semantic network models

Semantic network models seek to overcome the disparity of keywords and preferences inherent in keyword models by expressing broader *concepts* rather than keywords. Semantic networks consist of a weighted network of nodes, where nodes represent keywords and edges represent connections between terms (cf. Chapter 3). Both nodes, and edges can be weighted. The notion of edges and weights depends on the application.

[80] use semantic networks as a solution to create concepts, i.e., broader preferences. Their user models consist of a set of nodes that represent concepts. Additionally, their models can contain keywords that are connected with concepts. Keywords are connected with concepts by weighted edges. The edges represent the affiliation of a keyword to a concept for the specific user. This representation allows to designate keywords to concepts to disambiguate different meanings of the keywords. For example, two models may contain the keyword *java*. In one model the term is connected to a concept *programming*; in another it is connected with *indonesia*. Hence, this representation of the user model supports disambiguation of the terms by the connection to concept nodes. Additionally, concept nodes can be connected to each other by weighted edges when they are semantically associated.

Ontology models

Ontology models are a more complex representation for user models. They have been introduced independently from the Semantic Web [122, 53] but gained new attention with its development. Two general approaches for ontology models exist. The first models concepts in a shared vocabulary; the

second uses ontology languages to structure the user model internally.

(1) The first approach seeks to extend the semantic network approach with more expressive descriptions of concepts [77, 236, 7, 208, 173]. Ontology models consist of a set of weighted concepts. The concepts are related to each other, so that they form a network. Edges in the network describe semantic relations between the concepts, e.g., hierarchy. Typically, ontology models map keywords against existing reference ontologies (cf. [77]). A number of systems have shown how user models can be mapped onto Semantic Web ontologies [77, 97]. The idea of this approach is to attribute terms in the model to concepts in an ontology. This approach of ontology models thus addresses the above-discussed problems of keyword and semantic network models; they model semantic, and thus hierarchical, relations between concepts. Therefore, similar to semantic network profiles, synonym terms can be disambiguated more easily. Additionally, hierarchical structures apparent in ontologies can be leveraged by the system, e.g., to make generalizations (cf. [77]): when a user model contains a preference for the concept *social media*, a system may infer that the user is also interested in *foursquare*, if *foursquare* is modeled as a sub-concept of *social media*.

A key restriction of this approach of ontology models is that it often requires existing ontologies which the user models are mapped to. Ontologies may not exists for all domains or topics that are covered by a user model. For many concepts, no appropriate ontological mapping may be available. Additionally, it is rather restrictive to map terms onto existing ontologies, because it allows only to map to predefined categories. On the Web, new concepts emerge constantly which are possibly not covered by the mapping ontology used.

(2) The second approach for ontology models seeks to share and interlink user models across applications by using ontologies [235, 100, 25, 7, 101]. The quality of the adaptation in personalized systems depends on the size and quality of user models. Thus, aggregation and integration of user models is desirable. Ontology models have the potential to create re-usable and sharable user models. Two requirements have to be fulfilled to achieve this. First, the models must use a shared representation of concepts, like it is done

in the first approach. Second, the models must use a shared vocabulary to describe the structure of the model – users, items, preferences, weight, etc.

Until now, a shared ontology that is used successfully across existing, real-world, personalized systems on the Web does not exist. In the following section, we discuss the reasons for this, and further challenges that have to be overcome for distributed user modeling on the Web.

2.3 User modeling on the Web

Early adaptive Web systems were non-distributed, or closed, self contained, systems [43, 129]. In such systems, it is well-known which software components exist, and which role they have. All steps of the personalization process are conducted by a closed and usually centralized system. User modeling is typically conducted on a centralized user modeling server.

More recent systems have a distributed software architecture [66]. They allow to share user models between loosely coupled system components that are distributed across the Web and made user modeling more powerful. In these web-based systems, a variety of heterogenous user models are distributed over various independent user modeling components. Distribution of user modeling across the Web yielded new challenges to the software architecture and user model representation. *User model integration* is a prerequisite of distributed adaptive Web systems. The concept refers to the syntactic and semantic integration of user models generated by various independent software components involved in a distributed user modeling system. Distributed personalized systems share user models across components. Several components of such systems can create, modify, and apply user models. User model integration ensures that all components involved in the system can interpret the shared user models.

With the current establishment of personalized systems on the Social Web, distributed user modeling enters a next stage [207, 128]. On the Social Web, user modeling systems scale up to a so far unequalled size. User modeling has the potentials to grow to a web-scale – hundreds of Social Web services with millions of users can be integrated into distributed user

modeling systems. *Web-scale* user modeling yields new challenges for user model integration not present in early distributed user modeling systems. The Social Web is an open world where everybody can contribute anything. In this world, user model standardization and integration are complicated. User model integration attempts have to consider the existing architecture and technical standards of the Web. They also have to take into account its openness regarding data publishing, which implies the questions of trust, and (in)completeness, contradiction, and reconciliation of user models. Moreover, the sheer quantity of data published on the Social Web requires systems that are extensible and highly scaleable.

In the remainder of this section, we discuss the key challenges of web-scale user modeling (Section 2.3.1) and important solutions that address the challenges (Section 2.3.2). We conclude the section with a discussion of the existing solutions (Section 2.3.3).

2.3.1 Challenges

We highlight three challenges of user model integration on a web-scale: heterogenous and incomplete user models, technical unreliability of system components distributed across the Web, and privacy considerations (cf. [66]).

User model heterogeneity

An important challenge for user model integration is the heterogeneity of the user models. The heterogeneity problem exists on two levels, on the content and on the format level. The two levels refer to two aspects of user model integration: *what* is shared, and *how* is it shared.

Heterogeneity of user model *content* refers to *what* is shared. The user model content heterogeneity results from the diverse internal functionality of systems that create user models. Most personalized systems tailor user models to their specific goals and requirements [21]. The requirements of a system are often defined by the application domain that is served by the personalized system. Systems from other domains often cannot re-use these models, because they require different content.

Heterogeneity of the user model *representation* refers to *how* user models are shared. The problem here is, that no established, commonly agreed-on standard representation for user models exists. Most existing systems internally represent user models in proprietary or unspecified data schemas and formats [213]. Thus, it is likely that different systems on the Web store the same information in different representations. The lack of a shared data format partly results from the fact that it must be defined *what* should be modeled before it can be defined *how* it should be modeled. Components of a distributed user modeling system have to agree on the content that should actually be modeled before they can agree on a shared format.

The representation heterogeneity problem comprises two dimensions: *syntactic* and *semantic* heterogeneity. Syntactic heterogeneity refers to the data format in which user models are stored or published, e.g., the markup language. Semantic heterogeneity refers to the data model schema, i.e., how the user models are organized and should be interpreted.

Content and representation integration entails another challenge: inconsistency of information [174]. Due to the openness of the Web, it is likely that systems face conflicting and inconsistent user model data when integrating user models from various sources. The resolution of conflicting information from resources is thus an inherent problem in the integration of heterogeneous resources.

Technical unreliability

Web-scale personalized systems consist of a number of independent systems of different providers connected over the Web. Many of these systems themselves rely on further systems. This raises practical challenges concerning the performance of user model integration. User model integration requires data transfer through potentially slow, unreliable, and error-prone software components [21]. Integration of user models faces the same technical challenges as all other distributed systems on the Web: systems may be unavailable due to connectivity problems, workload, or other reasons [65, 201]. The architecture of personalized systems on the Web must thus be fault-tolerant for all

components involved; e.g., fall-back strategies must exist if a component does not react (in time) [201]. Scalability and extensibility are another relevant aspect of practical implementation of user model integration [37]: open-world systems can grow very large in an uncontrollable manner; moreover, additional, unpredicted requirements can emerge after a system has been setup. User model integration has to comply with these technical requirements of web-scale distributed systems.

Privacy

Privacy has always been a critical issue for user modeling and is a key challenge for user model integration on the Web. User models inherently contain private and sensitive user-specific information. Privacy becomes even more relevant in systems that are distributed on the Web [205, 130, 52, 179]. In self-contained systems, the user usually gives sensitive information to a single provider of a personalization system. In web-scale systems, user information is potentially shared with third parties. Third parties can aggregate information from further data providers and re-publish newly created user-specific information. Nobody wants their private information to be shared in-transparently and uncontrollably with other, unknown, parties. Users must be given control about their user model information. Many reputable systems therefore affirm in their privacy policies that user information is not shared with third parties – an exclusion criterion for data sharing. However, privacy protection does not go along with data enclosure. Users can be given control over their data to enable sharing of user model information with other systems. However, this requires transparency and controllability for user modeling. Creation of trust in systems trough transparency and user control is the key to face information enclosure by users [52]. In trustworthy systems, users are more like to actively consent to data sharing [179]. Users must be allowed to actively decide whether and which data they share with which parties. Technical solutions for privacy-preserving user model integration are thus essential for web-scale personalized systems.

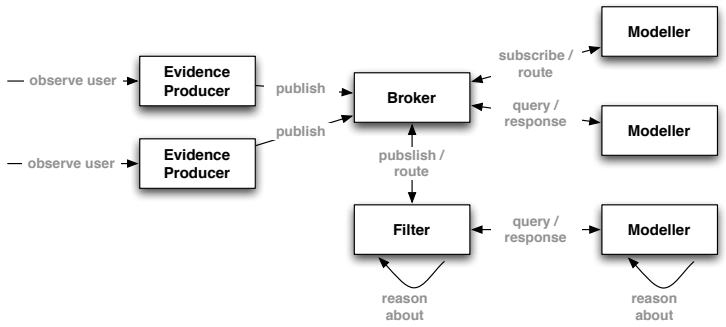

Figure 2.6: MUMS architecture (based on [37])

2.3.2 Approaches

In the following section, we discuss important approaches that address the challenges of user model integration on the Web from different perspectives.

MUMS

MUMS (Massive User Modeling System) is a framework to collect and disseminate user models on the Web [37]. It provides a generic architecture which addresses the architectural challenges of interoperability, scalability and extensibility of web-scale user modeling. MUMS has been developed for user modeling in web-based learning systems and can be generalized to user modeling on the Web.

MUMS contributes an architecture which logically decouples system components on the Web. The architecture defines four components involved in user modeling: *Evidence Producers*, *Brokers*, *Modellers*, and *Filters*, see Figure 2.6.

(1) *Evidence Producers* generate statements about a user. These statements do not represent user models, but *user opinions* – brief contextualized statements that describe user interaction that took place, or statements about user knowledge, goals, interest, etc. – in a certain context. Evidence Producers are typically applications or services that create user opinion through usage data logging. For example, a social bookmarking service could create an interest statement for a user and a topic when the user bookmarks a webpage on that topic.

(2) *Modellers* create a user model from the user opinions, usually by applying data analysis on the user opinions. Modellers may syndicate opinion statements from one or more Evidence Producers. Furthermore, they may be restricted to specific users or to specific purposes for which they create user models. For example, a Modeller may be specialized on modeling user interest in webpages for a recommender service. For this purpose, it may only consider certain types of user opinions.

(3) *Brokers* act as intermediaries between Evidence Producers and Modellers. They route user opinions between Evidence Producers and Modellers, or between various Modellers. The communication between Brokers and other components is either based on a query/response model, e.g., HTTP, or by a publish/subscribe model, e.g., XMPP.

(4) *Filters* serve for three purposes. First, they can act as Modellers. In this function, they apply data analysis on user opinions to generate higher-level knowledge. However, instead of generating complete user models, they enhance user opinions by newly created user opinions. For example, a Filter could receive user opinions of a specific user, which the Broker received from several Evidence Producers. The Filter applies data analysis on the aggregated user opinions and publishes derivate higher-level opinions to the Broker. Second, Filters can inject domain-specific rules into the user modeling process. For example, a Filter could filter user opinions according to specific domain knowledge. Modellers connected to this Filter only receive user opinions relevant or correct for the specific domain. Third, Filters can be chained together to add new value or filter out irrelevant value for a specific application.

The MUMS architecture addresses three problems of distributed user modeling: (1) interoperability, (2) extensibility and (3) scalability.

(1) MUMS ensures syntactical interoperability by using SOAP [5] for communication between decoupled components. The user opinions are expressed in RDF to ensure semantical interoperability. The components involved must agree on a set of ontologies that are used by the system. MUMS provides a database with agreed-on ontologies as a decision guide for developers of new

[5] http://www.w3.org/TR/soap12-part1/ (last access 2011-04-22)

components.

(2) Extensibility can be achieved through decoupling of components. In a decoupled system, individual components can be flexibly inserted into the system or removed. In MUMS semantic extensibility is ensured by the use of RDF. New ontologies can easily be integrated when required by future components.

(3) Scalability is addressed by MUMS mainly by leaving data storage and query methods unspecified. For example, the framework does not specify the transport mechanisms to be used. Thus, components may use any appropriate mechanisms, e.g., compressed formats. Furthermore, individual architectural components do not correlate with physical components. For example, a Modellers may physically be a cluster of computers.

MUMS contributes a viable generic architecture design for distributed user modeling on the Web. However, it leaves important issues open that are beyond architecture decisions. For example, the semantic heterogeneity of user models requires a shared ontology between components, which is problematic on a web-scale. MUMS does not provide further user model specifications that complete the generic architectural considerations.

APML

APML [6] is an attempt to overcome the issue of heterogeneity of user models on content and format level. It is an XML specification that allows users and services to create and share portable user model files that describe a user‘s *attention* (interest) for topics on the Web.

Listing 2.1: APML example

```
...
"implicitData": {
"sources":  {  "http://feeds.delicious.com/v2/rss/": {
     "name": "Delicious.com",
    "value": 0.7,
    "type":  "application/rss+xml"
    "authors": {
```

[6] http://apml.areyoupayingattention.com/ (last access 2011-04-22)

```
        "Sample":{
        "value": 0.5,
        "from": "GatheringTool.com",
        "updated": "2010-10-10T12:30:00Z"
...
```

The idea of APML is to create a standardized structured format that enables users of news feeds, blogs, etc. to exchange information about which topics they pay attention for. The format allows to describe attention for a topic, and to specify a degree of attention. Listing 2.1 shows an example APML snippet.

APML contributes two gains for web-scale user modeling. First, it explicitly addresses privacy issues by following a centralized approach: user models are stored and managed transparently and are fully user-controllable on client-side. Second, APML is a light-weight format that bases on simple and widely adopted technologies. Therefore, it could be easily adopted and implemented by existing services on the Web.

To ensure privacy, APML follows a centralized approach for user modeling: Web services create APML profiles that users can download and store locally. Users can then manually share their APML file with other services on the Web that support APML-based personalization. The centralized approach is one major shortcoming of APML. It requires users to actively create, manage, and share their own profile files – a tedious work which often does not provide obvious benefits for the users in the short term. A further shortcoming of APML is its limitation to attention. As discussed above, personalized systems often require more information like knowledge or background to make powerful adaptions. APML could not establish in the real-world. However, it has been one of the very few hands-on attempts for a standardized exchange format for user models that focuses on requirements of the Web.

IMS LIP

IMS LIP [7] is a specification that provides a more powerful standardized user model representation format for distributed user modeling. IMS LIP has been developed for distributed learning systems. The specification defines characteristics of a learner: learning activity, progress, goals, awards, preferences, etc. The IMS LIP specification categorizes user attributes into *data objects*. For example, the *Identification* data object describes individual information such as biographic or demographic data relevant for learning; the *Goal* data object models the learning objectives of the learner; the *Activity* data object models learning-related activity, containing formal and informal activity – work experience, training, civic service, etc. The main contribution of IMS LIP is that it allows to split user models into smaller data objects. Components of a distributed system may not require all user model information. To reduce complexity, such system components may only implement support for selected data objects of the IMS LIP specification and leave out the unnecessary parts. This approach reduces the obstacles of syntactic and semantic integration on a web-scale.

[173] show how IMS LIP can be applied for distributed user modeling in a distributed knowledge management system. An RDF binding of IMS LIP has been applied on a user modeling server, OntobUM. The user modeling server stores and manages IMS LIP user models as RDF/RDFS. However, OntobUM remains one of the rare implementations of IMS LIP. The standard could not establish in real-world applications. A key drawback of IMS LIP is that it needs to be extended by external vocabulary that describes the actual content of the specified data objects. For example, while IMS LIP defines that a system may model interest of a learner in a topic, it does not specify how interest should be modeled, i.e., how the data object is structured internally. Furthermore, the standard is very complex, hence too complicated to implement for many light-weight applications [8].

[7] http://www.imsglobal.org/profiles/ (last access 2011-04-22)
[8] http://standards-catalogue.ukoln.ac.uk/index/IMS_LIP (last access 2011-04-22)

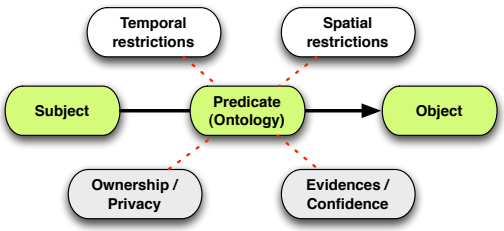

Figure 2.7: Situational statement (based on [99])

UserML and GUMO

UserML and GUMO are two complementary approaches for distributed user modeling from the context of ubiquitous computing research. The approaches contribute a more generic syntactic and semantic user model standard for web-scale user modeling.

UserML is a XML-based user model exchange language [99]. The approach for user model exchange in UserML bases on the notion of *situational statements*. Situational statements are RDF triples that are extended by additional information to enhance the expressivity statement. RDF triples are extended by temporal and spatial restrictions and metadata, e.g., ownership, confidence, or privacy information. The resulting situational statement is a 7-tuple, see Figure 2.7.

Listing 2.2: UserML Situational statement example

```
<SituationalStatement
    subject = ‘‘UbisWorld:Tom‘‘
    predicate = ,,UserOL:CognitiveLoad‘‘
    object = ,,High‘‘
    start = ,,20030517.140334‘‘
    duration = ,,600s‘‘
    owner = ,,UbisWorld:Tom‘‘
/>
```

The representation of extended triples is not possible straightforward inside the RDF data model. Therefore, [99] propose an XML syntax to model situational statements, i.e., RDF statements together with their restrictions and metadata. Listing 2.2 shows a situational statement that describes the

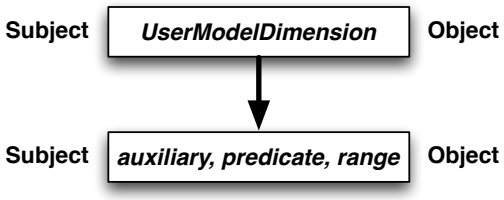

Figure 2.8: GUMO User Model Dimension (based on [101])

fact that the subject Tom has high cognitive load. The statement also contains metadata: start time, duration, and owner.

With the situational statements, UserML contributes a separation of the representation format of user models from the user modeling ontologies used, so that each system can use its own ontologies along with the shared representation syntax. However, the semantic extensibility of UserML is also its main shortcoming: UserML does not define any agreed on ontologies that should be used. For example, the situational statement in the example above has the predicate `CognitiveLoad` from an external ontology `UserOL` which is not defined by the UserML standard. Thus, while the approach provides a common syntactical model representation, it does not address the necessity of a shared vocabulary for semantic integration.

GUMO addresses the issue of semantic heterogeneity that was omitted by UserML [101]. GUMO is based on the situational statements that were introduced by the UserML approach. It extends this approach with a shared and modularized ontology which defines the semantics of the predicates used in situational statements.

Similar to UserML, GUMO extends RDF triples to situational statements. In GUMO, the predicate of each triple is replaced by a *user model dimension*. Each user model dimension consists of *auxiliary*, *predicate*, and *range*, see Figure 2.8

The semantics of user model dimensions are defined in the GUMO ontology. Thus, GUMO provides a shared syntax and semantics to describe users. A problem when designing an ontology to describe user model dimensions is, that a wide variety of user model dimensions must be modeled to cover all possibly relevant dimensions of a user. For example, users may

have interest in virtually anything. Theoretically, the GUMO ontology must model all possible user model dimensions for interest: *interest in social media*, *interest in music*, *interest in programming*, etc. To address this problem, GUMO follows a modularized approach, similar to the data objects in IMS LIP. Distinct ontologies for *user model dimensions* can be used or extended as needed by specific application contexts. For simple applications, a basic set of dimensions defined by a *UserModelDimensions* ontology may suffice. For example, GUMO provides a *BasicUserModelDimensions* ontology, which defines only basic user dimensions, like example emotional state, characteristics, demographics, abilities and proficiencies.

GUMO is an early attempt to apply Semantic Web technologies for user modeling. It has been used in academic systems for ubiquitous computing [35, 34, 135] but could not establish in real-world systems. One limitation of GUMO is the technical representation of situational statements. While technological infrastructure exists for RDF-based data models, extended statements – 7-tuples – require implementation of specialized solutions for storage, reasoning, and querying. Thus, the implementation of the extended statement approach is relatively complex. Furthermore, despite the modularized GUMO ontology, the problem remains, that a large number of user model dimensions exist, which makes the GUMO ontology large and complex.

Generic ontologies

With the growth of real-world Semantic Web services and the Linked Data Web [9], a number of generic RDF vocabularies became popular on the Web. These vocabularies are not devoted to user modeling explicitly, but still contribute to web-scale user modeling. Particularly three of the most often deployed RDF vocabularies on the Web are interesting in this context: SKOS, FOAF, and Dublin Core (cf. [10])

[9] http://linkeddata.org/ (last access 2011-04-22)
[10] http://www4.wiwiss.fu-berlin.de/lodcloud/state/ (last access 2011-04-22)

SKOS The Simple Knowledge Organization System [11] (SKOS) is a data model for representation of structures – relations between things – on the Semantic Web. It is formally defined as an OWL ontology. SKOS is a bridging technology between structures with limited semantics, e.g., folksonomies, and highly expressive OWL ontologies. Ontologies are capable to organize knowledge that is highly formal. On the Social Web, however, many systems contain less formal classification schemes, e.g., folksonomies, that are not sufficiently formal to model classifications with the standard ontology languages RDFS and OWL, cf. Chapter 3. SKOS provides a means to express both, these less discrete structures and fully-fledged ontologies [154].

Listing 2.3: SKOS relations example

```
ex:Geofencing a skos:Concept;
    skos:narrower ex:Foursquare;
    skos:related ex:Mobile_Web.
```

SKOS contributes to user modeling in that it provides a widely agreed-on semantic and syntactic standard to model and interrelate knowledge of users. SKOS allows to model concepts and semantic relations. Concepts are defined as „units of thought, ideas, meanings, or (categories of) objects and events – which underlie many knowledge organization systems“ [12]. For example, a concept may represent a topic of interest of a user. Semantic relations are links between two SKOS concepts. Generally, they model a related meaning of two concepts. SKOS distinguishes two categories of semantic relations: hierarchical and associative relations. Hierarchical relations model the assumption that one concept is more generic – broader – than another narrower concept. Associative relations model that two concepts are semantically related but do not state that one of the concepts is broader or narrower. Associative semantic relations are modeled by the `skos:related` property. Hierarchical relations are defined by the `skos:narrower` and `skos:broader` properties. Listing 2.3 shows hierarchical and associative relations modeled with SKOS.

[11] http://www.w3.org/TR/2009/REC-skos-reference-20090818/ (last access 2011-04-22)
[12] http://www.w3.org/TR/2009/NOTE-skos-primer-20090818/ (last access 2011-04-22)

FOAF FOAF [13] is a vocabulary that allows to model people and their social relations on the Web. The initial focus of FOAF has been on machine-readable descriptions of people on the Social Web. Besides characteristics of people, FOAF provides vocabulary to model projects, organizations and groups as other kinds of agents on the Web [89].

FOAF contributes to user modeling in two regards. First, FOAF provides vocabulary for community membership modeling – for describing people and their basic characteristics such as name, gender, age, etc. Hence, FOAF can be used to model background information and interest of users. The `foaf:interest` property can be used to model interest of a user in a topic. Second, FOAF can model relations between persons. The `foaf:knows` property can be used to model a social link between two people. The intended meaning of the `skos:knows` relation is roughly a personal acquaintance. Listing 2.4 shows basic user characteristics, user interest, and a social relation modeled in FOAF.

Listing 2.4: FOAF example

```
ex:Tom a foaf:Person;
    foaf:age 30;
    foaf:gender ,,male‘‘
    foaf:name ,,Tom‘‘
    foaf:interest ex:Social_media;
    foaf:knows ex:Eric.
```

Dublin Core The Dublin Core metadata element set defines *core metadata* for simple and generic resource descriptions [14]. It provides shared term definitions for metadata on the Web. The *Qualified Dublin Core*, an extension of the original Dublin Core, defines three additional elements – audience, provenance, and rights holder. Additionally, it refines the semantic specification of all elements, thus ensures semantic interoperability.

Listing 2.5: Dublin Core example

```
ex:AUserModel dct:created 2010-12-12;
```

[13] http://www.foaf-project.org/ (last access 2011-04-22)
[14] http://dublincore.org/ (last access 2011-04-22)

```
dct:provenance ex:AService.
```

Dublin Core can be used to enhance RDF user models with metadata descriptions. For example a user model can be specified with creation time and provenance information expressed with Dublin Core elements. Listing 2.5 shows metadata of a user model modeled with Dublin Core.

The described RDF vocabularies, SKOS, FOAF, and Dublin Core, are interesting for user modeling for three reasons. First, they are widely adopted on the Web, so that they ensure syntactic and semantic integration with existing data and systems. Second, they can be implemented into existing Web infrastructure with low effort. Third, they can be combined with each other and further vocabularies, hence they are easily extensible. However, all three vocabularies have serious limitations for user modeling, because they do not per se allow the representation of more complex relations required for user modeling, as we discuss in more detail in Chapter 5.

Cognitive Characteristics Ontology

Besides the described generic vocabularies, also light-weight RDF vocabularies dedicated for user modeling have been developed, which allow hands-on implementation of user models on the Web. The Cognitive Characteristics Ontology [15] (CCO) is an OWL ontology to model cognitive patterns of users on the Semantic Web. Besides the cognitive patterns, the ontology allows to model additional features of the patterns, e.g., context and temporal dynamics.

CCO defines a number of RDF properties that can be attributed to users to model their cognitive characteristics: `cco:interest`, `cco:belief`, `cco:expertise`, and `cco:skill`. All of these properties are sub-properties of `cco:cognitive_characteristic`. The property `cco:interest` aims at modeling preference of a user for an item. The properties `cco:belief`, `cco:skill`, and `cco:expertise` model a user‘s competences: `cco:belief` models beliefs, which can also be misconceptions; `cco:skill` models the user‘s ability to do something,

[15] http://smiy.sourceforge.net/cco/spec/cognitivecharacteristics.html (last access 2011-04-22)

e.g., play the piano; `cco:expertise` models the knowledge or expertise in a domain or topic, e.g., a programming language.

Listing 2.6: Property-centric representation of interest

```
ex:Tom cco:interest ex:Social_media .
```

CCO provides two alternative ways to model cognitive characteristics: *property-centric* and *class-centric* modeling. The first way uses the described RDF properties – `cco:cognitive_characteristic` and its sub-properties – to model cognitive characteristics by using single RDF statements. The strategy associates topics to users. Listing 2.6 shows a simple RDF statement that uses the property-centric vocabulary to model interest of a user Tom in the topic *social media*.

A limitation of this property-centric approach is, that it is not possible to model more detailed cognitive characteristics. For example, `cco:cognitive_characteristic` and its sub-properties do not allow to model weighted interest (see Chapter 5 for a more detailed discussion). To overcome this limitation, the ontology provides a RDF class to model detailed cognitive patterns: `cco:CognitiveCharacteristic`. Cognitive characteristics can be modeled as instances of this class instead of using properties. This *class-centric* approach allows to add further details of the pattern as properties to the class. For example, interest of a user for a topic can be enhanced by properties like `cco:appear_time`, or `cco:attention_duration` to describe the features of the interest property.

Listing 2.7: CCO interest characteristic

```
ex:ACharacteristic a cco:CognitiveCharacteristic ;
    cco:agent ex:Tom ;
    cco:topic ex:Social_media ;
    cco:characteristic cco:interest ;
    wo:weight [
        a wo:Weight ;
        wo:weight value 0.8 ;
        wo:scale ex:AScale
        ] .
```

CCO contributes to user modeling, because it provides a class-centric approach for RDF modeling and hence allows to model more complex user attributes than generic vocabularies (e.g., SKOS, FOAF, and Dublin Core). Class-centric modeling enables further properties from external ontologies to be easily be integrated into the vocabulary – it renders the ontology extensible. Listing 2.7 shows a class-centric cognitive pattern that is enhanced by a weight property from an external ontology. The example models the fact that Tom has interest in *social media* with a weight of 0.8 on a scale that is defined elsewhere. We discuss the issue of class-centric modeling with RDF in more detail in Chapter 5

Generic User model Component

The described light-weight RDF ontologies address semantic and syntactic integration of user models on the Web. They do, however, not specify how the should be applied by web-scale systems, i.e., they do not specify the system architecture. [213] present a user model server for semantic integration of user models on the Semantic Web. They contribute a real-world system that shows how several RDF ontologies can be combined and applied in a distributed system on the Semantic Web.

The key contribution of [213] is a server – GUC (Generic User model Component) – that stores user models from different user modeling services involved in a web-based personalization system. GUC applies Semantic Web technologies, RDF, OWL, and the OWL-based rule language SWRL [16], for user model integration. The server integrates semantically heterogeneous user models by semantic translation based on manually created mapping rules.

Several applications that use heterogeneous user model formats can subscribe to the GUC server to retrieve remote user models and store their own user models on the server. All subscribing applications must use RDF-based user models, i.e., the server requires a shared syntax for user models. The applications can use arbitrary RDF vocabularies, i.e., proprietary schemas that

[16] http://www.w3.org/Submission/SWRL/ (last access 2011-04-22)

define the semantics of the user model. The assumption underlying GUC is, that different user models modeled with different vocabularies usually contain overlapping information, i.e., they contain similar information represented differently. The GUC contains a user model schema translation component to semantically integrate these information. This translation component allows to exchange data between different heterogeneous user models on the server. To enable translation, all applications must provide OWL descriptions of their specific user model schema to the server. Based on these OWL schema descriptions of the user models, mappings between user models are created. Each class of a proprietary schema is mapped to a corresponding class of another schema, if possible. This translation must be configured manually by a mapping engineer once.

The translation process relies on SWRL. SWRL rules describe the mappings of OWL classes between different vocabularies. One of two strategies can be used for SWRL-based schema translation: (1) direct mapping, and (2) shared vocabulary mapping.

(1) To translate between the user model schemas of two applications, a direct mapping can be created. Here, elements of one schema are directly mapped to a fitting element of another schema with an SWRL rule. The problem with this approach is, however, that many mappings become necessary if more than two applications are involved.

(2) To reduce the complexity of mappings in scenarios with many applications, GUC offers another mapping strategy – a shared user model vocabulary. Here, a mediating user model vocabulary is created which models the most commonly used concepts of all applications. Then, all applications create mappings to this shared vocabulary. The advantage of this approach is that less mappings are required. The key limitation is, however, possible loss of information during the translation process. Some applications may model information that are not present in the mediating vocabulary. These information would get lost in the shared user model strategy.

While GUC shows how a distributed system can integrate heterogeneous user models represented with RDF ontologies, it has serious limitations on a web-scale. Scalability is a key limitation of GUC. GUC assumes a centralized

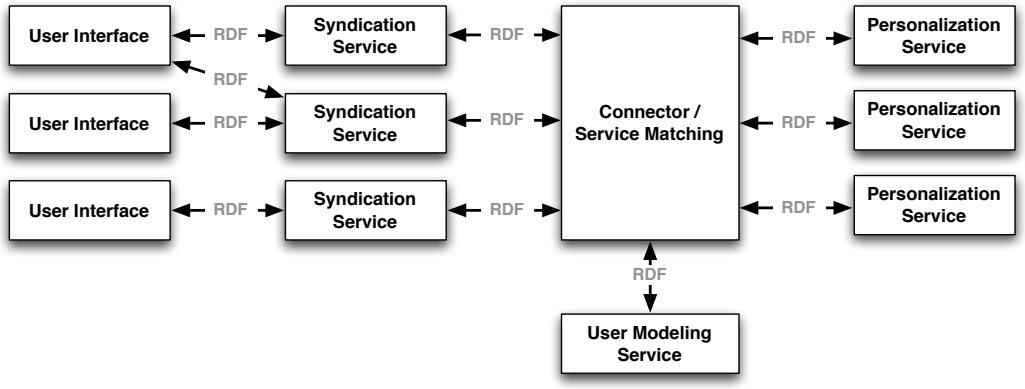

Figure 2.9: Generic personalization architecture (based on [38])

architecture – it requires a centralized server. For an open, web-scale system that potentially grows extensively this is not viable. Moreover, the translation strategies are not suitable on a web-scale. The first strategy requires to map each user modeling component to all other components, which is obviously only feasible for a very limited number of components. The second strategy assumes a shared minimal vocabulary for all involved components, which is highly problematic in an open-world system.

A personalization architecture for the Semantic Web

[38] present a generic architecture for personalized systems that contributes a highly scalable architecture and explicitly addresses user modeling on a web-scale, see Figure 2.9. Like GUC, the architecture fully relies on Semantic Web technologies. The architecture involves four loosely coupled components that communicate over Semantic Web standards.

Personalization services Personalization Services provide personalized content for a specific domain. Therefore, the service needs access to a user model. For example, a Personalization Service could filter information provided by some service according to a user model, transform the personalized output to RDF, and provide it to the user.

Syndication services Syndication Services create user models through analysis of data retrieved from other services and output them as RDF. For example, a Syndication Service may apply machine learning to usage log data to enhance a user model. The user model is then provided to the User Modeling Service in an RDF vocabulary.

User Modeling Service A User Modeling Service stores the user models and manages access to the user model. The User Modeling Service manages aggregation, alignment, and updating of user models from different services. Additionally the User Modeling Service controls access and therewith ensures privacy; e.g., it decides which services may or may not add, use, or remove which data in the user model [1].

Connector The Connector integrates Syndication Services, Personalization Services, and the User Modeling Services. Therefore, the connector decides which services are combinable. This matching can be done for example by describing the functionality – input and output – of all services in a common RDF language.

User interface User interfaces deliver personalized content to the user. The idea here is to detach content from presentation through Semantic Web technologies, so that generic user interfaces are suitable for different kinds of information encoded in RDF provided by various Syndication Services.

The described personalization architecture contributes an architecture design that allows for user modeling on a web-scale with Semantic Web technologies. It has, however, an important shortcoming: the architecture requires that all involved components fully rely on Semantic Web technologies. In the real-world, however, many Web services have only limited support for semantic Web technologies. The architecture requires radical technological changes from a large number of existing services. This imposes serious limitations in practice, because technological changes are costly and not possible to implement in all services.

2.3.3 Conclusion

We discussed important approaches for user modeling on the Web. We described MUMS – a generic architecture for distributed user modeling systems which addresses the architectural challenges of interoperability, scalability and extensibility of web-scale user modeling. Then, we described important attempts to develop shared user model languages: APML, a light-weight XML-based user model format; IMS LIP, a user model specification that splits user models into smaller modules; UserML and GUMO, two complementary formats from the context of ubiquitous computing that provide semantic integration but are problematic because they are complex and difficult to implement. Subsequently, we described more light-weight approaches: SKOS, FOAF, and Dublin Core, popular, generic RDF vocabularies which are limited to simplistic user models; and CCO, a hands-on ontology that allows for more complex modeling. Finally, we discussed two solutions that address an architectural level how user model integration can be implemented on the Semantic Web. GUC illustrates how semantic integration can be achieved in a heterogenous environment with a translation server; its centralized architecture does, however, not scale well. The *generic personalization architecture* by [38] shows how a Semantic Web architecture enables web-scale user modeling; however, the solution completely relies on Semantic Web technologies, which are not fully supported by many existing services on the Web today.

The state of the art of distributed user modeling suggests the following essential characteristics of viable user modeling on a web-scale:

(1) Semantic Web standards should be applied. Semantic Web technologies are a viable platform to achieve syntactic and semantic integration of user models. The ontology languages RDF, RDFS and OWL are important standards for knowledge representation on the Web (cf. Chapter 3). Widely deployed ontologies like FOAF, SKOS, and Dublin Core exist and can be applied for user modeling. Moreover, the openness of Semantic Web standards, established software, and wide acceptance on the Web suggest Semantic Web technologies for a web-scale user model integration effort.

(2) The most feasible architecture for web-scale user model integration is an intermediate solution that bridges the gap between the existing service-oriented Web architecture and a future fully-fledged Semantic Web. Integration of existing technology and services is important, because user modeling requires existing systems that create, share, and consume user models. A large number of candidate systems for user model extraction exist on the Social Web and extraction strategies are being developed (cf. Section 2.4). For a viable web-scale user modeling architecture, these Web services should be able to engage with minimal effort. Therefore, technological changes should be made in small steps and established standards should be applied whereever possible.

(3) A proper level of specificity of the user models is an important factor for adoption on the Web. The user model representation should be powerful enough but not overly complex. The language should orient towards successful existing languages and possibly re-use them. SKOS, FOAF, and Dublin Core have proven successful for a variety of applications on the Web. They are a good starting point for user model representation attempts.

(4) The architecture of the system should be decentralized to be scalable and flexible. The system should grow effortless when further components engage in the system. Furthermore, the architecture should be independent from implementation details, so that the components can be adapted when new demands emerge.

2.4 The Social Web

In this thesis, we address user modeling and exploratory search inside the ecosystem of the Web, and particularly the Social Web. The Social Web is valuable for user model extraction. It opens up new possibilities for user modeling that have not existed before. Users are becoming increasingly interconnected; they create and share information, resources, opinions over Web services; they communicate and interact over messaging, social networking, and collaboration services. The information they create and share yields new opportunities to harvest knowledge about users. Users create and main-

tain user models in many Social Web services; additionally, they interact with services, thus creating implicit information about their interests, preferences, knowledge, and opinions. Automated, manual, or semi-automated extraction methods can harvest knowledge from the structure of data generated by Social Web services. User models derived from information of Social Web services can greatly improve exploratory search. Our goal is to exploit the data inherent in social bookmarking services for user models to enable collaborative filtering and recommendations in exploratory search engines.

2.4.1 Characteristics

The *Social Web* [17] describes a combination of cultural aspects and technical developments that lead to a new form of how we use the Web.

From a cultural perspective, the Social Web can be seen as the habit of users to interact, communicate, socialize, and collaborate over the Web. The two important ideas of the Social Web are *user generated content* and *collaboration*. Social Web services focus on supporting these ideas – popular classes of Social Web services are *social networking services*, *social messaging services*, *social tagging services*, and *wikis*. The idea of user engagement and collaboration also flows into many other services on the Web. As a result, the conception of how we use the Web has changed with the emergence of Social Web services.

Technically, the Social Web is promoted by a set of technologies which enable a Web of services [36]. An increasingly *service-oriented* Web is about to establish, which enables or simplifies development of Social Web services. On this service-oriented Web, content is created and consumed not only by human users, but also by machines. An ecosystem of Web services retrieves and processes data and produces new content. Web services are implemented through Web APIs (Application Programming Interfaces) which provide programmatic access to data and application logic over the Web. *Software-as-a-service* on the Web allows to shift data storage as well as application logic to

[17] The Social Web has been also referred to as „Web2.0“, cf. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (last access 2011-04-22)

the Web. Most of today's Web services not only use their own data and logic, but also connect and integrate with further services. These *data mashups* aggregate data from multiple other services, process them for any purpose and possibly supply new data for further use. The focus of service-oriented Web applications is not anymore to provide and interlink relatively coarsely structured documents – webpages designed for human consumption – but services that provide data for machine consumption. The service-oriented Web technically relies on a set of protocols and standards, that allow to build a data and service-oriented web of services atop of the existing Web infrastructure. XML provides a common meta format for a variety of data formats relevant for the Social Web. Additionally, Web service APIs implement a set of design principles called REST [73]. REST specifies a set of architectural principles for designing server-client systems. RESTful Web services make use of the architectural features of the Web. RESTful APIs this make use of the existing HTTP architecture of the Web for Web services. They are more scalable and flexible than SOAP/WDSL [18] based Web services. The two predominant formats for data exchange on the Social Web are RSS [19] and Atom [20]. RSS and Atom are XML-based syndication formats that structure Web content in a common format and thus enable Web content to be exchanged on a finer-grained level than documents. They allow services to publish information by providing a common format and were initially developed for weblogs and news content. Content published in a syndication format is referred to as *feed*. Feeds are published under a URL by a service and can be subscribed to by multiple consumers. Feeds are designed to improve machine consumption of web content and are typically consumed by *feed aggregators*. Aggregators are applications that may run on client side or on another server and regularly poll feeds from a URL to process the retrieved data. The data provided by feeds are XML-based documents. They describe the content of the feed resource with a set of common entities like title, author, abstract, and eventually the payload of the resource.

[18] http://www.w3.org/TR/wsdl (last access 2011-04-22)
[19] http://www.rssboard.org/rss-specification/ (last access 2011-04-22)
[20] http://www.atomenabled.org/ (last access 2011-04-22)

The potentials of the Social Web for user modeling are twofold:

(1) User generated content and user engagement creates user-specific data which we can exploit for user models [153]. The information provided by Social Web services differ between different classes of services. Which data can be collected and how the extraction of user models can be achieved depends on the kind of the information the service provides, but generally more and more data become available promoted by user engagement. Users create content and communicate through Web services „leaving a virtual trail of information strewn across the web revealing their tastes, interests, and activities“ [199]. More data are available for user modeling through more active participation of users: „the active participation of a user [...] can provide a lot of information that can be used by an adaptive system to define more precise user models“ [44]. Much of the generated data is available publicly on the Web, for example social bookmarks.

(2) The service architecture of the Social Web enables new methods for data gathering [153, 36]. APIs and syndication formats allow to retrieve and process data from the Web with relatively low effort. The technological ecosystem of the Social Web make automated gathering and processing of information easier compared to the document-oriented Web. Instead of crawling and mining HTML documents, the Social Web allows to gather and aggregate information through feeds and APIs, which requires less computational effort. Usage data can be collected directly from the services, no data collection on the client side is necessary. Additionally, Social Web services often require user accounts. Thus, users are easier to identify over longer time periods, which improves user modeling because it allows to maintain long-term user models. Links between several accounts of a single user even allow to combine data from various services. Combination of user model data across Web services can greatly improve the user modeling process.

2.4.2 Social bookmarking

In this thesis, we extract user models from social bookmarking services. Social bookmarking services are a class of Social Web services which apply social

tagging to allow users to store, organize and share bookmarks on the Web. In this section, we introduce key concepts related to social bookmarking.

Tagging is a method applied by many Web services that allows users to annotate resources, like webpages, photos, or scientific articles [76, 147]. Tagging means that users can annotate a resource by assigning one or more text labels to it [192]. By tagging a resource, the user describes and categorizes the resource. Tagging enables the user to later retrieve the resource in an easy way through its tags. Tagging is an open way to assign keywords to resources; the number of tags on a resource is usually not limited. Users can assign as much tags to a resource as they want. Furthermore, tags are not restricted to a fixed classification schema or vocabulary – „[as] opposed to a classical taxonomy-based categorization system, they are usually non-hierarchical, and the vocabulary is open, so it tends to grow indefinitely“ [241]. Tagging systems do not prescribe to the user which tags to use to annotate a certain resource. As a result, the same resource can be tagged with different tags by different users.

Social tagging extends the idea of tagging to a community of users [86, 85]. Social tagging allows users to share their tags and tagged resources with others: „a tagging system becomes social when its tag annotations are publicly visible, and so profitable for anyone. The fact of a tagging system being social implies that a user could take advantage of tags defined by others to retrieve a resource“ [241]. By sharing their tags with others, users of social tagging systems create a *shared*, *user-generated* categorization of resources. Social tagging is not restricted to resources of any special kind. Social tagging systems have been used in various application contexts. The social tagging service Delicious [21] is amongst the most-cited applications of social tagging. This service uses social tagging to annotate bookmarks. Another popular example is the photo community Flickr [22], which applies social tagging to organize photos. An academic reference management service, CiteULike [23], applies social tagging for annotating links to academic papers. These are

[21] http://www.delicious.com (last access 2011-04-22)
[22] http://www.flickr.com (last access 2011-04-22)
[23] http://www.citeulike.org (last access 2011-04-22)

just some popular examples, social tagging can be used to annotate all kinds of content on the Web – for example emails [24], presentations [25], videos [26], or books [27].

Social bookmarking services are one popular application of social tagging [96, 144]. They allow users to save links to webpages and annotate them with tags. These annotated links are referred to as *bookmarks*. The goal of social bookmarking services is to enable the user to save, organize and retrieve webpages. Social bookmarking works similar to bookmarking in the user‘s local browser. However, it extends the functionality of local bookmarking with two features: users can annotate bookmarks with *tags* and they can share bookmarks with others. Annotation of bookmarks with tags allows individual users to organize bookmark collections with relatively low effort in the social bookmarking services. In addition to their own tags and bookmarks, users can see and browse the tags and bookmarks added by other users. In the remainder of this thesis, we refer to data that we collected from two popular social bookmarking services: Delicious and Connotea [28]. Both services are relatively similar in their functionality and differ mainly in the application domain focus. Delicious is intended for social bookmarking of any webpages. In practice, the users of Delicious are largely technically savvy. As a result, tagged resources are often webpages related to technical topics. Connotea is intended as a social reference manager for scientific resources and also allows to bookmark any kind of webpages. The bookmarks in Connotea are to large parts scientific papers, blog articles or other webpages related to scientific research.

The term *folksonomy* describes the result of a social tagging process. Social tagging enables a large number of users in a system to organize resources. Tagging facilities promote user generated categorization because they impose low effort to the users. Tag-based organizational structures, folksonomies, emerge from the communal tagging behavior of the users in a system. „A

[24] http://www.googlemail.com (last access 2011-04-22)
[25] http://www.slideshare.net (last access 2011-04-22)
[26] http://www.youtube.com (last access 2011-04-22)
[27] http://www.librarything.com (last access 2011-04-22)
[28] http://www.connotea.org (last access 2011-04-22)

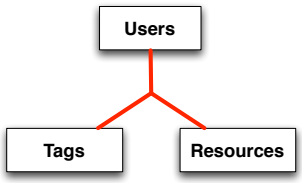

Figure 2.10: Folksonomy

folksonomy is also known as a community-based taxonomy, where the classification scheme is plain, there are no predefined tags, and therefore users can freely choose new words as tags“ [241]. The term folksonomy combines the words *folk* and *taxonomy* into one word to describe the „user-created bottom-up categorical structure development with an emergent thesaurus“ [217]. An important aspect of folksonomies is their self-organizational nature. Folksonomies „arise from data about how people associate terms with content that they generate, share, or consume“ [92].

Folksonomies are a set of users, tags, and resources which are related to each other, see Figure 2.10. To describe folksonomies formally, we adopt the usual notation (cf. [111, 136]): U, T, R are finite sets, whose elements are called users, tags, and resources. A folksonomy is a quadruple $F := (U, T, R, Y)$. Y is a ternary relation between these sets: $Y \subseteq U \times T \times R$. The elements of Y are referred to as tag assignments. This structure can equivalently be described as a tripartite undirected hypergraph $G = (V, E)$, where $V = U \cup T \cup R$ denotes the set of nodes and $E = \{\{u, t, r\} | (u, t, r) \in Y\}$ denotes the set of hyperedges.

2.4.3 Potentials for exploratory search

In the following section, we discuss the potentials of social bookmarking for exploratory search. Social bookmarking systems are not only used to tag webpages, but to organize, share, search, discover webpages. Social bookmarking can have several functions for knowledge organization and retrieval [86, 85, 192]. Attempts have been made to leverage social bookmarking for information retrieval. Social bookmarking has proven to be relevant for search in two ways. (1) Folksonomies allow new methods to rank search

results. New search ranking algorithms become possible that leverage the structure of folksonomies for result ranking. (2) Social bookmarking systems empower exploratory search with methods that go beyond the query-response paradigm and apply filtering, browsing, and recommendations.

(1) Social bookmarking systems provide data on the Web that can be leveraged for new search ranking algorithms. Ranking algorithms of traditional Web search engines leverage three types of data: webpage content, link structure, and query or click-through usage logs [105]. Folksonomies open up a new data source for ranking algorithms. A number of algorithms have been developed that leverage structural features of folksonomies for result ranking [16, 231, 134, 239, 105, 172, 112, 230]. Some of the algorithms rely completely on the folksonomies to compute the ranking [105]. Others use folksonomies as a complementary datasource to improve traditional Web search [172, 231]. [231] have shown that incorporating data from social bookmarking services into standard link-based ranking algorithms significantly increases precision of these algorithms.

(2) Social bookmarking systems enable new methods for information retrieval that extend the conventional query-response pattern and direct towards exploratory search. Services that merge social bookmarking and search into a single application have the potential to provide new interaction patterns, i.e., interactive browsing and filtering, in order to improve the user experience in exploring information created in social bookmarking systems [56, 155, 23]. It has been shown that search engines which incorporate social bookmarking bring up interesting new search methods and have the potential to improve exploratory search [155, 121, 229].

Exploratory browsing is one of the ties that connect bookmarking and exploratory search. It is one of the most frequently used features of social bookmarking services. In social bookmarking systems, exploratory browsing is also referred to as community browsing [155, 86, 85]. The idea of community browsing in social bookmarking is that users explore bookmarks that have been created by similar users. Bookmarks of similar users thus act as social navigation cues to explore tags and webpages. Users explore webpages starting from their *social context* – bookmarks of similar users. The commu-

nity browsing facilities of social bookmarking have been utilized for search engines to create navigational cues for exploratory browsing [75, 155, 121]. Navigational cues – „signposts from the masses“ [121] – allow for exploration of the information space through browsing. In exploratory search engines, navigational cues for browsing supports exploration of the information space by suggesting related resources to a search query. Navigational cues are thus recommendations which are specific to a user and a search query. Collaborative filtering-based recommenders have proven to give good recommendations when applied to folksonomies [229, 163, 116, 106, 29, 238]. Collaborative filtering techniques can be applied to suggest webpages, tags, or users. User models are a prerequisite to compute recommendations with collaborative filtering algorithms. Thus, by empowering collaborative filtering recommenders, user models „offer a good basis for exploratory search interfaces, even for users who are not using social bookmarking sites“ [121].

The key potential of social bookmarking for exploratory search is that it makes new data structures available for statistical analysis that were not present on the Web before. Folksonomies provide data for structural analysis. Analysis of the action of tagging of a user community is valuable for exploratory search. Social bookmarking services allow to extract knowledge through structural analysis of the folksonomies. A key strength of this content-agnostic approach for knowledge extraction is its computational efficiency. It does not require a qualitative analysis of some kind of content, e.g., full text crawling and mining, which is computationally expensive. Analogous to link-based ranking algorithms from traditional Web search, we can leverage the structural properties of folksonomies to extract knowledge for exploratory search.

Structural analysis of folksonomies has been successfully applied to infer knowledge about users – user models [119, 45, 12, 44, 125, 140, 229, 234]. We distinguish three types of user properties that can be extracted from folksonomies to support exploratory search (cf. [229]): user interest, user communities, and user-specific tag semantics, i.e., tag similarity and hierarchy.

Interest extraction

Tagging behavior is a highly reliable evidence to detect *user interest*. Traditional methods for interest detection often rely on a user viewing a resource. However, it is not easy to decide why a user views a resource: „view a resource can be out of curiosity, tagging usually not“ [44]. It is unlikely that users would tag resources in which they are not interested in. Thus, tagging is a more reliable and accurate data source to infer interest. As a result, the modeling process can be performed faster. In traditional user modeling methods, much of the usage logs must be filtered out to reduce noise – false positives. Social bookmarking data can be used without much filtering, because bookmarks are unlikely to have posted accidentally.

While the bookmark collection of users inform us about their interest, the tags assigned to the resources can also indicate user opinions – they indicate what the users think of resources [238, 215, 141]. A tag can be regarded as an expression of the user‘s personal opinion. The tagging process can be regarded as an implicit rating. „Thus, the tagging information implies user's important personal interest and preference information, which can be used to greatly improve personalized searching and recommendation making“ [141].

Community extraction

Tagging behavior can also be used to detect *user communities* [44, 229, 239]. User communities are groups of users with similar or overlapping interests or opinions. Social bookmarking allows to derive communities from shared tags or from shared sets of bookmarked webpages. Usually these user communities are not static but dynamic ad-hoc communities which emerge and dissolve while users bookmark and tag new resources. The detection of user communities is important for exploratory search, because similarity of users is an important indicator to assess how valuable the interest or opinion of one user is for another user. The value of an opinion can be assessed only if one knows who provided it [215]. For example, an exploratory search engine could recommend a webpage to a user because another user with highly similar interest bookmarked it. It would make less sense to recommend a

webpage because a user with contrary interests bookmarked it.

Tag semantics extraction

Tagging behavior leads to emergence of implicit semantics. The structure of folksonomies can be mined to extract some form of more explicit *tag semantics* – relations between tags. Tag relations describe *semantic relatedness* between tags. They define how semantically similar two tags are or of which type their relation is. For example, concepts described by two tags can have a hierarchical semantic relation – a concept is semantically *a kind of* or *part of* another concept. A number of algorithms have been developed to extract semantic structures from folksonomies [104, 111, 229, 183, 232, 186, 233, 49].

Extraction of more explicit tag semantics improves the system interpretation of the annotations. For example, it helps to identify topics in the contents or the annotations of documents [239]. Tag semantics also allow for tag meaning disambiguation – synonym detection [49]. Or, they allow to create concept hierarchies [49]. Ultimately, a richer tag semantics can greatly improve the quality of search engines, because it enhances recommendations with additional information. They allow to provide users with more detailed navigation cues to browse the information space. They enable an exploratory search engine to not only suggest tags related to the search, it can also inform users about the type of relation, *how* tags are related, „giving deeper understanding into the relationships between tags“ [106]. Knowledge about how tags are related can help users to identify topic domains, to disambiguate tags, or to deduce the meaning of a tag. Particularly, hierarchical tag relations are important for navigation through the information space.

2.5 Summary

In this introductory chapter, we provided the background for the goals and questions of this thesis and summarized the state of the art of the research.

We introduced the concept of *exploratory search*. We pointed out that exploratory search relies on personalization and user models. The key challenge

of exploratory search is to guide users through information – to suggest trails trough the information space – to facilitate *domain discovery* and *domain learning*. *Recommendations* are an important method to implement these navigation facilities. They rely on user-specific information – *user models*.

We described how user models are involved in the personalization process that underlies adaptive Web systems. User models are central to these systems. An important decision when designing a personalized system is *what* information to store in the user models and *how* to structure and represent these information. We described categories of *user model content* and strategies for *user model representation*.

A goal of this thesis is to foster user modeling on a *web-scale*. Therefore, we discussed the challenges of distributed user modeling on the Web. We described existing approaches of distributed user modeling, discussed their limitations and summarized required characteristics of a viable web-scale user modeling.

Finally, we introduced key concepts of the *Social Web* and *social bookmarking*. In this thesis, we leverage *folksonomies* for user model extraction. Therefore, we discussed the potentials of user model extraction from folksonomies for exploratory search: *user interest*, *user community*, and *tag semantics* extraction.

In the remainder of this thesis, we address the question how user modeling for exploratory search can be realized on the Social Web. We show how user models can be extracted from folksonomies and present a strategy to evaluate the quality of extracted models for exploratory search. Subsequently, we show how user modeling can be distributed across the Web to yield better exploratory search engines. Finally, we implement our findings in an exploratory search engine.

Chapter 3

Contribution 1: A user model extraction method

In the following chapter, we contribute a method for algorithmic user model extraction from social bookmarking systems. Our user model extraction method applies collaborative filtering algorithms to construct user models for exploratory search from folksonomies.

Folksonomies implicitly contain much information relevant for user modeling: user tagging behavior, webpages preferred or known by individual users, user similarities, popularity of webpages, etc. The challenge faced by this thesis is to understand and exploit the information inherent in folksonomies for user modeling with the purpose to utilize it for explorative search. Folksonomies don't contain user models per se. Although much knowledge about individual users is inherent in folksonomies, it requires some effort to create user models from folksonomies. The problem of folksonomies is that much information relevant for user modeling is present only *implicitly*. For example, folksonomies don't contain similarities between tags, users, or webpages explicitly, although all of this information is implicitly available and can be extracted from folksonomies. To apply and process the knowledge effectively, it has to be made *explicit*. The challenge is to transfer the implicit knowledge represented in folksonomies into a more formal representation. We extract user models with information relevant for exploratory search

from folksonomies algorithmically and thus transfer the implicit knowledge of folksonomies to a more formal representation.

A successful form of formal knowledge representation already exists on the Web: ontologies. Ontologies have gained popularity with the development of Semantic Web technologies. They enable more automated processing of knowledge than other forms of knowledge representation, because they represent information in a highly expressive form. Furthermore, they integrate well into Semantic Web technologies – Semantic Web standards and tools exist to represent, process and share ontologies on the Web. Representing user models as ontologies renders knowledge explicit, thus allowing to process it algorithmically. Additionally, it integrates user models into the technology stack of the Semantic Web, which allows for sharing the information over the Web, applying existing software to reason on it and integrating it with other information. Ontologies are thus a viable form for user model representation on the Web.

We aim at transferring folksonomies into ontologies. Our goal is to achieve the transformation algorithmically, which imposes a number of challenges, because fundamental differences exist between both representation forms. Ontologies and folksonomies are contrary in how they model knowledge, what knowledge they contain, and how they are created and maintained. Ontologies allow for extensive computational exploitation of modeled knowledge. However, their creation and maintenance is cumbersome, so that today only a small portion of information on the Web is represented as ontologies, which besides must be maintained with high effort to keep up with the fast-changing Web. Folksonomies are far less cumbersome to create – they just emerge through a collaborative process of annotation. Their drawback is that they are too implicit to be used for automated computation. Our goal is to mediate between folksonomies and ontologies – to bridge the gap between both forms of knowledge representation. Mediation between folksonomies and ontologies is not straightforward: „[r]ecently the two ideas have been put into opposition, as if they were right and left poles of a political spectrum. This is a false dichotomy; they are more like apples and oranges“ [92]. In this chapter, we address the challenges of mediating between both ideas and work

out a method for algorithmic extraction of ontologies from folksonomies.

With our user model extraction method, we contribute a strategy for knowledge modeling on the Social Web. We demonstrate how data gathered from folksonomies can be utilized to leverage the inherent user-specific information for user modeling. Our user modeling strategy extracts user models that contain three categories of information: *user interest*, *user similarity*, and *user knowledge*. User knowledge itself comprises three categories of user-specific knowledge which is significant not only for exploratory search: associative and hierarchical semantic relations between tags. With our knowledge extraction method, we contribute a strategy for algorithmic ontology modeling from folksonomies. Our method bridges the gap between dynamic user models, which reflect the dynamic nature of the Social Web, and ontology modeling which produces formalized knowledge.

Our user modeling method addresses key limitations of knowledge modeling methods on the Web. In particular, it has advantages over traditional ontology modeling methods in two aspects: First, it relies on implicit usage data, tagging activity, and does not require explicit, manual or semi-automated, knowledge modeling efforts. This makes our method more appropriate for knowledge modeling on a large scale. Second, our modeled knowledge evolves dynamically without human intervention. Thus, our method can keep up with the rapidly changing information on the Social Web. In our method, extraction is not a singular process; rather, ontologies are constantly adapted to evolving user characteristics: user interest evolves as users make new search queries; user-specific knowledge and similarity to other users constantly adapt to a user‘s search and tagging behavior. Usually, ontologies that model constantly changing information, as inherent in social tagging systems, must be maintained, which is a cumbersome process and usually involves manual engineering efforts. Our method maintains ontologies automatically, thus explicitly addressing an important issue for knowledge modeling on the Web: ontology evolution and maintenance in a dynamically evolving environment like the Social Web.

We apply collaborative filtering to extract user models. A key feature of collaborative filtering for our needs is that it is content-agnostic, hence

working equally good for all types of content. The ability of collaborative filtering to avoid content analysis has potentials for leveraging data from contexts where content plays a minor role or cannot be analyzed easily, e.g., social networking services, communication networks, usage log data, and social bookmarking services. Collaborative filtering relies solely on structural features of the underlying data. This also keeps the computational effort relatively low compared to content analysis [6]. In social bookmarking systems, content is produced constantly on a massive scale and is rapidly changing. Collaborative filtering has decisive performance advantages over content-based strategies in this context.

The remainder of this chapter is organized as follows. In Section 3.1, we discuss forms of knowledge representation on the Web and the differences between folksonomies and ontologies. In Section 3.2, we discuss ontology modeling strategies and their limitations for modeling folksonomies. In Section 3.3, we introduce an abstract model to mediate between folksonomies and ontologies. In Section 3.4, we give a brief introduction to collaborative filtering, which we apply in our user model extraction algorithms. In Section 3.5, we describe the output of our extraction method, the user model content. In Section 3.6, we describe our extraction algorithms. In Section 3.7, we conclude with a discussion of our extraction method.

3.1 Knowledge representation on the Web

Our goal is to explicate knowledge inherent in folksonomies for exploratory search. Therefore, we must transfer implicit user-specific information from folksonomies into an explicit form which allows to use it in an exploratory search engine. In the following section, we summarize popular forms of knowledge representation on the Web that vary in complexity, expressivity and formal explicitness. Figure 3.1 shows forms of knowledge representation arranged on a scale, ordered by complexity [153]. The scale reaches from glossaries, which carry little semantics, to semantically highly expressive ontologies.

(1) Glossaries are the least complex structures. They are controlled vo-

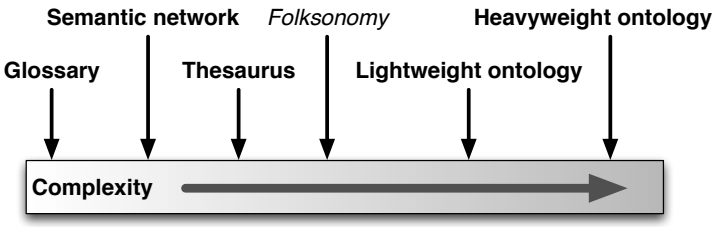

Figure 3.1: Knowledge representation structures (based on [153])

cabularies that define a shared meaning of a set of terms for a community. The terms don't relate to each other in any form. (2) Semantic networks extend the expressivity of glossaries. They describe graphs of terms that relate terms to each other [228] . Glossaries and semantic networks carry only very little semantic meaning. (3) Thesauri allow to model three kinds of relations between terms: hierarchy, association, and synonymy. Hierarchy modeled with thesauri is typically transitive, i.e., a term that belongs to a narrower category is automatically narrower than all terms that belong to broader categories. Thesauri carry more semantics than glossaries and semantic networks and allow to apply logical reasoning to transitive hierarchical structures to decide whether a term belongs to a category. Reasoners can classify terms by implicit knowledge. For example, they can find implicit sub-terms of a term by reasoning on a transitive classification. (4) Folksonomies don't contain hierarchical relations and differ fundamentally from the other structures, as we point out in Section 3.1.1. (5) Lightweight ontologies are what is most commonly referred to as ontologies in the Semantic Web context. They contain classes, instances, and properties, but still contain only minimal descriptions of them [153]. Concepts are typically modeled by classes. Individual instances of classes are the actual entities that are described. Lightweight ontologies extend beyond thesauri in that they allow to model arbitrary relations between concepts. For example, an instance of a class *user* and an instance of a class *bookmark* can be related by *creator of*. Classes as well as properties can be restricted to certain values or cardinality. For example, an entity of a class *bookmark* may have only one property *creator*. The entity referred to by *creator* may also be restricted to the class

user. (6) Heavyweight ontologies contain more detailed descriptions on how classes are composed (of other classes), or which properties are applicable for specific classes. Highly expressive ontologies that support first order logic allow for even more sophisticated logical reasoning. The boundaries between lightweight and heavyweight ontologies are fluent. In practice, it is often the tradeoff between application requirements and modeling costs that decides how expressive an ontology is modeled. Generally, the more expressive an ontology is, the more costly it is to model and maintain the ontology. On the other side, more expressive ontologies allow for better computational processing [93].

Ontologies are a popular explicit knowledge representation form that is suitable for user modeling. Our goal is to model the knowledge inherent in folksonomies using ontologies. In the following sections, we discuss the differences between ontologies and folksonomies and the challenges of mediating between them.

3.1.1 Ontologies versus folksonomies

Ontologies differ fundamentally from folksonomies. The diversity of both representation forms results from their different backgrounds – ontologies and folksonomies are phenomenons of two divergent worlds [92, 221]. As we discussed in Section 2.4, folksonomies are a phenomenon of the Social Web, which traditionally relies on non-semantic technologies. Ontologies are phenomenons of the Semantic Web.

We begin our discussion with a brief introduction to the Semantic Web to clarify the idea of ontologies. Ontologies are an enabling technology for the Semantic Web [92, 211, 61, 110]. The Semantic Web is a currently evolving extension of the current Web [22, 187, 9, 108]. Semantic technologies use, leverage, and extend the available Web infrastructure, which is can be seen as a technology stack of URIs used for resource identification, HTTP used as a transport protocol and XML acting as a common format for structural description of data exchanged on the Web. The key idea of the Semantic Web is to represent information in a computer-interpretable form. Resources

on the Web that are extended by computer-interpretable information on the semantics of the resources. On the envisioned Semantic Web, semantics – *meaning* – of information resources and services is defined. Computers can interpret information not only on a structural, but also on a semantic level. One objective of the Semantic Web is to share and reuse data across services. Instead of having proprietary data formats for specific services or applications, the vision of the Semantic Web is to create a global Web of data based on interchangeable and reusable data in a shared format: ontologies [63].

To achieve computer-interpretable semantics, semantic technologies add further layers to the Web technology stack. The Semantic Web is based on two pillars.

(1) Globally shared data formats make data sharable and reusable for all components of the Semantic Web. Key formats are RDF [1] and the ontology languages RDFS [2] and OWL [3]. RDF provides a common format for semantic description of data on the Web. On top of RDF, RDFS and OWL formalize knowledge represented by RDF resources as ontologies. Ontologies describe relations between RDF resources and model generic or domain specific knowledge that can be shared on the Web. They describe concepts and their relationships to model shared, formal conceptualizations of a domain [91], i.e., they are domain models that represent context knowledge using a computer-interpretable formal language.

(2) Human-like logic can applied to the ontologies to infer implicit knowledge from the data. Reasoners use formal logic to derive further knowledge from ontologies [210, 95, 191, 70]. SPARQL, a semantic query language enables querying of knowledge in ontologies [170]. Implementations of SPARQL engines may include reasoners that apply formal logic to the queries.

Semantic Web ontologies differ from folksonomies on many levels – representation, expressivity, goals, creation methods, and dynamics [92]. Subsequently, we discuss the key differences between ontologies and folksonomies.

[1] http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (last access 2011-04-22)
[2] http://www.w3.org/TR/rdf-schema/ (last access 2011-04-22)
[3] http://www.w3.org/TR/owl2-overview/ (last access 2011-04-22)

(1) A major difference between ontologies and folksonomies is their representation. As a phenomenon of the Semantic Web, ontologies are expressed in formal languages with well-defined semantics. Ontology languages are Semantic Web technologies: RDF, RDFS, and OWL provide syntactical formalization and enable logical reasoning which results in semantics. Folksonomies are a phenomenon of the Social Web. They traditionally rely on non-semantic technologies. They don't have a defined syntactical structure and may appear in arbitrary database schemas.

(2) Ontologies and folksonomies differ in expressivity. Ontologies classify concepts, i.e., they categorize concepts into classes and subclasses. The ontology definition does not define the level of expressivity or complexity of a domain model that is required to form an ontology; in practice, ontologies greatly differ in complexity [153]. However, all ontologies have in common that they categorize into classes and create relationships between instances and classes. Ontologies model formal semantic relations by relating concepts. A key strength of ontologies is that they can model hierarchy – *is kind of* or *is part of* relations between concepts. In folksonomies, instead, relationships are merely implicit. They don't formally express semantic meaning. Tags, users, and webpages are related *somehow* in folksonomies, but folksonomies don't formally model *how* they are related. Furthermore, folksonomies don't model any hierarchy – in the spirit of folksonomies „everything is miscellaneous“ [220]. Thus, folksonomies are less expressive compared to ontologies.

(3) Ontologies and folksonomies differ in their objectives. The main purpose of ontologies is to model a shared meaning within a community. This presumes agreement over concepts and relations of a domain among a community. A conceptual domain model must be shared by several individuals to form an ontology – „there is no such thing as a ‚personal ontology'“ [153]. In folksonomies, knowledge doesn't have to be agreed on. Instead, it emerges when several users share interest, preferences, or opinions. Opposed to ontologies, folksonomies do therefore not require a shared vocabulary and classifications. The creation of folksonomies is simple, effortless and collaborative, because no agreement is required across individuals. There is no pre-defined organizational structure to follow when modeling knowledge in a folksonomy.

As a result, the creation and maintenance of knowledge in folksonomies is achieved with considerably lower effort than that of ontologies.

(4) Ontologies and folksonomies differ in *what* they model. The range of the knowledge modeled in ontologies is usually clearly defined. Ontologies model either generic knowledge on a high-level, or specific knowledge for a limited and fixed topic domain. Folksonomies, instead, cover a broad variety of domains without clear-cut limitations. The crowd-sourced approach of folksonomies produces domain knowledge of much larger range than that of ontologies. Closely related to this difference is the *dynamic* of the modeled knowledge. Folksonomies are inherently open-ended and can be easily extended – they are in a constant flow. Folksonomies dynamically develop over time; they are much more dynamic than ontologies. In folksonomies, new tags are used as soon as they are needed – new vocabulary develops as soon as it is required. The notion of *users* and *time* is much more dominant in folksonomies. Folksonomies are created by a community of users. Users may enter or leave the community. Commitments of users change as new resources are tagged. The consensus of shared knowledge in folksonomies is in constant change. A new consensus may contradict, and thus invalidate, domain knowledge that users agreed on earlier [152]. In folksonomies, knowledge is „an emergent effect of the system as opposed to be a fixed, limited contract of the majority“ [152]. Ontology maintenance, instead, is a tedious process, which involves definition of new vocabulary, classifications and relations and consensus on the newly created knowledge [229].

(5) While ontologies focus on knowledge representation, the focus of folksonomies is more on the social knowledge acquisition process. While ontologies don't model the process of *knowledge creation*, folksonomies inherently address this complex process. [152] highlight the highly dynamic and collaborative nature of folksonomies. Their idea of folksonomies „is a community of self-organizing, autonomous, networked and localized agents co-operating in dynamic, open environments, each organizing knowledge (e.g. document instances) according to a self-established ontology, establishing connections and negotiating meaning only when it becomes necessary for co-operation“ [152]. They suggest the term *emergent semantics* to emphasize the dynam-

ics of semantic structures in folksonomies. In ontologies, instead, knowledge creation and maintenance is usually detached from ontology usage. This practice „cuts off valuable feedback and actually makes the social agreement over ontology elements brittle and vague“ [212]. Folksonomies blend knowledge creation, maintenance, and usage. Therefore, integrating folksonomies with ontologies has the potential to make ontology creation and agreement over semantics faster and easier.

3.1.2 Connect ontologies and folksonomies

Attempts to integrate the Social Web and the Semantic Web are gaining popularity. Recently the divergent worlds of the Semantic Web and the Social Web begin to merge [90, 28, 36, 111]. Semantic Web technologies become important for the Social Web, because the predominant technologies of the Social Web yield limitations for further developments. On the other side, the Social Web offers new potentials to promote the Semantic Web. The convergence of the Social Web and the Semantic Web opens up two key potentials:

(1) The Social Web enables new strategies for ontology creation and maintenance in which folksonomies play an important role [193, 229, 212]. We discussed that a key problem of folksonomies is their lack expressivity: unlike ontologies, folksonomies don't explicitly express hierarchy and semantic meaning of relations. But folksonomies offer huge potentials for ontology creation. Even if they are less expressive than traditional domain ontologies, much implicit knowledge is inherent in social tagging systems. Folksonomies thus have „the potential of becoming a technological infrastructure for harvesting social knowledge“ [229]. Integration of folksonomies into the ontology creation process has the potential to produce more dynamic ontologies and ontologies that describe also less popular domains – the *long tail* of domains that is not (yet) covered by existing ontologies.

(2) The Semantic Web can be an enabler of *data integration* on the Social Web [36]. Semantic Web technologies are an important step for data integration. For example, most user models that are currently generated by

Social Web services are bound to a single service due to proprietary data formats and schemas. Semantic technologies can integrate user models across services. Additionally, user models can be combined with other data. In this context, ontologies are an important technology to organize, link, and share user models. They provide a powerful conceptual and technical basis to explicate and formalize user models and benefit user model representation as well as their linkage and dissemination.

Our goal is to leverage the potentials that arise from convergence of the Social and the Semantic Web for user modeling. We create semantically interpretable, ontology-based, user models from folksonomies. Therefore, we formalize knowledge inherent in folksonomies.

3.2 Ontology modeling methods

In the following section, we discuss four approaches for ontology creation – ontology engineering, ontology learning, crowd-sourced ontology engineering, and semantics extraction – and discuss their limitations for creating ontology-based user models from folksonomies.

3.2.1 Ontology engineering

Ontology engineering describes manual creation of ontologies where ontology engineers model a certain knowledge domain in an ontology language [145, 196, 167]. Ontology engineers are typically domain experts; to model a domain, they need exhaustive knowledge on the domain. The process of manual ontology engineering involves three steps (cf. [161]):

In a first step, the ontology engineer decides the range of the domain that is to be covered and the purpose the ontology should serve. This decision defines the range of the ontology and how expressive the ontology will be. For example, depending on the purpose, certain classes of properties may be required, or property restrictions may or may not be necessary. The purpose and domain defines what concepts and properties are used for the ontology. For many domains, ontologies may already exist that can be

reused or extended. Once the classes and properties for the ontology are defined, class hierarchies have to be defined. Three approaches of hierarchy modeling exist: *top-down*, *bottom-up*, and a combination of both [211]. The top-down development starts with defining the most general concepts and incrementally defines more specialized concepts. The bottom-up approach starts with the most specific concepts and incrementally groups the concepts into more general concepts. The combined approach starts with defining the most important concepts and incrementally defines more generalized and specific concepts. The result of the class hierarchy development is a hierarchical taxonomy of classes, where an instance of a specific class is also an instance of the superclass of its class. In other words, the taxonomy describes *is-a* relations between classes. Class hierarchies are typically transitive, i.e., a class is not only a subclass of its direct superclass, but also a subclass of all superclasses of its superclass.

In a second step, the ontology engineer defines the properties of classes. For each class, properties are defined that may be used to further describe an item of this class. The properties are also applicable for all subclasses of this class. The domain of a property defines which classes can use the property. The range of a property defines which class a property can have as its value. Properties can be further specified by defining value-type, e.g., a property value may be specified only by a string value, and cardinality; in other words, a class may only have exactly one property of a certain type.

In a third step, instances are created for classes in the ontology. For each instance, the ontology engineer chooses an appropriate class and defines the values for the properties.

The benefits of manual ontology engineering are high quality of the modeled knowledge. The method is, however, not appropriate for explicating knowledge in folksonomies. Ontology engineering relies on extensive domain knowledge of the engineer. Knowledge in folksonomies, however, covers a wide range of topic domains. This is an exclusion criterion for manual ontology engineering. There is probably no engineer who has knowledge on all topic domains covered in a folksonomy. Furthermore, manual ontology engineering is a tedious process which requires much work and is hence ex-

pensive, even when engineering tools are applied to support the engineer. Folksonomies are large and dynamic hence go beyond what can be modeled manually. Usually, manual engineering is applied to fixed topic domains, where the focus is on high-quality and static domain models for a relatively small domain. It is not a feasible strategy to transfer folksonomy knowledge into ontologies.

3.2.2 Ontology learning

Ontology learning describes a partly automated ontology acquisition process [145]. Ontology learning seeks to support the cumbersome ontology engineering process by automation. Typically, ontology learning involves natural language processing or machine learning to extract concepts and relations from data sets. However, ontology learning is still not a fully automated process but a semi-automated process that requires human intervention at several points. The non-automated steps of the ontology learning process are again carried out by an ontology engineer. [145] refer to this semi-automated approach as *balanced cooperative modeling*. Instead of fully automating the ontology creation process, ontology learning aims at facilitating the modeling process for a technically trained person. [145] distinguish five steps involved in the ontology learning process:

In step one, existing ontologies relevant for the given domain context are identified and prepared for reuse. Existing ontologies may be domain models, which roughly cover the given purpose. Also cross-domain, top-level ontologies may be useful as initial structures when creating domain-specific ontologies. Imported ontologies must be merged or aligned to form a single ontology as a basis for the next step.

Step two is ontology extraction. This step exploits existing domain-relevant data sets – free text, semi-structured or structured data sets – to automatically model large parts of the ontology rather coarsely grained. This step is usually automated and extracts coarse-grained domain models from data sets with knowledge extraction algorithms. Typically, this involves also pre-processing of the collected data. Which algorithms can be applied to the

pre-processed data depends on the kind of data. Semi-structured documents, like webpages, may be reduced to free text, which then can be analyzed with natural language processing algorithms to extract domain knowledge [159]. Structured data, like data sets from databases, may be mapped to a specific data model and processed with machine learning algorithms.

In step three, the created ontology is pruned to adjust it to its purpose. The goal of this step is to produce an ontology which is expressive for the application context but excludes unnecessary knowledge. Knowledge irrelevant for the application context that makes the ontology less manageable and computationally less efficient may be dropped.

In step four, the ontology is refined. This step completes the modeling of the extraction step at a more granular level. The ontology is adapted – fine-tuned – to better serve the application requirements. This step usually involves similar algorithms than the extraction step. However, in contrast to the extraction step, refinement can consider existing ontologies.

In step five, the application for which the ontology is created validates its quality. Since an ontology is usually created to serve a specific purpose, the application can measure how well the ontology fulfills the requirements. Optionally, the learning process can be iterated with the resulting ontology to improve or extend the ontology.

Ontology learning is a more feasible method to model ontologies on a large scale, hence it overcomes the most serious limitation of manual engineering. However, it still has drawbacks for knowledge extraction from folksonomies. Ontology learning aims at extracting knowledge from a data set by identifying instances and mapping them to existing classes. It relies on existing domain ontologies on which information from the input data can be mapped. This implies that ontologies exist which model the topic domain covered by the input data. Furthermore, it implies that it is known in advance which domains are covered in the input data. Both is not the case in folksonomies where topic domains may emerge which have not been modeled yet in an existing ontology and where it is not possible to know what topics will emerge in the future by user tagging behavior.

3.2.3 Crowd-sourced ontology engineering

The crowd-sourced ontology engineering approach extends the traditional ontology engineering approach in another direction. Instead of a single ontology engineer, many users collaboratively model domain knowledge. The crowd-sourced ontology engineering approach enables a large number of non-technical users to contribute ontological knowledge. The design of a crowd-sourced ontology engineering system must be inherently different from ontology engineering systems for a single ontology engineer [139, 15].

Crowd-sourced ontology engineering has been successfully applied for web-scale knowledge modeling, but it still faces three challenges that have to be overcome for knowledge modeling in the context of social bookmarking systems. First, users have no special technical knowledge on ontology engineering. Crowd-sourced ontology engineering systems must break down the modeling process to simple steps that can be carried out without any knowledge on ontologies. They must also provide easy-to-use user interfaces to enable users to contribute. Second, users are usually not obliged to fulfill a task. Crowd-sourced ontology engineering systems must provide a reason or incentive for users to contribute to the modeling process. The systems should also make ontological engineering effortless to keep the entrance barrier for contribution low. Third, users may have different and contradicting opinions on a domain. The system must consider the possibility of contradictory conceptions of different users – it must be able to deal with contradicting opinions.

In the context of the Social Web, two forms of crowd-sourced ontology engineering systems have been developed: *semantic wikis* and *Extreme Tagging systems*. Both seek to integrate crowd-sourced ontology engineering with the ideas of the Social Web.

Semantic wikis

Semantic wikis [4] are a form of crowd-sourced ontology engineering systems commonly used in the Semantic Web context [15, 13]. Semantic wikis allow for Web-based, distributed knowledge engineering. They provide an easy-to-use interface that enables non-technically trained users to collaboratively create and maintain ontologies or to semantically annotate resources on the Web. They extend the concept of traditional *wikis* to the Semantic Web, i.e., they seek to make semantic knowledge engineering and maintenance easy for average users and thus to leverage a large number of technically untrained domain experts as ontology engineers.

Semantic Wikis strongly simplify knowledge creation and representation for the user. For example, they provide graphical user interfaces like *what you see is what you get* ontology editors. They also address the issue of contradicting opinions. Like traditional wikis, semantic wikis provide tools for change-tracking, discussion, commenting, and rating to allow consensus building. This, however, is still problematic, since different parallel opinions may exist or even be required depending on the use case. Depending on the ontology purpose, different definitions of a topic may be required to be reflected in an ontology.

Extreme Tagging

Extreme Tagging is a crowd-sourced ontology engineering approach that specializes on social tagging systems [176, 200]. It aims at utilizing the popularity of social tagging systems for crowd-sourced ontology engineering. The approach combines social tagging and manual ontology engineering. Usually, social tagging systems allow to tag resources, e.g. webpages. Extreme Tagging additionally allows to tag tags themselves, relations between tags, and relations between tags and resources. Goal of the additional tagging is to enhance the semantic expressivity of the underlying folksonomy.

An example for the benefit of Extreme Tagging is disambiguation of var-

[4] http://www.gi.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/semantic-wiki-174.html (last access 2011-04-22)

ious meanings of a tag. Tags may have different meanings in different contexts. In Extreme Tagging systems, users can tag existing tags to clarify the tag meaning. For example, a tag *apple* that describes a resource may be tagged with *computer* to disambiguate its meaning as a company from another possible meaning, the fruit.

A further benefit of Extreme Tagging is that it can render associations between tags semantically expressive. Relations between tags in a folksonomy may carry several semantic meanings, e.g., *is kind of*, *is part of*, or *singular of*. To explicitly model the semantic association between tags, users can tag the relations. For example, a user could tag the relation of two tags *foursquare* and *social media* with *application of*. Tagging of relations between resources and tags works the same way. Tags can have various functions in a social tagging system. A tag may describe the content of a resource, e.g. *geofencing*; or it may be functional, e.g., it may express interest – *toread* or *remember*. A user can tag a resource-tag relation with *describes* to clarify that the tag is descriptive. Tagging of resource-tag relations thus makes the function of tags semantically expressive.

Allowing users to tag the tags and relations promotes collaborative enhancement of semantic expressiveness of folksonomies. Extreme tagging addresses two issues of collaborative ontology engineering: it ensures a low entrance barrier for user contribution, and it shows how to use existing social tagging services for ontology creation. However, the Extreme Tagging approach faces an unsolved problem: there are no constraints for *how* relations can be tagged. Unlike ontology engineering, where engineers stick to a pre-defined vocabulary, Extreme Tagging allows for free annotation – users are not encouraged to stick to common classifications or vocabulary. Extreme tagging thus does indeed add further structure to folksonomies; the problem is, however, that the added structure does not necessarily provide more semantic expressivity, because the tags that describe tags or relations are semantically as arbitrary as the original tags. Extreme tagging does therefore not yield enhanced *formal* semantics.

3.2.4 Ontology extraction

Ontology extraction is a method that *automatically* explicates semantics inherent in folksonomies. It exploits manually created content from social tagging systems, folksonomies, for knowledge modeling. Semantics extraction systems use folksonomies as an input for lightweight ontology creation. They require no explicit knowledge modeling of the users. Instead, social tagging behavior is regarded as an implicit knowledge modeling process. Semantics are extracted through machine learning algorithms. Ontology extraction algorithmically explicates semantic relations that are implicit in folksonomies through analysis of relations between tags, users, and tagged resources. Ontology extraction is a promising approach, particularly because it allows for high automation of knowledge modeling.

3.3 An abstract model for semantics extraction from folksonomies

Algorithmic ontology extraction can be used to extract knowledge relevant for exploratory search from folksonomies. [152] contribute a generic, abstract model to describe emergent semantics in folksonomies, the *Actor-Concept-Instance* model of ontologies. A problem of ontologies is that they don‘t cover all aspects relevant for knowledge representation in social bookmarking services. The Actor-Concept-Instance model extends the ontology model to better address the specific features of folksonomies. The model explicitly models the fact that all meaning depends on users that contribute in a community. It extends the ontology model with a social dimension to better describe crowd-sourced semantics emergence in folksonomies. The Actor-Concept-Instance model is a formal basis for the semantics extraction process from folksonomies.

The formal definition of folksonomies (see Chapter 2) describes folksonomies as a tripartite graph. The Actor-Concept-Instance model is inspired by this model. It also models the folksonomy as a tripartite graph and involves several graph conversions to apply analysis methods on the

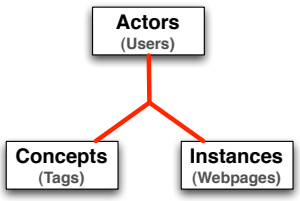

Figure 3.2: Actor-Concept-Instance graph

graph. Applying machine learning techniques, like classification, clustering, or recommenders, on folksonomies is a viable strategy for semantics extraction. The problem is, however, that these techniques cannot be applied to folksonomies per se, because they cannot be applied to tripartite graphs directly. The Actor-Concepts-Instance model formalizes folksonomies and converts them to formal structures that can be utilized by machine learning techniques. In the following, we describe the graph modeled by the Actor-Concept-Instance model and the conversion steps that are necessary to apply established analysis methods for semantics extraction.

3.3.1 A tripartite graph

The Actor-Concept-Instance model is an extension of the ontology model. The traditional ontology definition models concepts and instances. The Actor-Concept-Instance model extends this definition with actors. It models a folksonomy as a tripartite graph with hyper-edges. Nodes of this folksonomy graph are partitioned into three disjoint sets: actors (users), concepts (tags), and instances (tagged resources, e.g. webpages); formally $A = \{a_1, \ldots, a_k\}, C = \{c_1, \ldots, c_l\}, I = \{i_1, \ldots, i_m\}$. The graph is a hypergraph with ternary edges; each edge represents a concept associated with an instance by an actor, see Figure 3.2.

3.3.2 Affiliation graphs

Goal of the Actor-Concept-Instance model is to provide a formal model of folksonomies on which analysis methods can be applied for semantics extraction. Established analysis methods for semantics extraction already exist,

e.g., from the field of machine learning. A problem for these analysis methods is that folksonomies formally take the form of a tripartite graph with hyper-edges (see Chapter 2). Such graphs, however, are complicated to work with. To apply established analysis methods, we must convert the folksonomy into a simpler formal structure. The Actor-Concept-Instance model helps to formally convert the original tripartite folksonomy graph with hyperedges into three bipartite graphs with regular edges:

(1) An *actor-concept graph* models affiliations between users and tags. It links users to the tags they have used to describe at least one resource. Associations between actors and concepts can be weighted. The weight models how often a user has used a tag.

(2) A *concept-instance graph* models affiliations between tags and resources. This graph can also have weighted edges, which model how often a resource has been tagged by a certain tag.

(3) An *actor-instance graph* models affiliations between users and resources. Because users usually tag a resource only once, weighting makes no sense in this graph.

3.3.3 Similarity

These three affiliation graphs allow to extract similarities between nodes in the graphs, i.e., similarities between actors, concepts, or instances. Similarity can be derived from overlapping of pairs in the affiliation graphs. Similarity is weighted; the weight of an edge is higher when more pairs overlap. For example, the similarity of two users from an actor-concept graph is higher, when the two users have a larger number of shared tags (tags that both users have used to tag a resource).

Three similarity graphs can be created from the affiliation graphs: (1) a user similarity graph (overlapping agents), (2) a tag similarity graph (overlapping concepts), and (3) a resource similarity graph (overlapping instances).

3.3.4 Statistical Analysis

Similarity graphs are an appropriate input source for established statistical analysis methods that can be applied for semantics extraction. These methods are usually machine learning algorithms. For example, recommenders can utilize similarity graphs to recommend tags or webpages to users [163, 141, 115, 106, 29]. Another potential is hierarchy extraction [104, 183, 125, 127]. Folksonomies are flat structures, which is a shortcoming in many regards. Machine learning algorithms can extract hierarchy from similarity graphs to enhance the expressiveness of folksonomies.

3.3.5 Semantic interpretation

A problem with statistical analysis for semantics extraction is that the extraction methods can extract relations between tags, users, and webpages; however, they cannot automatically define how to semantically interpret the extracted relations [49]. For example, the semantic meaning of a tag hierarchy relation is different than user hierarchy. Tag hierarchy usually typically expresses taxonomic classifications – *is-a* relations. User similarities are explications of social networks implicit in folksonomies. User hierarchy typically expresses affiliation of a user to a certain community. Any semantics created through statistical analysis must be accompanied by interpretation of its results.

3.4 Collaborative Filtering

Our user modeling algorithms apply collaborative filtering for semantics extraction. Collaborative filtering was first introduced by [84]. It is a machine learning technique that allows to compute similarities between arbitrary things (*items*, in the terminology of collaborative filtering) or users. Item and user similarities are then used to predict *preferences* of particular users for items [102, 132]. Collaborative filtering techniques have been widely applied in *recommender systems* [198]. Recommender systems analyze user

models of users in a community to help individual users of that community identifying relevant information or filtering out irrelevant information [132].

Recommenders are thus an information filtering technique that allows to identify items which a specific user is likely to have interest in. To make predictions, recommender algorithms compare user models with some other characteristics. Recommenders aim at presenting only those information to a user that the user is interested in. Although preferences of people are highly individual, they do follow certain patterns. People tend to prefer things that are similar to other things they like [162]. Because Tom, the user from our introductory example, is interested in *social media* and *mobile web*, we can guess that he is also interested in *geofencing*, which is a closely related topic. Communities of people that express opinions on things provide valuable data to make recommendations. In a community, people tend to prefer things that similar people prefer. The idea of recommender systems to use these preference patterns in communities to predict preferences of individual people: „people who like *mobile web* also like *geofencing*“.

Recommenders predict users' preferences for items based on a set of *ratings* of other users. Ratings are opinions – likes and dislikes – that users have expressed about items. The goal is to predict items that a specific user has interest in from existing ratings of the user and that of the other users. How ratings are represented depends on the recommender; ratings may be votes on a numerical scale, 1-5 to express the level of preference, or just a binary value, preference or no preference. The prediction process atop of the preference ratings consist of two steps. First, the recommender estimates pairwise similarities between all items – it creates item-item similarities based on the ratings. Alternatively, the recommender can also estimate pairwise similarities between all users. Both strategies are possible and reflect two perspectives on the recommendation problem, as we will point out below. Second, the recommender computes predictions from these similarities. For each item, the recommender estimates the probability that a particular user likes it. It suggests a list of items that the user presumably has interest in. The predictions can also include items that were not yet rated by the user – *novel* items. A key strength of recommenders is to predict preference pat-

terns, and to use these patterns to discover items that are useful for specific users but which the users did not already know about [162].

Recommender systems can be categorized into two types: *content-based* and *collaborative filtering* recommenders. Both types differ in the methods they use to compute similarity between items or users.

Content-based recommendation techniques compute user preferences and make recommendations considering attributes of the items or users [143, 164]. For example, a content-based recommender could recommend a webpage to a user because the user has shown interest in other webpages from the same Web domain, e.g., *acm.org*. In this case, the recommender leverages attributes of the URL to predict a user interest. The prediction could also rely on metadata embedded in a webpage, or the actual content of the page. Content-based recommendations work well in many contexts, however, they have drawbacks for our needs. A key shortcoming of content-based recommendation techniques is that they require to process the content of resources. This imposes considerable computational effort, particularly when scaling to large and dynamic data sets on the Social Web. For example, to predict webpages to users in a folksonomy based on the page content, we would have to be retrieve and mine the text and multimedia content of all webpages included in the folksonomy.

Collaborative filtering computes user preferences and recommendations based on user and community data: „[c]ollaborative filtering systems use the item ratings by users to come up with recommendations, and are typically content agnostic“ [60]. They are a complementary approach to content-based filtering. Collaborative filtering recommendation techniques don't require to process the properties of items to make recommendations. They are solely based on knowledge of users' relationships to items. Hence, collaborative filtering is agnostic to what the items are or which attributes they have – the same collaborative filtering technique can be applied to recommend webpages, tags, or something completely different, as long as relationships between users and items exist. A key advantage over content-based techniques is that collaborative filtering can be easily adapted to other data – images, music, videos – where it is hard to analyze the underlying content.

Furthermore, by avoiding content analyses, which are often computationally costly, collaborative filtering usually imposes less computational effort.

We adopt the formalization of collaborative filtering provided by [181]: The goal of collaborative filtering is to recommend an item or to predict the preference of an item for a particular user. There is usually a list of m users $U = \{u_1, u_2, \ldots, u_m\}$ and a list of n items $L = \{i_1, i_2, \ldots i_n\}$. Each user has a list of items I_{u_i}, which the user has expressed their opinion about. Opinions can be gained from explicit ratings, or implicitly from analysis of usage data, like bookmarks, search query logs, or click-through data. There exists a particular user $u_a \in U$ – the *active* user – for whom the result of the algorithm is computed. According to [181], the result can be either a *recommendation* or a *prediction*.

A recommendation is a list $I_r \subset L$ of N items that the active user u_a will prefer the most. The list must not contain items which the user has already expressed their opinion about, i.e., $I_r \cap I_{u_a}$. This result is referred to as *top-N recommendation*.

A prediction is a numerical value which represents the predicted preference $P_{a,j}$ of the active user u_a for a particular item $i_j \notin I_{u_a}$. The prediction values are represented in the same scale than the expressed opinions of u_a.

We extend the formalization by another result type, which reflects a further possible task of collaborative filtering algorithms: collaborative filtering can be applied to suggest *similar items*. Similar items are a list $I_r \subset L$ of N items that have the highest similarity to an active item i_a for the active user u_a. We refer to this result as *top-N similar items recommendation*. Note, that the result of similar items is specific to the active user. The top-N similar items are influenced by existing ratings of the active user. For example, the similar items can be filtered so that they don't contain items which the user has already rated, i.e., $i_j \notin I_{u_a}$.

Preference matrix. The collaborative filtering process relies on preferences, which represent expressed opinions – ratings – of users about items. The preferences are a $m \times n$ user-item matrix M_p, where each entry $m_{i,j}$ represents the preference rating of $user_i$ on $item_j$. A rating can be empty

indicating that a user has not expressed an opinion about an item.

Similarity matrixes. From the preference matrix, the recommender algorithms computes a similarity matrix. Similarities are computed either between users or between items. The two approaches reflect two strategies of collaborative filtering: *user-based* and *item-based* recommenders. [162] point out that „[t]he label ‚user- based' is somewhat imprecise, as any recommender algorithm is based on user- and item-related data. The defining characteristic of a user-based recommender algorithm is that it is based upon some notion of similarities between users". User-based collaborative filtering algorithms compute a pairwise user similarity matrix $M_u(u_k, u_l)$. Accordingly, item-based algorithms compute a pairwise item similarity matrix $M_i(i_k, i_l)$. Both similarity matrixes are computed based on the ratings users expressed for items and certain similarity measures. The similarity matrixes are typically computed offline. During runtime, only recommendations, predictions, or similar items are computed from the matrixes.

Similarity measures. An important element of collaborative filtering algorithms is the computation of which users, or items respectively, are similar to others. *Similarity measures* are applied to compute pairwise similarities between users or items based on the preference matrix. Common similarity measures include the Euclidean distance [5], the Pearson correlation coefficient [6], the cosine similarity [7], and the Tanimoto coefficient (see Equation 3.1 on page 237). Which similarity measures produce the best results depends on the application context and the features of the preferences. For example, some similarity measures, e.g., cosine similarity, are inappropriate for binary preferences. Other measures don't consider the actual preference value for similarity computation, hence are suited only for binary preferences.

Collaborative filtering has proven successful in the context of social bookmarking systems to support users in annotating resources, browsing through

[5] http://en.wikipedia.org/wiki/Euclidean_distance (last access 2011-04-22)
[6] http://en.wikipedia.org/wiki/Pearson_correlation (last access 2011-04-22)
[7] http://en.wikipedia.org/wiki/Cosine_similarity (last access 2011-04-22)

resources, and detect new resources [238, 141, 116, 182, 163, 115]. We use collaborative filtering to extract user models and recommend resources to users based on the user models. Our user models aim at supporting the two key tasks of exploratory search – domain discovery and domain learning. Collaborative filtering recommendations support these tasks in three regards:

(1) Collaborative filtering can suggest novel items to users. A strength of collaborative filtering is to predict preferences for items that a user does not know yet. Suggesting novel items enables domain discovery. In an exploratory search engine that bases on folksonomies, collaborative filtering recommenders can suggest novel tags or webpages to users to support discovery of novel topic domains.

(2) Collaborative filtering is an appropriate technique for semantics extraction from folksonomies. We apply collaborative filtering to extract hierarchical associations between tags. Tag hierarchy is valuable for exploratory search because it supports domain learning: hierarchical relations between tags can guide users from general to specific information and vice versa.

(3) Collaborative filtering integrates well into personalization. Collaborative filtering is inherently user-specific. Exploratory search can greatly profit from personalized navigation facilities for domain discovery and domain learning. Collaborative filtering algorithms support adaptation of recommendations to personalize navigation cues for exploratory search engines.

3.5 Output

Our user model extraction method extracts three user properties from folksonomies: *user interest*, *user similarity*, and *user knowledge*.

(1) We extract the interest of individual users in a topic. We create a keyword-based interest model. We consider tags to be keywords that describe a concept. Interest is represented by tags that a user has used more than once to bookmark webpages, i.e., whenever a user uses a tag more than once, the tag is added to the user interest model. The extracted interest is binary – users have or don't have interest in a topic.

(2) We extract the similarity between individual users. We assume that

two users are more similar if they share more interests. Hence, our user similarity extraction method relies on common interests of users. Our user similarity extraction algorithm computes similarity values between all users. The value is higher, the more tags two users have been used in common. Hence, user similarity is weighted – users are more or less similar to others. Our user models store the top-N similar users with a similarity weight value.

(3) We extract knowledge of individual users. User knowledge describes semantic relations between tags that are valid for a specific user. The knowledge extraction process involves two steps. First, we extract the top-N similar tags for all tags in which an active user has interest. Second, we extract semantic relations between similar tags individually for each user. Semantic relations between tags can be either associative or hierarchical. Associative relations model *semantic similarity* between two tags. Hierarchical relations model *semantically broader* or *semantically narrower* relations between tags. Our tag hierarchy algorithm extracts the relations considering tag similarities and popularity of tags. Our user models store weighted similarity relations and model their semantic relation type.

3.6 Extraction algorithms

In the following section, we describe our collaborative filtering user model extraction algorithms. Our extraction method involves three steps. First, we extract interest properties for individual users with our user *interest prediction* algorithm. Second, we extract similar users for individual users with our *similar users prediction* algorithm. Third, we extract knowledge for individual users, which involves two algorithms: the *similar tag prediction* algorithm and the *tag hierarchy* prediction algorithm.

3.6.1 User interest prediction

Our user interest prediction algorithm requires a folksonomy as input source. We assume a preference of a user in a tag when a user tags a webpage with a tag. We create a binary preference matrix from the input data. This pref-

Table 3.1: User interest prediction output

User	Item
Tom	social media
Tom	mobile web
Tom	geofencing
Eric	web search

erence matrix is a list of users relating to tags. Each entry of the matrix models a preference of a user for a tag. We construct binary preferences, i.e., users either do or don't prefer a tag; we don't assign preference values. Note, that it would be possible to create weighted preferences from our input data, e.g., by considering how often a tag was used by the user. However, these additional data don't necessarily improve recommendation results. In our extraction method, we neglect some of our available input data, namely the information how often a user used a tag, because an informal evaluation indicated that numerical preference values don't produce better results but complicate preference data collection and similarity computation. It seems that much of the quantitative preference information in folksonomies results in noise rather than in valuable information for our extraction method. To further reduce noise in our models, we exclude tags from the preference matrix that have been used only once. The output of the user interest prediction algorithm is a binary preference matrix, $M_p(u, t)$, that models unweighted interest of users u in tags t, see Table 3.1.

3.6.2 Similar users prediction

Our similar users prediction algorithm requires the binary preference matrix output by the user interest prediction algorithm, $M_p(u, t)$. We create a pairwise user similarity matrix for all users in the preference matrix. This user similarity matrix models how similar any two users of M_p are. It stores a similarity value for each combination of users. Table 3.2 illustrates the structure of the similarity matrix.

The user similarity matrix models a similarity value between 0 and 1 for all users for which some similarity exists. Note, that the similarity between

Table 3.2: User similarity matrix

	Tom	Eric	Steve	Bill
Tom		0.87	0.71	0.61
Eric	0.87		0.11	0.28
Steve	0.71	0.11		0.04
Bill	0.51	0.38	0.04	

Table 3.3: Similar users prediction output

User	**User**	**Similarity**
Tom	Eric	0.87
Tom	Steve	0.71
Tom	Bill	0.51

users may be *null* if no similarity exists. We compute the similarity values for the user preference matrix with the Tanimoto coefficient. The Tanimoto extends the cosine similarity to calculate similarities for binary attributes. It is thus appropriate for our preference matrix, which models binary preferences. The Tanimoto coefficient is defined as shown in Equation 3.1. It returns a similarity coefficient between 0 and 1.

$$T(A,B) = \frac{A \times B}{\parallel A \parallel^2 + \parallel B \parallel^2 - A \times B} \tag{3.1}$$

To populate the user models with user similarities, we predict the top-N similar users for each user based on the user similarity matrix. The output of the similar user prediction algorithm is a matrix $M_u(u_a, u_b, sim)$ with all users u_b, their top-N similar users u_a, and a similarity value sim between 0 and 1, see Table 3.3.

3.6.3 User knowledge extraction

User knowledge extraction involves two algorithms. First, we predict tag similarities. For each user, we predict the top-N similar tags for all tags that the user is predicted to have interest in. Second, we predict semantic relations between tags. A tag hierarchy prediction algorithm extends the output of the tag similarity algorithm by semantic relations between tags. Based on

the output of the similar tags prediction algorithms and the popularity of tags, it computes semantic relations between similar tags. Similarity between tags models a semantic *association* by default. In some cases, our hierarchy algorithm can enhance the associative semantic relations with *hierarchy*, i.e., it predicts semantical *narrower* and *broader* relations between similar tags.

Similar tags prediction

Our similar tags prediction algorithm requires the binary preference matrix, $M_p(u,t)$, output by the user interest prediction algorithm. We create a pairwise tag similarity matrix for all tags of the preference matrix. This tag similarity matrix models how similar any two tags of M_p are. It stores a similarity value for each combination of tags. The result is a tag similarity matrix that models a similarity value between 0 and 1 for all tags for which some similarity exists. The similarity may be *null* if no similarity exists. Like in the previous algorithm, we compute the similarity values for the tag preference matrix with the Tanimoto coefficient.

To populate the user model with tag similarities, we recommend the top-N similar tags for each tag from the tag similarity matrix – we predict the top-N similar tags for all tags that an active user has interest in. We can re-score similarities between tags for individual users in this step. This individual re-scoring allows us to bias the result of the algorithm towards certain features. For example, we can decrease a tag similarity value for the active user, when one of the involved tags has already been used by the user in a bookmark or query. Thus, we can bias similar tag suggestions towards novel tags – re-scoring allows to personalize the recommendations. Or, we can increase similarity values for tags which are amongst the most popular tags to bias recommendations towards popular tags. The baseline version of our tag similarity extraction algorithm does not re-score the similarities. However, we present two alternative versions of the algorithm in Chapter 4 which re-score the similarities to bias recommendations according to tag popularity. The output of the similar tags prediction algorithm is a matrix, $M_s(t_a, t_b, sim)$, of all tags in which the active user has interest in t_a, their top-N similar tags

Table 3.4: Similar tags prediction output

Tag	Tag	Similarity
social media	marketing	0.87
social media	mobile web	0.71
social media	geofencing	0.51

t_b, and a similarity value sim between 0 and 1, see Table 3.4

Tag hierarchy prediction

Our tag hierarchy prediction algorithm extends the tag similarity algorithm by a hierarchy prediction step. The algorithm requires two input sources: (1) the weighted tag similarity matrix, $M_s(t_a, t_b, sim)$, output by the similar tags prediction algorithm, and (2) a popularity matrix $M_p(t, pop_t)$ that stores the popularity for each tag of the tag similarity matrix. The popularity describes how often a tag has been used in the underlying folksonomy. Additionally, the algorithm assumes a threshold value $thresh$ to categorize the results.

The tag hierarchy algorithm computes the semantic relation between tags in the user model of an active user. Note, that the computed hierarchy is valid only for the active user, not for all users. By default, any similarity between two tags is semantically *associative – semantically related*. The tag hierarchy extraction algorithm can enhance these associative similarities to hierarchical relations – *semantically broader* or *semantically narrower*, when certain criteria apply. Our algorithm adapts the algorithm presented by [104]. The algorithm computes semantic relations between two tags, which can be associative, broader, or narrower.

For each row in M_s, the algorithm computes whether t_a and t_b have an associative semantic relation, or if t_b is semantically broader or narrower than t_a. By default, t_a and t_b are categorized as associatively related. t_b is categorized as semantically broader than t_a if pop_{t_b} is larger than pop_{t_a} and sim_{t_a,t_b} is smaller than $thresh$. t_b is categorized as semantically narrower than t_a if pop_{t_b} is smaller than pop_{t_a} and sim_{t_a,t_b} is larger than $thresh$.

The output of the similar tags prediction algorithm is a matrix $M_h(t_a, t_b, sim, rel)$ of all tags in which the active user has interest in t_a, their top-N similar tags

Table 3.5: Tag hierarchy prediction output

Tag	Tag	Similarity	Relation type
social media	marketing	0.87	associative
social media	mobile web	0.71	narrower
social media	geofencing	0.51	narrower

t_b, a similarity value between 0 and 1 sim, and a flag indicating whether the similar tag is associated, narrower or broader rel. Table 3.5 illustrates the output.

3.7 Conclusion

The gain of this chapter is a user modeling method that creates user models required for exploratory search. In Chapter 2, we discussed why the Social Web, and particularly social bookmarking services, have potentials for user modeling for exploratory search: first, the service-oriented architecture and user-generated content available over APIs allows to retrieve data appropriate for user modeling at a large scale; second, the structure of the data underlying social bookmarking services allows for user model extraction with collaborative filtering algorithms. Collaborative filtering allows to mine knowledge in folksonomies with relatively low computational effort, hence being a fast and computationally cheap technique. We presented a method for algorithmic user model extraction from folksonomies. Our extraction method applies collaborative filtering to folksonomies to extract and explicate three categories of user properties – user interest, user similarity, and user knowledge. The resulting user models are an important enabler for exploratory search.

We discussed why ontologies are a viable form to represent our user models. Ontologies represent knowledge in a highly formalized, explicit, form. Therefore, they are more expressive than other forms of knowledge representation, like folksonomies. The problem of folksonomies is that relationships between users, tags, and resources in folksonomies are semantically undefined. Folksonomies implicitly define *that* its elements are related, however, they don't define formally *how* they are related. The formal semantics that

ontologies provide are an important step towards more powerful Semantic Web services. Ontologies integrate well into Semantic Web, because ontology modeling languages exist that allow to store and process ontologies with Semantic Web technologies. A key problem of ontologies is, however, that they are cumbersome to model and maintain. Our user modeling method is a strategy to automate the ontology modeling and maintenance process. It creates user-specific knowledge that is explicit, hence can be formalized in ontologies as we show in Chapter 5. Our method does not require manual engineering like other ontology modeling approaches we discussed; it relies solely on algorithmic extraction. Furthermore, our method is not restricted to fixed topic domains, because it does not involve mapping of the data set to existing domain ontologies. It is dynamic, because is develops ontologies automatically and on-the-fly, while the underlying data, the folksonomy, develops. For these reasons, our method is a viable approach for semantics extraction from folksonomies.

Our user modeling method creates explicit semantic relations between users and user-specific semantic relations between tags that are important for exploratory search. In particular, we extract three user-specific properties: user similarity, user knowledge, and user interest.

User similarity is important for exploratory search because it facilitates detection of communities of interest. Communities of interest are valuable to enable collaboration in exploratory search engines. Similar users can be presented to users to support domain discovery. They can act as navigational cues to help users exploring their social context: in a social tag search engine, users may browse bookmarks of similar users to discover new topics *accidentally*. Chances are good that two users which are highly similar share preferences for topic domains and webpages.

User knowledge is likewise important for exploratory search. In our user models, user knowledge represents user-specific tag semantics. Tag semantics are modeled by similarity and hierarchy relationships between tags. These tag semantics enable exploratory search engines to suggest cues for navigation through the information space. Similar tags can act as navigation cues for domain discovery. A tag search engine may present similar tags to a query

to foster exploratory browsing. Users can navigate to similar topics from the initial search context through suggested similar tags and discover other topics related to their initial search context.

Hierarchical relations model semantically *narrower* and *broader* relationships. Lack of hierarchical relations is a major limitation of folksonomies. Hierarchy is important for many applications. Exploratory search engines benefit from hierarchical relations between tags, because semantic hierarchy between tags enables domain learning. A tag search engine may present hierarchically related tags to a query to foster focused search. Users can navigate from general tags to more specific tags and vice versa through hierarchically related suggestions. Thus, hierarchical relations support navigation of users through an information space to dig deeper into a topic, or to get the whole picture by learning about the context of the topic. User knowledge is hence an enabler for navigation cues that guide users through information.

User interest is an important property to create the personalized user knowledge. User knowledge is inherently user-specific. Semantic relations between tags are different for each user. For a computer scientist, *apple* may be semantically related to *computer*, whereas for a biologist *apple* may be related to *fruit*. User interest is a prerequisite to personalize semantic relations for a user‘s knowledge. Furthermore, user interest is important for exploratory search because it improves domain discovery. A tag search engine may suggest only semantically related tags that are unknown – *novel* – to a user to foster discovery of new topics. To suggest novel topics and resources to users, a search engine must know which topics and resources the specific user already knows.

Finally, our method bridges the gap between two technologies: the Social Web and Semantic Web. Both worlds can profit from this mediation. On the one hand, we show an automated strategy to model and maintain knowledge with low effort. Our method creates more formal knowledge available on the Web – an important prerequisite for semantic Web services. On the other hand, we show how semantics can be incorporated into social bookmarking systems. Ultimately, we show how exploratory search can profit from semantics inherent in folksonomies.

Chapter 4

Contribution 2: A user model evaluation method

In the following chapter, we contribute an evaluation method for user models in the context of exploratory search. Our evaluation method comprises a set of experiments to evaluate how user models support domain discovery and domain learning. We exemplarily apply our evaluation method to the user models created by our *similar tag* extraction algorithm.

Evaluation of exploratory search differs greatly from the evaluation of query-and-response search engines. Traditionally, Information Retrieval focuses on providing users with all relevant results for a search query. Evaluation has concentrated on accuracy, completeness, and ranking of search results; the key measures for query-and-response search engines are *precision* and *recall* [14, 218]. Exploratory search has another focus and requires different evaluation measures and strategies. Although exploratory search involves querying and ranking of results, it is much more interactive than query-and-response search. Search results must not be evaluated in isolation from the surrounding context. Evaluation of exploratory search engines must consider the tasks that users want to fulfill and how users interact with the search engine [225, 223]. Evaluation aims at assessing how well some intended goals have been accomplished. The goals of exploratory search process are, however, ill-defined and dynamic. Exploratory search involves multiple or

multi-faceted goals which cannot be clearly defined (cf. Chapter 2). Furthermore, exploratory search is a long-term process where goals often change during the search process. It is difficult to seize clear goals for evaluation.

Our method measures how well an exploratory search engine supports the two key tasks involved in exploratory search: domain discovery and domain learning (cf. Chapter 2). In our experiments, we measure how good the user models extracted by our algorithms support these tasks in an exploratory search engine. For the evaluation, we implemented an exploratory tag search engine similar to that from our introductory example. In this search engine, we facilitate the domain discovery and domain learning tasks by recommending related tags for search queries. Our search engine allows users to query for tags to retrieve webpages for the tags – webpages that have been tagged with the query tags. Additionally, our search engine recommends the top-N similar tags for the query tag. Users can click on the recommendations to expand the search query with the recommended tag. They can further click on the initial tag to remove it from the query. Thus, users have two options to interact with the search engine: they can expand a search query by one or more recommendations, and they can replace a query tag by a recommendation. The recommendations facilitate domain learning and domain discovery by enabling *navigation* through the information space. The recommendations act as navigation cues that suggest *trails* through the information space. They guide users to related information that help them to (1) discover novel topics, or (2) learn more on an already known topic.

(1) The recommendations facilitate domain discovery by suggesting topics related to the current search query that are *novel* for the user. Users can follow recommendations to navigate through information, i.e., recommendations suggest a navigation trail through the information space. To foster discovery of topics, the recommended tags should be novel to the user. In our introductory example, Tom discovers a topic that was novel to him, *geofencing*, because the search engine suggested a trail from his initial query to the topic.

(2) The recommendations facilitate domain learning by suggesting topics that enable users to get more specific information on a topic or to con-

textualize a topic. In our introductory example, the search engine suggests *foursquare* as a more specific topic related to *geofencing*. Tom follows the trail to gain deeper insight into the topic. In the other direction, recommendations can guide to more general tags to provide context. A further function of the recommendations that enables domain learning is *result filtering*. Adding recommendations to the initial search query allows to narrow down the initial search results to a more specific context. In our introductory example, Tom adds a recommendation, *social media*, to his initial search query, *geofencing*, to narrow down the results to more specific information: webpages on *social media* and *geofencing*.

Our evaluation method measures how appropriate the navigation trails provided by the recommendations are to support domain discovery and domain learning. In other words, we measure how well the recommendations guide users through the information space to tags that yield domain discovery or domain learning. We do this by analyzing the structural properties of a set of pre-compiled recommendations. The aggregate of all recommendations in our search engine can be imagined as a directed network of tags interconnected through edges where recommendations exist. The topology, i.e., the internal structure, of this network informs us about its navigability. We measure the network connectivity to evaluate how users can navigate between tags. For example, we measure how likely a tag is to be recommended by counting the ingoing edges that refer to this tag from other tags. Our method extends the network analysis by combining the network properties with properties of the underlying folksonomy. For example, we complement the network analysis by relating the recommendations to the popularity of tags in the folksonomy and to the webpages tagged with these tags to measure how the recommendations affect the search results in our search engine.

We exemplarily evaluate three recommender algorithms. The baseline algorithm is identical to the *similar tag prediction* algorithm described in Chapter 3. For the evaluation, we implemented two modified versions of the recommender algorithm that we compare with the baseline algorithm.

The remainder of this chapter is organized as follows. In Section 4.1, we briefly review existing evaluation strategies for recommender systems and

discuss their shortcomings for our needs. In Section 4.2, we describe our method for evaluation of recommenders applied to user modeling for exploratory search, which extends the reviewed methods. In Section 4.3, we describe three recommender algorithms that we evaluate exemplary: our *similar tag prediction* algorithm and two alternative versions. In Section 4.4, we describe the experiment setup and results. In Section 4.5, we conclude with a discussion of the evaluation results.

4.1 Evaluation strategies

In the following section, we review existing recommender evaluation strategies and measures. The ultimate goal of any recommender system is to make recommendations that help a given user in a given context [162]. Evaluation aims at answering the question how well a recommender achieves this goal. Several measures and strategies exist that try to answer this question from different perspectives. They focus on different aspects of the recommender quality [188].

4.1.1 Predictive Accuracy

The focus of recommenders evaluation has long been on their *predictive accuracy* [94]. Accuracy evaluation measures describe how accurate the recommendations are. They seek to assess the quality of recommendations by evaluating if predicted ratings for items are the same as the actual ratings that a user would give to the items. The evaluation process for predictive accuracy involves three steps. First, existing preferences are split into a training set and a test set. Second, recommendations are generated by the recommender algorithm considering only the training set. Third, the ratings of recommended items are compared with the preferences in the test set to evaluate their predictive accuracy. The predictive accuracy describes how much the ranking of the predicted items differs from the ranking of these items in the test set. Important predictive accuracy measures are Mean Absolute Error, Precision/Recall, and F1.

The Mean Absolute Error [1] measures the average deviation between a rating predicted by a recommender and the user's actual rating. Other predictive accuracy measures related to MAE are Mean Square Error [2] and Root Mean Square Error [3]. All of these measures evaluate the deviation between existing and predicted ratings and differ in how they penalize deviations. Predictions are considered more accurate when the deviation of their rating compared to the test set is low. A shortcoming of these measures is that they make less sense for recommenders where rating values are unimportant. For example, in our search engine it is only important *which* tags are recommended, and not *in which order*. The discussed measures cannot properly evaluate the prediction quality in this context.

Binary classification measures precision, recall [4] and F1 [5], are more useful in contexts where the ratings don't affect the quality of recommendations. The measures classify items into relevant and non-relevant items. Additionally, they classify items into items that are recommended and items that are not recommended. Precision is then defined as the ratio of relevant items recommended to number of items recommended. It represents the probability that a recommended item is relevant. Recall is defined as the ratio of relevant items recommended to total number of relevant items available. It represents the probability that a relevant item will be recommended [103]. Usually, precision and recall are considered together. The F1 score combines precision and recall into one value. A problem of the binary classification in a recommender system is the subjectivity of relevance. The notion of objective relevance makes no sense in recommender systems, which recommend items based on the likelihood that they will meet a specific user's taste, and thus are inherently subjective [103].

A shortcoming of all predictive accuracy measures is that they can assess the rating or relevance only for those items that users have already rated. Accuracy measure cannot evaluate recommendations which are not in the

[1] http://en.wikipedia.org/wiki/Mean_absolute_error (last access 2011-04-22)
[2] http://en.wikipedia.org/wiki/Mean_squared_error (last access 2011-04-22)
[3] http://en.wikipedia.org/wiki/Root_mean_square_deviation (last access 2011-04-22)
[4] http://en.wikipedia.org/wiki/Precision_and_recall (last access 2011-04-22)
[5] http://en.wikipedia.org/wiki/F1_Score (last access 2011-04-22)

test set. This affects the evaluation results when recommenders are applied in a context where *novel* items should be suggested to users. Novel items are necessarily not in the test set, therefore, all accuracy measures by definition penalize prediction of novel items although these items may be relevant.

The quality of recommenders as quantified by predictive accuracy does not necessarily match their overall quality in a particular application context [189, 74, 132, 79]. A proper evaluation strategy should consider criteria that go beyond predictive accuracy [149]: *non-accuracy measures*. *Novelty* and *coverage* are two important non-accuracy measures

4.1.2 Novelty

Novelty describes wether a recommended tag is unknown for a user. It aims at evaluating the *nonobviousness* of recommendations [103]. Recommending items that yield discovery of novel, yet interesting items is a key strength of recommenders. Recommenders are often applied to recommend items that users would not have discovered without the recommendations. Novelty is closely related to the notion of *serendipity*: „[a] serendipitous recommendation helps the user find a surprisingly interesting item he might not have otherwise discovered“ [103].

Novelty is an important measure for exploratory search. Novel recommendations yield better exploration, because they foster discovery of unknown topics. In our introductory example, the search engine suggests a novel tag, *geofencing*, to Tom which yields the discovery of a topic that has been unknown to Tom but is interesting for him. Recommending a tag already known by Tom would probably not have yielded the discovery of a novel topic. For example, recommending the tag *programming* to a user who already knows the tags *java*, *c++*, and *python*, does probably not yield discovery of much new, because the recommendations are not serendipitous; we can assume that the user would have discovered all tags also without the recommendation.

4.1.3 Coverage

Coverage has been used to describe two distinct concepts: „(1) the percentage of the items for which the system is able to generate a recommendation, and (2) the percentage of the available items which effectively are ever recommended to a user“ [79]. We adopt the second definition, also referred to as *catalogue coverage* [103]. For example, in our search engine the catalogue coverage describes how many of the tags of the folksonomy are recommended effectively when assuming a set of queries for all tags in the folksonomy. Coverage is important in the context of exploratory search, because a search engine should make suggestions that cover all topics modeled in the data, i.e., it should not be limited to a subset of available topics. *Diversity* is a further measure which is closely related to coverage [148, 240]. It measures the percentage of available *topics* covered by the effectively recommended recommendations [79].

A high coverage of recommendations in an exploratory search engine yields better exploration, because recommendations are not restricted to a limited set of topics. For example, a recommender may effectively suggest only 10 percent of the available items. Although the recommendations may be accurate, this recommender would inhibit discovery of the remaining 90 percent of the items, hence would be less appropriate for exploratory search.

4.1.4 User-perceived quality

Predictive accuracy, novelty, and coverage allow only to evaluate the quality of a recommender *indirectly*. They must be interpreted considering the application context in which the recommender is used. *User-centric* evaluation measures the recommender quality more directly and is independent from the context in which the recommender is applied [123, 216]. It measures the *user-perceived quality* of the recommendations. User-perceived quality describes the usefulness of recommendations as perceived by users. User-centric evaluation strategies collect user feedback on the recommendations to measure the user-perceived quality [123]. This means that the evaluation setup explicitly asks users whether they find recommendations useful, or that

usage data are collected and analyzed to retrieve implicit user feedback. For example, a system may log whether users follow a recommendation to decide whether this recommendation is useful.

A problem of user-centric evaluation is that it requires either user feedback collection through a survey or through usage logs. Surveys require explicit user involvement which is not feasible in all systems. Collected usage data may not be available in some cases. For example, it is not possible to evaluate the user-perceived quality of a system before it has been deployed and actively used.

4.2 Our evaluation method

In the following section, we describe our evaluation method. The goal of our evaluation is to measure how recommender algorithms support exploratory search. Domain learning and domain discovery are the key tasks that an exploratory search engine must facilitate. We evaluate the quality of recommender algorithms implemented in an exploratory search engine.

Our method comprises a set of experiments that primarily evaluate the novelty and coverage of recommender algorithms. Therefore, we evaluate the navigability of the underlying recommendation networks. Additionally, our method evaluates how recommendations affect the retrieved search results when applied in a search engine. Therefore, we relate recommended tags to their popularity score and to the search results.

We exemplarily apply the evaluation method to our tag similarity extraction algorithm. For the evaluation, we implemented two alternative algorithms that extract tag similarities from folksonomies. We populated user models with all three algorithms by recommending the top-N similar tags for all available tags in the folksonomy. Our baseline algorithm recommends the top-N similar tags as described in Chapter 3. A second algorithm biases the recommendations towards popular tags. A third algorithm biases the recommendations towards unpopular tags. We apply our evaluation method to compare these three algorithms.

Table 4.1: Input dataset features

Bookmarks	209307
Tags	566782
Tags (popularity $>$ 1)	169899
Users	209307
URLs	2790459

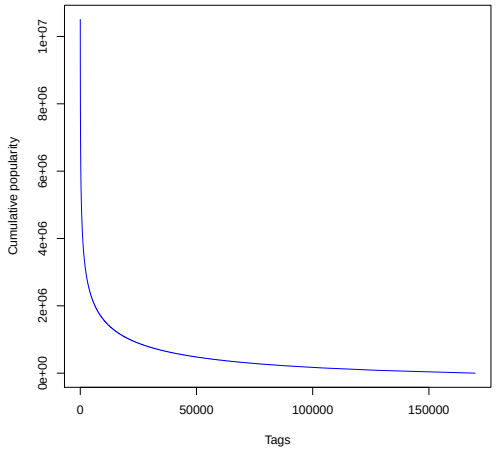

Figure 4.1: Cumulative tag popularity distribution

4.3 Input dataset

The input dataset for our experiments are bookmarks that we collected from two social bookmarking services – Delicious and Connotea – over 30 days. Each bookmark contains one user (the creator of the bookmark), one URL (the reference to the bookmarked webpage) and one or more tags. We did not collect bookmarks without tags, because they cannot be used for recommendations by our algorithms. Furthermore, our dataset excludes tags which were used only once. We assume that these tags have only little relevance for recommendations or result from typos. Table 4.1 shows the features of the collected data.

Table 4.2: Quantiles of the popularity curve

Min	1st Quantile	Median	Mean	3rd Quantile	Max
2	3	5	61.84	13	166200

4.3.1 Tag popularity

The popularity of the tags in our input dataset is an important feature for our evaluation method. We define tag popularity as the number of times a tag has been used. Our dataset shows a popularity distribution for the tags that is characteristic for social tagging systems: a small number of tags have a high popularity and the majority of tags have a relatively small popularity. In other words, users of social tagging systems tend to use a small number of tags very often. Popular tags are likely to be tags which are being used by a large number of users. The majority of tags is used only rarely. These tags are likely to be tags which are used only by a smaller number of users. Figure 4.1 shows the cumulative tag popularity distribution of our input dataset excluding tags with popularity < 2.

In our experiments, we want to evaluate how our algorithms perform for tags with different popularity scores. Therefore, we model a tag popularity distribution curve. We divide the distribution curve into a head, center, and tail part, see Table 4.2. To decide the split points, we calculate the 1^{st} and 3^{rd} quantile of the curve. We define the head of the curve as all tags with a popularity $>$ the 3^{rd} quantile, the center of the curve as all tags with popularity between the 1^{st} quantile and the 3^{rd} quantile, and the tail of the curve as all tag with a popularity smaller than the 1^{st} quantile.

4.4 Recommender algorithms

In the following section, we describe three collaborative filtering-based recommender algorithms that we evaluate. All algorithms apply the Tanimoto coefficient to generate similarities between the tags in our input data.

4.4.1 Algorithm 1: Unbiased

The baseline recommender algorithm, T(s), recommends the top-N similar tags for a tag directly based on a pairwise tag similarity matrix. We use the Tanimoto coefficient to generate the similarity matrix. T(s) corresponds to the algorithm described in Chapter 3.

4.4.2 Algorithm 2: Biased towards popular tags

The second algorithm, T(p), biases the recommendations towards popular tags. It recommends only tags from the head and center of the popularity distribution curve. Hence, T(p) excludes unpopular tags, tags from the tail of the curve, from the recommendations.

4.4.3 Algorithm 3: Biased towards unpopular tags

The third algorithm, T(u), biases the recommendations towards unpopular tags. It recommends only tags from the center and tail of the popularity distribution curve. Hence, T(u) excludes popular tags from the recommendations.

4.5 Evaluation datasets

For our experiments, we generate a set of recommendations for each algorithm and correlate the recommendations with the search results. Our evaluation data consists of two datasets.

(1) We create a pre-compiled set of recommendations for each algorithm. The recommendation sets relate all tags of our input data to their top-N similar tags. Formally, each of these recommendation sets can be seen as a directed network of tags related through edges, see Figure 4.2. In these *recommendation networks*, nodes represent tags and directed edges represent the similarity relations from tags to their top-N similar tags. The formalization of recommendations as networks allows us to apply network analysis algorithms to evaluate the structural properties of the recommendation sets.

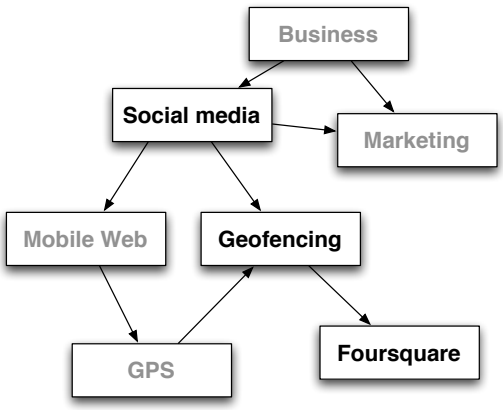

Figure 4.2: Recommendation network

For example, we measure network features to evaluate how well it is possible to navigate through recommendation networks by following recommended paths, i.e., we evaluate the navigability of the recommendations.

(2) We combine the recommendation networks with the query results of our search engine. We create a *query result matrix* that relates all tags of our input data to the URLs of the webpages which have been tagged with this tag and are retrieved by querying this tag. The query result matrix allows us to evaluate how query modification – expansion or replacement of the query – by following recommendations affects the results retrieved by our search engine. For example, we measure how the results change when an initial search query is replaced by a recommended similar tag.

4.5.1 Recommendation networks

To create the recommendation networks, we pre-compile the top-N similar tags for all tags in our dataset for each algorithm (see Table 4.3). The algorithms don't guarantee that n recommendations are available for each tag. Therefore, some tags may have less than n recommendations. For some tags, no recommendations may be available. Hence, for each tag in the compiled recommendation sets $\leq n$ recommendations exist. For each of our three recommender algorithms, we model two recommendation sets with different number of similar tag recommendations. The first set recommends

Table 4.3: Recommendation set

Tag	Recommendation
social media	marketing
social media	mobile web
social media	business
social media	geofencing
social media	social networks
geofencing	location-based services
geofencing	foursquare
geofencing	maps
...	

Table 4.4: Query result matrix

Tag	Results
internet marketing	[urlA, urlB, urlC,...]
social media	[urlB, urlF, urlG,...]
twitter	[urlA, urlP, urlQ,...]
...	

the top-5 similar tags (n=5), the second recommends the top-10 similar tags (n=10).

In our experiments, we want to compare how the tag popularity affects the results of our algorithms. Therefore, we split the recommendation sets of each algorithm into three subsets for the evaluation. The subsets contain recommendations for tags only from the head, center, or tail parts of the popularity curve. The head sets contain only recommendations for popular tags (with a popularity > 13), the center sets contain recommendations for tags with popularity (with a popularity between 3 and 13), and the tail sets contain the recommendations for unpopular tags (with a popularity < 3). The result of this partition are eight recommendation networks for each algorithm: overall, head, center and tail; each for the top-5 and the top-10 recommendation set.

4.5.2 Query result matrix

We want to evaluate not only the recommendations as such, but also how they affect the *retrieval results*. Therefore, we relate all tags in our recommendation networks to the URLs retrieved by querying this tag. To compare the query results of individual tags or combinations of tags, we create a matrix in which each row contains a tag and its query result vector, see Figure 4.4. The query result vector is a vector of URLs that are retrieved by querying the tag.

In some of our experiments, we relate the recommendation networks to the query result matrix. This allows us to evaluate the results of individual tags or a combination of tags, e.g., for a combination of a tag and its top similar tag. The query result matrix provides data for two types of analyses. First, the length of the result vectors allow for a quantitative analysis of the result vectors, i.e., we can compare the number of results for individual queries. For example, we measure how strong the expansion of the initial search tag by a recommended tag reduces the number of retrieved results. Second, it allows to compare which results the query result vectors contain for individual tags or a combination of tags, i.e., we can carry out a qualitative analysis of the result vectors. For example, we measure how many of the retrieved results remain the same when the initial query is replaced by a recommendation.

4.6 Experiments

In the following section, we describe our evaluation experiments. We conduct two sets of experiments that evaluate the quality of our recommender algorithms in two aspects: domain discovery and domain learning.

4.6.1 Domain discovery

In the first experiment set, we evaluate how well the algorithms support domain discovery. In our search engine, users can query for one or more tags. The recommender suggests the top-N similar tags for the query. Users

can then click on the recommendations to expand the search query with the recommended tag. They can further click on the initial tag to remove it from the query. Thus, users can modify the search query, i.e., they can replace a query tag by a recommended tag. In this way, users can navigate through the tag space.

We formalize the tag space as a directed network of tags, which corresponds to the recommendation networks of our evaluation dataset. Tags are connected to their top similar tags with egdes. Edges allow users to navigate from one tag to another. Thus, we image tag recommendations as navigation trails through the tag network. The tag recommendations provide navigation cues for users who navigate through tag networks. Users can follow the trails to discover novel tags and to retrieve novel search results.

Navigation through the information space is a crucial facility to discover novel or serendipitous tags and webpages. Tom, the user from our initial example who queries for *social media*, discovers a novel topic which he is interested in, *geofencing*, through a recommendation. By following the recommendation, he finds a further novel topic, *foursquare*. He explores the tag space by navigating through recommendations. Navigating through recommended tags yields discovery of topics by facilitating exploratory browsing. The ability to navigate from *social media* to *foursquare* yields in the discovery of a new topic for Tom. The recommendations must form a network of connected tags which users can traverse to get to novel topics. We define three criteria of the recommendations that affect domain discovery:

(1) Domain discovery relies on the ability to navigate to further tags through the recommendations. Recommendations should support navigation through the tag space. Navigation relies on recommendations that lead to further recommendations, because these recommendations allow to further explore the tag space. The algorithms should recommend tags for which further recommendations are available. For example *geofencing* leads to another recommendation, *foursquare*, which could lead to another recommendation *check-in*, and so on. Recommendations that don't lead to further recommendations or only lead back to already seen recommendations thwart the navigation through the tag space.

(2) Domain discovery relies on a high coverage and diversity of topic domains in the recommendations. Recommenders should be able to recommend a large portion of tags from the tag space, because domain discovery relies on the ability to navigate to a large number of topics. At best, recommendations should be available for all queries. The recommendations should also be diverse, so that all tags available in the input data can be effectively recommended. A recommender that recommends only a small part of the tags is not useful to discover novel tags, because it does not provide any navigation paths that lead to the rest of the tags.

(3) Domain discovery relies on diversity of *search results*. The ultimate goal of Tom in our example is to discover not only tags, but also the actual search results – webpages. This means that users should not only be able to discover tags, but also webpages. The topic represented by the results retrieved through recommended tags should therefore shift when the query is modified: query modification towards a recommended tag should shift the query results to another topic domain. When Tom navigates from the tag *social media* to *geofencing*, the search engine should reflect this shift by providing new results.

Experiments

In the experiments we evaluate the navigation facilities between topics by measuring the structure of the underlying recommendation networks that our algorithms produce. The internal structure of the recommendation networks strongly affects the navigation capabilities through the information space [126, 160, 51]. We measure how good the recommendation networks support navigation through the tag space and how the recommendation networks affect the query results.

Popularity-indegree correlation First, we want to measure how good the recommendation networks allow to transition between different topics. For domain discovery, it is beneficial if the recommended navigation path leads to discovery of new topic domains, so that the user can transition between many different topic domains in few navigation steps.

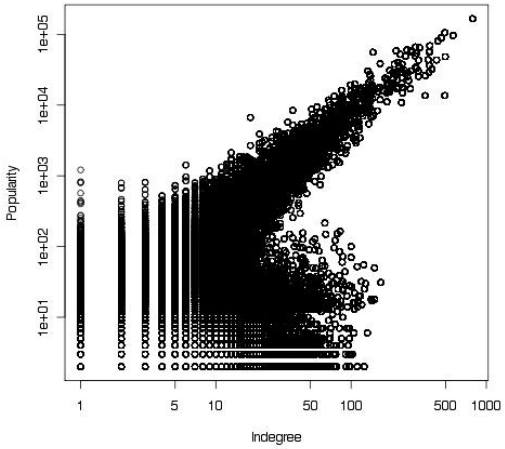

Figure 4.3: Indegree-popularity correlation plot for T(s,10) network (log-log)

Table 4.5: Indegree-popularity correlation coefficients

T(s,5) / T(s,10)	0.85 / 0.94
T(p,5) / T(p,10)	0.89 / 0.92
T(u,5) / T(u,10)	0.87 / 0.89

To evaluate how good recommendations allow to transition between domains, we look at the topology of our recommendation networks. An interesting measure in this context is *indegree centrality*. The indegree centrality of a tag informs us about how likely the tag connects distinct domains. Tags that connect distinct domains allow to transition from one topic domain to another. Hence, these tags should be favored in the recommendations to enable discovery of new domains.

Indegree of a node in a network measures the number of edges that have their endpoint on that node, i.e., the number of ingoing edges [33]. The indegree can be used as a measure for centrality of a node. Nodes with a high indegree centrality are strongly connected with other nodes and often act as *hubs* in a network, i.e., they are nodes that connect distinct sets of clustered nodes in a network. We assume that sets of clustered nodes are topic domains. Hubs, nodes with a high indegree, facilitate domain discovery, because they connect different domains. We hypothesize that popular tags have a high

indegree. To confirm this hypothesis, we evaluate if the popularity of recommendations correlates with their indegree in the tag-recommendation network. We test if the popular tags in the recommendation network are likely to have also high indegree centrality. Therefore, we measure the indegree of recommendations and correlate it with their popularity. We conducted the experiment for all recommendation networks.

We found that the popularity and indegree of recommendations show a high correlation coefficient (>0.87) for all algorithms. Figure 4.3 shows the correlation plot for the T(s,10) network. Table 4.51 shows the indegree-popularity correlation coefficients for all networks.

The results show that popular tags in our dataset are also hubs in the recommendation networks. Popular tags are more likely to connect different domains. The findings indicate that algorithms which recommend more popular tags are better for domain discovery. This implies that our algorithm T(p), which biases recommendations towards popular tags, enhances domain discovery opposed to T(s) and T(u), because its recommendation network contains more popular tags.

Clustering coefficient The findings of the popularity-indegree correlation experiment suggest that we should bias recommendations towards strongly connected tags to improve transition between sets of clustered tags. This biasing can, however, also have disadvantages. If a set of tags is strongly connected, it is possible that recommendations of these tags refer to tags inside the same cluster. For a cluster of strongly connected tags, it is also possible that the recommendations connect the tags mutually and don't refer to tags outside the cluster. Navigating these recommendation paths would not lead to a new tag cluster, but to the same tags inside a cluster again and again. Thus, these recommendations don't support domain discovery. This effect occurs particularly if a tag cluster is only weakly connected to other clusters, or even isolated.

Again, the structure of our recommendation network informs us about the quality of recommendations in this regard. We measure the global clustering coefficient of our network to evaluate how good new tag clusters can

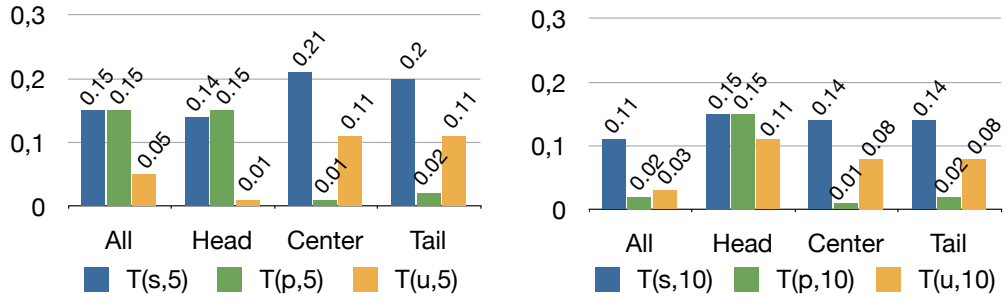

Figure 4.4: Clustering coefficients

be reached through recommendations. The global clustering coefficient of a network measures the probability that the adjacent nodes of a node are connected to each other [33]. A higher clustering coefficient indicates a higher probability to get stuck in unconnected small tag clusters when navigating through the recommendation network.

We found that the clustering coefficient is significantly lower for the T(u) networks in the head of the tag popularity curve and for the overall curve, see Figure 4.4. For the center and tail of tag curve, the clustering coefficient is significantly lower for T(p).

The results indicate, that the navigability of the network is sensitive to the popularity of the search query. For popular search tags, it seems beneficial to bias towards unpopular recommendations to avoid to get stuck inside clusters. For less popular tags, however, biasing towards popular recommendations improves the navigability between clusters. Without popularity-based adaption of recommendations, the results suggest that T(p,10) provides the best navigability, because the algorithm has the lowest clustering coefficient for the overall popularity curve.

Average path length The clustering coefficient experiment informs us about the navigability between tag clusters. But for a more complete picture of the navigability of the network, we have to consider that some tag clusters are completely isolated from each other. We hypothesize that it is not possible to navigate from a tag to all other tags in the network through

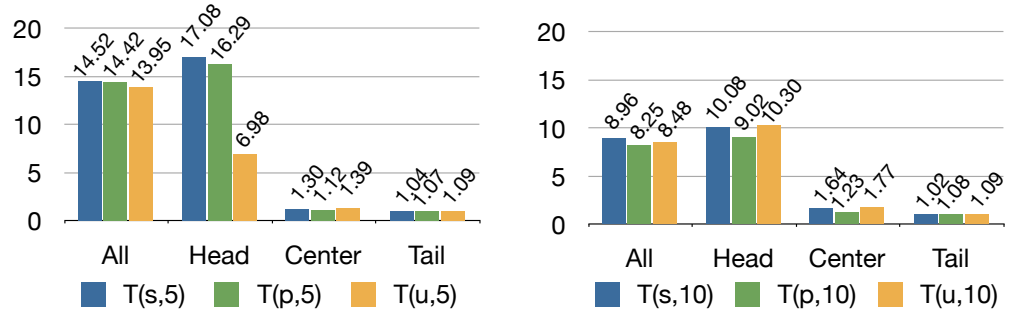

Figure 4.5: Average path lengths

recommendations, because navigation paths don't exist between all tags. Instead, we assume that a number of unconnected subnetworks of tags exist.

To learn about the size of these subnetworks, we measure the average path length of the network, see Figure 4.5. The average path length for a network measures the average path length between all nodes in the graph [33]. In our context, the average path length indicates how large the subnetworks are. It indicates how many clicks are required to navigate from one tag to any other tag inside a subnetwork. We assume that a low average path length is an indicator for *unconnectedness* of the network. Unconnectedness of a network results in a low average path length because missing edges between nodes are not considered in the calculation of the path length.

We found no significant differences between our algorithms when we look the overall networks. When we look at the parts of the popularity curve individually, our results show that T(u,5) provides a significantly lower value than T(s,5) and T(p,5) for the head of the popularity curve. We also found that the average path length of the networks are significantly lower for the center and tail of the curve.

Our results indicate that the center and tail of the popularity curve show a lower connectivity of the subnetworks. In other words, we suspect that the tail of the graph consists of many small and unconnected subnetworks of tags. However, average path length may also indicate a better navigability of the network: another interpretation of the results could be that tags are more strongly connected on the average. To confirm our interpretation of

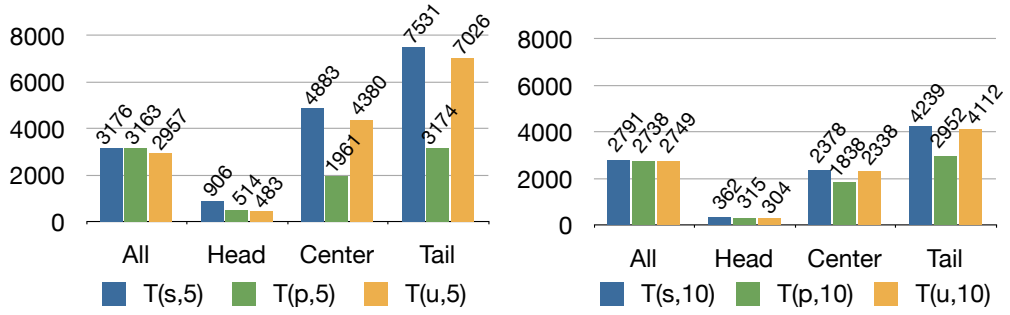

Figure 4.6: Number of clusters

the average path length, we conduct two further experiments.

Clusters To confirm that the tail of the popularity curve is less connected than the head, we measure the number of isolated clusters of tags in our networks. The number of clusters measure the connectivity of a network. Clusters are subnetworks which are isolated from other subnetworks. A large number of clusters indicates a poor navigability of the network, because users cannot transition between clusters through recommendation paths. Many small and unconnected clusters indicate that it is less likely possible to navigate between tags through recommendations, on the average [33].

We found that the center and tail of the tag curve show a significantly larger number of unconnected clusters than the head, see Figure 4.6. The number of clusters is the highest in the tail. We also found that T(p) reduces the number of clusters opposed to T(s) and T(u) when looking at the head, center, and tail of the tag-curve individually. The reduction of the number of clusters for T(p) is particularly significant in the tail area. Also, a higher n parameter reduces the number of clusters. The T(10) networks are more connected than the T(5) networks.

The results confirm our interpretation of the short average path length in the tail. They indicate that the navigability of the recommendation networks is limited in the tail of tag-curve. Thus, the popularity of the search query strongly affects the quality of the recommendations in this context. Furthermore, the results indicate that, particularly for the tail, T(p) improves the

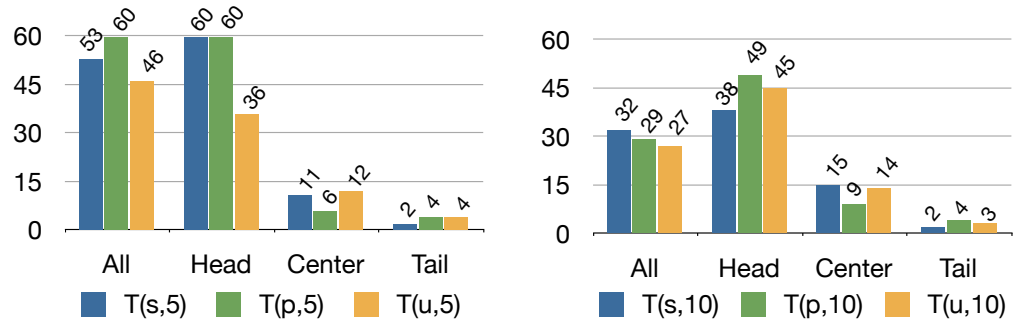

Figure 4.7: Diameters

connectivity of the network, because it creates a smaller number of unconnected recommendation networks. Additionally, a higher n seems beneficial for connectivity of the networks.

Diameter To further back our interpretation of the navigability of our networks, we measure their diameter. The diameter is the longest geodesic of a network. A low diameter for a network with many tags indicates that the network consists of small subnetworks.

We found that the diameter is significantly lower in the tail of the tag curve for all networks, see Figure 4.7. The results show no clear beneficial tendencies for any of our algorithms, although differences exist. T(p,5) provides a larger diameter for the overall network opposed to T(s,5) and T(u,5), but the beneficial effect cannot be found for the center and tail of the tag curve. Additionally, we looked at the parts of the popularity curve individually and found that the networks for the head have a significantly larger diameter.

The results confirm the above findings that the tail networks consist of small and unconnected clusters, which negatively affects their navigability.

The combined results of the average path length, cluster, and diameter experiments indicate that the tail of the tag curve is highly unconnected and hence provides limited navigability. The unconnectedness of the recommendation networks for the tail of the tag curve seems to be a major shortcoming of our recommender algorithms. For unpopular queries, our al-

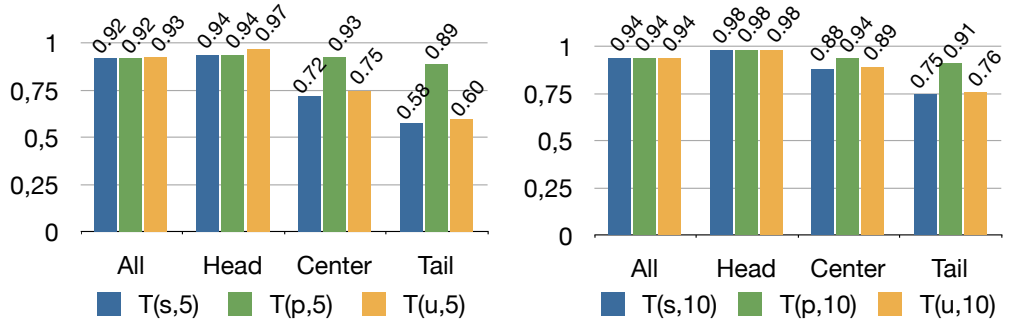

Figure 4.8: Strong Giant Components

gorithms provide only a small number of recommendations (on the average), which makes it difficult to navigate between unpopular tags. On the other side, the algorithms provide good navigability for the head of the popularity curve.

Strong Giant Component In a further experiment, we evaluate the *coverage* of the recommendation networks, i.e., how many of the overall tags are effectively recommended by our algorithm. The topology of our recommendation networks informs us about the coverage of our algorithms. We assume that our networks have one large component – or cluster – of interconnected tags. Our above findings indicate that these are probably popular tags which are more strongly connected. We measure the Strong Giant Component (SGC) of the networks. The SGC measures the percentage of tags in a network which belong to the largest cluster. In our context, it measures the percentage of tags in the network that can be reached from a query tag which belongs to the Strong Giant Component, through recommendations.

We found that T(p) positively affects the SGC opposed to T(s) and T(u) in the center and tail, see Figure 4.8. We also found that the SGC is significantly lower for the tail of the tag curve. In other words, in the tail of the tag curve, a lower percentage of tags belong to the largest cluster. A larger number of tags is unconnected from this component. These tags are not reachable through recommendations from a tag in the SGC.

The results indicate that the coverage of recommendations is strongly

affected by the popularity distribution. In the head of the tag curve, the SGC and hence the coverage of the recommendations is fairly high. In the tail of the tag curve, the coverage is significantly lower for T(s) and T(u). T(p) improves the coverage of the recommender in the center and tail. In the head, we found no significant differences between the algorithms. Hence, it seems beneficial for the coverage to bias towards popular recommendations for queries, at least for the tail of the tag curve.

Query result similarity In a further experiment, we evaluate how navigation through the recommendation networks affects the results of the search query. The ultimate goal of our search engine is to retrieve novel webpages for the user, so that the user can discover novel domains. Hence, we must evaluate the quality of the *retrieval results*. The role of the recommendations is to provide a navigation path through a network of tags. Users can modify the search query by navigating through recommendations. The query modification leads to a modification of the retrieved results.

To evaluate the effects of the recommendations on the query results, we measure how the results change when a user replaces a query through a recommendation. Therefore, we calculate the similarity of the result vectors of the initial query tag and the replaced query – the recommendation. The result set similarity indicates how strong the results shift when the initial query tag is replaced by a recommendation, i.e., how many of the results remain the same for the new query. We assume that a lower similarity between the results indicates better support for discovery of new webpages, because users usually replace a query to retrieve new results.

In our experiment, we compare the result vectors of the initial query and the replaced query. We measure the similarity of both result vectors, i.e., how strong the results of the initial and the new query overlap. The query result similarity informs us about how many of the results of the initial query are retained when the user changes the query to a recommendation. We define the query result similarity S as number of results r retrieved by the initial query q_1 and the replaced query q_2, divided by number of results of the replaced query, see Equation 4.1.

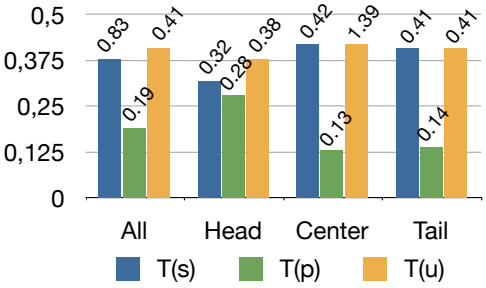

Figure 4.9: Result similarities

$$S_{q_1,q_1} = \frac{r_{q_1} \cup r_{q_2}}{r_{q_2}} \tag{4.1}$$

We found that T(p) has a lower result similarity than T(s) and T(u), see Figure 4.9. We looked at the parts of the popularity curve individually and found that T(u) and T(s) differ mainly in the head of the popularity curve, where T(u) has a higher similarity. In the center and tail of the curve, both algorithms have similar result similarity. For T(s) and T(u), we also found that the similarity is higher in the tail than in the head. For T(p), the opposite is true – the similarity is lower in the tail party than in the head.

The results indicate that T(p) significantly reduces the result similarity. This algorithm retrieves the most new results when a query is modified to a recommendation, which is desirable for domain discovery. Hence, biasing towards popular recommendations improves the discovery of new webpages. The positive effect of T(p) is particularly significant for unpopular queries.

Conclusion

To conclude, we discuss our findings regarding the domain discovery capabilities of our search engine.

We found that domain discovery is supported best by biasing recommendations towards popular tags. The results of our above experiments show that, across the board, T(p) benefits the navigability and coverage of the recommendation networks. We have shown this by evaluating the average

path length, number of clusters, diameter, and strong giant component for the individual areas of the tag curve. For some measures, however, the popularity of the query tag influences which algorithm supports domain discovery best. The results of the clustering coefficient experiment show that although T(p) reduces the possibility to get stuck in small connected clusters of recommendations for query tags with moderate or little popularity, biasing towards popular recommendations has the opposite effect for popular queries. Hence, it seems beneficial for domain discovery to adjust the biasing direction of the algorithm according to the popularity of the search query.

The number of recommendations, i.e., the n parameter of the recommendation networks, also influences their navigability. We compared only two n values, $n = 5$ and $n = 10$, out of which $n = 5$ performed slightly better. However, a viable n parameter certainly affects other factors that are beyond the scope of our evaluation. For example, the user interface of the search engine is certainly an influential factor to decide how many similar tags can be presented to the user without overtaxing the user‘s perception.

Altogether, the quality of the recommendations of all of our algorithms is very sensitive to the popularity of the query tag. Regarding domain discovery, we can state that all of our algorithms provide significantly better recommendation networks for popular tags. We have shown with the evaluation of clusters and diameter of the recommendation networks that all of our algorithms provide only poorly navigable graphs when we isolate the unpopular tags. A key reason for the low quality of the recommendation networks in the tail of the tag curve is the low connectivity. The sparsity of the recommender networks of all of our algorithms in the tail area of the tag curve is their main weakness. Domain discovery could greatly benefit from a higher connectivity of the recommendation networks for the tail of the tag curve. We have shown with the evaluation of clusters, diameter, and strong giant component that T(p) significantly improves the connectedness of the recommendation networks in the tail of the tag curve. It is an important feature of search engines to provide quality results not only for popular queries. Thus, it seems a major challenge for our system to provide recommender algorithms that further increase the connectivity of recommendation networks

for less popular query tags.

The low connectivity for recommendation networks of unpopular query tags results from the sparsity of the input data set. T(p) can improve connectivity of the graph but the improvements are limited. Sparsity of input data remains a key problem for our algorithms and for recommenders in the context of search engines in general. The quality of the recommendation networks for the tail of the tag curve could benefit from a better input data set. The quality of the data set for domain discovery depends on two features. First, the *size* of the data set is clearly an important factor. However, our evaluation results for the head area of the popularity curve show that our algorithms provide good recommendations already with a medium-sized input data set. Second, the *coverage* and *diversity* of topic domains in the input data seems crucial for domain discovery. For domain discovery, it is important that a large number of domains are covered in the input data.

4.6.2 Domain learning

In a second experiment set, we evaluate how well our algorithms support domain learning. In our exploratory search engine, users can query for tags and manually expand queries through recommendations, i.e., users can add a recommended tag to the initial query. The query expansion supports domain learning, i.e., users that are already familiar with a topic domain can further explore the domain. We define two criteria for recommendations to support domain learning.

(1) Recommendations should allow to learn more on a domain by discovering more tags on the domain, which are potentially more expert tags. These tags are likely to be more specific and less popular. For example, users that want to learn more on *social media* could learn more about the topic by getting more specific recommendations like *foursquare*. Also the other direction can support learning: more general recommendations can support users in contextualizing a tag. To support domain learning, our search engine should provide equally good recommendations for both non-expert and expert users, i.e., it should recommend both popular and unpopular tags to increase the

diversity of recommendations. Additionally, it should allow to navigate from non-expert to expert tags and vice versa. Thus, domain learning relies on a good transitivity of the recommendation network throughout the popularity curve. It should be possible to transition from popular to unpopular tags to deepen knowledge of a topic. In the opposite direction, it should be possible to transition from unpopular to popular tags to contextualize a tag.

(2) Recommendations allow to filter the results through query expansion. In our search engine, query expansion narrows down the results to webpages which have been tagged with *all* tags in the query. When a large number of results is available for a topic, filtering helps to sort out webpages. For example, users that are experts in a domain can filter out general, non-expert webpages. Domain learning relies on filtering out irrelevant results. Query expansion through recommendations should support result filtering, i.e., it should decrease the number of results at a sensible rate.

Experiments

In the experiment set, we measure the domain learning support of our algorithms.

Popularity correlation First, we want to find out how good our algorithms are capable to make recommendations from popular and unpopular parts of the popularity curve. We assume that unpopular tags are more likely to be expert tags and popular tags are more general tags. To support domain learning, our algorithms should make recommendations that allow to transition from popular to unpopular tags.

We measure the *Pearson Correlation Coefficient* for the popularity score of the tags in the recommendation networks. The Pearson Correlation Coefficient describes the *popularity correlation* between tags and their recommendations, i.e., how likely it is that a recommendation has the same popularity than the initial tag. The popularity correlation of tags and their recommendations indicates how likely it is to transition between popular and unpopular tags. When popularity of tags and their recommendation correlate, it is hard to transition between popularity levels. For example, a query for a popular

Table 4.6: Popularity correlation

T(s,5) / T(s,10)	-0.001 / 0.000
T(p,5) / T(p,10)	-0.001 / 0.001
T(u,5) / T(u,10)	-0.002 / 0.000

tag cannot lead to unpopular tags when all recommendations for the tag have similar popularity. A low correlation between the popularity of tags and recommendations indicates that it is possible to transition between parts of the popularity curve, and thus to navigate to tags with all popularities from queries of all popularities. The popularity of tags and their recommendations should not correlate to support transition between popularity levels.

We found no correlation for any of our algorithms, see Table 4.6; the correlation coefficient is $\leq 0,002$ for all recommendation networks. The results indicate, that all of our algorithms support transition between tags with different popularity scores. Furthermore, all algorithms cover popular as well as unpopular recommendations, hence are appropriate to make recommendations throughout the head, center, and tail of the tag popularity curve.

Reduction rates In a further experiment, we evaluate how good query expansion through recommendations supports filtering of query results. When users expand a query by a recommendation in our search engine, the result set is narrowed down to webpages which have been tagged with both tags. For example, a user can expand the query *social media* by *mobile web* to narrow down the results to webpages which are related to the *social media* and *mobile web*. We assume that this is a viable strategy to filter results for domain learning. An appropriate filtering rate helps to identify different aspects of a topic, thus improves domain learning.

We measure the reduction rate for query expansion, i.e., the length difference between the result vectors of the initial and the expanded query. Our experiment is set up as follows. First, we evaluate the result vector lengths for all tags in the recommendation networks, i.e., how many results does a query with a single tag retrieve. Second, we evaluate the result vector lengths for queries with tags and their top recommendation, i.e., how many results

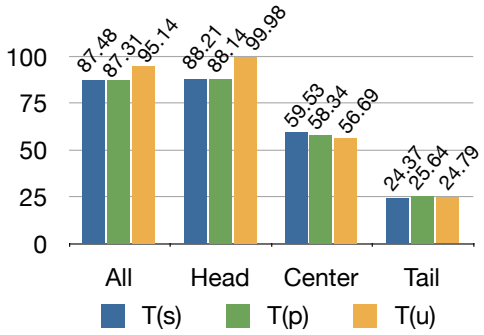

Figure 4.10: Result reduction rates (in percent)

does a query for both tags (initial tag and top similar tag) retrieve. We compare the lengths of both result vectors to get their length difference. The length difference of the result vectors indicates how good query expansion through a recommendation filters the result set.

We found that, for the head of the popularity curve, T(u) significantly increases the result reduction rate opposed to T(p) and T(s), see Figure 4.10. T(u) reduces the results by almost 100 percent, which means that almost all results are filtered out. In the center and tail of the popularity curve, the algorithms don't significantly differ in the reduction rate. All algorithms show a reduction rate around 58 percent in the center and 25 percent in the tail.

The results indicate that biasing the recommendations towards popular or unpopular tags has no strong effects on the reduction rate. Only T(u) increases the reduction rate for popular query tags. The reduction rate for all algorithms seems very high, but it is difficult to interpret the results without consideration of some absolute values.

Mean result lengths In a next experiment, we look at the absolute numbers of the result lengths to interpret the reduction rates, i.e., to decide whether the reduction rates of our algorithms lead to appropriate filtering. First, we measure the average absolute numbers of results for the initial, non-expanded, queries. The absolute number of results for the initial query indicates if it is necessary to filter the results at all. Filtering is rather irrel-

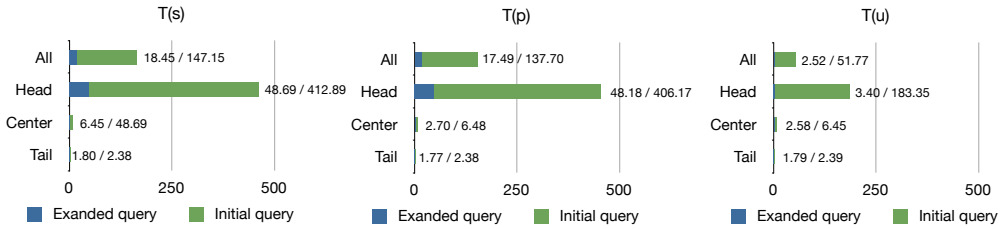

Figure 4.11: Result length for queries and expanded queries (in percent)

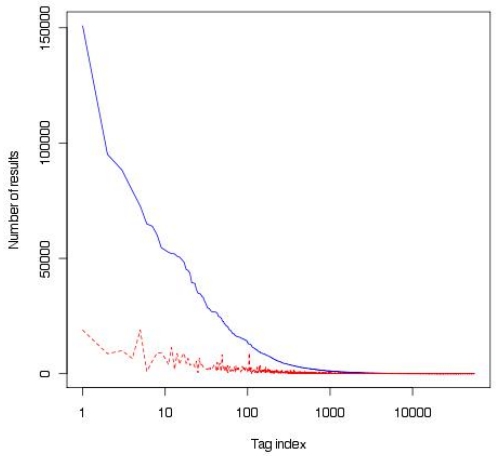

Figure 4.12: Result vector length for queries (blue) and expanded queries (red dotted) for T(s)

evant for queries with very few results. We suppose that the result lengths differ between queries for popular and unpopular tags, therefore, we split the evaluation into head, center, and tail parts of the popularity curve. Second, we measure the average absolute result numbers of expanded queries. This number indicates if the reduction leads to a result set of sensible size.

We found a dramatic drop of result vector length between head part of the tag popularity curve and its center and tail, see Figure 4.11. In the center and tail of the popularity curve, the number of results for the initial query is so low that filtering is not necessary in most cases. Figure 4.12 illustrates this phenomenon, the graph shows the result lengths for initial queries and expanded queries for T(s). [6]

[6]Note, that the number of query results slightly differs between the algorithms, because

The results indicate that query expansion is more likely to be required for popular queries. Here, the number of results for the initial queries are so high that filtering is required to help users to cope with the large result number. In the center and tail of the popularity curve this is not the case. The average result number is so low here, that users can easily look through all results without further filtering through query expansion. Furthermore, our results confirm our hypothesis that the reduction rates are too high in some cases. For popular queries, the reduction leads to a sensible number of results. In the center and tail the result filtering makes no sense. On the other side, it is not required here, so that we can neglect the filtering rate for these parts of the curve.

Our findings of the experiment indicate that, in the tail of the popularity curve, our dataset is too sparse to support domain learning. We suspect that the result of this experiment is largely due to the size of our data set and due to the sparsity of our input data set. With a larger dataset, the query result length for the tail of the curve would certainly grow to a number where filtering becomes necessary.

Expansion rate with zero results We conduct a further experiment to evaluate how query expansion affects the query results. A phenomenon of our collaborative filtering algorithms that is noticeable here is that two tags may be highly similar, according to the recommender, but may still not retrieve any results when combined in a search query. This phenomenon results from the fact that our recommender algorithms only consider whether different users have used the same tags. The algorithms don't in involve the webpages that have been tagged. For example, two tags *programming* and *pizza* could be similar, because many users that preferred programming also preferred *pizza*. This does not necessarily entail that webpages exist that are tagged with both tags. Hence, query expansion even with a highly similar recommended tag can reduce the result set to zero results.

---

we excluded tags from our evaluation, for which no recommendations exist. The algorithms don't guarantee that recommendations exist for all tags; T(s) provides recommendations for more tags than T(p) and T(u), therefore the query result is higher on the average.

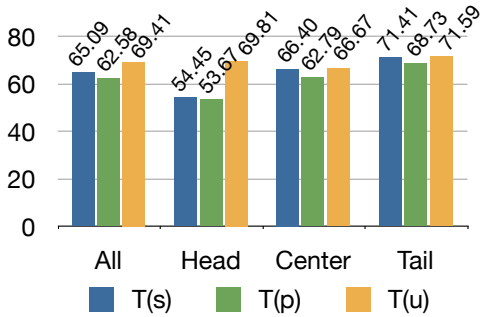

Figure 4.13: Query expansions without results

We measure the rate of query expansions that return zero results. Obviously, expansions without query results don't help users. Therefore, a lower rate of zero result query expansions indicates better domain learning support.

We found a fairly high average rate of query expansions that retrieve zero results across all of our algorithms, see Figure 4.13. For all algorithms, the rate is $>$ 53 percent. T(p) reduces the rate of expansions without results. This is particularly intense in the center of the popularity curve. T(u) significantly increases the number of zero result expansions in the head part of the popularity curve.

The results indicate that T(u) has a significantly negative effect on domain learning for popular queries. T(p) improves domain learning, particularly for queries of average popularity. The high zero result expansion rate for all of the algorithms also indicates that, in our context, algorithms which involve not only tags and users but also the resources for tag similarity calculation may give better results.

Conclusion

To conclude, we discuss our findings regarding the domain learning support of our algorithms.

All of our recommender algorithms seem to be appropriate for domain learning. We have shown that all algorithms make recommendations from the head, center, and tail of the curve. Furthermore, all algorithms support transition between different popularity levels. We found this by evaluating

the correlation between popularity of query tags and their recommendations. The ability to transition between tags from the whole range of the tag curve is crucial for domain learning. It indicates that all domains represented in the input data set are equally likely to be recommended. All algorithms can recommend tags from the head, center, and tail of the tag curve, independent from the popularity of the query tag. The independence of popularity of tags and recommendations is a key strength of our system for domain learning.

Filtering of results is a strength of recommenders in the context of exploratory search. It supports domain learning by helping users to find more specific or more unspecific tags and webpages. In other words, it helps to transition between expert and non-expert resources. We have shown that biasing towards unpopular tags has negative affects on the filtering. T(p) and T(u) perform better in this regard. We found this by comparing the lengths of the result sets between queries for a single tag and expanded queries for a tag and its recommendation. We found that our algorithms have only poor filtering results for unpopular tags by looking at the absolute numbers of results

The size of the dataset and the resulting data sparsity seems to be a problem for domain learning. Result retrieval and filtering are problematic for unpopular queries. We have shown that the result length for single tag queries dramatically drops between popular tags and less popular tags. Queries for unpopular tags retrieve only very few results. While our system provides good domain learning for popular queries, the quality is poor for unpopular queries. The poor query results don't depend on our algorithms; they result directly from the low size and sparsity of the input data set. However, the low result lengths affect the capabilities of our algorithms for domain learning. They clearly affect the necessity and quality of the result filtering of our algorithms. Our findings of the *mean result lengths* experiment indicate that a larger input dataset could improve domain learning particularly for unpopular tags. The dataset seems good enough for our algorithms for popular tags. For unpopular tags, our algorithms provide only poor domain learning support with the current input dataset.

Our collaborative filtering algorithms only include tags and users into the

similarity calculation. This can be a problem in the context of a search engine, because the algorithms don't consider the webpages that are retrieved. It could be beneficial in our context to consider the webpages from the input data for the tag similarity calculation. We measured the negative effects in the *zero query result rate* experiment. The findings suggest further experiments with algorithms that consider the webpages for the tag similarity calculation.

4.7 Conclusion

We presented an evaluation method for recommender algorithms in exploratory search engines. The method allows to assess the quality of user models generated with these algorithms. Our method combines a set of evaluation experiments that measure the navigability of recommendation networks and the effects in the search results. It measures the topology of the networks to evaluate how well recommendations provide navigation trails through the tag space which facilitate domain discovery and domain learning. Furthermore, it measures how navigation through the recommendations affect the search results.

The evaluation method focuses on non-accuracy measures, novelty and coverage, because these measures are more important than predictive accuracy in the context of exploratory search. Non-accuracy measures are more *fuzzy* to evaluate than predictive accuracy measures. They cannot be measured straightforward and must be seen in the context for which the recommendations are applied.

A popular strategy to include non-accuracy into the evaluation is to measure the user-perceived quality. However, this strategy requires user feedback and can be obtrusive for users. A major shortcoming of user-centric evaluation strategies is that they cannot evaluate algorithms before they are deployed in a running system. Furthermore, it is cumbersome to compare a larger number of algorithms, because all algorithms must be deployed in a running system for a certain time to collect user feedback.

A key strength of our method is that is can evaluate non-accuracy mea-

sures quick and with low effort – without requiring user feedback data. Our method solely relies on evaluating the structural features of pre-compiled recommendation networks and the properties of the underlying folksonomy. It allows to evaluate recommender algorithms before they are deployed in a running system. Because it requires no user feedback and is relatively simple, our method also allows to compare a larger number of algorithms with low effort.

We exemplarily applied our evaluation method to a portion of the user models that we generated with our user model extraction method (cf. Chapter 3): we evaluated how well the tag similarity modeled in our user models supports domain discovery and domain learning. Therefore, we assumed an exploratory tags search engine that recommends similar tags to a query. We compared our baseline algorithm with two alternatives which bias the recommendations towards popular or unpopular tags.

The evaluation results show that biasing recommendations towards popular tags provides the most advantages for domain learning and domain discovery: the algorithm T(p) is the winner for our context. We have shown that T(p) creates recommendation networks that a more navigable than T(s) and T(u). T(p) reduces the low navigability and coverage of the recommendation networks for unpopular tags. Furthermore, it provides the best results for the query result filtering. Biasing towards unpopular tags, T(u), reduces the recommendation quality for exploratory search compared to the baseline algorithm.

The results suggest that the quality of our algorithms is highly sensitive to the popularity of the search query tag. The results of our evaluation vary greatly between different parts of the popularity distribution curve. Depending on the query popularity, we found different or converse performances of our algorithms. We can state that our user models perform good for popular queries. Our experiments show that the navigability of the recommendation network is good for the head of the popularity distribution curve. Also the result set length and filtering rate is appropriate for popular queries. For the long tail of unpopular tags, all of our algorithms perform significantly worse. We found two reasons for this. First, the connectivity of our recommendation

networks is low in the tail of the popularity curve. This thwarts navigation through the network in these areas. We can improve the navigability in the tail by biasing recommendations towards popular tags, however, this does not completely avoid the problem. Second, our search engine retrieves too little results for unpopular queries. This issue is beyond our algorithms, but depends on our input data set. We clearly need to collect more data to provide a reasonable number of search results also for less popular queries. Our evaluation shows that a search engine that works with the current input set and the current user models is suitable for exploration of popular topic domains, but shows deficiencies for unpopular domains.

Our findings imply that input data are a crucial factor for the quality of an exploratory tag search engine. Clearly, a larger input data set could provide better query results; however, the pure quantity is not the only relevant issue with the input data. Our results for the head and center of the popularity curve show that already a medium-sized data set can provide good results. Another issue of the input data that affects the evaluation results is the *diversity* of topic domains covered in the input data. The search activity of a user is usually not restricted to specific topic domains, but ranges across numerous domains. To support exploration for search queries from a wide range of possible domains, user models must cover the whole domain range. Probably, a folksonomy of a single bookmarking system cannot provide the range of data necessary for this task. Our findings indicate that data from a single bookmarking systems are too sparse and too domain-specific for web search. Other studies confirm these findings [105, 134]. To achieve a better size and a better coverage of topic domains in the user models, and thus to overcome the key shortcomings that our evaluation exposed for our user modeling approach, we suggest to share user models and connect user models from various services on a web-scale. We believe that this can be achieved by a web-scale user modeling integration strategy that comprises user models from larger number of popular Social Web services. In the following chapter, we contribute a method for web-scale distributed user modeling to make this possible.

Chapter 5

Contribution 3: A user model integration method

In the following chapter, we contribute a user model integration method that addresses the challenges of web-scale user modeling. The goal of our method is to represent and publish user models for exploratory search engines on the Web. Our contribution comprises three aspects: (1) a generic user modeling architecture, (2) a user model scheme, and (3) a user model representation method.

(1) We contribute an *architecture* for distributed user modeling systems on a web-scale. The architecture takes into account the requirements of distributed personalized systems and the existing architecture and characteristics of the Social Web. Thus, it bridges the gap between the existing service-oriented Web architecture and a fully-fledged Semantic Web.

(2) We contribute a *user model scheme* that takes into consideration the characteristics of web-scale user modeling. The user model scheme aims at providing a reasonable abstraction of user properties. The goal is to make user models generic enough to be re-used by many services on the Web. At the same time, they should be specific enough to provide helpful information for popular analysis techniques. Our user model scheme models user interest, user knowledge, and user similarities.

(3) We contribute an RDF-based *user model representation method* that

allows for syntactic and semantic integration on the Web. The representation method applies Linked Data standards. It re-uses or maps to the widely adopted RDF ontologies SKOS, FOAF, and Dublin Core. The representation method is capable of integrating further vocabularies and is thus extensible.

The overall objective of our user modeling method is viable user modeling on a web-scale. This goal implies three sub-goals:

(1) User modeling should naturally integrate with the distributed and open architecture of the existing Web. The Web is an open world – new services can emerge and engage in a web-scale distributed system at any time. This entails three design principles: first, open standards to ensure interoperability; second, an extensible architecture to enable adaptation to new requirements; third, a highly scalable architecture to scale up when required.

(2) User models should model information that can be practically used by real-world services on the Web. This entails user models that are neither too generic to be used efficiently by existing services nor too specific to be shared across services. In addition, the user model content should reflect the requirements and characteristics of the current Web topology. For example, it should model social relationships between users, which are an important aspect of the Social Web.

(3) User models must be syntactically and semantically aligned. However, syntactic and semantic integration is just *one* prerequisite for shared data. Data must also be actually published by user modeling services. Although this not merely a technical issue but depends largely on business models (cf. [2]), technology can simplify and motivate data publishing and dissemination.

The remainder of this chapter is organized as follows. In Section 5.1 we describe our architecture for web-scale user modeling. In Section 5.2, we describe our user model scheme. In Section 5.3, we describe our user model representation method. We conclude with a discussion of our contribution in Section 5.4.

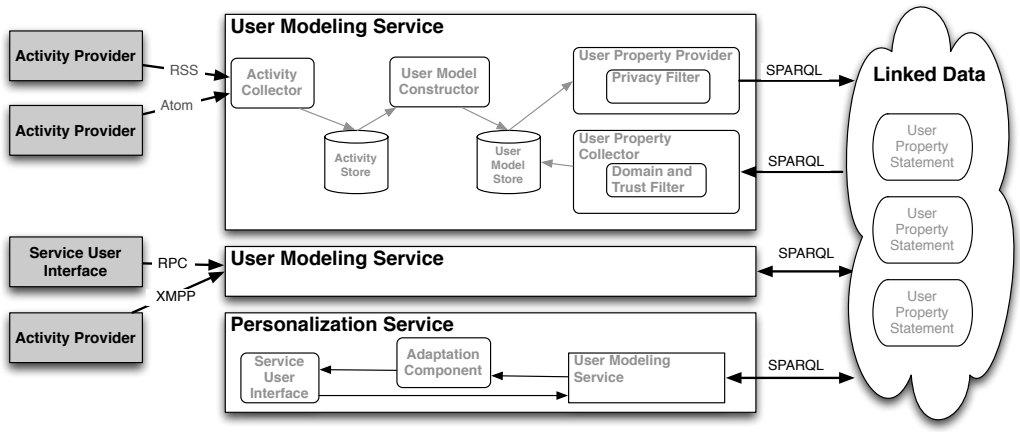

Figure 5.1: User modeling architecture

5.1 A user modeling architecture

In the following section, we describe our generic architecture for web-scale user modeling. The architecture provides a framework for flexible, scalable, extensible distributed user model integration. Furthermore, it incorporates existing Web services (and the technologies used by these services for data publishing: Atom, RSS, and XMPP[1], cf. [36]) and Semantic Web technologies. The main components of the architecture are *Activity Providers*, *User Modeling Services*, and *User Property Statements*, see Figure 5.1. The components exist independently from each other and are only loosely coupled. They are distributed on the Web and communicate over HTTP. All components are logical units only; each component may well be distributed on several physical devices, e.g., on a server cluster or in a cloud-based environment.

Activity Providers produce *Activity Data* – usage data and user generated content. For example, social bookmarking services publish data streams that represent folksonomies, social networking services publish social relations between people. Furthermore, also non-social services may provide Activity Data streams, e.g., logged usage data. Activity Data are collected by *User Modeling Services* and serve as the data basis for the user modeling process.

[1]http://xmpp.org/rfcs/rfc3920.html (last access 2011-04-22)

User Modeling Services produce higher-level knowledge of users – *User Property Statements*. Therefore, they collect Activity Data as well as existing User Property Statements and apply analysis techniques on these data to produce user models. The user models are then published as User Property Statements. User Modeling Services may optionally be integrated into a *Personalization Service*. Personalization Services are services which not only create user models, but also apply the models to adapt some information presented to users. User Modeling Services are independently working components. Thus, the number of User Modeling Services in a system is in principle not limited.

User Property Statements are statements about user characteristics which are produced by the User Modeling Services. They are the building blocks for the user models. For example, User Property Statements may describe a user‘s interest, knowledge, or social relations. Depending on the User Modeling Service, User Property Statements may vary in the topic domain. A set of User Property Statements forms the user model. A key feature of User Property Statements is that they can be interpreted independently from each other. This characteristic distinguishes User Property Statements from traditional user models. Technically, all User Property Statements are represented with RDF. Linked Data standards and existing vocabulary are largely applied. User Property Statements take the form of a *cloud* of interlinked statements on the Web which describe users. The User Property Statements cloud is published to the Web and is accessible through SPARQL queries.

We introduced the main components of our architecture – *Activity Providers*, *User Modeling Services*, and *User Property Statements*. Subsequently, we describe their function and internal structure in more detail.

5.1.1 Activity Providers

Activity Providers produce initial data on which all user modeling and personalization grounds. Activity Providers are part of the already existing Web. They are existing Web services, e.g., social bookmarking services, that produce data streams.

Activity Providers and User Modeling Services communicate via HTTP. User Modeling Services *subscribe* to Activity Providers to receive their activity stream. Usually, Activity Providers use the standard exchange formats for Web services, RSS, Atom or XMPP, for data publishing. In detail, two data publishing methods exist. In the first method, activity is published by Activity Providers through *feeds*. User Modeling Services subscribe to feeds, i.e., they *poll* published data in regular intervals, e.g., as Atom feeds. In the second method, Activity Providers *push* activity data to User Modeling Services over a push protocol, e.g., XMPP.

The actual content provided by the data stream varies depending on the kind of Activity Provider. Content streams relevant for user modeling can be provided by virtually all kinds of Web services. For example, social bookmarking services produce tagging activity. Their streams describe which users tag which resources with which tags at which time. Social networking services produce a variety of activity, for example, community affiliation, similarity relations between users, interest groups, or demographic information. Streams of location-based services may publish activity data that describe user location. Feed aggregators like Feedburner [2] or Yahoo Pipes [3] can act as intermediate Activity Providers. They combine and possibly filter streams of several Web Services and produce aggregated activity streams.

5.1.2 Service User Interfaces

Service User Interfaces are an alternative provider of activity data. They create traditional usage data that is logged from user interaction. Usage logging is usually tightly integrated with the application logic. Thus, these feedback data are not publicly published as a feed. Instead, a user interface of a service may directly push the data to the service‘s own logging component via remote procedure calls (RPC) over HTTP.

[2] http://www.feedburner.com (last access 2011-04-22)
[3] http://pipes.yahoo.com/ (last access 2011-04-22)

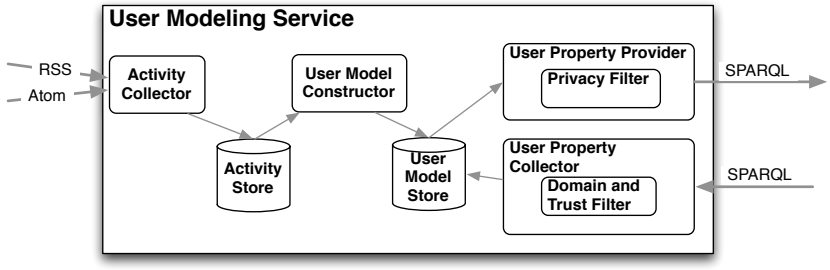

Figure 5.2: User Modeling Service

5.1.3 User Modeling Services

User Modeling Services are the key component of our architecture. They contain the logic that converts activity data into User Property Statements. User Modeling Services consume activity data streams as input and produce User Property Statements as output. Typically, User Modeling Services are either (1) domain-specific, (2) user-specific, or (3) purpose-specific. This categorization is non-technical; rather it corresponds to the underlying business models.

(1) *Domain-specific* User Modeling Services collect and process activity data of a specific topic domain. For example, a User Modeling Service may collect activity from a scientific social reference manager, like CiteULike, to produce User Properties related to users‘ *scientific* knowledge.

(1) *Purpose-specific* User Modeling Services collect and process activity data to serve a specific purpose. The activity data relevant to serve a certain purpose may involve various topic domains. For example, the purpose of a User Modeling Service may be to produce User Property Statements for a recommender that suggests products to a user in an online shop.

(3) *User-specific* User Modeling Services collect and process data concerning a specific user or user group, independently from domain and purpose. For example, a user-specific User Modeling Service may produce User Property Statements for a single user based on the overall activity of this user aggregated from all services used by the user.

The idea of the presented architecture is, that User Modeling Services on the Web exist independently from each other, similar to existing Web

services. In the following, we describe the internals of User Modeling Services, see Figure 5.2.

Activity Collector

Each User Modeling Service contains zero or more *Activity Collectors*. Each Activity Collector subscribes to an Activity Provider. One User Modeling Service can implement several Activity Collectors to collect activity from several Providers. It receives and parses data streams from the Activity Provider to extract relevant information from the input stream and store it into an Activity Store. Data may be stored in any format, depending on the requirements of the application. An Activity Collector can also receive usage data provided by Service User Interfaces.

Activity Store

The *Activity Store* caches collected Activity Data in a local data store. The Activity Store makes the data available for the User Model Constructor. The purpose of local caching of Activity Data in the User Model Service store is two-fold:

First, data access over feeds is usually slow and limited. Statistical analysis techniques on the activity data usually require fast and frequent access on the data. An acceptable access speed is typically not realistic with remote data querying over the Web. Furthermore, access to data published as feeds by Activity Providers is usually limited. For example, a feed usually contains only a small portion of the overall data available. To retrieve more data, the feed subscriber has to poll the data iteratively in small portions. Feed APIs usually only allow a limited number of queries per subscriber in a given time span.

Second, data analysis techniques require data in a special format, depending on the analysis technique or software that is used. Usually, feeds provide data in a form that must be cleaned or parsed to comply with that format. For example, a feed from a social bookmarking service may implicitly express preferences of users for tags. However, usually, this information has to be

converted for the usage in a recommender software, which may require a list of user-tag pairs.

User Model Constructor

The *User Model Constructor* is a data analysis component that creates or enhances user models. The User Model Constructor usually implements machine learning techniques, e.g., collaborative filtering algorithms. The User Model Constructor has two possible input sources – existing user models in the *User Model Store* or the data in the Activity Store. Depending on the application context, the User Model Constructor uses one of the two sources, or both. The output of the user model construction process is a user model.

User Model Store

The *User Model Store* is a local data store that stores the user models created by the User Model Constructor. The user models can be modeled and stored in any arbitrary format and scheme, depending on the requirements of the application context.

User Property Provider

The *User Property Provider* publishes the output of the User Modeling Service, the user models, as *User Property Statements* on the Web over a public interface. Technically, User Property Statements are small groups of RDF statements that encode user models according to Linked Data standards. The User Property Provider involves two steps: *data mapping* and *data publishing*. The data mapping step constructs RDF-based User Property Statements from the internally used user model. Data elements of the internal User Model Store are therefore mapped to an appropriate RDF vocabulary. The data publishing step makes the created RDF statements available and query-able on the Web. Technically, the User Property Provider is a SPARQL endpoint that can be queried over the Web.

Privacy Filter

The *Privacy Filter* is part of the User Property Provider. It decides which information from the internal user model is made available to the User Property Provider for publication. The internally used user model may contain information that should not be made public for some reason. For example, a service provider may not want to share sensitive information about a user that need to be kept private for privacy protection. The privacy filter applies internal rules to the User Property Provider, so that User Property Statements are generated for non-private data only.

User Property Collector

The *User Property Collector* retrieves and caches User Property Statements that are already available on the Web. It converts the User Property Statements to the internal user model format and stores them to the local data store. The purpose of caching remote User Property Statements is *performance improvement*. Linked Data are distributed across servers on the Web. Conceptually, it is very convenient to be able to query data independently from their physical location. Practically, however, this method for querying raises performance issues. As discussed above, machine learning techniques often require fast and frequent data access, which is not feasible with distributed queries. Therefore, relevant data are cached in the local User Model Store in an appropriate format. Technically, this component is a SPARQL interface that dereferences URLs of RDF statements.

Domain and Trust Filter

The *Domain and Trust Filter* is a part of the User Property Collector. Its purpose is to decide which existing User Property statements to consider for data analysis and cache in the local User Model Store. This decision can be made according to two criteria – *domain* and *trust*.

(1) The *domain* of User Properties is defined by the content that is modeled. The content may or may not be relevant for the application, depending

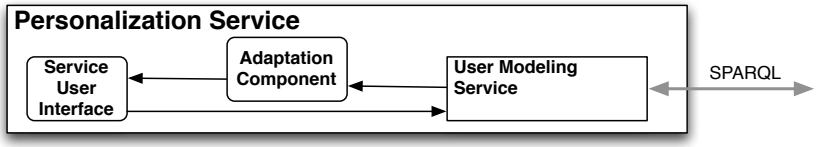

Figure 5.3: Personalization Service

on the application domain. For example, location information may not be relevant for a user modeling service of a book store.

(2) *Trust* in User Properties is defined by the origin of the data. In an open environment like the Web, every service can publish User Property Statements of varying quality. User Modeling Services may want to rely only on User Property Statements of certain trusted Services.

5.1.4 Personalization Service

Personalization Services are optional components of our architecture that *contain* User Model Services, see Figure 5.3. They extend the functionality of User Modeling Services by an *Adaptation Component* that personalizes content based on the user model, and a *Service User Interface* that delivers the adapted information to the user. Typically, Personalization Services also collect usage data from the Service User Interface.

Adaptation Component

The *Adaptation Component* is part of the Personalization Service. It adapts content, structure, or form of the information delivered to the Service User Interface. The adaption component can rely on arbitrary data formats for input and output. Hence, established methods and software can be retained by Personalization Services.

5.1.5 User Property Statements

User Property Statements form the user models that are shared on the Web. The idea of User Property Statements is to decompose user models to independently interpretable small units of user model information. A User

Modeling Service does not publish a complete user model that must be integrated by another service. Instead, it publishes a model as User Property Statements. Further services can integrate all of these User Property Statements or only portions of them. Individual statements can be modeled in different schemas. Hence, a user model expressed by User Property Statements may be composed of a number of schemas. User Property Statements address two problems of the traditional user modeling approach:

(1) Traditional user models require a defined schema or vocabulary to model the user information. This is problematic for two reasons. First, these schemas must be powerful enough to cover all desired information that may be required by application. This is hard to achieve when requirements change in a dynamic context like the Web. Second, schemas must be simple enough to be implemented by a large number of applications with a reasonable amount of effort. A compromise of the two requirements for user model schemas has shortcomings on the one or the other side. Therefore, user model schemas which completely fulfill the demands of services across very limited domains seems not feasible at present. User Property Statements overcome this problem, because an existing simple schema can be extended by other existing schemas as required.

(2) User Modeling Services are usually specific to a certain application context. They create specific user models for a certain problem, domain, or user. Hardly ever do other services re-use models of other services completely. This is because they usually address distinct problems, although the problems might overlap in some degree. For example, an educational system may model a user‘s programming skills as well as official certificates gained at a university. Another service that personalizes search results according to the user‘s programming skills does not need information about official certificates in its user model, neither does it support the vocabulary to describe this information. With the traditional user modeling approach, developers of the latter service would have to deal with the complete user model schema of the first service, even if only small parts of the schema are to be supported. Independent User Property Statements make it easier to integrate only the portions of user model schemas required by a service, because services do not

have to parse and interpret complete user models. Instead, services consume only relevant User Property Statements.

To sum up, we presented an abstract architecture for web-scale user modeling. It allows for scalable and extensible user modeling. All components are only loosely coupled, no centralized service is necessary to mediate between components; the system can hence scale effortless. The architecture also considers characteristics of existing services on the Web: it integrates established standards for data publishing, RSS, Atom and XMPP. At the same time, the architecture applies Semantic Web technologies for data publishing. User models are published as RDF statements in the form of User Property Statements. User Property Statements decompose user models into small units which can be interpreted independently from each other. Hence, the architecture acts as a link between Social Web services and a Semantic Web.

5.2 A user model scheme

In the following section, we describe our lightweight and extensible user model scheme for the Web. First, we define the overall approach that we follow. In general, two divergent kinds of user model schemes exist: (1) generic models and (2) application-specific models.

(1) *Generic* user model schemes model users at a high abstraction level. Here, the reference to the domain of the application is rather low. For example, user models that represent individual traits or, tasks, goals, or opinions of users are rather generic and have only weak linkage to the application domain. The advantage of generic user models is, that they can be re-used more easily across domains. The downside is, however, that it is difficult to interpret generic models to obtain concrete directives for adaptation. Machine learning techniques cannot process highly generic user model content; at least, elaborate reasoning is required to process generic user models [219].

(2) *Application-specific* user model schemes model content so that it is customized for the application‘s purpose or technology. Application-specific models are typically close to the raw data which are being logged. For exam-

ple, a search engine may store queries of users. The advantage of application-specific user models is that they are pragmatic and usually require no complex user modeling schema. Since application-specific modeling is often optimized on specific analysis software or techniques, it requires no intermediate interpretation steps and allows for straight-forward statistical analysis. The downside of application-specific user models is that it is hard to share models across services, analysis techniques, and domains.

Our method pursues a *semi-generic* user modeling approach – our user model scheme has an *intermediate* abstraction level. The approach is a compromise to meet the requirements of user modeling on a web-scale. It seeks to provide content that is practically useful for established analysis techniques. At the same time, it seeks to be shareable across domains, purposes, and techniques. To keep the model on a level that is generic enough to share it between many services, we exclude temporal information from the *content level*. For example, we don't model duration or time of appearance of a user characteristic on the content level. Instead, we shift these information to a *metadata level*. Hence, we distinguish between information that is modeled within the user model and metadata that describe the user model itself, or parts of it. We define three user properties that are relevant for web-scale user modeling: *user interest*, *user knowledge*, and *user similarity*. On the content level, we model these user characteristics. On a metadata level, we model properties of the user model: *creation time* and *provenance*. A key advantage of separating content and metadata is, that the user model scheme becomes more generic and thus more easy to re-use across different application contexts.

In the following sections, we describe the components of our user model scheme on content and on metadata level. First, we describe the properties that we model on the content level: user interest (Section 5.2.1), user knowledge (Section 5.2.2), and user similarity (Section 5.2.3). Subsequently, we describe what we model on the metadata level (Section 5.2.4).

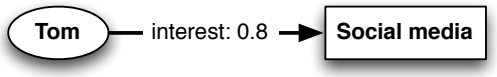

Figure 5.4: User interest example

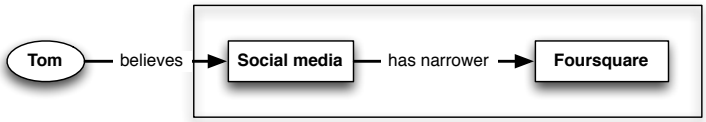

Figure 5.5: User knowledge example

5.2.1 User interest

User interest defines topics in which the user is assumed to be interested in. It relates users with topics of interest. A topic can be a term or a set of terms, e.g., a tag from a social bookmarking system. The interest relation has a weighting. The weight factor indicates how strong the assumed interest of the user in the topic is. For example, a user Tom may be interested in the topic *social media* with a weight of 0.8. Figure 5.4 schematically shows weighted user interest in a topic.

5.2.2 User knowledge

User knowledge models *what a user knows*. User knowledge is subjective – it may well contain misconceptions. Therefore, user knowledge could be equivalently referred to as *user belief*. We stick, however, to the term knowledge, since is fairly established in related literature. We model knowledge as a set of believed relations between concepts. These relations can be either associative or hierarchical. Associative relations are undirected; hierarchical relations are directed. For example, Tom may believe that *foursquare* is a subclass of *social media*. The sum of believed relations model a user‘s knowledge. Figure 5.5 schematically shows user knowledge.

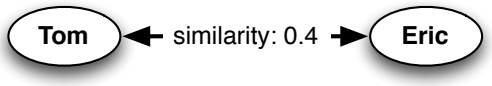

Figure 5.6: User similarity example

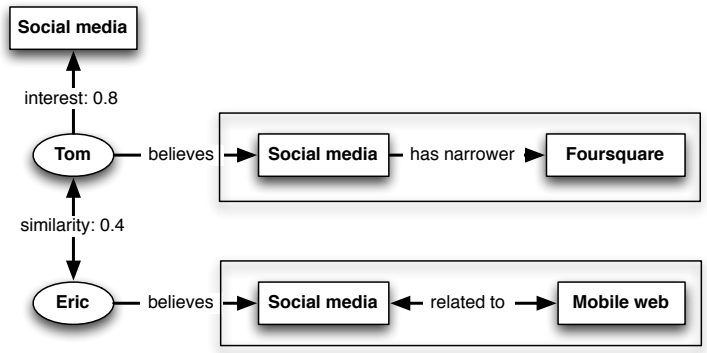

Figure 5.7: Overall user model content example

5.2.3 User similarity

User similarity models relations of a user to other users on the Web. User similarity relations are undirected and weighted. The weight factor indicates how strong the similarity between two users is. Usually, similarity relations between users base on similarity of interest, knowledge, or behavior. Hence, they are implicit semantic relations – they represent affiliation of a user to an (ad-hoc) community of interest. For example, a service may identify a group of users based on their interest in similar topics or some kind of similar behavior in the past. Figure 5.6 shows a user similarity relation.

We defined three user characteristics that we model – user interest, user knowledge, and user similarity. The sum of these three characteristics defines the scope of the content level of our user model. Figure 5.7 schematically shows an overall example of our user model scheme on the content level.

5.2.4 Metadata

The described user model content excludes two important dimensions: *time* and *provenance*. We exclude these dimensions from the content level and

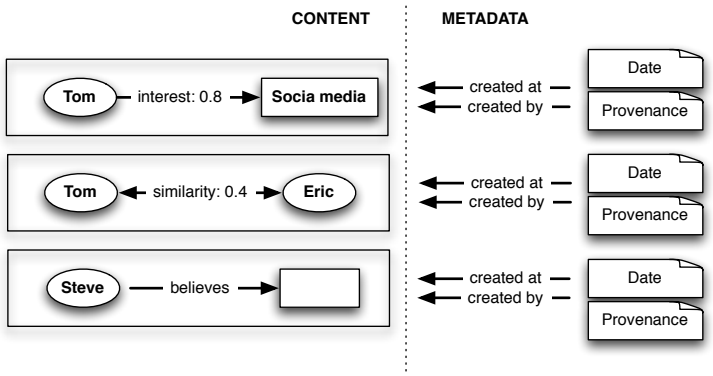

Figure 5.8: User Property Statements with metadata

instead model temporal and provenance information at a *metadata* level. The benefit is a more lightweight and generic user model scheme. To describe individual parts of the user model content with metadata, we decompose the user model into small units that can individually annotated with metadata, see Figure 5.8. The individual units are User Property Statements. In the following, we describe the function and implications of time and provenance modeling at the metadata level in detail.

Creation time models *when* a User Property Statement was published. It does not provide temporal information of cognitive characteristics; for example, it does not model appearance or duration of interest of a user in a topic. Instead, our user modeling approach takes a more generic view on cognitive characteristics. We assume that a user property is valid at the time when it is published. For example, a service may publish a User Interest Statement that describes weighted interest. Later, the service may withdraw the User Property Statement and publish a new User Property Statement with another interest weight, because new data became available for analysis meanwhile. Thus, User Property Statements are considered to represent the currently valid state of a user.

Provenance models *who* published a User Property Statement. It provides information about the service that created a User Property Statement. It helps consuming services to decide how trustworthy the data are. Consuming services may want to consider only information from certain, trusted, data

publishers. Or, they may want to decide how to interpret contradicting information from two services. Provenance metadata do, however, not model how to interpret the provenance information. For example, they does not imply a trust value for the User Property Statement.

Excluding time and provenance from the user model content has important implications for the user model scheme. The metadata describe when User Property Statements were created and who created them, but they do not give information on how to interpret this information. For example, time metadata do not describe how long a User Property Statement is valid, or when it expires. Accordingly, provenance metadata don't model whether a User Property Statement is trustworthy. Services that *consume* User Property Statements must implement application logic that decides how to interpret metadata. We consider this approach more suitable for an open environment like the Web, because it keeps the user model more generic and flexible. Services from different domains can interpret the metadata differently, according to their specific needs. For example, one service may want to consider all User Property Statements that are younger than one month; another service may consider all available User Property Statements and only downgrade the influence of older ones. The approach allows new services that want to interpret information differently to be integrated in the system more easily.

To sum up, we presented a semi-generic user model scheme for web-scale user modeling. We divide the user model scheme into a content level and a metadata level. On the content level, we model three user properties: user interest, user knowledge, and user similarity. User interest is a set of weighted relations between users and topics. User knowledge is a set of believed associative or hierarchical relations between topics. User similarity is a set of weighted relations that define a similarity value between two users. On the metadata level, we model time and provenance of parts of the user model. Therefore, we decompose the user model into small units, which we refer to as User Property Statements.

5.3 A user model representation method

In the following section, we describe our method to represent the described user model scheme in RDF. The method overcomes important problems of user modeling with RDF and existing RDF vocabularies. We re-use or map to widely deployed vocabularies where possible and introduce a new ontology to overcome constraints of existing vocabularies. In Section 5.3.1, we discuss key problems of user model representation with RDF. In Section 5.3.2, we review patterns to overcome these constraints. In Section 5.3.3, we describe our method to represent user models as User Property Statements.

5.3.1 Representation problems

Semantic integration of shared user models on the Web calls out for the use of widely adopted vocabularies like SKOS, FOAF, and Dublin Core for user model representation. However, these vocabularies have serious limitations for user modeling. In the following, we illustrate why SKOS, FOAF, and Dublin Core are not per se powerful enough to serve as vocabulary for user model representation on the Web.

Generally, user models model complex relations between an arbitrary number of things. The discussed RDF vocabularies do not provide facilities to model these complex structures. The basic cause for the problem is the RDF data model, which is limited in that is allows only to model binary relations [4] [64]. We demonstrate this problem with two examples. The examples illustrate two patterns that are problematic to model with RDF, (1) *qualified relations* and (2) *n-ary relations*.

(1) Our user model scheme requires to model the following kind of facts: „Tom is interested in *geofencing* with a weight of 0.8“. The fact is a *qualified relation*. Figure 5.9 schematically shows the relations that must be expressed to model the fact. The challenge here is to *specify* a RDF property instance, *interest*, thats link two individuals, *Tom* and *geofencing*. Intuitively, FOAF seems to be a suitable vocabulary to model interest: its `foaf:interest` prop-

[4] http://www.w3.org/TR/swbp-n-aryRelations/, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ (last access 2011-04-22)

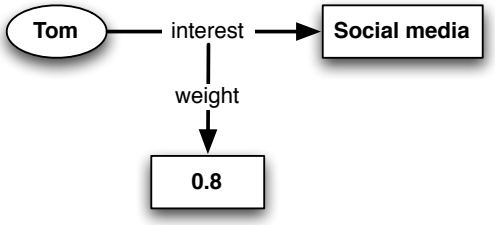

Figure 5.9: Qualified relation example

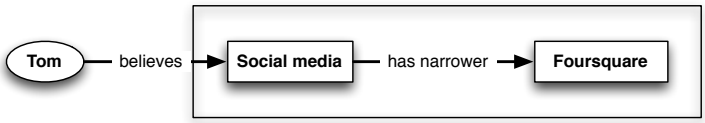

Figure 5.10: N-ary relation example

erty allows to model an interest relation between Tom and *geofencing*. The challenge is to specify the `foaf:interest` property with a weighting. The RDF data model does not support annotation of property instances. Hence, it is not per se possible to qualify the `foaf:interest` property instance with a weight value.

(2) Our user model scheme requires to model the following kind of facts: „Tom beliefs that *foursquare* is a subclass of *social media*. This fact is an *n-ary relation*. There is a statement that describes the binary relation between *foursquare* and *social media* and there is a property describing a limitation of the statement. Figure 5.10 schematically shows the relations that must be expressed to model the fact. Intuitively, two existing vocabularies seem appropriate to model the relation properties involved in this fact. First, the hierarchical semantic relation between *foursquare* and *social media* can be described with the `skos:narrower` property of the SKOS ontology. Second, a belief relation can be modeled with the `cco:belief` property of the Cognitive Characteristics Ontology. The challenge of this example is to represent a relation between more than two individuals. We want to model a belief relation between *Tom* and the statement that *foursquare* is a subclass of *social media*. The RDF data model does not support these n-ary relations. It is not per se possible to model a relations between statements; RDF statements

cannot have another statement as an object to model the intended meaning.

The two given examples illustrate structural patterns that must be represented in our user model scheme. The challenge is to represent these structures inside the RDF data model and to apply existing vocabularies.

5.3.2 Representation patterns

In the following section, we describe four representation patterns that address the challenges of qualified and n-ary relation modeling with RDF: *named graphs*, *property customization*, *reification*, and *class-based relation modeling*.

Named graphs

Named graphs are an extension to the RDF data model [47]. The extension is backwards compatible with the original RDF model to ensure easy integration of named graphs into existing applications. Named Graphs are thus a minimal step approach to extend the RDF model with little effort to encourage quick deployment on the Web. The initial motivation of named graphs was to add provenance information to RDF statements. Provenance can act as an indicator to deduce trust. The Web assumes an open world – everyone can publish anything. Data consumers have to decide which information are trustable. Provenance tracking is one approach for a trust layer on the Semantic Web.

The idea of named graphs bases on *quads*. Quads extend RDF triples with a fourth element which carries arbitrary information. The intention of quads is to add more information to triples. The key problem of quads is semantic arbitrariness. The semantics of the additional element in the RDF statement is beyond the RDF data model. Therefore, existing RDF infrastructure cannot per se semantically interpret quads. *Named Graphs* seek to overcome this problem by specifying both syntax and semantics of the extension. The Named Graph data model defines the formal semantics of the additional element as *context*. By defining the context of an RDF statements, Named Graphs allows to add meta information to RDF statements and to express relations between sets of RDF statements.

Listing 5.1: Named Graph example

```
ex:TomGraph {
    ex:social_media skos:narrower ex:foursquare
}
ex:EricGraph {
    ex:social_media skos:narrower ex:facebook
}
```

In the context of user model representation, Named Graphs can be used for scoping assertions and logic [5]. For example, a statement about a relation between two concepts can be restricted to individual users. Listing 5.1 shows individual Named Graphs of two users. Both contain a relation between concepts. The statements are semantically restricted to the Named Graphs of the individual users.

Named graphs can only partly overcome the problems of user model representation discussed above for two reasons. First, the formal semantics as defined in [47] are not commonly agreed on in real-world [6]. Existing RDF tools and reasoners do not agree on the formal semantics of Named Graphs. Although Named Graphs are backwards compatible, they remain outside the RDF data model with the consequence of not being agreed on by the RDF infrastructure. Second, Named Graphs do not provide a generic solution for qualified and n-ary relations. Rather, they are restricted to model provenance information. This does not solve the problems of all cases of n-ary and qualified relations.

Property customization

Listing 5.2: Property customization example

```
ex:social_media ex:Tom_narrower ex:foursquare
ex:social_media ex:Eric_narrower ex:facebook
```

[5] http://www.ninebynine.org/RDFNotes/UsingContextsWithRDF.html (last access 2011-04-22)

[6] http://www.semanticoverflow.com/questions/757/which-owl-reasoners-understand-named-graphs (last access 2011-04-22)

Property customization is a hands-on approach to n-ary relations that does not extend the RDF data model. The idea of property customization is to define more specific RDF properties as sub-properties of existing vocabulary. Listing 5.2 shows how relations between concepts are semantically restricted to individual users with property customization. The properties `ex:Tom_narrower` and `ex:Eric_narrower` are sub properties of `skos:narrower`. In this example, properties are customized for each user, to restrict the scope of statements to individual users.

Property customization is easy to implement and has therefore been adopted by some systems on the Web. The idea of property customization is also used by GUMO [101], where predicates of statements, properties, are replaced by User Model Dimensions (cf. Chapter 2). The approach is, however, rather naive and does not scale well. Although semantics are formally defined, it is hardly feasible to make reasonable queries over a larger number of customized properties. For each condition, a new property has to be defined. Hence, the number of properties grows very fast when properties are scoped to users, time, and other conditions. This fundamental problem of property customization has already been stated in [101].

RDF reification

RDF reification defines a mechanism to describe a RDF statement using a built-in RDF vocabulary. A description of a statement using this vocabulary is called a *reification of the statement* [7]. The RDF reification vocabulary contains the class `rdf:Statement` and three properties: `rdf:subject`, `rdf:predicate`, and `rdf:object`. The vocabulary allows to model a RDF statement with four reification statements. Therefore, the original statement is assigned a URI and the statement is then described with the reification properties. The resulting representation of a reified statement is *class-centric* – the original statement becomes an instance of the class `rdf:Statement`. As a result, reification allows to add arbitrary additional information to a statement. For example, RDF reification can record the provenance or creation time infor-

[7] http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification (last access 2011-04-22)

mation of statements. Listing 5.3 shows a statement in its original and its reified form with additional weight information.

Listing 5.3: RDF reification example

```
# Original statement
ex:Tom foaf:interest ex:social_media .

# Reified form
_:stmt a rdf:Statement ;
    rdf:subject ex:Tom ;
    rdf:predicate foaf:interest ;
    rdf:object ex:social_media ;
    wo:weight 0.8 .
```

RDF reification must be used carefully. The semantics of the RDF reification mechanism are not as intuitively perceived: RDF reification has no formally defined semantic relation between the original statement and its reification form. This implies that „RDF Reification does not assert the original triple, therefore one cannot infer the statement form from the reified form [, hence] asserting the reification is not the same as asserting the original statement, and neither implies the other“ [8]. For example, a RDF reasoner that is given the reified form of the statement from the above example cannot assert its original shortcut form.

In practice, some applications in fact assume a semantic correlation of a statement and its reified form. However, these semantics are outside the RDF data model, i.e., it is not what the RDF reification mechanism defines. Other applications and reasoners possibly do not interpret reification statements as intended.

Property reification

Property reification addresses the problem of missing semantics on the content-level of reified statements, which is the key weakness of the RDF reification mechanism. Property reification is a generic approach to solve the problems of qualified and n-ary relations in user modeling. Property reification has

[8] http://esw.w3.org/PropertyReificationVocabulary (last access 2011-04-22)

two benefits. First, it allows to model qualified and n-nary relations with formally clear semantics. Second, it provides a solution to extend existing vocabulary with capabilities for qualified and n-ary relation modeling. In the following, we describe the property reification process, which involves two steps: *class-centric vocabulary re-definition*, and *reification mapping*.

Listing 5.4: Property reification example

```
# Shortcut form
ex:Tom foaf:interest ex:social_media .

# Reified form
_:stmt a ex:ExtendedInterest ;
    ex:relationType foaf:interest ;
    ex:user ex:Tom ;
    ex:topic ex:social_media ;
    wo:weight 0.8 .
```

In a first step, existing vocabulary is adapted – *re-defined* – to meet the requirements of property reification. Most popular RDF vocabularies, like FOAF, SKOS, and Dublin Core, are *property-centric*: the vocabularies use RDF properties to model relations between things. In RDF, the property instances cannot be further specified. Instances of RDF classes, however, can be specified easily by adding further properties to the instance. Therefore, relevant properties are re-defined as classes. RDF classes must be defined for all properties of an existing vocabulary that are to be reified. Listing 5.4 shows a fact in its shortcut form and in its reified form. The property `foaf :interest` is re-defined as a class `ex:ExtendedInterest` in order to specify the interest with date information.

Listing 5.5: Property reification mapping example

```
# Re-defininition of foaf:interest property as ex:
   ExtendedInterest class
ex:ExtendedInterest a rdfs:Class .
ex:subj a rdf:Property .
```

```
ex:topic a rdf:Property.
ex:relation_type a rdf:Property .

# Reification mapping
_:InterestReification a prv:PropertyReification ;
    prv:reification_class ex:ExtendedInterest ;
    prv:shortcut foaf:interest ;
    prv:shortcut_property ex:relation_type ;
    prv:subject_property ex:user  ;
    prv:object_property ex:topic .
```

In a second step, the reified statements are semantically *mapped* to their shortcut forms with a reification vocabulary. One best-practice on the Semantic Web is to re-use existing vocabulary where possible. A drawback of the vocabulary re-definition – i.e., the re-definition of a existing RDF property as a new RDF class – is, that the result of this process is formally a new vocabulary. The classes defined during the re-definition are semantically not related to the original properties. A connection between both has to be created to semantically map the newly-created classes to existing properties. This can be achieved with a *property reification vocabulary* for semantic mapping between shortcut and reified expression [9]. A property reification vocabulary defines at ontology-level (as opposed to instance-level) how the reified – *class-centric* – form of a statement semantically relates to its original form. Listing 5.5 shows a reification mapping between the two forms of the statement from the example above.

The example applies the *Property Reification Vocabulary* [10]. The key element of the vocabulary is the class `PropertyReification`. It allows to describe the relations of a property reification. Therefore, instances of class `PropertyReification` have the properties `reification_class`, `shortcut`, `shortcut_property`, `subject_property`, and `object_property`.

The property `reification_class` relates to the class which re-defines the original property as a class, `ex:ExtendedInterest` in the example above. The property shortcut relates to the property of the original shortcut statement,

[9] http://esw.w3.org/PropertyReificationVocabulary (last access 2011-04-22)

[10] http://smiy.sourceforge.net/prv/spec/propertyreification.html (last access 2011-04-22)

`foaf:interest` in the example. `shortcut_property` relates to the property of the reification class (`ex:ExtendedInterest`) that defines the shortcut property of the relation, `ex:relation_type` in the example. `subject_property` and `object_property` relate to the subject and object of the original shortcut form, `ex:user` and `ex:topic` in the example above.

We illustrated how property reification allows to extend RDF statements by defining a class-based – reified – form of RDF properties. Property reification ensures semantic coherence between both forms by mapping them with a property reification vocabulary.

To sum up, we can state that Named Graphs, property re-definition, and RDF reification do not, or only partly, solve our problems of user model representation. We explained why property reification is the most viably approach to overcome the representation problems for RDF-based user model representation: n-ary and qualified relation modeling. Property reification allows to extend existing vocabulary with more powerful class-centric form. Thereby, it applies reification vocabulary to maintain semantic coherence between the original and the extended vocabulary.

5.3.3 User Property Statements

In the following section, we describe our method for user model representation. User models are represented as a collection of *User Property Statements*. Conceptually, User Property Statements have two important features: (1) they can be interpreted independently from each other, and (2) they use a non-proprietary, syntactically and semantically shared format. Technically, these conceptual features of User Property Statements are realized with RDF vocabulary and Linked Data standards. User Property Statements are small groups of RDF statements that model individual user properties. The RDF statements use, combine, and extend existing RDF vocabularies in large parts. All User Property Statements are published on the Web and can be queried with SPARQL.

One advantage of RDF for content representation is that any number of

vocabularies can be combined individually in a single user model. Consuming services do not necessarily have to be able to interpret all vocabularies used in a user model to interpret parts of the model. Rather, services can pick out content required for their needs. For example, a user modeling service may publish user models with the RDF vocabularies in CCO and FOAF. A consuming service may only support FOAF. This service can then simply drop information modeled in CCO and still interpret the parts of the user model modeled in FOAF. Hence, a user modeling service can provide user models for various topic domains using an arbitrary number of vocabularies. Developers of consuming services can decide which parts of the user model their service must support to retrieve reasonable information. This imposes little restrictions to service providers and keeps the overall system flexible.

Our user model representation approach aims at fulfilling two goals. First, existing vocabulary should be possibly re-used to ensure semantic integration. A key challenge here is that existing vocabularies often do not allow for modeling of qualified or n-ary relations. Second, newly created vocabularies should remain extensible for possible later enhancements. The two objectives can be achieved by property reification, which applies class-centric modeling and semantically maps classes of newly created vocabulary to existing vocabulary where possible.

According to the user properties defined in the user model scheme – interest, knowledge, and user similarity – we distinguish between three types of User Property Statements: *User Interest Statements*, *User Knowledge Statements*, and *User Similarity Statements*. In the following, we describe how the three types of User Property Statements are represented in RDF.

User Interest Statements

Listing 5.6: User Interest Statement

```
ex:AUserPropertyStatement a cco:CognitiveCharacteristic ;
    cco:agent ex:Tom ;
    cco:topic ex:social_media ;
```

```
    cco:characteristic cco:interest ;
    wo:weight [
        a wo:Weight ;
        wo:weight_value 0.8 ;
        wo:scale ex:AScale
        ] .
```

A *User Interest Statement* (Listing 5.6) models weighted interest of a user in a topic. We apply the CCO ontology and the Weighting Ontology [11]. Each User Interest Statement is an instance of the class `cco:Cognitive Characteristics`.

Listing 5.7: Weighting scale

```
ex:AScale a wo:Scale ;
    wo:min_weight 0.0 ;
    wo:max_weight 1.0 ;
    wo:step_size 0.1 .
```

Each instance has the following properties to describe the cognitive characteristic *interest*: `cco:agent`, `cco:topic`, `cco:characteristic`, and `wo:weight`. The `cco:agent` property defines the described user, an instance of `foaf:Agent`. The `cco:topic` defines the topic of interest, an instance of `owl:Thing`. The `cco:characteristics` defines the type of cognitive characteristic, `cco:interest`. The `wo:weight` defines the weight of the interest, an instance of `wo:Weight`, on a weighting scale used by the service. Each User Modeling Service can publish an instance of `wo:Scale` to describe the weighting scale used, see Listing 5.7. This scale must be published only once for each User Modeling Service. The classes and properties for the weighting are defined in the external Weighting Ontology. This vocabulary aims at modeling weightings and their referenced scales.

Listing 5.8: Property reification

```
ex:InterestReification a prv:PorpertyReification ;
    prv:shortcut foaf:interest ;
    prv:reification_class cco:CognitiveCharacteristic ;
```

[11] http://smiy.sourceforge.net/wo/spec/weightingontology.html (last access 2011-04-22)

```
    prv:shortcut_property cco:interest ;
    prv:subject_property cco:agent ;
    prv:object_property cco:topic .
```

An instance of the class `prv:PropertyReification` from the Property Reification Vocabulary maps the `cco:interest` property used in the User Property Statement to its shortcut form, the `foaf:interest` shortcut property, which is more commonly used on the Web (see Listing 5.8). This enables a larger number of services to consume the published interest information. This property reification must be published only once for each User Modeling Service.

User Knowledge Statements

Listing 5.9: User Knowledge Statement

```
ex:AUserPropertyStatement a cco:CognitiveCharacteristic ;
    cco:agent ex:Tom ;
    cco:topic [
        a cr:ConceptRelation ;
        cr:subj ex:social_media ;
        cr:relationType skos:narrower;
        cr:obj ex:foursquare
        ];
    cco:characteristic cco:belief.
```

A *User Knowledge Statement* (see Listing 5.9) models the belief of a user that two concepts are related in a certain way. Each User Property Statement is an instance of the class `cco:CognitiveCharacteristic` from CCO. The properties are identical to those of the User Interest Statement. The `cco:characteristics` defines the type of cognitive characteristic: `cco:belief`. The `cco:topic` defines the believed concept relation. The SKOS vocabulary allows to model relations. However, we already discussed its constraints in modeling detailed relations. SKOS does not allow to model more extensive, weighted, relations. Therefore, we present our *Relation Ontology* to model weighted relations between things.

Listing 5.10: The Relation Ontology

```
cr:  rdf:type owl:Ontology ;
      dc:creator "Mirko Gontek"^^xsd:string ;
      dc:date "2010-10-18T12:30:00+01:00"^^xsd:dateTime ;
      dc:description "A vocabulary for describing semantic
         relations between concepts"@en ;
      dc:title "Concept Relation Ontology"@en .

cr:ConceptRelation
      rdf:type rdfs:Class , owl:Class ;
      rdfs:comment "A Class to describe relations between
         skos:Concepts"@en ;
      rdfs:isDefinedBy cr: ;
      rdfs:label "Concept Relation"@en ;
      rdfs:subClassOf owl:Thing .

cr:subj
      rdf:type owl:ObjectProperty , rdf:Property ;
      rdfs:comment "A link from a Semantic relation to the
         subject Concept"@en ;
      rdfs:domain cr:ConceptRelation ;
      rdfs:isDefinedBy cr: ;
      rdfs:label "has subject"@en ;
      rdfs:range skos:Concept .

cr:obj
      rdf:type owl:ObjectProperty , rdf:Property ;
      rdfs:comment "A link from a Semantic relation to the
         object Concept"@en ;
      rdfs:domain cr:ConceptRelation ;
      rdfs:isDefinedBy cr: ;
      rdfs:label "has object"@en ;
      rdfs:range skos:Concept .

cr:relationType
      rdf:type owl:ObjectProperty , rdf:Property ;
      rdfs:comment "A link from a Concept Relation to the
         relation type that relates the concepts"@en ;
      rdfs:domain cr:ConceptRelation ;
```

```
    rdfs:isDefinedBy cr: ;
    rdfs:label "has relation type"@en ;
    rdfs:range skos:semanticRelation .
```

In the following, we describe our *Relation Ontology*, see Listing 5.10. The Relation Ontology aims at modeling relations between things. The key element of the ontology is the class `cr:ConceptRelation`. The Relation Ontology is an extension of the SKOS vocabulary; it allows to model more detailed relations between concepts by applying the class-centric modeling pattern. For example, it allows to qualify relations with a weight factor.

The `cr:ConceptRelations` class has the properties `cr:subj`, `cr:obj`, and `cr :relationType`. The `cr:subj` and `cr:obj` properties define the two concepts to be related. Both refer to `skos:Concepts`. Relations are directed from the instances referred to by `cr:subj` to those referred to by `cr:obj`. The property `cr:relationType` defines the type of relation to be modeled, a `skos: semanticRelation` or one of its subclasses.

Listing 5.11: Property Reification

```
ex:RelationReification a prv:PropertyReification ;
    prv:shortcut skos:semanticRelation ;
    prv:reification_class cr:ConceptRelation ;
    prv:shortcut_property cr:relationType ;
    prv:subject_property cr:subj  ;
    prv:object_property cr:obj .
```

A property reification (see Listing 5.11) maps the `cr:relationType` property used in the `cr:ConceptRelation` instance, a `skos:semanticRelation` or one of its subclasses, to its property shortcut form. The property reification must be published only once for each User Modeling Service. The mapping maps the relation statements to the corresponding SKOS vocabulary.

User Similarity Statements

Listing 5.12: User Similarity Statement

```
ex:AUserPropertyStatement
     a cr:ConceptRelation ;
     cr:subj ex:Tom ;
     cr:relationType skos:semanticRelation;
     cr:obj ex:Eric ;
     wo:weight [
         a wo:Weight ;
         wo:weight_value 0.3 ;
         wo:scale ex:AScale
         ] .
```

A User Similarity Statement models a relation between two users. FOAF is intended to model users and their relations on the Web. However, FOAF has only a very limited possibilities for social relation modeling: the only property it offers for relation modeling is the `foaf:knows` property. The semantics of this property differs from that of our user model scheme. The `foaf:knows` property aims at modeling an acquaintance of users; our user model scheme describes user relations that are more implicit: ad hoc relations that base on user similarities. Additionally, our relations have weightings. FOAF does not provide an appropriate vocabulary to model these semantics. Therefore, we use our Relation Ontology to model weighted similarity between users, as shown in Listing 5.12.

Listing 5.13: Property Reification

```
ex:RelationReification a prv:PropertyReification ;
    prv:shortcut skos:semanticRelation ;
    prv:reification_class cr:ConceptRelation ;
    prv:shortcut_property cr:relationType ;
    prv:subject_property cr:subj  ;
    prv:object_property cr:obj .
```

Again, a property reification mapping (see Listing 5.13) maps the `cr:relationType` property used in the `cr:ConceptRelation` instance to its property shortcut form. The property reification must be published only once for each User Modeling Service.

5.3.4 Metadata

Listing 5.14: Metadata example

```
ex:AUPStmt a prov:DataItem ;
    prov:createdBy _:ADataCreation ;
    dct:creationTime "2010-10-10T23:30:00+08:00"^^xsd:
       dateTime ;
    dct:provenance ex:AUserModelService .

_:ADataCreation a prov:DataCreation ;
    prov:performedAt "2010-10-10T23:30:00+08:00"^^xsd:
       dateTime ;
    prov:performedBy ex:AUserModelService .

ex:AUserModelService a prov:DataProvidingService ;
    rdfs:label "An example User Modeling Service" ;
    prov:operatedBy ex:AServiceOwner .
```

Finally, we describe how we model the metadata of the User Property Statements. We model two metadata properties: *creation time* and *provenance*. The metadata can be extended by further information, e.g., licensing information, when required. We use the *Provenance Vocabulary* [12] for metadata modeling. This vocabulary allows to model more extensive provenance specifications than Dublin Core. For example, it allows to model further information on the service that created a User Property Statement, and the owner of the service. Although the Provenance Vocabulary is more powerful for provenance modeling, Dublin Core is far more often deployed on the Web and the de facto standard for metadata modeling on the Web. Therefore, we publish substantial properties of the metadata with Dublin Core in parallel. In Dublin Core, we model creation time and provenance with the properties `dct:creationTime` and `dct:provenance`. Listing 5.14 shows metadata of a User

[12] http://sourceforge.net/apps/mediawiki/trdf/index.php?title=Provenance_Vocabulary (last access 2011-04-22)

Property Statement modeled with the Provenance Vocabulary and Dublin Core.

5.4 Conclusion

In this chapter, we presented a method for web-scale user model integration that meets the requirements of the Social Web. Our method enables a non-revolutionary development of a user modeling architecture atop of the existing Web. The benefit of our method comprises three features: (1) the method integrates the existing service architecture of the Web, (2) it provides a simple, lightweight and intermediary generic user model, and (3) it uses Semantic Web technologies for user model representation.

(1) Our method provides an architecture that bridges the gap between Social Web services and the Semantic Web. Technologies intended to be deployed on a web-scale need to attach great importance on simplicity and on incremental steps (cf. [47]). Our approach is *non-revolutionary*. It leverages the characteristics of the Web architecture for user modeling. Therefore, it reuses and extends established standards and technology as much as possible. Furthermore, it integrates with existing services and content published by these services. Our method aims at keeping required changes in existing services low. Hence, the method does not require major changes on existing services. The result is a user modeling method that is simple, hence feasible in a real-world scenario, and yet powerful. A key advantage is that it can be implemented simple and quick on a large scale.

(2) Our method models users on an intermediary generic level to make the models useful and sharable. User information is abstracted to a non-application-specific level that makes sharing reasonable but not too generic to be used efficiently by statistical analysis techniques. The method creates user models that are shareable across domains, purposes, and techniques. Our user models are modularized: we introduced User Property Statements that can be retrieved and processed by other Web services independently from each other. The result is a lightweight user model that models weighted interest, knowledge, and social relations.

(3) Our method uses Semantic Web technologies for user model integration. Semantic Web technologies enable a distributed, extensible, and scalable architecture for a Web of data. We illustrated the problems of n-ary and qualified relation modeling in RDF and contributed a representation method that overcomes these problems of RDF-based user modeling. We envision our user models as building blocks of a Web of data. A technical infrastructure and non-proprietary tools for Linked Data publishing already exists. In addition, shared vocabularies, published data, and real-world services exist, which can be connected to create synergy effects. Our user models integrate well into this Linked Data *ecosystem*. They base on independent User Property Statements: User Interest Statements, User Knowledge Statements, and User Similarity Statements. We used two specialized vocabularies for user modeling: CCO, and our Relation Ontology, an ontology that extends the SKOS vocabulary to model more detailed relations. Our representation method semantically maps the CCO and the Relation Ontology to the more widely-deployed vocabularies FOAF and SKOS to obtain semantic integration with more existing services. We achieved the semantic integration by property reification. Furthermore, we model metadata of the User Property Statements with the Provenance Vocabulary and the more widely-adopted Dublin Core vocabulary in parallel to enable extensible semantics and retain maximum semantic compatibility.

Chapter 6

Contribution 4: Earlybird – an exploratory tag search engine

In the following chapter, we describe Earlybird, a reference implementation of an exploratory tag search engine. Earlybird is a web-based tag search engine that facilitates explorative search on data from social bookmarking services. Earlybird implements the strategies for user model extraction and integration that we described in the previous chapters. It is a proof-of-concept of our findings. The purpose of Earlybird is to enable exploratory search on folksonomies. We demonstrate that the methods that we developed are viable to be adopted by services on the Social Web. Therefore, we provide a software that implements our methods. We show that our algorithms and strategies work in a real-world application, and that they can be implemented successfully in a Web-based service. Particularly, we show the following aspects of our findings on exploratory search:

Earlybird demonstrates the potentials of algorithmic user model extraction from social bookmarking systems. (1) We show that social bookmarking services are a valuable data source for exploratory search engines because they allow for algorithmic user model extraction. Social bookmarking data can be exploited for exploratory search; we can explicate knowledge inherent in folksonomies by collaborative filtering-based data analysis. The extracted knowledge can be applied to populate user models. (2) We show how to ex-

tract user models on a large scale with low effort. A key potential of our user model extraction strategy is extraction of user models for a large number of users from a relatively small input data set with low computational effort. Our user model extraction relies solely on the structure of folksonomies; it does not require content analysis. (3) We show that user models are not only valuable for adaptive Web systems. Our extracted user models contain three categories of content: user interest, user knowledge and user similarities. We demonstrate that our user models can also improve unpersonalized exploratory search. In our software, we use the output of our user model extraction algorithms as common *social* knowledge which provides cues for social navigation even in unpersonalized contexts.

Earlybird is an enabler for personalized services on the Web. We aggregate, create, and share user models with other services on the Web in Semantic Web standards. We show how to lock open user models, which reside in proprietary formats in most applications today. We make user models interchangeable and linkable across services. A further important aspects of Semantic Web standards is that we make user modeling more transparent and controllable for the user, because we use open standards and non-proprietary data formats. We further show how user model integration is feasible with the existing technological infrastructure of the current Web. Earlybird implements our user model integration architecture for web-scale user modeling. We show that our method can be integrated with technologies that are well-established for web application development today.

A further potential of Earlybird is to identify the strengths and limitations of our studies in a real-world context. Earlybird can act as a technical framework to enable further research on the presented methods. The findings of this thesis suggest further research in different directions. Earlybird is a platform that enables various further experiments; we want to name only two examples here. First, we discussed the potentials of specialized collaborative filtering algorithms that better leverage the nature of folksonomies for exploratory search, e.g., by considering the tagged resources for tag similarity computation. Further studies in this direction could implement and evaluate alternative algorithms. We implemented the Mahout machine learning

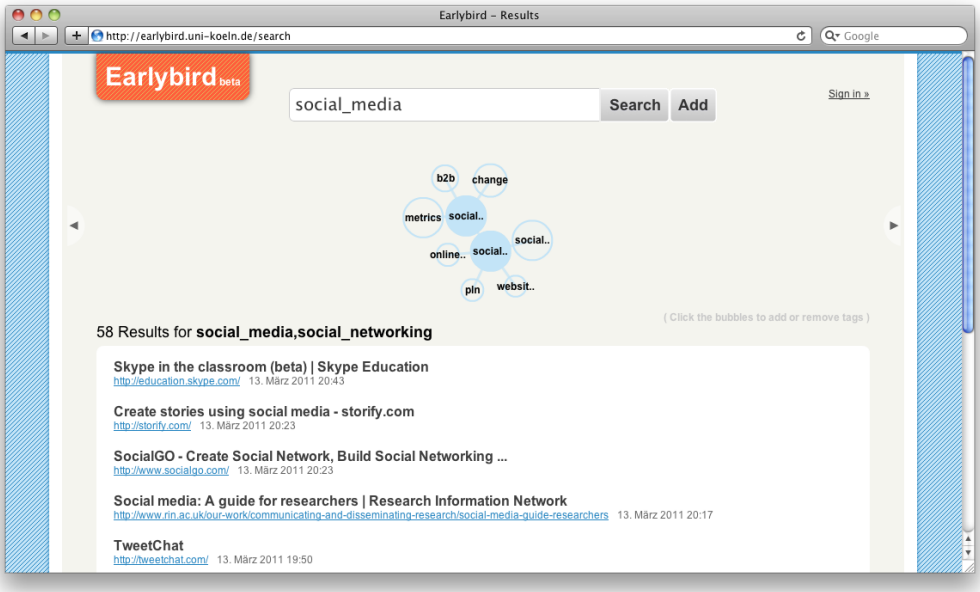

Figure 6.1: Screenshot of the Earlybird search result site

framework in our software that provides a technical infrastructure to experiment with different algorithms. Second, we discussed the limitations of our user model evaluation strategy for measuring user perceived quality. A real-world application like Earlybird allows for complementary evaluation strategies that fill the gaps of our studies. For example, usage data analysis and A/B tests allow to measure the user-perceived quality of the search engine. We equipped Earlybird with usage logging to prepare the ground for further research in this direction.

The remainder of this chapter is organized as follows. In Section 6.1, we describe the functionality of Earlybird. In Section 6.2, we describe its 4-tier software architecture and document which technologies we used. In Section 6.3 through 6.6, we describe the key challenges of the implementation and document how we solved them in the individual software tiers of Earlybird.

6.1 Functionality

Earlybird [1] is a Web-based social tag search engine, see Figure 6.1. It allows to search for webpages by tags. Earlybird retrieves webpages that have been tagged with the query tags in the popular social bookmarking services Delicious and Connotea during the last 30 days. The search results are ranked by date, most recently tagged webpages are the top results. Additionally, Earlybird recommends tags that are related to the current query tags. The search engine recommends the 5 most similar tags for each query tag and visualizes the recommendations in an interactive tag network. Users can browse the tag network to discover new tags and webpages. To browse the tag network, users can click on recommended tags to add them to the search query. Furthermore, users can add arbitrary tags to the query over an input field. A click on a tag that is in the current query removes it from the query. The search results adopt accordingly.

Users can register to Earlybird to receive personalized recommendations. At registration, users can optionally enter their Delicious or Connotea account to import their existing bookmarks from the bookmarking services. For registered users, Earlybird creates persistent user models. The bookmarking accounts of registered users are connected with their Earlybird account, so that also future bookmarks are added to the user model in real-time. Additionally, search queries in Earlybird are logged an added to the user models. Thus, Earlybird personalizes recommendations according to users' bookmarks and logged search queries.

Input data Earlybird aggregates and makes searchable the data from two social bookmarking services, Delicious and Connotea. It allows to retrieve webpages and recommend tags which have been posted in these services. To process and analyze the data, we cache them in a local database. Both bookmarking services provide their bookmarks over Web APIs as RSS feeds which contain the latest bookmarks. We poll these feeds regularly and cache the new bookmarks to our local database. To keep the data set of manageable

[1] available at http://earlybird.uni-koeln.de

size for our experiment setup, and to comply the terms and conditions of the bookmarking services, we store the bookmarks only for 30 days. We delete cached bookmarks from our database when they exceed the age of 30 days.

Query Results Earlybird retrieves webpages that have been bookmarked by users of Delicious or Connotea and tagged with all of the query tags during the last 30 days. When more than 100 results are retrieved, only the latest 100 results are displayed. The user interface displays the results as a list of URLs along with the webpage title and the date when the webpage was last bookmarked. The search results are ranked by date, latest tagged webpages first.

Recommendations Along with the query results, Earlybird recommends the top-5 similar tags for each of the current query tags. The recommendations allow to navigate in a tag network – the recommender adds a *context of related tags* to the current query. The recommender is a collaborative filtering tag recommender which suggests similar tags for a given tag. The recommendations are generated from the input data, the cached bookmarks, in three steps. First, we compute a preference matrix from the folksonomy and search logs. Second, we compute a tag similarity matrix from the preference matrix. These two steps are done offline; both matrixes are pre-computed independently from search requests and are updated every 24 hours. Only the third step is performed online, i.e., during the search process. Here, we re-score similarities between tags and deliver the recommendations. When a user searches for a tag, the recommender retrieves similar tags from the pre-computed tag similarity matrix, re-scores according to certain criteria, and delivers the top-5 similar tags. At present, Earlybird implements two re-scoring policies. First, we down-score unpopular tags, so that we bias recommendations towards popular tags. Popular tags are those tags that have been tagged more often in the bookmarking services in the last 30 days. The popularity re-scoring is performed for all users and all tags. Second, we up-score tags which are in a user‘s user model, so that we bias recommendations towards tags which a user has already used to tag a webpage or to search for

a webpage. The re-scoring is only performed for registered users, for which a persistent user model is available.

Tag network Recommendations are visualized in an interactive tag network. Users can extend or replace the search query by navigating through the tag network visualization. The search results adapt accordingly. To add a recommendation to a query, users can click on the recommendation in the visualization. A click on tags which are already in the current query removes the tag from the query. The size of the tags in the visualization indicates the their popularity: popular tags are larger than unpopular. The layout of the network visualization is computed by a physical layout algorithm, i.e., physical forces are attached to the tags to arrange the tags in the two-dimensional space.

Personalization Users can register to Earlybird to receive personalized recommendations. On a registration webpage, users can create an account. Optionally, they can enter their bookmarking accounts from Delicious and Connotea to connect them to their Earlybird account. For registered users with connected bookmarking accounts, all existing and future bookmarks of the accounts are imported to a persistent user model. Additionally, all search queries and recommendations are recorded and stored to the user model. For registered users that did not connect a bookmarking account, we store only search queries and recommendations. Our recommender algorithm re-scores tag similarities for registered users according to their user model.

Logging Earlybird logs search queries of unregistered and registered users to prepare the ground for usage data evaluation. We log three types of information. First, we record all individual search query tags – the tags that a user queries. Second, we log the recommendations for each query – the similar tags that the recommender suggests for the query. Third, we log the recommendations that the user clicks – the recommendations that the user finds helpful. For registered users, we link these logs to the user account ID. This allows for analyzing individual usage behavior in the long-term.

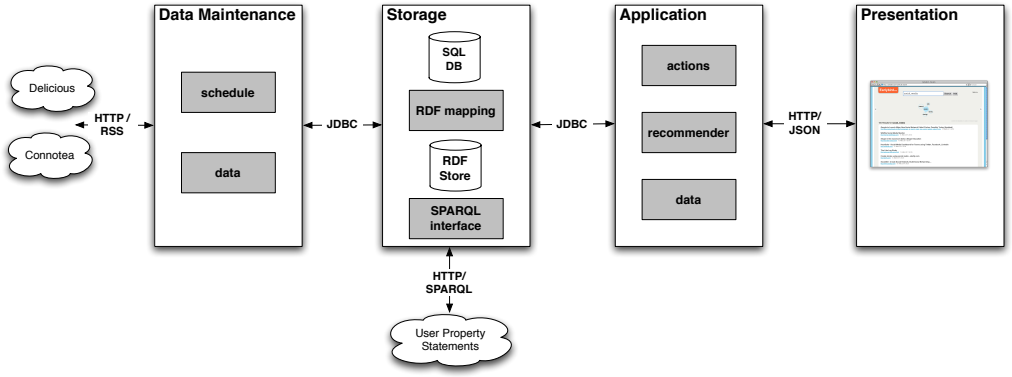

Figure 6.2: Software architecture

User model integration For registered users, Earlybird is a user model publishing service. It allows to publish user models on the Web as RDF. Other services on the Web can use the information in the user models, e.g., for personalization. We publish user models as User Property Statements, which we introduced in Chapter 5, over a SPARQL web interface. Remote web services can query our SPARQL endpoint to retrieve the User Property Statements of registered users.

6.2 Software architecture

Earlybird is a web application that consists of loosely coupled components which form a distributed server-client architecture. Figure 6.2 shows the overall architecture of our software. We distinguish four software tiers: *Data Maintenance*, *Presentation*, *Application*, and *Storage*.

The server-side Data Maintenance and Application tiers of our search engine are implemented in Java and run in a Tomcat [2] servlet container. We use the Struts2 [3] framework to implement the model-view-controller architecture. For the recommender logic, we implemented the Apache Mahout [4] machine learning and data mining framework. Additionally, we use some

[2]http://tomcat.apache.org/ (last access 2011-04-22)
[3]http://struts.apache.org/ (last access 2011-04-22)
[4]http://mahout.apache.org/ (last access 2011-04-22)

smaller APIs, e.g., the ROME API [5] to deserialize RSS streams and the Quartz Scheduler API [6] to schedule tasks.

For the Presentation tier, the browser-based user interface of our software, we use JSP, Javascript, and Flash. The Presentation tier communicates with the Application tier over HTTP. Inside JSP webpages, we use Javascript for asynchronous client-server communication. For the interactive tag network which visualizes the recommendations, we use Flash. A Flash movie is embedded in JSP and communicates with Javascript over a Flash library – the External Interface API [7]. The Flash movie implements a further API, Box2DFlashAS3 [8], to simulate physical forces. We use this physics library to create the dynamic layout and interactive look-and-feel of the tag network.

For the Storage tier, we use the open source edition of the OpenLink Virtuoso [9] database server. The Data Maintenance and Application tiers communicate with the Storage tier over JDBC. Here, we store all data which we collect from the bookmarking services. Furthermore, we use Virtuoso to publish the user models over a SPARQL endpoint and to store usage logs.

In the following sections, we describe the tasks and challenges of the individual tiers and document important details of their implementation.

6.3 Data Maintenance

Earlybird bases on data from social bookmarking services, i.e., it makes their bookmarks searchable. Therefore, we cache and analyze bookmarks from running social bookmarking services. The current implementation connects to two services: Connotea and Delicious. The software can be easily extended to further services. The task of the Data Maintenance tier (see Figure 6.3) is to cache all bookmarks that are generated in these services.

The bookmarking services are live systems, where new data are being cre-

[5] http://java.net/projects/rome/ (last access 2011-04-22)
[6] http://www.quartz-scheduler.org/ (last access 2011-04-22)
[7] http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/external/ExternalInterface.html (last access 2011-04-22)
[8] http://www.box2dflash.org/ (last access 2011-04-22)
[9] http://virtuoso.openlinksw.com/ (last access 2011-04-22)

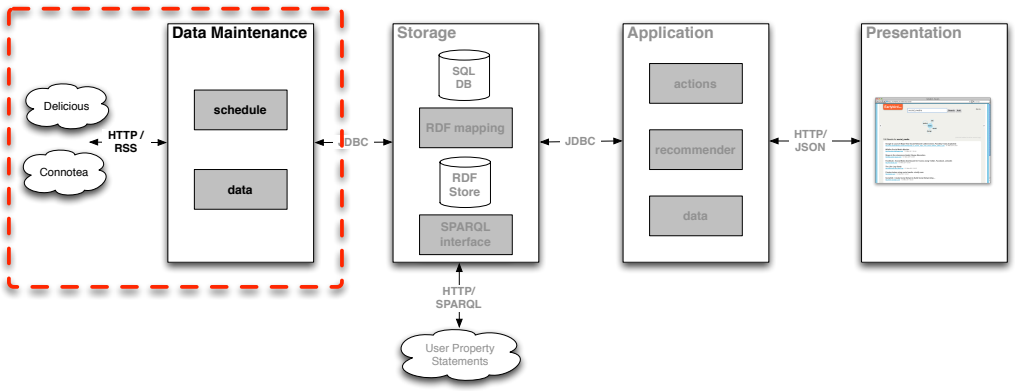

Figure 6.3: Data Maintenance tier

ated by users constantly. Depending on the service, the traffic can be high: Delicious generates about 60 bookmarks per minute on the average; Connotea generates about 5 bookmarks per minute. We retrieve the bookmarks generated by both applications. Our exploratory search should be able to explore the *live* data of both systems. It is important for our application, that we retrieve the data constantly, as soon as they are available. Therefore, we cannot simply import a dump of the bookmarks, but we need a live connection to the bookmarking services which we want to integrate into our search engine. Our live data caching process caches all bookmarks generated by the connected services to a local data store in near real-time. To comply the terms and conditions of the bookmarking services and to keep the data size manageable, we cache the data only for a limited time: until they exceed the age of 30 days. The live caching offers three challenges. First, we need to retrieve the data fast – as soon as they are available – over HTTP. Second, we need to insert the data to our data store efficiently – we must be able to retrieve, process and insert more than 65 posts per minute. Third, we need to identify and delete bookmarks older than 30 days efficiently. Delicious and Connotea provide Web APIs that allow to poll recently created bookmarks over HTTP. The bookmarks are delivered as RSS feeds. We use these RSS feeds to poll the bookmark data and cache them in our local database. To insert and delete bookmarks, we call stored procedures in the Storage tier over a JDBC connection. The Data Maintenance tier contains two packages

that implement the data caching: *schedule* and *data*.

Schedule The *schedule* package schedules Quartz jobs which regularly poll RSS feeds with the 100 most recent bookmarks from the Delicious and Connotea Web APIs. The API requests require different parameters, and the RSS feed responses have different structure for each service. Therefore, the schedule package contains a client for each connected service. Each client sends a HTTP request to the service's API to retrieve the feed data as an RSS input stream. We deserialize the response stream and store the bookmarks to our local database by calling an insert procedure in the Storage tier through an SQL query over JDBC. We check whether a bookmark belongs to registered users that connected their bookmarking accounts. We store bookmarks that belong to a registered user under their local account ID. Other bookmarks are stored under the original bookmarking account ID. To keep account IDs unique across several services, we attach a service prefix „con_" or „del_", so that user are stored as „con_USERNAME" and „del_USERNAME". A challenge of the scheduler is to poll bookmarks in intervals so that no bookmarks are omitted at traffic peaks. We set the polling interval of the scheduler to a fixed interval of 30 seconds for Delicious and 10 minutes for Connotea; Connotea requires less polls, because the service creates less bookmarks. A problem of our polling strategy is that we retrieve some bookmarks more than once. Therefore, we keep the date of the last imported bookmark in the client at each poll. In the subsequent poll, we only import those bookmarks that have been posted after this date.

A further task of the schedule package is to delete bookmarks which are older than 30 days from the local database. We call a delete procedure in the Storage tier through a SQL query every hour.

Data The *data* package connects the Data Maintenance component with the Storage tier. The package connects the Storage tier as a JDBC datasource. Furthermore, we define all SQL queries required for data maintenance in the data package. We moved some of the query logic to the Storage tier. For example, insert and delete requests are performed as procedure

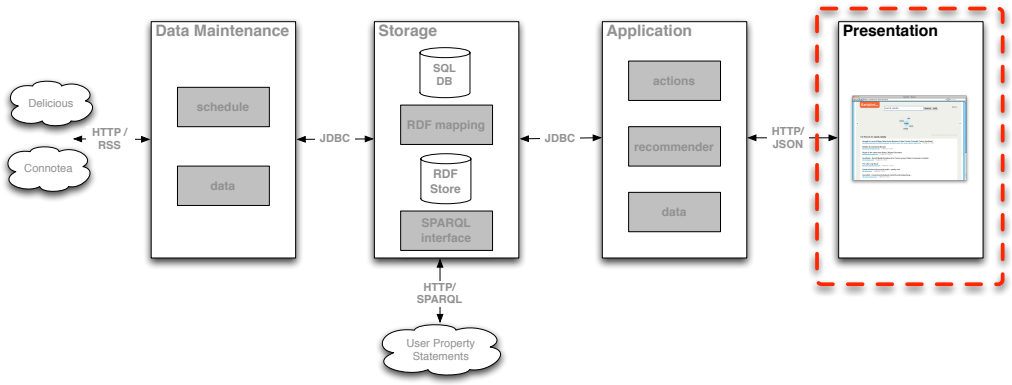

Figure 6.4: Presentation tier

calls to enclose database schema-related logic from the application logic, and to speed up the processes. Procedures are scripts at the Storage tier that are called through SQL queries over the JDBC connection. We describe the functionality of the used procedures in Section 6.6.

6.4 Presentation

The Presentation tier is the client-side component of our software, see Figure 6.4. It implements a user interface for explorative search. The Presentation tier is browser-based; it delivers webpages where users can search by tag, explore related tags through recommendations, and register and login to Earlybird.

The task of the Presentation tier is to provide an easy-to-use user interface for searching by tag and for exploring the results and recommendations for the search query. A key feature of Earlybird are recommendations that suggest similar tags for the query tags. For each query tag, Earlybird suggests 5 similar tags. To support explorative search, our application requires navigation and filtering functionalities for the tags and recommendations. We suggest two important interaction functionalities to support exploration. First, users must be able to replace a search query by a recommendation. This allows to browse the recommendations – to navigate through the tags. Second, users must be able to add recommendations to the query to expand

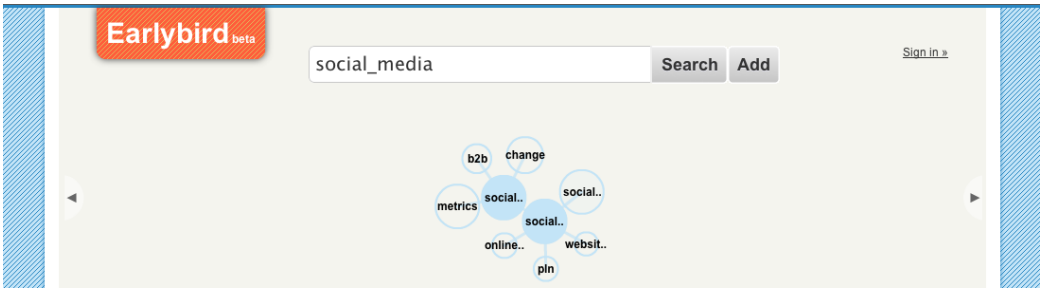

Figure 6.5: Tag cloud with scaled tags and hierarchy

the original query by a recommendation. This allows to narrow down the result set – to filter the results.

In the Presentation tier, we implemented a user interface that meets the discussed requirements for explorative search. In-depth research on the user interface is beyond the scope of this thesis; however, we are aware that user interfaces play an important role for explorative search engines, e.g., they have a great influence on the perceived quality of a system. With our user interface implementation, we want to prepare the ground for further research at this open end of our research. Our user interface is extensible and easily adaptable to support further experiments.

We implemented the desired exploratory search facilities in an interactive tag network visualization, which is a central part of the Presentation tier. In this interactive visualization, we visualize query tags and their recommendations, see Figure 6.5.

Tags in the network are visualized as circles that surround the actual tag strings. Edges connect the tags and their recommendations to indicate to which tag a recommendation belongs. Tag circles have different sizes. The size of the circles reflect the tag popularity. Larger circles represent more popular tags. The circle sizes are scaled relative to all other circles in the currently displayed network. Recommendations can be associatively or hierarchically related to the search tags. Associative relations mean that two tags are semantically similar. We visualize recommendations with associative relations to the query tag as circles with a single line. Hierarchically related tags are semantically narrower or broader. Hierarchy can provide valuable

navigation cues to the user. We visualize recommendations that are narrower than the query tag as circles with a double line and recommendations that are broader than the query tag as circles with a single thin line. To further support navigation in the tag network, we provide buttons to navigate back and forward in the search history.

An important design decision for the tag network visualization was the network layout – how are the tags arranged. We implemented a *physical layout* for the tag network visualization. A physical layout algorithm allows to easily implement a flexible and dynamic tag network visualization. Physical graph layouts are better suited for dynamic visualizations than layered graph layouts, because they allow to insert and remove nodes from the network without re-calculating the layout. This is particularly important for interactive visualizations. Our layout algorithm implements the Box2DFlashAS3 physics library. The physics engine simulates physical objects with mass connected by rigid levers in a world with gravitation. A gravitation force in the center of our visualization attracts the objects in the simulated world: all objects – the tag circles – gravitate around the center of the Flash movie. Objects repulse each other and rigid levers – the edges between tags in the tag network – connect tags with their recommendations.

Our tag network visualization achieves a good look-and-feel and is easy to implement and adapt. With the interactive tag network visualization, users can navigate through recommendations and filter the result set by adapting the search query. Recommendations and search results adapt when the search query is modified. It supports exploration of the tag space, hence meets the discussed requirements for exploratory search interfaces. We also choose this layout because it allows for further research experiments with the user interface. Changes, e.g., different numbers of recommendations or different scaling parameters for tags, can be realized easily with the current implementation.

The key technologies used in the Presentation tier are JSP, Struts2, Flash, and Javascript. The tag network visualization is implemented in a Flash movie, which is embedded into the JSP webpage and communicates with the Application tier through HTTP requests. To update search results, the Flash

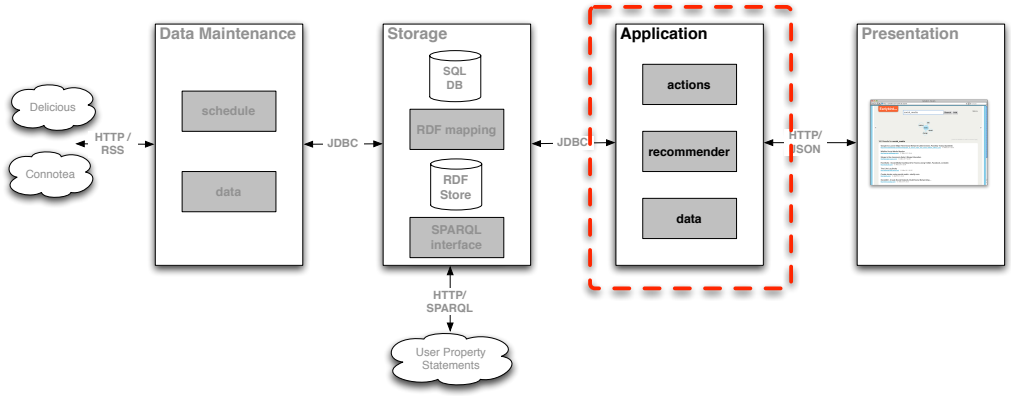

Figure 6.6: Application tier

movie sends HTTP requests to the Application tier. The movie also calls Javascript functions embedded in the JSP webpages for client-side updates. The communication between Flash and Javascript is implemented with the External Interface API, which allows to call external methods from within a Flash movie and to call flash methods from external scripts.

6.5 Application

The Application tier implements the server-side application logic of the exploratory search engine, see Figure 6.6. The main function of the Application tier is to retrieve search results and recommendations for queries. Additionally, it implements utility tasks of Earlybird: login, registration, and usage logging. Java servlets connect the Application tier with the Presentation tier. Communication with the Storage Tier is realized with a JDBC connection.

A key challenge of the Application tier is to provide recommendations *fast* – speed of the recommender is a crucial factor for our search engine. Our application requires to make recommendations in real-time. Therefore, we must pre-compute at least parts of the recommendation process to boost the performance and yet keep up-to-date with the constantly changing input data. We implemented a recommender which retrieves preferences from the Storage tier in regular intervals and stores them to the filesystem on our

server for faster access. From the preference file, we pre-compute a similarity matrix which resides in the memory. The memory-based similarity matrix is then used to compute the recommendations online, i.e., at request from the Presentation tier. A further task of the Application tier is to connect the user models in the Storage tier with the recommender. We implemented a re-scorer which biases recommendations according to the user model. The recommender application logic retrieves the user models from the Storage tier and injects them into the recommendation process. Additionally, we pass the results of the recommender to the Storage tier for user modeling. The Application tier contains three packages: *actions*, *recommender*, and *data*.

Actions The *actions* package implements the interfaces for the communication with the Presentation tier. It contains a number of servlets that handle HTTP requests from the client-side. The servlets handle three categories of requests: (1) Login/logout and registration requests, (2) search and recommendation requests, and (3) usage logging requests.

The `Register` servlet handles the requests from the registration form of the Presentation tier. It validates the input, sends a confirmation code for the double opt-in registration process to the user‘s email address and triggers the creation of a new user account in the database. A further servlet, `Confirm`, handles the input of the confirmation code; the account must be activated with the confirmation code to validate the user‘s email address. When the user enters a bookmarking account at registration, the `Register` servlet triggers an import of the user‘s existing bookmarks after the registration. The logic for the bookmark import resides in the *data* package. `Login` and `Logout` servlets handle the requests from the login form and the logout link on the webpage. The `Login` servlet validates the login credentials and adds a user ID to the user‘s session. `Logout` deletes the session. Additionally, we implemented an `Autologin` servlet, which allows to persist the user ID in a cookie, if the user checks this option at login. `Autologin` checks if a cookie with valid credentials is available when the start page is requested by the client.

A `Search` servlet handles search requests from the Presentation tier. The

servlet deserializes the search query, which is a parameter in the HTTP request. It triggers a database query to retrieve the query results. The results contain the URLs, titles and dates of the result webpages. The servlet serializes the results to JSON [10] and sends them to the client in a HTTP response. Listing 6.1 shows a serialized response for a search request.

Listing 6.1: Response of Search servlet

```
{"identifier":"id","items":[{
    "id":0,
    "timestamp":"31. Januar 2011 17:52",
    "title":"Java",
    "uri":"http:\/\/java.sun.com"
},
...
]}
```

Listing 6.2: Recommendation type extraction method

```
1  private int calcRecType(int queryScore, int recommScore,
       float similarity) {
2      float threshold = 0.15f;
3      if (similarity < threshold) {
4          return 0; // associative relation
5          }
6      else if (recommScore < queryScore) {
7          return 1; // recommendation is narrower
8          }
9      return 2; // recommendation is broader
10     }
```

A `Recommendation` servlet handles the recommendations for a search query. The servlet deserializes the search query and triggers a recommendation request to the *recommender* package. The servlet also initiates the recommendation re-scoring. It biases the recommendations towards known tags for logged in users. Therefore, the servlet triggers a query which retrieves all known tags of the user from the database and creates a class instance `Rescorer` for these tags. The logic of the `Rescorer` is defined in the *recom-*

[10] http://www.json.org/ (last access 2011-04-22)

mender package. The `Rescorer` is passed through to the `Recommender` which returns recommendations that comply to the re-scoring policy. The results of the `Recommender` are processed in a second step to compute the recommendation type: a relation between a tag and its recommendation can be associative or hierarchical; if a hierarchical relation exists, the recommendation can be narrower or broader than the query tag. Our hierarchy extraction algorithm allows for fast and easy hierarchy computation from recommendations. For the hierarchy extraction, we first set a threshold value for the algorithm; informal experiments indicated that 0.15 is appropriate. We pass three values to the method: the popularity score of the query tag, the popularity score of the recommendation, and their similarity value returned by the recommender. If the similarity value is below the given threshold, the method returns an associative relation. If the similarity is above the threshold, the method checks whether the recommendation is narrower or broader than the query tag. The recommendation is considered narrower if the score of the recommendation is smaller than the score of the query. Otherwise, the recommendation is considered broader than the query. Listing 6.2 shows the code for the recommendation type computation.

Listing 6.3: Response of Recommendation servlet

```
{"identifier":"id","items":[{
    "id":0,"searchTag":"java",
    "recType":2,
    "recTag":"programming",
    "searchTagScore":16750,
    "recTagScore":58861}
},{
    "id":1,"searchTag":"java",
    "recType":0,
    "recTag":"opensource",
    "searchTagScore":16750,
    "recTagScore":25598
},
...
]}
```

The recommendations with the hierarchy information are then serialized

to JSON and delivered to the client in an HTTP response. The response contains (1) the query tags and their popularity score and an identifier for better data handling on the client-side, (2) the top-5 similar recommendations for each query tag and their popularity score, and (3) a numerical value for the recommendation type – „0“ for associative, „1“ for narrower or „2“ for broader – for each recommendation. Listing 6.3 shows a serialized response for a recommendation request.

The servlet logs all recommendations to the usage logs. Therefore, it calls a log query which stores the recommendations for the current user to the database.

Recommender The *recommender* package contains a collaborative filtering-based recommender that implements our tag similarity extraction method. The recommender package implements interfaces from the Apache Mahout framework: `IDMigrator`, `ItemSimilarity`, `ItemBasedRecommender`, and `Rescorer`.

The `IDMigrator` allows to map between `String` and `Long` values. The Mahout `ItemRecommender` internally requires tags and users as 64-bit numeric values – `Long` IDs – instead of strings. Our software stores user and tags as strings. Hence, we must convert all strings to Longs for the recommender. The mapping from `String` to `Long` is deterministically computable; however, the reverse conversion is not. Therefore, we need to store a mapping between strings and their `Long` IDs. We implemented a memory-based `IDMigrator` which stores these mappings.

Listing 6.4: Recommender initialization process

```
1 private void refreshRecommender(File preferenceFile){
2     // ...
3     // CREATE MAHOUT DATAMODEL ON PREFERENCE MATRIX
4     DataModel dm = new FileDataModel(preferenceFile);
5     // CREATE CUSTOM ITEM SIMILARITY MATRIX
6     CachingItemSimilarity itemSim = new CachingItemSimilarity
          (new BiasingTanimotoCoefficientSimilarity(dm), dm);
7     // CREATE ITEM RECOMMENDER WITH SIMILARITY MATRIX
8     RecommenderComponent.setItemRecommender(new
          GenericItemBasedRecommender(dm, itemSim));
```

```
9       // ...
10      }
```

A scheduled Quartz job initializes the tag recommender every 24 hours. The initialization refreshes the recommender to include newly added input data. The procedure implies three steps. First, we initialize a new preference matrix. Therefore, we trigger a database query to retrieve a list of user-tag relations, which excludes tags with a popularity score < 2. We exclude these tags to reduce noise and size of our input data set. We convert all strings in the user-tag matrix to `Long` IDs with the `IDMigrator` and store the numerical preference matrix to the local file system on the server. The file-based preference matrix allows for faster access than a JDBC-based solution. Second, we initialize a tag similarity matrix from the preference matrix. We compute the similarity for all tags with the a custom implementation of `ItemSimilarity`. Our algorithm computes the Tanimoto coefficient but biases towards popular tags by ignoring tags below a certain popularity score. This similarity algorithm corresponds to the algorithm T(p) presented in Chapter 4. We store the similarity matrix in memory to achieve adequate performance of the recommender. Third, we instantiate an `ItemBasedRecommender` on the similarity matrix, which allows to recommend similar tags to a query tag. Listing 6.4 shows the instantiation of a new `ItemBasedRecommender` from a preference file during the initialization process.

Listing 6.5: NoveltyRescorer implementation

```
1  public class NoveltyRescorer<LongPair> implements Rescorer<
       LongPair>{
2       Set<LongPair> knownSimilarities;
3       // ...
4       public NoveltyRescorer(Set<LongPair> knownSimilarities){
5           this.knownSimilarities = knownSimilarities;
6           }
7
8       @Override
9       public double rescore(LongPair pair, double originalScore
            ){
10          if (knownSimilarities.contains(pair)){
```

```
11              return originalScore;
12              }
13          return originalScore/2;
14          }
```

Listing 6.6: Retrieve top-5 similar tags from recommender

```
1 Set<LongPair> knownSimilarities = new HashSet<LongPair>();
2 //...
3 Rescorer<LongPair> rescorer = new NoveltyRescorer<LongPair>(
    knownSimilarities);
4 List<RecommendedItem> recommendedItems = RecommenderComponent
    .getItemRecommender().mostSimilarItems(
    RecommenderComponent.getIdMigrator().toLongID(queryTag),
    5, rescorer);
```

The recommender package also implements a custom implementation of `Rescorer`: the `NoveltyRescorer`, see Listing 6.5. This `NoveltyRescorer` biases recommendations for registered users towards unknown tags. It up-scores similarity values for tags which are already in the user profile of a certain user by the factor 2. We up-score known tags to recommend tags that are already familiar to the user to support domain learning on known topics. In the `Recommendation` servlet, where the recommender is called, the `NoveltyRescorer` is passed through to the `mostSimilarItems()` method of the recommender, which returns the top similar items. Listing 6.6 shows how the top-5 similar tags for a query tag are retrieved from the recommender in the `Recommendation` servlet with a re-scoring policy.

Data The *data* package connects the Application tier with the Storage tier over a JDBC connection and defines all SQL queries that are used in the Application tier. Additionally, the data package implements two importer classes for bookmark accounts: a `DeliciousImporter` and a `ConnoteaImporter`. The importers import all bookmarks of a user from Delicious and Connotea at registration, if the user enters their bookmarking account. Both importers retrieve an RSS feed with the user‘s bookmarks. They deserialize the bookmarks and trigger a database query to add the bookmarks to the user model.

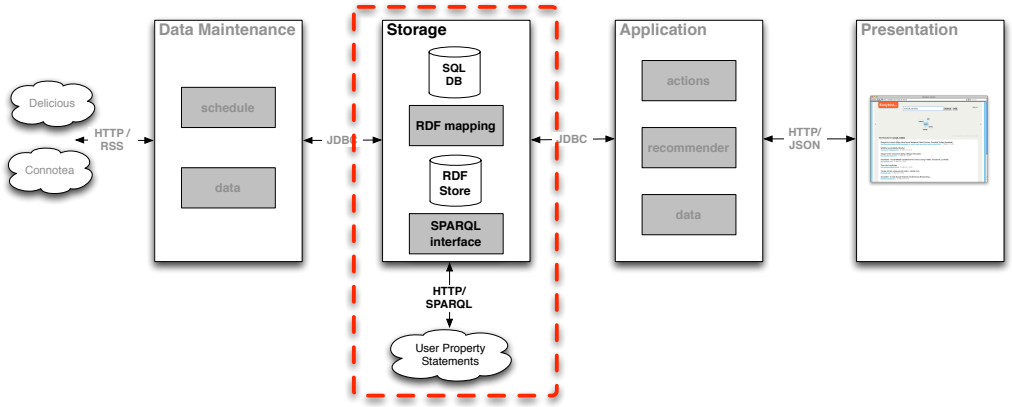

Figure 6.7: Storage tier

6.6 Storage

The Storage tier, see Figure 6.7 is an OpenLink Virtuoso database server. It implements our method for user model publishing described in Chapter 5. We distinguish two tasks of the Storage tier: *data storage* and *data publishing*.

(1) The *data storage* task involves storage of all data required by the Application tier. It stores the retrieved bookmarks, generated user models, and usage logs in a local object-relational database. Additionally, the Storage tier defines the SQL database schema and encloses data insert, update, and remove procedures from the other tiers. The bookmark caching process, which is performed by the Data Maintenance tier, requires fast insert and remove operations on the data. For the Application tier, we need to model the data so that they can be queried efficiently. The key challenge of the data storage is to define a database schema which meets the requirements of both tiers. We use stored procedures and triggers in the Storage tier to maintain the data efficiently.

(2) The *data publishing* task involves two steps: user modeling and user model supply. First, we model RDF-based user models, User Property Statements, from the user models stored in our internal data schema. In the Storage tier, we create User Property Statements from the user model data generated by the Application tier. The desired output of the user modeling process are User Property Statements that conform to our RDF user model

schema, cf. Chapter 5. Second, we supply the user models on the Web, so that other services can query and access the user models. The key challenge of the data publishing is to transform the user models from the database schema into our RDF user model schema efficiently. The user models in the database update constantly, namely when users search or bookmark. The modeling process must update the User Property Statements dynamically according to the user models created dynamically by the Application tier. Additionally, we must pass the created User Property Statements to an interface that makes them accessible over the Web. Ideally, the Storage tier should provide a strategy that performs the user model publishing in real-time.

We opted for the Virtuoso database server to implement the Storage tier because it provides three of features that support the described tasks.

(1) Virtuoso provides an object-relational SQL database with a JDBC driver and other data-access interfaces. It is a fully-fledged database server that allows to use established frameworks and tools for web application development. Additionally, Virtuoso provides a script language, *Virtuoso/PL* [11], for writing *stored procedures* and SQL triggers in Virtuoso. Stored procedures enclose complex data operations to the database server. In a web application, they reduce traffic between the database server and other components of a distributed architecture. Hence, stored procedures can improve the performance of complex data manipulation operations compared to application side solutions. In Earlybird, we use Virtuoso/PL for fast insert, update and remove operations to speed up the data storage task.

(2) Virtuoso provides an RDF store and supports the SPARQL query language. RDF data can be stored in a native *quad store*. For example, we can import our created OWL schemas to the Virtuoso quad store as RDF data. Virtuoso supports SPARQL queries on these data. To simplify the integration into non-RDF applications, Virtuoso allows to send SPARQL queries inside SQL queries. Thus, we can query the Virtuoso quad store through standard JDBC/SQL connections. We can use SPARQL and SPARQL extensions where we can use SQL. Besides the quad store which allows to

[11] http://docs.openlinksw.com/virtuoso/sqlprocedures.html (last access 2011-04-22)

store and retrieve native RDF data, Virtuoso also provides *RDFization* facilities for automated RDF creation from relational data sources. It allows to *map* relational data into RDF with so-called *RDF Views*. RDF Views are a dynamically created RDF representation of relational data sources. The mapping is dynamic , i.e., changes to the underlying data are reflected immediately in the RDF representation. Virtuoso provides a declarative *Meta Mapping Language* [12] for generating RDF views from SQL. The Meta Mapping Language is itself implemented as an extension of SPARQL, hence can be used inside SQL queries. To access RDF data, Virtuoso comes with a SPARQL web interface. Clients can pass SPARQL queries as HTTP parameters to this interface for querying RDF data.

(3) A key feature of Virtuoso for our application is, that the database server contains two data stores from two technological camps, an object-relational SQL database and an RDF quad store. The potential of their combination in one database server is to bridge the technological gap between both technologies. With Virtuoso, we can tightly integrate existing technical infrastructure with Semantic Web technologies. To create RDF views, no changes are required to the underlying relational database schema. In Earlybird, we can use efficient standard technologies established for web application development, JDBC/SQL and related frameworks, and add a semantic layer with relatively low effort. Thus, we show how to integrate semantic technologies into the established technology stack of Web services.

In the Storage tier, we use the relational database facilities of Virtuoso for data storage tasks – bookmark management, user account management, and usage logging. The bookmark management performs the local caching of recent bookmarks retrieved from the bookmarking services for 30 days. Account management includes registration and login of registered users. Usage logging records search queries and recommendations for registered users. For the user model publishing task, we use the RDF facilities. We map the relational user model schema to our RDF User Property Statements and make the user models accessible through a SPARQL web interface. In the following sections, we document how we implemented data storage and data

[12] http://docs.openlinksw.com/virtuoso/rdfrdfviewgnr.html (last access 2011-04-22)

Table 6.1: SQL table EARLYBIRD_ENTRIES

EARLYBIRD_ENTRIES
id (primary key)
post_uid
tag
url
user_uid
provider
post_datetime

Table 6.2: SQL table EARLYBIRD_SCORES

EARLYBIRD_SCORES
tag (foreign key)
score
tag
cur_timestamp

publishing in the Storage tier.

6.6.1 Bookmark management

The data schema contains four SQL tables relevant for bookmark management: `EARLYBIRD_ENTRIES`, `EARLYBIRD_SCORES`, and `EARLYBIRD_URLS`, see Tables 6.1, 6.2, and 6.3. The data schema allows for efficient queries for search and recommendations from the Application tier. In the following section, we describe how we implemented the bookmark insert and delete process and the user account management.

Table 6.3: SQL table EARLYBIRD_URLS

EARLYBIRD_URLS
url (foreign key)
title
last_posted

Insert bookmarks

We insert bookmarks that are polled continuously by the Data Maintenance tier or imported nonrecurring for each registered user by the Application tier into the table `EARLYBIRD_ENTRIES`. One bookmark (or *post*, in our database terminology) consist of one or more rows, one row for each tag. A post is identified by a random post ID. Each row contains the random *post ID*, a *tag*, the *URL* of the bookmarked webpage, the *account ID* of the creator, and the *service* where the post was imported from. We assign *„delicious“*, *„connotea“*, or *„local“* to distinguish the services; *„local“* labels posts of registered users. Note, that posts may lack the URL in one special case: due to API restrictions of Connotea, we cannot import the existing bookmarks of users. Instead, we can only import a list of tags that have been used the user. For the Application tier, this is not a problem because we neglect URLs for the user modeling. However, we must consider the special case at the data storage process.

Listing 6.7: Virtuoso/PL insert bookmark procedure definition

```
CREATE PROCEDURE DB.DBA.EARLYBIRD_ADD_ENTRY (IN in_post_uid
    VARCHAR, IN in_tag VARCHAR, IN in_url VARCHAR, IN
    in_user_uid VARCHAR, IN in_provider VARCHAR, IN
    in_post_datetime DATETIME, IN in_title VARCHAR){

INSERT INTO DB.DBA.EARLYBIRD_ENTRIES (post_uid, tag, url,
    user_uid, provider, post_datetime)
VALUES (in_post_uid,in_tag,in_url,in_user_uid,in_provider,
    in_post_datetime);

INSERT SOFT DB.DBA.EARLYBIRD_SCORES (tag, score) VALUES (
    in_tag, 0);
UPDATE DB.DBA.EARLYBIRD_SCORES SET score=score+1 WHERE tag=
    in_tag;

IF (in_url is null)
RETURN 1;
```

```
INSERT SOFT DB.DBA.EARLYBIRD_URLS(url, title, last_posted)
   VALUES (in_url, in_title, in_post_datetime);

UPDATE DB.DBA.EARLYBIRD_URLS SET title=in_title, last_posted=
   in_post_datetime WHERE url=in_url;
RETURN 2;
}
```

We implement the bookmark insert logic in the Storage tier with the Virtuoso/PL procedure language. Procedures are functions which we can call and pass parameters to from the Application and Data Maintenance tiers through SQL queries. We use Virtuoso/PL to enclose the insert process in a procedure. The bookmark insert is performed a procedure `EARLYBIRD_ADD_ENTRY`, see Listing 6.7.

The insert process implies three steps. First, we insert a row for each bookmark tag into `EARLYBIRD_ENTRIES`. Second, we increase the popularity score of the tag by 1 in the table `EARLYBIRD_SCORES` which is required by the Application tier for the recommender logic. Therefore, we insert the tag with score 0 if the tag is not already in the table and increase the value by 1 afterwards. Third, we update the title and the date of the URL and the date when the URL was last posted in the table `EARLYBIRD_URLS`. If the URL is not already in the table, we insert a new row with the date and URL title of the current bookmark. If no URL parameter is passed to the function, we skip the URL update step. The table `EARLYBIRD_URLS` is required by the Application tier for the search logic.

Listing 6.8: Insert bookmark procedure call

```
1 // GET JDBC CONNECTION
2 Connection con = dataSource.getConnection();
3 //...
4 // CLEAN UP TAGS
5 p_subject.replace(',', ' ');
6 p_subject = p_subject.trim();
7 p_subject = URLEncoder.encode(p_subject, "UTF-8");
8 // CALL ADD ENTRY PROCEDURE IN SQL
9 PreparedStatement entryStmt = con.prepareStatement("SELECT DB
   .DBA.EARLYBIRD_ADD_ENTRY(?,?,?,?,?,?,?)");
```

```
10 // PASS PARAMETERS
11 entryStmt.setString(1, post_uid);
12 entryStmt.setString(2, p_subject);
13 entryStmt.setString(3, p_url);
14 entryStmt.setString(4, URLEncoder.encode(p_username, "UTF-8")
       );
15 entryStmt.setString(5, dataProvider);
16 entryStmt.setString(6, p_dateTime);
17 entryStmt.setString(7, p_title_enc);
18 //EXECUTE QUERY AND CLOSE CONNECTION
19 entryStmt.execute();
20 entryStmt.close();
21 //...
22 con.close();
```

The procedure is called by the data packages of the Data Maintenance and Application tiers. Listing 6.8 shows how the procedure is called in the Data Maintenance tier.

Delete bookmarks

Every hour, we delete bookmarks which are older than 30 days and which don't belong to registered users. The delete process implies three steps. First, we delete all post entries from `EARLYBIRD_ENTRIES` which have been created more than 30 days ago and which were not created by registered users. Posts of registered users are labelled with „local“ in the *provider* column of `EARLYBIRD_ENTRIES`. Second, we clean up our table `EARLYBIRD_URLS`, which stores all URLs with their title and date. We delete all URLs which are not contained in `EARLYBIRD_ENTRIES` from `EARLYBIRD_URLS`. Third, we decrease the popularity score of all tags of the removed posts in `EARLYBIRD_SCORE`.

Listing 6.9: Delete bookmark procedure definition

```
create procedure DB.DBA.EARLYBIRD_DELETE_OLD_ENTRIES(){

DELETE FROM DB.DBA.EARLYBIRD_ENTRIES WHERE dateadd('day', 30,
    post_datetime) < now() AND provider <> 'local';
```

Table 6.4: SQL table EARLYBIRD_USERS

EARLYBIRD_USERS
user_uid (foreign key)
email
pwd
delicious_acc
connotea_acc
conf_code
conf_flag
cur_timestamp

```
DELETE FROM DB.DBA.EARLYBIRD_URLS WHERE NOT EXISTS (SELECT 1
   FROM DB.DBA.EARLYBIRD_ENTRIES WHERE (DB.DBA.
   EARLYBIRD_ENTRIES.url = DB.DBA.EARLYBIRD_URLS.url));
}
```

Listing 6.10: Decrease score trigger definition

```
create trigger DECREASE_SCORE after delete on DB.DBA.
   EARLYBIRD_ENTRIES referencing old as DELETEDROW{

UPDATE DB.DBA.EARLYBIRD_SCORES

SET DB.DBA.EARLYBIRD_SCORES.score = DB.DBA.EARLYBIRD_SCORES.
   score - 1 where DB.DBA.EARLYBIRD_SCORES.tag = DELETEDROW.
   tag;
}
```

We enclose the first two steps of the delete process into a Virtuoso/PL procedure, see Listing 6.9. The second step is implemented in an SQL trigger, which is triggered when a row in `EARLYBIRD_ENTRIES` is deleted, see Listing 6.10.

6.6.2 Account management

A further task of the Storage tier is user account management for registration and login. We store all registered users to the table `EARLYBIRD_USERS`, see Table 6.4. Each row in the table contains a user ID, email address, an MD5 hash of the password, the registration code , a flag whether the account has

been activated, a flag whether the user subscribed to the newsletter, and the registration date. Optionally, the row contains the connected Delicious and Connotea bookmarking account IDs.

The user account data are accessed from the Application tier through simple SQL queries, no performance issues exist here.

6.6.3 RDF user model publishing

A key functionality of the Storage tier is *user model publishing*. We publish the user models that have been created by our algorithms on the Web in our RDF user model schema. In the following section, we describe how we implemented creation and publishing of user models in the Storage tier.

The user model publishing process involves three steps. First, we create user model content in the Application tier and store the created user properties to an SQL table. Second, we map the content of the SQL table to our RDF user modeling schema with the Virtuoso Meta Schema Language. Third, we publish the RDF views along with our ontologies on the Web over a SPARQL endpoint.

(1) We create persistent user models for registered users through the recommender logic in the Application tier. We derive user models from the search queries and related recommendations. At present, our software implements the modeling of two categories of user properties: *user interest* and *user knowledge*. The user interest property models an assumed interest of a user in a tag. Interest is created from user search queries. We assume interest in all tags that have been searched by the user in Earlybird. The user knowledge property models individual users‘ knowledge of how two tags semantically relate. The input source for knowledge is our tag recommender, which implements our algorithms for tag similarity and tag hierarchy extraction (cf. Chapter 3). The recommender suggests related tags for all tags in the search queries. The recommender result also contains the relation type between two tags: associative, narrower, or broader. For the user knowledge, we assume the semantic relation that our recommender suggests.

To persist the user models – the search queries and related recommender

Table 6.5: SQL table EARLYBIRD_USERMODELS

EARLYBIRD_USERMODELS
id (primary key)
user_uid
search_tag
rec_tag
type
cur_timestamp

results – for all registered users, we log the output of the `Recommendation` servlet of the Earlybird component into the SQL table `EARLYBIRD_USERMODELS`, see Table 6.5. Each row of the table contains a unique row ID, the user ID, the search query tag, the related recommended tag, a numeric flag to indicate the relation type, and the current timestamp.

As the result of this step, we have the user models for registered users stored in our proprietary relational database schema.

(2) Now, we convert the user model content that we persisted in the SQL table `EARLYBIRD_USERMODELS` to our RDF schema for user modeling (cf. Chapter 5). This is the crucial step of our user model publishing process. Virtuoso provides a Meta Schema Language for defining a mapping of SQL data schemas to RDF ontologies. The declarative language allows to dynamically map data in SQL to arbitrary RDF schemas by defining RDF statements that contain data from SQL table columns in the SQL database schema.

Listing 6.11: URI class definition

```
sparql
prefix eb: <http://localhost:8890/earlybird/>
create iri class eb:uks_iri "http://localhost/earlybird/uks#%
   d" (in id integer not null) .
create iri class eb:uis_iri "http://localhost/earlybird/uis#%
   s" (in id varchar not null) .
create iri class eb:user_iri "http://localhost/earlybird/
   users#%s" (in user_uid varchar not null) .
create iri class eb:tr_iri "http://localhost/earlybird/tr#%d"
    (in id integer not null) . ;
```

The mapping from the SQL user models to our RDF user model schema involves two steps. In a first step, we define how URIs are created from SQL table columns. This step is required because subjects and objects of RDF statements must be URIs, by definition. The result of the mapping process are RDF statements. To map the content of SQL column to a RDF subject or predicate, we must define how the content of the column is converted to a URI. For example, when we want to map the column `EARLYBIRD_USERMODELS.user_uid` to an RDF subject in an RDF statement, we must convert the *varchar* content of the column, e.g., „Tom“, to a URI, `http://localhost/earlybird/users#tom`. We do this in a URI class definition script. We define URI patterns for specific column names in `EARLYBIRD_USERMODELS`, e.g., `http://localhost/earlybird/users#` for the column user_uid. The actual content of the columns is appended to these URI patterns later, e.g., `http://localhost/earlybird/users#tom`. Listing 6.11 shows the URI class definition script. The scripts defines four URIs that refer to the knowledge and interest statement, the user, and the topic relation.

Listing 6.12: RDF mapping

```
1  sparql
2  prefix qs: <http://localhost:8890/rdfv_demo/quad_storage/>
3  prefix cr: <http://earlybird.hki.uni-koeln.de/ontology/cr#>
4  prefix cco: <http://purl.org/ontology/cco/core#>
5  prefix dct:  <http://purl.org/dc/terms/>
6  prefix foaf: <http://xmlns.com/foaf/spec/>
7  prefix eb: <http://localhost:8890/earlybird/>
8  prefix skos: <http://www.w3.org/2008/05/skos#>
9
10 create quad storage qs:default from DB.DBA.
      EARLYBIRD_USERMODELS as um_tbl{
11
12 # User Knowledge Statements
13 create qs:knowledge as graph <http://localhost/um/knowledge>{
14     eb:uks_iri(um_tbl.id) a cco:CognitiveCharacteristic ;
15     cco:agent eb:user_iri(um_tbl.user_uid) ;
16     cco:topic eb:tr_iri(um_tbl.id) ;
```

```
17      cco:cognitiveCharacteristic cco:belief ;
18      dct:created um_tbl.cur_timestamp .
19      eb:tr_iri(um_tbl.id) a cr:ConceptRelation ;
20      cr:subj um_tbl.search_tag ;
21      cr:obj um_tbl.rec_tag ;
22      cr:relationType skos:semanticRelation where (^{um_tbl.}^.
           type = 0) ;
23      cr:relationType skos:narrower where (^{um_tbl.}^.type =
           1) ;
24      cr:relationType skos:broader where (^{um_tbl.}^.type = 2)
            .
25      eb:user_iri(um_tbl.user_uid) a foaf:Person .
26      } .
27
28  # User Interest Statements
29  create qs:interest as graph <http://localhost/um/interest>{
30      eb:uis_iri(um_tbl.post_uid) a cco:CognitiveCharacteristic
            ;
31      cco:agent eb:user_iri(um_tbl.user_uid) ;
32      cco:topic um_tbl.search_tag ;
33      cco:cognitiveCharacteristic cco:interest ;
34      dct:created um_tbl.cur_timestamp .
35      eb:user_iri(um_tbl.user_uid) a foaf:Person .
36      }
37  }.;
```

In a second step, we define the actual RDF statements that we create from the SQL columns to get our target RDF model. The target RDF model are User Interest Statements and User Knowledge Statements as described in Chapter 5. We store the RDF statements in two RDF graphs: `http://localhost/um/interest` and `http://localhost/um/knowledge` . Both reside in a quad store named `http://localhost:8890/rdfv_demo/quad_storage/default` which contains the mapped information. A mapping script written in the Meta Schema Language creates the quad store and the RDF graphs, see Listing 6.12. For each graph, the script defines patterns which describe the output RDF statements. The content of the SQL table can be integrated either as URIs which have been defined in the class definition script, cf. line 17, or as literal values, cf. line 19. The mapping patterns

can contain SQL expressions to restrict the creation of RDF statements to certain conditions. In lines 22-26, we use SQL expressions to define the relation type of the `cr:ConceptRelation` instance that we create. We set the relation type – *associative* (`skos:semanticRelation`), *broader* (`skos:broader`), or *narrower* (`skos:narrower`) – according to the value in the SQL column.

The result of the mapping process so far are two categories of User Property Statements: User Interest Statements and User Knowledge Statements. Both are modeled in our RDF user model schema. They are stored as RDF statements in a quad store on the Virtuoso database server.

(3) The goal of the user model publishing is to supply the data on the Web so that other services can query them. In a third step, we publish the User Property Statements on the Web, i.e., we make the data accessible through SPARQL queries. We implemented this functionality with a public SPARQL interface which allows to query the user Property Statements and additional RDF data required for a proper interpretation of the statements. Additional data contain a cached copy of the ontologies applied in the User Property Statements – our *Concept Relation Ontology*, FOAF, DC, SKOS, and CCO – and metadata, a description of the service that publishes the data. The SPARQL interface allows to query RDF across both data: the native RDF data and the mapped RDF views.

Listing 6.13: SPARQL query example

```
http://earlybird.uni-koeln.de:8890/sparql?query=select * from
    <http://localhost/um/interest> where { ?s ?p ?o }
```

The SPARQL interface of the Storage tier is available on the Web [13]) and handles HTTP requests. The HTTP requests must contain a SPARQL query in the *query* parameter. Listing 6.13 shows an example query to the SPARQL endpoint.

Listing 6.14: SPARQL query example 2

```
prefix cco: <http://purl.org/ontology/cco/core#>
select DISTINCT(?user) where {
?uis a cco:CognitiveCharacteristic ;
```

[13] http://earlybird.uni-koeln.de:8890/sparql

```
    cco:cognitiveCharacteristic cco:interest ;
    cco:topic "social_media";
    cco:agent ?user .
}
```

The interface can be used by Web services to retrieve User Property Statements, or to query on the knowledge created by the Application component. For example, a service could pass a SPARQL query to retrieve all users which have interest in a specific tag, see Listing 6.14.

Listing 6.15: Query response example

```
<rdf:RDF xmlns:res="http://www.w3.org/2005/sparql-results#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:nodeID="rset">
<rdf:type rdf:resource="http://www.w3.org/2005/sparql-results
   #ResultSet" />
    <res:resultVariable>user</res:resultVariable>
    <res:solution rdf:nodeID="r0">
      <res:binding rdf:nodeID="r0c0"><res:variable>user</res:
          variable><res:value>http://localhost/earlybird/users
          #tom</res:value></res:binding>
    </res:solution>
  </rdf:Description>
</rdf:RDF>
```

The response of the interface is the SPARQL result set. The HTTP request supports a *format* parameter to adapt the result format. For example, services can request RDF/XML to retrieve a result as shown in Listing 6.15

6.6.4 Usage logging

A further task of the database server is *usage logging*. In the current implementation, we do not use the logged data. However, we wanted to equip Earlybird for usage-based evaluation in future research experiments. Analysis of usage data can lead to deeper insights into explorative search. User-centric analysis can answer questions that are interesting starting points to improve Earlybird: *what do users search, do they tend to search for popular or unpopular tags, where does our recommender fail to make recommendations, how*

Table 6.6: SQL table EARLYBIRD_LOG_SEARCH

EARLYBIRD_LOG_SEARCH
id (primary key)
query
user_uid
session_id
cur_timestamp

Table 6.7: SQL table EARLYBIRD_LOG_RECOMMENDED

EARLYBIRD_LOG_ RECOMMENDED
id (primary key)
search_tag
rec_tag
score
is_novel
user_uid
cur_timestamp

do people use the navigation facilities. At present, we cannot answer these questions, because the number of users that actively use Earlybird is still too low. We hope, that we can record enough usage data over time to gain insight in the behavior of our users.

We log three types of usage information, which we believe to be appropriate for extensive usage data analyses: (1) search queries, (2) the returned recommendations for the queries, and (3) clicked recommendations. The

Table 6.8: SQL table EARLYBIRD_LOG_LIKED

EARLYBIRD_LOG_LIKED
id (primary key)
search_tag
rec_tag
score
is_novel
user_uid
session_id
cur_timestamp

click-through rate describes the effectiveness of the recommendation [79]. Tables 6.6, 6.7, and 6.8 show the logging tables. The storage of the logs does not raise any special problems and can be done over simple SQL queries from the *data* package of the Application tier.

6.7 Conclusion

In this chapter, we described Earlybird, our implementation of an exploratory tag search engine. Earlybird is similar to that envisioned in the introductory example. It renders webpages bookmarked in social bookmarking services searchable and facilitates exploratory browsing and focused search in social bookmarking data. It implements collaborative filtering algorithms for user model extraction and publishes created user models on the Web. Furthermore, Earlybird implements usage logging to enable user-centric evaluation of the exploratory search facilities in the future.

Earlybird shows how a prolific combination of exploratory browsing and focused search can be realized. It implements the key ideas of exploratory search, exploratory browsing and focused search, in a real-world application. Our vision was an exploratory search engine that integrates with social bookmarking services. We have shown that combining the ideas of social bookmarking and exploratory search is a fruitful approach. Earlybird can be seen either as an extension of social bookmarking with search facilities, or as an extension of a search engine through social bookmarking. Earlybird implements two facilities to enable exploratory search: exploratory browsing and focused search. The facilities support domain discovery and domain learning, realized mainly through adaptable tag recommendations visualized in an interactive tag network. The tag network enables domain discovery by presenting similar tags for the search query tags. For example, Tom, the user from our introductory example, can navigate the tag network starting from his initial search *social media*. The network allows to replace the query with recommendations to adapt the search. While Tom navigates through tags, new tags – *geofencing*, *foursquare* – are added to the networks and webpages are retrieved for these tags. Interactive browsing through the tag recommen-

dation network allows Tom to discover novel topics starting from his initial search context. The tag network also enables domain learning. Earlybird biases recommendations towards already known tags, i.e., tags that users have already used to search or bookmark a webpage. These known recommendations allow one to *contextualize* novel tags. For example, Tom could search for a topic which he is largely unfamiliar with: *geofencing*. Earlybird would recommend *mobile web* to Tom, because it is a similar tag and Tom has already searched for the tag in the past. The recommendation helps Tom to contextualize the novel tag with his existing knowledge, thus facilitates learning on the topic. Tom can also filter and specify results by adding more recommendations to the search query, e.g., *foursquare*. And Earlybird enables domain learning in a further way: the results are websites recently tagged in social bookmarking services, which update constantly while users contribute new bookmarks. Tom will find novel webpages on *social media* each time he queries for the tag, since users of Delicious and Connotea constantly contribute bookmarks to newly occurring webpages on the topic. Thus, Earlybird enables him to stay up-to-date with social media marketing and to increase his expertise on the topic over time.

Earlybird shows how exploratory search engines can profit from web-scale user modeling. It shows how user modeling can be implemented on a web-scale. We apply collaborative filtering to folksonomies to empower the recommendations for the tag recommendation network. Recommendations play an important role for exploratory search. We show how to create recommendations from folksonomies with collaborative filtering techniques. Hence, we show how social bookmarking services can be exploited for user modeling and ultimately used for exploratory search. Our collaborative filtering algorithms extract user models from folksonomies that yield successful recommendations for popular search queries. However, our evaluation results indicate that the recommendations are relatively sparse for unpopular tags (cf. Chapter 4). Therefore, we propose to integrate the user modeling with further Social Web services. We think that user models, recommendations, and ultimately the exploratory search facilities, can greatly profit from more and topically broader user data. This can only be realized by integrating user

models from multiple services. Earlybird connects with other services on the Social Web over RSS and Atom interfaces. From this perspective, we show how exploratory search engines can integrate with Social Web services to collect user data. Earlybird applies the established Web technologies, feeds and relational database models, to implement user model collection. From another perspective, Earlybird enhances the collected data and publishes newly created user models on the Web. We extract semantically explicit user models from folksonomies and publish them on the Web with Semantic Web technologies. Earlybird acts as a content provider for user models on the Social Web: it publishes information on interest and knowledge of users on the Web. These information are valuable for other exploratory search engines. Our vision is a Web where users can allow user modeling services to create and share user-specific information with other services to improve adaptation. Earlybird shows how user modeling services can share re-usable user models semantically explicit and with non-proprietary Semantic Web standards: ontology-based user models and SPARQL interfaces. From this perspective, Earlybird is a Semantic Web content provider.

Altogether, the technologies used in Earlybird comprise technologies from the service-oriented Social Web and from the data-oriented Semantic Web. With the integration of both technologies into our software, we show how to connect both technologies for prolific web-scale user modeling.

Chapter 7

Conclusions

Technological innovations have triggered strong cultural developments on the Web. The way we use the Web has greatly changed over the last five years. The Web has developed from a *digital library* to a *social* place – we not only retrieve information from the Web, but we also communicate and collaborate on the Web. The ubiquity of the Social Web and its collaborative character has greatly changed how we deal with information – and how we search. We take the Web with us; we retrieve information when we need it; we share information with others; we playfully explore and discover information – Web search becomes *ambient* [157]. The *digital library* metaphor isn't really fit to describe how we search and explore the Web anymore. Information Retrieval has yet to adopt these changes. One of the grand challenges of Information Retrieval over the course of the next few years will be to supply models and pragmatic solutions that suit the needs of the emerging Social Web. We believe that exploiting the social knowledge that underlies the Web, i.e., utilizing the communication and collaboration links between users, can impact Web search as much as the exploitation of the link structure between documents did 15 years ago.

In our thesis we have shown a strategy to exploit the social knowledge inherent in social bookmarking systems for exploratory search. In the following chapter, we conclude with a discussion of our research contributions. We summarize the main contributions of this thesis in Section 7.1 and discuss its

limitations, along with rewarding further research possibilities in Section 7.2.

7.1 Summary of contributions

The gain of this thesis comprises four contributions: a user model extraction method, a method to evaluate our user models, a method for web-scale user model integration, and a real-world exploratory tag search engine.

7.1.1 User model extraction

We have shown how to exploit social bookmarking systems to create user models. We presented a method for algorithmic user model extraction from folksonomies that applies collaborative filtering algorithms to create user models that contain user interest, user similarity, and semantic relations between tags: tag similarity and tag hierarchy. Our user model extraction method has the following benefits:

(a) it formalizes implicit knowledge and enhances the semantic expressivity of folksonomies.

(b) it exploits the structure of folksonomies for knowledge extraction, i.e., it avoids content analysis.

(c) it shows a completely automatized ontology creation and maintenance strategy.

(d) it creates ontologies without requiring existing domain knowledge, i.e., it is not limited to pre-defined knowledge domains.

(e) it bridges the gap between folksonomies and ontologies, i.e., it mediates between the Social Web and the Semantic Web.

7.1.2 User model evaluation

We have shown how to evaluate exploratory search engines. We presented a method to evaluate how well user models facilitate exploratory search. The

method measures structural properties of pre-compiled recommendations. The evaluation method has the following benefits:

(a) it evaluates non-accuracy measures, novelty and coverage, which are difficult to measure but important for exploratory search.

(b) it considers the impact of user models on the search results in a search engine.

(c) it allows to compare user modeling algorithms with low effort.

(d) it does not require user feedback data, i.e., algorithms can be evaluated before being deployed in a running system.

(e) it complements our exploratory tag search engine with a method that enables experimenting with different user modeling methods.

7.1.3 User model integration

We have shown how to create synergy effects for exploratory search by integrating user modeling services on the Web. We presented a method to integrate user models on a web-scale to enable better user modeling. The method comprises a web-scale user modeling architecture, a user model content schema, and a user model representation format. The user model integration method yields the following benefits:

(a) it syntactically and semantically aligns user models.

(b) it overcomes the sparsity problem by integrating user models from various sources.

(c) it aggregates RSS data from Social Web services and publishes Semantic Web content, i.e., it mediates between Social Web and the Semantic Web.

(d) it is a pragmatic method that can be implemented without revolutionary changes in existing Web services.

7.1.4 An exploratory tag search engine

We implemented our findings in a real-world exploratory tag search engine named Earlybird. Earlybird is a proof-of-concept of our findings. Furthermore, it is a software framework that enables further research on exploratory search in folksonomies. Our software has the following benefits:

(a) it implements explorative search facilities for folksonomies of two popular social bookmarking systems, Delicious and Connotea.

(b) it allows to easily implement and compare different algorithms.

(c) it collects usage logs to enable future user-centric evaluations.

(d) it allows to adjust and compare other components involved in exploratory search process: input data, user modeling algorithms, and the user interface.

(e) it is a real-world service that publishes user models to the Linked Data Web.

We believe that an important gain of this thesis results from the sum of the individual contributions. Our thesis exploits and advances academic research on the areas of Information Retrieval, user modeling, machine learning, and Linked Data publishing. We combine these different lines of research to address the challenges of exploratory search on the Social Web. The combination of usually disconnected techniques and methods allows for new perspectives on the problem and yields new insights.

7.2 Limitations and further research

The interdisciplinary perspective applied in this thesis is its key benefit. However, it makes necessary a certain degree of generality, which implies limitations. This thesis combines a range of problems, each of which deserves to be further researched. We covered several topics involved in web-scale

user modeling for exploratory search. Our goal was to contribute a real-world application. Clearly, each problem that was broached – user model extraction, evaluation, and integration – is complex enough to fill a thesis alone. However, our goal was to highlight how combining various research lines normally viewed as separate can produce new insights into the problem that underlies this thesis. We believe that the result, a real-world exploratory search engine, exhibits well-conceived paths that can be taken to address individual problems. This thesis provides a viable starting point for in-depth research on individual issues, and the software that we contribute provides a technical framework that promotes further research on this topic.

We propose only *some* important open endings of the problems covered in this thesis which suggest further research.

7.2.1 User model extraction

Our collaborative filtering user model extraction method implements algorithms that very much rely on established algorithms. Our evaluation and related research suggests that new collaborative algorithms can improve the recommendations in the context of folksonomies. A number of more specialized collaborative filtering algorithms have been developed recently that leverage the properties of folksonomies better [238, 163].

An important aspect of our user model extraction method is the semantic enhancement of folksonomies through tag relations. We extract semantics that are user-specific and formalize these semantics in ontologies. Promising research on hierarchy extraction from folksonomies and semantic enhancement exists [8, 111, 183, 176]. It would be interesting to evaluate how different approaches perform in the context of exploratory search.

A limitation of our knowledge extraction method is that extracted semantics remain user-specific. This reveals a gap between the folksonomy world and the ontology world that our method does not bridge: our semantics have still limited shared meaning. Our ontologies are personal, going against the spirit of ontologies and their reliance on shared meaning [153]. A viable further research direction would be to map our personal semantics

onto ontologies shared amongst user communities or even amongst all users. Promising strategies for automated ontology mapping are being developed which could be integrated into our user model extraction method [68, 55].

7.2.2 User model evaluation

An important benefit of our user model evaluation method is that it can be applied before any user feedback is available. This allows the evaluation of algorithms *offline*, i.e., before they are deployed. Our method applies network-centric evaluation measures. A limitation of our evaluation is that it does not cover all possible aspects of quality. The user-perceived quality of the recommendations is an important aspect for exploratory search [189, 50]. Our method is limited to the navigability of the recommendations and corresponding search results. Other aspects of our system are also important to assess the exploratory search facilities: the user interface, for example, clearly plays a central role for how users perceive the quality of the recommendations. Our method does not allow one to measure the user-perceived quality of exploratory search engines. We suggest using a user-centric analysis to complement our evaluation method, i.e., an evaluation of user feedback data. We have provided the ground for further research on user-centric evaluations by collecting usage logs in Earlybird.

7.2.3 User model integration

User modeling affects the domains of two research communities. First, user model extraction and processing is largely driven by machine learning. Second, web-scale user model integration currently engages the Linked Data community. The developments originate from different historical backgrounds and therefore have different requirements in regards to the content and the representation of user models. User model extraction and processing methods heavily rely on machine learning techniques which require *simple* user models that can be processed efficiently and fast. This line of research focuses more on the quantity rather than quality of information. User modeling integration originates, for the most part, from human-centered research focused

on the quality of user models. User models are typically more elaborate and complex. Our user modeling method mediates between both camps. We believe that it is viable to further connect both lines of research to find more elaborate user model integration strategies that comply with the needs of both research areas.

Privacy is an important challenge for web-scale user modeling [130]. As user modeling becomes more ubiquitous and users become more elaborate on the Web, they must be given appropriate control to determine who collects their data and what data is to be collected. In this thesis, we have taken the route towards openness and transparency. We apply our user model extraction on bookmarks which have been made available on the Web explicitly by users. Additionally, we use open standards for user model publishing. However, a web-scale user modeling effort must go further. For example, it should require automized control to determine who is allowed to collect user data and how aggregated user data can be published. Ideas and first implementations of *trust* that point in this direction have already emerged in the Semantic Web community [83, 206].

7.2.4 Theory and practice

Finally, we believe that research on Information Retrieval on the Web can profit considerably from experimental research. Innovation on the Web is largely driven by real-world applications. The Web is an open world, where technological evolution involves convincing many parties to adopt these new inventions. We think that the impact of academic innovations on the Web can be increased if they are implemented and demonstrated in real-world applications that prove and illustrate their surplus value. Our thesis aims at embracing academic research and pragmatic application.

Bibliography

[1] Fabian Abel, Juri De Coi, Nicola Henze, Arne Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *The Semantic Web*, Lecture Notes in Computer Science, chapter 1, pages 1–14. Springer, 2008.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Personalization technologies: a process-oriented perspective. *Commun. ACM*, 48(10):83–90, 2005.

[3] Stefano Aguzzoli, Paolo Avesani, and Paolo Massa. Collaborative case-based recommender system. In *Proceedings of the ECCBR Conference*, pages 460–474, 2002.

[4] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[5] Sarabjot Anand and Bamshad Mobasher. *Intelligent Techniques for Web Personalization*. Springer, 2005.

[6] Antonio Anaya and Jesús Boticario. Content-free collaborative learning modeling using data mining. *User Modeling and User-Adapted Interaction*, pages 1–36, January 2011.

[7] Anton Andrejko, Michal Barla, and Maria Bielikova. Ontology-based user modeling for web-based information systems. In *Advances in Information Systems Development*, chapter 38, pages 457–468. Springer US, Boston, MA, 2007.

[8] Sofia Angeletou, Marta Sabou, and Enrico Motta. Semantically enriching folksonomies with FLOR. In *Proceedings of the 5th ESWC. workshop: Collective Intelligence and the Semantic Web*, 2008.

[9] G. Antoniou and Frank Van Harmelen. *A Semantic Web primer*. MIT Press, Cambridge, Mass., 2nd edition, 2008.

[10] Liliana Ardissono, Luca Console, and Ilaria Torre. An adaptive system for the personalized access to news. *AI Commun.*, 14(3):129–147, 2001.

[11] Liliana Ardissono and Anna Goy. Tailoring the interaction with users in web stores. *User Modeling and User-Adapted Interaction*, 10(4):251–303, 2000.

[12] C. M. Au-Yeung, N. Gibbins, and N. Shadbolt. A study of user profile generation from folksonomies. In *Proceedings of the WWW 2008 Social Web and Knowledge Management, Social Web Workshop*, April 2008.

[13] Sören Auer, Sebastian Dietzold, and Thomas Riechert. Ontowiki - a tool for social, semantic collaboration. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.

[14] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval*. ACM Press books. ACM Press, New York, 1999.

[15] Jie Bao and Vasant Honavar. Collaborative ontology building with wiki@nt: Multi agent based ontology. Technical report, Iowa State University, 2004.

[16] Shenghua Bao, Guirong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing web search using social annotations. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 501–510, New York, NY, USA, 2007. ACM.

[17] John Battelle. *The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture*. Portfolio Hardcover, September 2005.

[18] Ian H. Beaumont. User modelling in the interactive anatomy tutoring system anatom-tutor. *User Modeling and User-Adapted Interaction*, 4(1):21–45–45, March 1994.

[19] Nicholas J. Belkin. Some(what) grand challenges for information retrieval. *SIGIR Forum*, 42(1):47–54, June 2008.

[20] P. Berkhin. A survey of clustering data mining techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle, editors, *Grouping Multidimensional Data*, chapter 2, pages 25–71. Springer-Verlag, Berlin/Heidelberg, 2006.

[21] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, 18(3):245–286, August 2008.

[22] T. Berners-Lee, J. Hendler, O. Lassila, and Others. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[23] Bin Bi, Lifeng Shang, and Ben Kao. Collaborative resource discovery in social tagging systems. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 1919–1922, New York, NY, USA, 2009. ACM.

[24] Mária Bieliková. M.: Personalized faceted navigation for multimedia collections. In *Proceedings of the 2nd Int. Workshop on Semantic Media Adaptation and Personalization*, 2007.

[25] Mária Bieliková and Jaroslav Kuruc. Sharing user models for adaptive hypermedia applications. In *ISDA '05: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 506–513, Washington, DC, USA, 2005. IEEE Computer Society.

[26] Daniel Billsus and Michael J. Pazzani. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10(2-3):147–180, 2000.

[27] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2007.

[28] Andreas Blumauer. *Social Semantic Web: Web 2.0 - was nun?* X.media.press. Springer, Berlin, 2009.

[29] Toine Bogers. *Recommender Systems for Social Bookmarking*. PhD thesis, Tilburg University, December 2009.

[30] Craig Boyle and Antonio O. Encarnacion. Metadoc: An adaptive hypertext reading system. *User Modeling and User-Adapted Interaction*, 4(1):1–19, March 1994.

[31] T. Brailsford, C. Stewart, M. Zakaria, and A. Moore. Autonavigation, links and narrative in an adaptive web-based integrated learning environment. In *Eleventh International World Wide Web Conference*, May 2002.

[32] Giorgio Brajnik, Giovanni Guida, and Carlo Tasso. User modeling in intelligent information retrieval. *Inf. Process. Manage.*, 23(4):305–320, 1987.

[33] Ulrick Brandes and Thomas Erlebach. Network analysis. methodological foundations. *Network Analysis, Lecture Notes in Computer Science*, 3418, 2005.

[34] Boris Brandherm and Michael Schmitz. Presentation of a modular framework for interpretation of sensor data with dynamic Bayesian networks on mobile devices. In A. Abecker, S. Bickel, U. Brefeld, I. Drost, N. Henze, O. Herden, M. Minor, T. Scheffer, L. Stojanovic, and S. Weibelzahl, editors, *LWA 2004, Lernen Wissensentdeckung Adaptivität*, pages 9–10, 2004.

[35] Boris Brandherm and Tim Schwartz. Geo referenced dynamic Bayesian networks for user positioning on mobile systems. In Thomas Strang and Claudia Linnhoff-Popien, editors, *Proceedings of the International Workshop on Location- and Context-Awareness (LoCA), LNCS 3479*, pages 223–234, Munich, Germany, 2005. Springer-Verlag Berlin Heidelberg.

[36] John G. Breslin, Alexandre Passant, and Stefan Decker. *The Social Semantic Web*. Springer, 1 edition, 2009.

[37] Christopher Brooks, Mike Winter, Jim Greer, and Gordon Mccalla. The massive user modelling system (mums). In *In: Proceedings of the Seventh International Conference on Intelligent Tutoring Systems*, pages 635–645, 2004.

[38] Brunkhorst, F. Abel, A. Arnaiz, M. Baldoni, C. Baroglio, N. Henze, M. Klopotek, and V. Patti. Personal information systems in the semantic web- lesson learnt and best practices. Technical report, University of Hannover, 2007.

[39] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2):87–129, July 1996.

[40] Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1):87–110–110, March 2001.

[41] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web: Methods and Strategies of Web Personalization (Lecture Notes in Computer Science / Information Systems and Applications, incl. Internet/Web, and HCI)*. Springer, 1 edition, June 2007.

[42] Peter Brusilovsky and Mark T. Maybury. From adaptive hypermedia to the adaptive web. *Commun. ACM*, 45(5):30–33, May 2002.

[43] Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321

of *Lecture Notes in Computer Science*, chapter 1, pages 3–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[44] Francesca Carmagnola, Federica Cena, Luca Console, Omar Cortassa, Cristina Gena, Anna Goy, Ilaria Torre, Andrea Toso, and Fabiana Vernero. Tag-based user modeling for social multi-device adaptive guides. *User Modeling and User-Adapted Interaction*, 18(5):497–538, November 2008.

[45] Francesca Carmagnola, Federica Cena, Omar Cortassa, Cristina Gena, and Ilaria Torre. Towards a tag-based user model: How can user model benefit from tags? In *UM2007, User Modeling: Proceedings of the Eleventh International Conference*, volume 4511 of *Lecture Notes in Computer Science*, pages 445–449, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[46] David Carmel, Naama Zwerdling, Ido Guy, Shila O. Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized social search based on the user's social network. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1227–1236, New York, NY, USA, 2009. ACM.

[47] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):247–267, December 2005.

[48] C. A. Carver, R. A. Howard, and W. D. Lane. Enhancing student learning through hypermedia courseware and incorporation of student learning styles. *IEEE Transactions on Education*, 42(1):33–38, Feb 1999.

[49] C. Cattuto, D. Benz, A. Hotho, and G. Stumme. Semantic analysis of tag similarity measures in collaborative tagging systems. In *Proceedings of the 3rd Workshop on Ontology Learning and Population (OLP3)*, July 2008.

[50] O. Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, 2008.

[51] Òscar Celma and Perfecto Herrera. A new approach to evaluating novel recommendations. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 179–186, New York, NY, USA, 2008. ACM.

[52] Ramnath K. Chellappa and Raymond G. Sin. Personalization versus privacy: An empirical examination of the online consumers dilemma. *Information Technology and Management*, 6(2):181–202, April 2005.

[53] Weiqin Chen and Riichiro Mizoguchi. Communication content ontology for learner model agent in multi-agent architecture. In *Proceedings of the 7th International Conference on Computers in Education*, pages 95–102, 1999.

[54] Ed Chi, Peter Pirolli, and Shyong Lam. Aspects of augmented social cognition: Social information foraging and social search. In *Proceedings of the 2nd international conference on Online communities and social computing*, pages 60–69, 2007.

[55] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41, September 2006.

[56] Sheung-On Choy and A. K. Lui. Web information retrieval in collaborative tagging systems. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 352–355, Washington, DC, USA, December 2006. IEEE Computer Society.

[57] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[58] C. Cool. Issues of context in information retrieval (IR): an introduction to the special issue. *Information Processing & Management*, 38(5):605–611, September 2002.

[59] Crestani, Fabio, Ruthven, and Ian. Introduction to special issue on contextual information retrieval systems. *Information Retrieval*, 10(2):111–113, April 2007.

[60] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 271–280, New York, NY, USA, 2007. ACM.

[61] J. Davies, Dieter Fensel, and Frank Van Harmelen. *Towards the semantic web: ontology-driven knowledge management*. J. Wiley, Chichester, England, 2003.

[62] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7–7, February 2001.

[63] Li Ding, Pranam Kolari, Zhongli Ding, and Sasikanth Avancha. Using ontologies in the semantic web: A survey. In Raj Sharman, Rajiv Kishore, and Ram Ramesh, editors, *Ontologies*, volume 14 of *Integrated Series in Information Systems*, pages 79–113. Springer US, 2007.

[64] Leigh Dodds and Ian Davis. Linked data patterns: A pattern catalogue for modelling, publishing, and consuming linked data. Published online, 2010.

[65] Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series)*. Addison Wesley, May 2005.

[66] P. Dolog and W. Nejdl. Challenges and benefits of the semantic web for user modelling. In *Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems 2003*, 2003.

[67] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, February 2004.

[68] Marc Ehrig. *Ontology alignment: bridging the semantic gap*, volume 4. Springer, New York, 2007.

[69] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Inter. Tech.*, 3(1):1–27, February 2003.

[70] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Axel Polleres. Rules and ontologies for the semantic web. In *Reasoning Web*, pages 1–53. Springer, 2008.

[71] Miguel Encarnaç. Multi-level user support through adaptive hypermedia: a highly application-independent help component. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 187–194, New York, NY, USA, 1997. ACM.

[72] Usama M. Fayyad, Georges G. Grinstein, and Andreas Wierse. *Information visualization in data mining and knowledge discovery*. Morgan Kaufmann Publishers, San Francisco, 2002.

[73] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[74] François Fouss and Marco Saerens. Evaluating performance of recommender systems: An experimental comparison. In *WI-IAT '08: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 735–738, Washington, DC, USA, 2008. IEEE Computer Society.

[75] Wai T. Fu, Thomas G. Kannampallil, and Ruogu Kang. Facilitating exploratory search by model-based navigational cues. In *Proceedings of the 14th international conference on Intelligent user interfaces*, IUI '10, pages 199–208, New York, NY, USA, 2010. ACM.

[76] George W. Furnas, Caterina Fake, Luis von Ahn, Joshua Schachter, Scott Golder, Kevin Fox, Marc Davis, Cameron Marlow, and Mor Naaman. Why do tagging systems work? In *CHI '06 extended abstracts on Human factors in computing systems*, CHI '06, pages 36–39, New York, NY, USA, 2006. ACM.

[77] Susan Gauch, Jason Chaffee, and Alexander Pretschner. Ontology-based personalized search and browsing. *Web Intelligence and Agent Systems*, 1:1–3, 2003.

[78] Susan Gauch, Mirco Speretta, Aravind Chandramouli, and Alessandro Micarelli. User profiles for personalized information access. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 2, pages 54–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[79] Mouzhi Ge, Carla D. Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 257–260, New York, NY, USA, 2010. ACM.

[80] G. Gentili, A. Micarelli, and F. Sciarrone. Infoweb: An adaptive information filtering system for the cultural heritage domain. *Applied Artificial Intelligence*, pages 715–744, 2003.

[81] Eric J. Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. Using web structure for classifying and describing web pages. In *Proceedings of WWW-02, International Conference on the World Wide Web*, 2002.

[82] Daniela Godoy and Analía Amandi. Modeling user interests by conceptual clustering. *Inf. Syst.*, 31(4):247–265, June 2006.

[83] Jennifer Golbeck, Bijan Parsia, and James Hendler. Trust networks on the semantic web. In *Proceedings of Cooperative Intelligent Agents*, pages 238–249, 2003.

[84] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

[85] S. Golder and B. A. Huberman. The Structure of Collaborative Tagging Systems. *Arxiv preprint cs.DL/0508082*, 2005.

[86] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198, 2006.

[87] Rafael A. Gonzales, Nong Chen, and Ajantha Dahanayake. *Personalized information retrieval and access : concepts, methods and practices.* Information Science Reference, Hershey, 2008.

[88] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[89] Mike Graves, Adam Constabaris, and Dan Brickley. Foaf: Connecting people on the semantic web. *Cataloging & classification quarterly*, 43(3):191–202, April 2007.

[90] T. Gruber. Where the Social Web Meets the Semantic Web. *Lecture notes in computer science*, 4273:994, 2006.

[91] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199, 1993.

[92] Thomas Gruber. Ontology of folksonomy: A mash-up of apples and oranges. *International Journal on Semantic Web & Information Systems*, 3(2):1–11, 2007.

[93] N. Guarino. Formal ontology and information systems, 1998.

[94] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, December 2009.

[95] Volker Haarslev and Ralf Möller. Racer: An owl reasoning agent for the semantic web. In *In Proc. of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with 2003 IEEE/WIC International Conference on Web Intelligence*, volume 13, pages 91–95, 2003.

[96] T. Hammond, T. Hannay, B. Lund, and J. Scott. Social Bookmarking Tools (I). http://www.dlib.org/dlib/april05/hammond/04hammond.html [last accessed: november 2008], 2005.

[97] Xiaogang Han, Zhiqi Shen, Chunyan Miao, and Xudong Luo. Folksonomy-Based ontological user interest profile modeling and its application in personalized search. In Aijun An, Pawan Lingras, Sheila Petty, and Runhe Huang, editors, *Active Media Technology*, volume 6335 of *Lecture Notes in Computer Science*, chapter 6, pages 34–46–46. Springer, Berlin, Heidelberg, 2010.

[98] S. P. Harter. Psychological relevance and information science. *J. Am. Soc. Inf. Sci.*, 43(9):602–615, 1992.

[99] Dominik Heckmann. Introducing situational statements as an integrating data structure for user modeling, context-awareness and resource-adaptive computing. In Andreas Hoto and Gerd Stumme, editors, *LLWA Lehren - Lernen - Wissen - Adaptivität (ABIS2003)*, pages 283–286, 2003.

[100] Dominik Heckmann, Tim Schwartz, Boris Br, and Er Kröner. Decentralized user modeling with UserML and GUMO. In *Proceedings of the Workshop on Decentralized, Agent Based and Social Approaches to User Modelling*, pages 61–65, 2005.

[101] Dominik Heckmann, Tim Schwartz, Boris Brandherm, Michael Schmitz, and Margeritta von Wilamowitz-Moellendorff. Gumo – the general user model ontology. In *User Modeling 2005*, pages 428–432. Springer, Berlin / Heidelberg, 2005.

[102] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM.

[103] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[104] Paul Heymann and Hector Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report 2006-10, Stanford University, April 2006.

[105] Paul Heymann, Georgia Koutrika, and Hector G. Molina. Can social bookmarking improve web search? In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 195–206, New York, NY, USA, 2008. ACM.

[106] Paul Heymann, Daniel Ramage, and Hector G. Molina. Social tag prediction. In Sung H. Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat S. Chua, Mun K. Leong, Sung H. Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat S. Chua, and Mun K. Leong, editors, *SIGIR*, pages 531–538, New York, NY, USA, 2008. ACM.

[107] T. Hirashima, K. Hachiya, A. Kashihara, and J. Toyoda. Information filtering using user‘s context on browsing in hypertext. *User Modeling and User-Adapted Interaction*, 7(4):239–256, 1997.

[108] Pascal Hitzler. *Semantic Web: Grundlagen.* eXamen.press. Springer, Heidelberg, 2008.

[109] Vera Hollink, Maarte Someren, and Bob J. Wielinga. Discovering stages in web navigation for problem-oriented navigation support. *User Modeling and User-Adapted Interaction*, 17(1-2):183–214, March 2007.

[110] Ian Horrocks. Ontologies and the semantic web. *Commun. ACM*, 51(12):58–67, December 2008.

[111] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Das Entstehen von Semantik in BibSonomy. *Social Software in der Wertschöpfung. Nomos*, 2006.

[112] A. Hotho, R. Jaschke, C. Schmitz, and G. Stumme. Information Retrieval in Folksonomies: Search and Ranking. *Lecture notes in computer science*, 4011:411, 2006.

[113] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[114] Mohsen Jamali and Hassan Abolhassani. Different aspects of social network analysis. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, pages 66–72, December 2006.

[115] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, pages 506–514. Springer-Verlag, Berlin, Heidelberg, 2007.

[116] Ae-Ttie Ji, Cheol Yeon, Heung-Nam Kim, and Geun-Sik Jo. Collaborative tagging in recommender systems. In *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*, pages 377–386, 2007.

[117] Xin Jin, Yanzan Zhou, and Bamshad Mobasher. Task-oriented web user modeling for recommendation. In *User Modeling*, volume 3538, pages 109–118, 2005.

[118] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 154–161, New York, NY, USA, 2005. ACM.

[119] Jason Jung and Geun-Sik Jo. Extracting user interests from bookmarks on the web. In *Proceedings of the 7th Pacific-Asia conference on Advances in knowledge discovery and data mining*, page 568, 2003.

[120] S. Jung, J. Herlocker, and J. Webster. Click data as implicit relevance feedback in web search. *Information Processing & Management*, 43(3):791–807, May 2007.

[121] Yvonne Kammerer, Rowan Nairn, Peter Pirolli, and Ed H. Chi. Signpost from the masses: learning effects in an exploratory social tag search browser. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 625–634, New York, NY, USA, 2009. ACM.

[122] Kay. Ontologies for reusable and scrutable student models. *AIED Workshop W2: Workshop on Ontologies for Intelligent Educational Systems*, pages 72–77, 1999.

[123] Diane Kelly. Methods for evaluating interactive information retrieval systems with users. *Foundations and Trends in Information Retrieval*, 3(1—2):1–224, 2009.

[124] Hyoung-rae Kim and Philip Chan. Personalized search results with user interest hierarchies learnt from bookmarks. In Olfa Nasraoui, Osmar Zaïane, Myra Spiliopoulou, Bamshad Mobasher, Brij Masand, and Philip S. Yu, editors, *Advances in Web Mining and Web Usage Analysis*, volume 4198, chapter 9, pages 158–176. Springer, Berlin, Heidelberg, 2006.

[125] Hyoung-Rae Kim and Philip Chan. Learning implicit user interest hierarchy for context in personalization. *Applied Intelligence*, 28(2):153–166, April 2008.

[126] Jon M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845, August 2000.

[127] Jonathan Klinginsmith, Malika Mahoui, Yuqing Wu, and Josette Jones. Discovering domain specific concepts within user-generated taxonomies. In *ICDMW '09: Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, pages 19–24, Washington, DC, USA, 2009. IEEE Computer Society.

[128] Evgeny Knutov, Paul De Bra, and Mykola Pechenizkiy. Ah 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques. *New Review of Hypermedia and Multimedia*, 15(1):5–38, 2009.

[129] Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1):49–63, March 2001.

[130] Alfred Kobsa. Privacy-enhanced personalization. *Commun. ACM*, 50(8):24–33, August 2007.

[131] Alfred Kobsa, Jürgen Koenemann, and Wolfgang Pohl. Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16:111–155, 2001.

[132] Joseph A. Konstan. Introduction to recommender systems: Algorithms and evaluation. *ACM Trans. Inf. Syst.*, 22(1):1–4, January 2004.

[133] S. Koshman. Visualization-based information retrieval on the web. *Library & Information Science Research*, 28(2):192–207, FebFeb 2006.

[134] Beate Krause, Andreas Hotho, and Gerd Stumme. A Comparison of Social Bookmarking with Traditional Search. In *Proceedings of the IR research, 30th European conference on Advances in information retrieval*, pages 101–113, 2008.

[135] Michael Kruppa, Dominik Heckmann, and Antonio Krüger. Adaptive multimodal presentation of multimedia content in museum scenarios. *KI*, 19(1), 2005.

[136] R. Lambiotte and M. Ausloos. Collaborative tagging as a tripartite network. *Lecture notes in computer science*, 3993, 2006.

[137] Ray R. Larson. Introduction to information retrieval. *Journal of the American Society for Information Science and Technology*, 61(4):852–853, 2010.

[138] Steve Lawrence. Context in web search. *IEEE Data Engineering Bulletin*, 23(3):25–32, 2000.

[139] Man Li, Dazhi Wang, Xiaoyong Du, and Shan Wang. Ontology construction for semantic web: A Role-Based collaborative development method. In *Web Technologies Research and Development - APWeb 2005*, Lecture Notes in Computer Science, pages 609–619. Springer, 2005.

[140] Xin Li, Lei Guo, and Yihong E. Zhao. Tag-based social interest discovery. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 675–684, New York, NY, USA, 2008. ACM.

[141] Huizhi Liang, Yue Xu, Yuefeng Li, and Richi Nayak. Tag based collaborative filtering for recommender systems. In Peng Wen, Yuefeng Li, Lech Polkowski, Yiyu Yao, Shusaku Tsumoto, and Guoyin Wang, editors, *Rough Sets and Knowledge Technology*, volume 5589, chapter 84, pages 666–673. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[142] Fang Liu, C. Yu, and Weiyi Meng. Personalized web search for improving retrieval effectiveness. *Knowledge and Data Engineering, IEEE Transactions on*, 16(1):28–40, 2004.

[143] Pasquale Lops, Marco Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco

Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, chapter 3, pages 73–105. Springer US, Boston, MA, 2011.

[144] B. Lund, T. Hammond, M. Flack, T. Hannay, and I. NeoReality. Social Bookmarking Tools (II). http://dlib.org/dlib/april05/lund/04lund.html [last accessed: november 2008], 2005.

[145] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2):72–79, March 2001.

[146] Gary Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, April 2006.

[147] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. Ht06, tagging paper, taxonomy, flickr, academic article, to read. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 31–40, New York, NY, USA, 2006. ACM.

[148] Lorraine McGinty and Barry Smyth. On the role of diversity in conversational recommender systems. In *Proceedings of the 5th international conference on Case-based reasoning: Research and Development*, ICCBR'03, pages 276–290, Berlin, Heidelberg, 2003. Springer-Verlag.

[149] Sean M. Mcnee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06: Extended Abstracts on Human Factors in Computing Systems*, pages 1097–1101, New York, NY, USA, 2006. ACM.

[150] Alessandro Micarelli, Fabio Gasparetti, Filippo Sciarrone, and Susan Gauch. Personalized search on the world wide web. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 195–230. Springer-Verlag, 2007.

[151] Alessandro Micarelli and Filippo Sciarrone. A case-based system for adaptive hypermedia navigation. In *EWCBR '96: Proceedings of*

the Third European Workshop on Advances in Case-Based Reasoning, pages 266–279, London, UK, 1996. Springer.

[152] Peter Mika. Ontologies are us: A unified model of social networks and semantics. *Web Semant.*, 5(1):5–15, March 2007.

[153] Peter Mika. *Social Networks and the Semantic Web (Semantic Web and Beyond)*. Springer, September 2007.

[154] Alistair Miles and José R. Pérez-Agüera. Skos: Simple knowledge organisation for the web. *Cataloging & Classification Quarterly*, 43(3):69–83, 2007.

[155] David Millen, Meng Yang, Steven Whittaker, and Jonathan Feinberg. Social bookmarking and exploratory search. In Liam Bannon, Ina Wagner, Carl Gutwin, Richard Harper, and Kjeld Schmidt, editors, *ECSCW 2007*, pages 21–40. Springer London, 2007.

[156] Roberto Mirizzi and Tommaso Di Noia. From exploratory search to web search and back. In *Proceedings of the 3rd workshop on Ph.D. students in information and knowledge management*, PIKM '10, pages 39–46, New York, NY, USA, 2010. ACM.

[157] Peter Morville. *Ambient Findability: What We Find Changes Who We Become*. O'Reilly Media, 1 edition, September 2005.

[158] Maurice D. Mulvenna, Sarabjot S. Anand, and Alex G. Büchner. Personalization on the net using web mining: introduction. *Commun. ACM*, 43(8):122–125, 2000.

[159] Roberto Navigli and Paola Velardi. Learning domain ontologies from document warehouses and dedicated web sites. *Comput. Linguist.*, 30(2):151–179, June 2004.

[160] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167, Mar 2003.

[161] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. http://www.ksl.stanford.edu/people/dim/papers/ontology-tutorial-noy-mcguinessabstract.html [last accessed: november 2008], March 2001.

[162] Sean Owen and Robin Anil. *Mahout in action (MEAP)*. Manning, 2010.

[163] Denis Parra and Peter Brusilovsky. Collaborative filtering for social tagging systems: an experiment with citeulike. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 237–240, New York, NY, USA, 2009. ACM.

[164] Michael Pazzani and Daniel Billsus. Content-Based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 325–341, 2007.

[165] Daniela Petrelli, Elena Not, Massimo Zancanaro, Carlo Strapparava, and Oliviero Stock. Modelling and adapting to context. *Personal Ubiquitous Comput.*, 5(1):20–24, 2001.

[166] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, and Constantine D. Spyropoulos. Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13(4):311–372, November 2003.

[167] Helena S. Pinto and J. P. Martins. Ontologies how can they be built. *Knowledge and Information Systems*, V6(4):441–464, July 2004.

[168] Peter Pirolli and Stuart Card. Information foraging. *Psychological Review*, 106(4):643–675, October 1999.

[169] M. C. Polson and J. J. Richardson, editors. *Foundations of intelligent tutoring systems*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1988.

[170] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Technical report, W3C, 2008.

[171] Feng Qiu and Junghoo Cho. Automatic identification of user interest for personalized search. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 727–736, New York, NY, USA, 2006. ACM.

[172] Daniel Ramage, Paul Heymann, Christopher D. Manning, and Hector G. Molina. Clustering the tagged web. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 54–63, New York, NY, USA, 2009. ACM.

[173] Liana Razmerita, Albert Angehrn, and Alexander Maedche. Ontology-based user modeling for knowledge management systems. In Peter Brusilovsky, Albert Corbett, and Fiorella Rosis, editors, *User Modeling 2003*, volume 2702 of *Lecture Notes in Computer Science*, chapter 29, page 148. Springer Berlin Heidelberg, Berlin, Heidelberg, June 2003.

[174] Luis F. Revilla and Frank M. Shipman. Managing conflict in multi-model adaptive hypertext. In *HYPERTEXT '04: Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, pages 237–238, New York, NY, USA, 2004. ACM.

[175] David Robins. Interactive information retrieval: Context and basic notions. *Informing Science Journal*, 3:57–62, 2000.

[176] Mike Rohland and Olga Streibel. Algorithmic extraction of tag semantics. Technical report, Networked Information Systems, Free University Berlin, 2009.

[177] D. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*, pages 13–19, 2004.

[178] Ian Ruthven. Interactive information retrieval. *Annual Review of Information Science and Technology*, 42(1):43–91, 2008.

[179] Stefan Sackmann, Jens Strüker, and Rafael Accorsi. Personalization in privacy-aware highly dynamic systems. *Commun. ACM*, 49(9):32–38, September 2006.

[180] Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, New York, 1983.

[181] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[182] J. Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative Filtering Recommender Systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 9, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[183] Christoph Schmitz, Andreas Hotho, Robert Jäschke, and Gerd Stumme. *Mining Association Rules in Folksonomies*, pages 261–270. Springer, 2006.

[184] Stephen J. Schultze. A collaborative foraging approach to web browsing enrichment. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 860–861, New York, NY, USA, 2002. ACM Press.

[185] Ingo Schwab, Alfred Kobsa, and Ivan Koychev. Learning user interests through positive examples using content analysis and collaborative filtering. Technical Report 30 2001, GMD, 2001.

[186] Eric Schwarzkopf, Dominik Heckmann, Dietmar Dengler, and Alexander Kroner. Mining the Structure of Tag Spaces for User Modeling. In *Proceedings of the Workshop on Data Mining for User Modeling at the 11th International Conference on User Modeling*, pages 63–75, 2007.

[187] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[188] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender Systems Handbook*, pages 257–297. Springer, 2011.

[189] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, chapter 8, pages 257–297. Springer US, Boston, MA, 2011.

[190] Xuehua Shen, Bin Tan, and Chengxiang Zhai. Implicit user modeling for personalized search. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 824–831, New York, NY, USA, 2005. ACM.

[191] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[192] Gene Smith. *Tagging: People-powered Metadata for the Social Web (Voices That Matter)*. New Riders Press, December 2007.

[193] Lucia Specia and Enrico Motta. Integrating folksonomies with the semantic web. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, pages 624–639, 2007.

[194] M. Speretta and S. Gauch. Personalized search based on user search histories. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 622–628, 2005.

[195] Amanda Spink and Charles Cole. *New directions in cognitive information retrieval*, volume 19. Springer, Dordrecht, 2005.

[196] Steffen Staab. *Handbook on ontologies*. International handbooks on information systems. Springer, Berlin, 2004.

[197] A. Stefani and C. Strappavara. Personalizing access to web sites: The siteif project. *Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98*, June 1998.

[198] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:2, January 2009.

[199] Martin N. Szomszor, Iván Cantador, and Harith Alani. Correlating user profiles from multiple folksonomies. In *HT '08: Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, pages 33–42, New York, NY, USA, 2008. ACM.

[200] V. Tanasescu and O. Streibel. Extreme tagging: Emergent semantics through the tagging of tags. In *Proceedings of the First International Workshop on Emergent Semantics and Ontology Evolution*, pages 84–94, 2007.

[201] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[202] F. Tanudjaja and L. Mui. Persona: A contextualized and personalized web search. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, page 67, 2003.

[203] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 449–456, New York, NY, USA, 2005. ACM.

[204] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Characterizing the value of personalizing search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 757–758, New York, NY, USA, 2007. ACM.

[205] Maximilian Teltzrow and Alfred Kobsa. *Impacts of user privacy preferences on personalized systems: a comparative study*, pages 315–332. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[206] Bhavani Thuraisingham. Confidentiality, privacy and trust policy enforcement for the semantic web. In *POLICY '07: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 8–11, Washington, DC, USA, 2007. IEEE Computer Society.

[207] Ilaria Torre. Adaptive systems in the era of the semantic and social web, a survey. *User Modeling and User-Adapted Interaction*, 19(5):433–486, December 2009.

[208] Joana Trajkova and Susan Gauch. Improving ontology-based user profiles. In *Proceedings of RIAO*, pages 380–389, 2004.

[209] Evangelos Triantafillou, Andreas Pomportsis, and Stavros Demetriadis. The design and the formative evaluation of an adaptive educational system based on cognitive styles. *Comput. Educ.*, 41(1):87–103, August 2003.

[210] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. *Lecture notes in computer science*, 4130:292–297, 2006.

[211] Mike Uschold and Michael Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[212] Céline Van Damme, Martin Hepp, and Katharina Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007)*, pages 57–70, 2007.

[213] Kees van der Sluijs and Geert-Jan Houben. A generic component for exchanging user models between web-based systems. *International*

Journal of Continuing Engineering Education and Life Long Learning, 16(1):64–76, January 2006.

[214] C. J. Van Rijsbergen. *Information retrieval*. Butterworths, London, 2nd edition, 1979.

[215] Mark van Setten, Rogier Brussee, Harry van Vliet, Luit Gazendam, Ynze van Houten, and Mettina Veenstra. On the Importance of "Who Tagged What". In *Workshop on the Social Navigation and Community based Adaptation Technologies*, 2006.

[216] L. van Velsen, T. van der Geest, R. Klaassen, and M. Steehouder. User-centered evaluation of adaptive and adaptable systems: a literature review. *The Knowledge Engineering Review*, 23(03):261–281, 2008.

[217] Thomas Vander Wal. Folksonomy coinage and definition. http://vanderwal.net/folksonomy.html, February 2007.

[218] Ellen Voorhees. The philosophy of information retrieval evaluation. In *Lecture Notes In Computer Science*, volume 2406, pages 355–370. Springer-Verlag London, UK, 2001.

[219] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1):19–29, March 2001.

[220] David Weinberger. *Everything is Miscellaneous: The Power of the New Digital Disorder*. Henry Holt, May 2008.

[221] Katrin Weller. Folksonomies and Ontologies: Two New Players in Indexing and Knowledge Representation. In *Applying Web 2.0. Innovation, Impact and Implementation: Online Information 2007 Conference Proceedings*, pages 108–115, 2007.

[222] Ryen W. White, Bill Kules, Steven M. Drucker, and Schraefel. Supporting Exploratory Search, Introduction. *Commun. ACM*, 49(4):36–39, April 2006.

[223] Ryen W. White, Gary Marchionini, and Gheorghe Muresan. Editorial: Evaluating exploratory search systems. *Inf. Process. Manage.*, 44(2):433–436, November 2008.

[224] Ryen W. White, Gheorghe Muresan, and Gary Marchionini. Report on ACM SIGIR 2006 workshop on evaluating exploratory search systems. *SIGIR Forum*, 40:52–60, December 2006.

[225] Ryen W. White and Resa A. Roth. Exploratory Search: Beyond the Query-Response Paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, January 2009.

[226] Dwi H. Widyantoro, Thomas R. Ioerger, and John Yen. Learning user interest dynamics with a three-descriptor representation. *J. Am. Soc. Inf. Sci. Technol.*, 52(3):212–225, 2000.

[227] Ian H. Witten, Katherine J. Don, Michael Dewsnip, and Valentin Tablan. Text mining in a digital library. *International Journal on Digital Libraries*, 4(1):56–59, 2004.

[228] W. A. Woods. What's in a link: Foundations for semantic networks. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 35–82. Academic Press, 1975.

[229] Harris Wu, Mohammad Zubair, and Kurt Maly. Harvesting social knowledge from folksonomies. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 111–114, New York, NY, USA, 2006. ACM.

[230] Shengliang Xu, Shenghua Bao, Ben Fei, Zhong Su, and Yong Yu. Exploring folksonomy for personalized search. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 155–162, New York, NY, USA, 2008. ACM.

[231] Yusuke Yanbe, Adam Jatowt, Satoshi Nakamura, and Katsumi Tanaka. Can social bookmarking enhance search in the web? In *Proceedings*

of the 7th ACM/IEEE-CS joint conference on Digital libraries, JCDL '07, pages 107–116, New York, NY, USA, 2007. ACM.

[232] Junjie Yao, Yuxin Huang, and Bin Cui. Constructing evolutionary taxonomy of collaborative tagging systems. In *Proceeding of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 2085–2086, New York, NY, USA, 2009. ACM.

[233] Ching-Man A. Yeung, Nicholas Gibbins, and Nigel Shadbolt. Tag meaning disambiguation through analysis of tripartite structure of folksonomies. In *Web Intelligence and Intelligent Agent Technology Workshops, 2007 IEEE/WIC/ACM International Conferences on*, pages 3–6, 2007.

[234] Ching-man A. Yeung, Nicholas Gibbins, and Nigel Shadbolt. A study of user profile generation from folksonomies. In *Proceedings of the WWW 2008 Social Web and Knowledge Management, Social Web Workshop*, 2008.

[235] Michael Yudelson, Tatiana Gavrilova, and Peter Brusilovsky. Towards user modeling meta-ontology. In *In Proceedings of the 10th International Conference on User Modeling*, pages 448–452, 2005.

[236] Hui Zhang, Yu Song, and Han-Tao Song. Construction of Ontology-Based user model for web personalization. In *User Modeling 2007*, pages 67–76, 2007.

[237] Jin Zhang. *Visualization for information retrieval.* Springer, Berlin, 2008.

[238] Shiwan Zhao, Nan Du, Andreas Nauerz, Xiatian Zhang, Quan Yuan, and Rongyao Fu. Improved recommendation based on collaborative tagging behaviors. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '08, pages 413–416, New York, NY, USA, 2008. ACM.

[239] Ding Zhou, Jiang Bian, Shuyi Zheng, Hongyuan Zha, and C. Lee Giles. Exploring social annotations for information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 715–724, New York, NY, USA, 2008. ACM.

[240] Cai N. Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

[241] Arkaitz Zubiaga. Understanding the meaning of tags. http://blog.zubiaga.org/, 2010.

[242] Ingrid Zukerman and David W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1):5–18, March 2001.

List of Figures

2.1 Search activities (based on [146]) 37
2.2 Exploration dimensions 44
2.3 Personalization process (based on [39]) 52
2.4 Scalar model 77
2.5 Overlay model 79
2.6 MUMS architecture (based on [37]) 105
2.7 Situational statement (based on [99]) 115
2.8 GUMO User Model Dimension (based on [101]) 118
2.9 Generic personalization architecture (based on [38]) 136
2.10 Folksonomy 157

3.1 Knowledge representation structures (based on [153]) 180
3.2 Actor-Concept-Instance graph 214

4.1 Cumulative tag popularity distribution 271
4.2 Recommendation network 276
4.3 Indegree-popularity correlation plot for T(s,10) network (log-log) 287
4.4 Clustering coefficients 291
4.5 Average path lengths 293
4.6 Number of clusters 296
4.7 Diameters 298
4.8 Strong Giant Components 300
4.9 Result similarities 304
4.10 Result reduction rates (in percent) 316
4.11 Result length for queries and expanded queries (in percent) . . 318

4.12 Result vector length for queries (blue) and expanded queries (red dotted) for T(s) . . . 318
4.13 Query expansions without results . . . 322

5.1 User modeling architecture . . . 340
5.2 User Modeling Service . . . 347
5.3 Personalization Service . . . 355
5.4 User interest example . . . 364
5.5 User knowledge example . . . 365
5.6 User similarity example . . . 366
5.7 Overall user model content example . . . 367
5.8 User Property Statements with metadata . . . 368
5.9 Qualified relation example . . . 374
5.10 N-ary relation example . . . 375

6.1 Screenshot of the Earlybird search result site . . . 419
6.2 Software architecture . . . 428
6.3 Data Maintenance tier . . . 431
6.4 Presentation tier . . . 437
6.5 Tag cloud with scaled tags and hierarchy . . . 439
6.6 Application tier . . . 443
6.7 Storage tier . . . 459

List of Tables

3.1 User interest prediction output 235
3.2 User similarity matrix 236
3.3 Similar users prediction output 238
3.4 Similar tags prediction output 241
3.5 Tag hierarchy prediction output 243

4.1 Input dataset features 270
4.2 Quantiles of the popularity curve 272
4.3 Recommendation set 277
4.4 Query result matrix 279
4.5 Indegree-popularity correlation coefficients 288
4.6 Popularity correlation 313

6.1 SQL table EARLYBIRD_ENTRIES 465
6.2 SQL table EARLYBIRD_SCORES 466
6.3 SQL table EARLYBIRD_URLS 466
6.4 SQL table EARLYBIRD_USERS 475
6.5 SQL table EARLYBIRD_USERMODELS 478
6.6 SQL table EARLYBIRD_LOG_SEARCH 490
6.7 SQL table EARLYBIRD_LOG_RECOMMENDED 490
6.8 SQL table EARLYBIRD_LOG_LIKED 491

Listings

2.1 APML example 110
2.2 UserML Situational statement example 116
2.3 SKOS relations example 122
2.4 FOAF example 125
2.5 Dublin Core example 126
2.6 Property-centric representation of interest 128
2.7 CCO interest characteristic 129
5.1 Named Graph example 379
5.2 Property customization example 380
5.3 RDF reification example 383
5.4 Property reification example 386
5.5 Property reification mapping example 387
5.6 User Interest Statement 394
5.7 Weighting scale 395
5.8 Property reification 396
5.9 User Knowledge Statement 397
5.10 The Relation Ontology 398
5.11 Property Reification 402
5.12 User Similarity Statement 403
5.13 Property Reification 404
5.14 Metadata example 405
6.1 Response of Search servlet 447
6.2 Recommendation type extraction method 447
6.3 Response of Recommendation servlet 450
6.4 Recommender initialization process 452

6.5 NoveltyRescorer implementation 455
6.6 Retrieve top-5 similar tags from recommender 456
6.7 Virtuoso/PL insert bookmark procedure definition 468
6.8 Insert bookmark procedure call 470
6.9 Delete bookmark procedure definition 473
6.10 Decrease score trigger definition 473
6.11 URI class definition 479
6.12 RDF mapping 481
6.13 SPARQL query example 486
6.14 SPARQL query example 2 486
6.15 Query response example 487