A Semantic Web Service-based Framework for Generic Personalization and User Modeling

Von der Fakultät für Elektrotechnik und Informatik der
Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

Doktor der Ingenieurwissenschaften

Dr.-Ing.

genehmigte Dissertation von

M.Sc. Daniel Krause

geboren am 20. Oktober 1981 in Hannover

2011

Referent:	Prof. Dr. Nicola Henze
Koreferent:	Prof. Dr. Julita Vassileva
Koreferent:	Prof. Dr. Wolfgang Nejdl
Tag der Promotion:	14. Dezember 2011

Kurzfassung

Die Menge an verfügbaren Daten im Web wächst rapide, so dass die Personalisierung des Informationsangebots auf den Nutzer und Anwendungsfall wichtiger denn je ist. Techniken zur Personalisierungung, wie Recommender Systeme, werden dafür in einer breiten Masse von Anwendungsfällen, wie Online Shops, Applikationen für Mobiltelefone oder E-Learning Systemen [Rossi et al., 2001] eingesetzt. Dennoch ist die Gesamtanzahl personalisierter Applikationen gering.

Um Probleme und Ansätze zur Aufwandsreduktion beim Einsatz von Personalisierung zu identifizieren wurde eine Literaturrecherche im Bereich der generischen und wiederverwendbaren Personalisierung durchgeführt. Die Ergebnisse der Literaturrecherche wurden durch eine Umfrage unter Experten überprüft. Das Ergebnis der Umfrage belegt, dass generische Personalisierungskomponenten, standardisierte Schnittstellen und Wiederverwendbarkeit als Schlüsseltechnologien angesehen werden. Basierend auf den Ergebnissen wird ein Framework vorgestellt, das den Lebenszyklus einer personalisierten Applikation ganzheitlich unterstützt.

Das Personal Reader Framework kapselt Personalisierungsfunktionalität in wiederverwendbaren generische Web Servicen, sogennante PServicen, und stellt damit den Stand der Technik dar. Für verschiedene Anwendungsfälle bietet das Framework fertige PService, die in bestehende Anwendungen integriert werden können. Der Meta-Personalisierungs-Matchmaker selektiert PService basierend auf Benutzerpräferenzen, verfügbaren Eingabedaten und angebotener Funktionalität. Die erzielten Ergebnisse übertreffen die aktueller nichtpersonalisierter Matchmaker.

Diese Arbeit geht im Bereich Benutzermodellierung über den Stand der Technik hinaus, da eine Zugriffskontrollkomponente vorgestellt wird, die auf zentralisierten Benutzermodellierungsservicen Zugriffsregeln implementiert, indem Anfragen an ein RDF Repository umgeschrieben werden. Die Benutzerprofile werden in einem gemeinsam genutzten RDF-Format gespeichert, sodass Interoperabilität und Wiederverwendbarkeit von Benutzerprofildaten zwischen Personal Reader Applikationen ermöglicht wird. Benutzerfreundliche Bedienoberflächen ermöglichen dem Endbenutzer das Benutzerprofil zu erforschen und fein-granulare Zugriffsregeln zu bestimmen. Eine Benutzerstudie zeigt, dass Anwender hiermit komplexe Zugriffsregeln erstellen können.

Der Thread Recommender ist eine von mehr als zehn Applikationen, die auf dem Personal Reader Framework beruhen. Dieser zeigt erstmals dass regelbasierte Personalisierung mit Collaborative Filtering in einem E-Learning Diskussionforum kombiniert werden. Die Sichtbarkeit des Frameworks innerhalb der Forschungsgemeinschaft ist durch erfolgreiche Zusammenarbeit mit internationalen Forschungspartnern und Publikationen auf hochrangigen Konferenzen (ISWC und AH) und in Fachblättern (TLT Journal, etc.) sichergestellt.

Schlagwörter: Personalisierung, Benutzermodellierung, Semantik Web

Abstract

The amount of data on the Web grows enormously. It is more important than ever to filter Web data by selecting the most appropriate information based on user and context. Personalization techniques, like recommender systems, have been successfully implemented in various scenarios, like online shops, mobile phone applications, or E-Learning systems [Rossi et al., 2001]. However, the amount of personalized applications is still limited.

In order to detect the main problems of creating personalized applications and analyze approaches for lowering the effort of using personalization, we inspected available approaches for generic and reusable personalization functionality. To verify our outcomes, we conducted a survey among experts in the fields of personalization and user modeling. The survey reveals that experts consider generic personalization components, standardized interfaces and reusability as key techniques to simplify the use of personalization. Based on the findings from related work and the survey, we modeled a framework that supports the entire life-cycle of a personalized application.

The Personal Reader Framework goes beyond state-of-the-art in the area of personalization by offering encapsulated personalization via reusable and generic Web Services, so called PService. For most personalization tasks, ready-to-run PServices are available to be integrated into existing applications. We present a meta-personalization matchmaker, which incorporates user preferences, available input data, and offered functionality to find best-matching PServices. Our evaluations prove that the proposed matchmaking algorithm outperforms non-personalized state-of-the-art algorithms.

In the area of user modeling this thesis contributes to the state-of-the-art by providing an access control component for a centralized user modeling service that enforces access policies by rewriting RDF queries. User models are stored in a shared RDF-based user profile storage format ensuring the interoperability and reuse of user profile data beyond single Personal Reader applications. The user modeling service is complemented by a user-friendly interface allowing the end user to explore profile data and define fine-grained access control policies.

The Thread Recommender is one example of more than ten different Personal Reader applications: It showcases the integration of rule-based personalization and collaborative filtering in an E-Learning discussion board. The visibility of the Personal Reader Framework within the research community is ensured by the successful collaborations with several international research partners, publications in highly ranked conferences (ISWC and AH) and journal articles (TLT journal, etc.).

Keywords: Personalization, User Modeling, Semantic Web

Acknowledgements

I would like to thank Prof. Nicola Henze for her great support and continuous motivation during my entire Ph.D. time – thank you Nicola for all the time you invested in improving this thesis! I enjoyed the great working atmosphere of L3S, established by Prof. Wolfgang Nejdl, making it possible to work in an international team right on my doorstep. My special appreciation goes to Prof. Julita Vassileva who gave me a warm welcome in Canada and made my research stay in Saskatoon successful, pleasant and unforgettable.

Most of the work that I conducted was joint work with Fabian Abel, my former colleague and office-mate for about 5 years. I thank Fabian for his engagement, his brilliant research ideas, the nice discussions we had and for all the great work he did.

I would further like to say thank you to Ig Ibert Bittencourt whom I met during my research stay in Canada. Thank you Arne Wolf Koesling, Daniel Olmedilla and Juri Luca De Coi for the successful research work in the area of access control. I enjoyed the agile discussions with Dimitris Skoutas and Anna Averbakh who helped me to realize and evaluate the idea of personalized matchmaking. Daniel Plappert, who supplied the best master thesis that I supervised, contributed excellent work in the area of the user modeling ontology - thank you. I want to further thank our former student assistants Peyman Nasirifard, Kashif Mushtaq and Kai Tomaschewski for all their implementation work in the Personal Reader project, Philipp Bähre and Zhivko Asenov for their help in the evaluation of the questionnaire and Nicole Ullmann for her implementation work of the policy editor.

Finally, I thank my family – especially my grandparents – for their manifold support, ranging from the help during homework in school, motivation to continue education, financial help which allowed me to focus on studying computer science and providing me over all the time free space to develop. Thank you Sandrina for taking the load off from me and ongoing support over the last four years.

This work was partially funded by the German Research Foundation (DFG).

Contents

Chapter 1

Introduction

Personalization, the task to adapt the functionality, interface, information content, or distinctiveness of a (software) system [Blom, 2000], is an important research area in computer science with a long history going back to the 1960s [Licklider et al., 1968]. Personalization techniques, like adaptive hypermedia or recommender systems have received attention inside and beyond the research community: Personalized recommendations, for example, generated millions of additional revenues and justified the success story of Amazon.

However, personalization strongly depends on high-quality input data. Today, this input data either consists of a huge automatically generated data collection, like sales-logs and weblogs or comparatively small hand-crafted data collections, like E-Learning courses with attached metadata or product catalogues containing specific features of an item-collection. The drawback of automatically generated data is the existence of noise and wrong information in the set: The gay community for example, exploited the Amazon recommender system to show recommendations to books from their community on the page of an gay adversarial book [Mehta and Hofmann, 2008]. The disadvantage of hand-crafted data are scalability issues and maintenance costs.

A data collection, that combines both properties, large scale data and human maintained information, is the Web – by far the largest and most-recent information space of human mankind. Thus, personalization has focused in the last decades on utilizing Web data. While 15 years ago the main task in the Web was information discovery, namely the discovery of related Web pages to a given keyword, nowadays, major search engines deliver millions of relevant pages for popular keywords. The success of personalized search is still limited as Web data is created mainly for humans and can be processed by machines only hardly and error-prone nor can information be combined in a generic fashion.

The combination of data from different sources in a large scale is a key aspect of the so-called Web 2.0, proposed by Tim O'Reilly. So-called Mashups

combine existing data from different Web applications and therewith create added value for the users. Moreover, Mashups do not only combine data but also functionality from various sources. A drawback of Web 2.0 Mashups is that they are statically created by humans: All the description of the offered data and functionality, encapsulated by so-called Web Services, which can by considered as interfaces, is hidden in plain-text API documentations.

The Semantic Web, proposed by Tim Berners-Lee, aims at making Web data machine processable by adding additional descriptions, so-called metadata. Today, the Semantic Web already contains billions of machine-readable information snippets, called RDF triples, which are linked to each other by the Linked Data paradigm[1]. The Semantic Web provides techniques to create automatic Mashups: Semantic Web Services offer a machine-readable description of their functionality. Programmers can specify required functionalities in an application without knowing if a service is available that offers such a functionality or where such a service can be located. So-called Semantic Matchmakers retrieve appropriate services that are invoked by the application at runtime.

In such a scenario, traditional monolithic applications, which combine all personalization related tasks, like user modeling, adaptation of the user interface and information filtering tightly coupled, become a distributed network of services, possibly run by different parties.

Successful frameworks and toolkits, like the Spring Framework[2], Ruby on Rails[3], etc. facilitate simple and still standard-compliant development of modular Web applications. The concept of frameworks served as key idea for proposing the Personal Reader Framework, a Semantic-Web based architecture that copes with the newly arisen research questions for supporting personalization:

1. Can the strongly-coupled personalization process of monolithic applications be divided into logically independent services?
2. Can such personalization services be reused in various applications?
3. How shall user profiles be stored, maintained, and accessed in a Semantic Web Service-based environment?
4. Can personalization be used to orchestrate personalized applications from single Web Services?
5. Which requirements need to be fulfilled by a personalization framework to ease the process of creating a personalized application and which support needs to be offered to assist the programmers in this process?

The thesis is structured as follows:

[1] http://www.w3.org/DesignIssues/LinkedData.html
[2] http://www.springframework.org
[3] http://www.rubyonrails.org

In Chapter 2, we identify integration opportunities and success factors of different state-of-the-art approaches as requirements for a framework that supports personalization and user modeling. We present the current state-of-the-art of generic personalization and a short introduction into the problems of privacy and Web Service discovery. Finally we discuss advantages and challenges of the presented techniques. To verify the results from literature, we designed a questionnaire to receive additional ideas for a personalization and user modeling framework from domain experts. The evaluation of the questionnaire reveals today's obstacles of using personalization in applications as well as promising techniques and trends for the future of personalization. Based on the results, we revisit and complete the requirements for our framework from the first part.

The requirements from the previous section serve as design principles for our Web Service-based Framework and its core components as described in Chapter 3. The core components are implemented as Web Services, namely Personalization Services, which encapsulate personalization algorithms, Syndication Services, which contain the business logics, as well as the Connector Service, which handles the communication between the services and provides centralized functionality. To enable the dynamic orchestration of personalized applications, we present a personalized matchmaker. In this chapter, the above defined research questions will be revisited.

For storing and exchanging user profile data among applications, we introduce a generic user modeling service, which is described in Chapter 4. The user model service provides two user interfaces to allow end-users to: a) inspect and modify their user profile and b) to ensure privacy by enabling them to specify fine-grained access rules. An RDF-based access control mechanism, called AC4RDF, enforces these rules and applies them to the user profile.

In Chapter 5, we evaluate the real-world usability of the proposed framework by three proof-of-concept applications: The Comtella-D Thread Recommender, the Personal Reader Agent and MyEar outline the benefits of applying the framework. User access statistics as well as an overview about the continuous development and cooperation with the research community outline the success of the Personal Reader Project.

The conclusion and an outlook to future research directions is given in chapter 6.

The research that I jointly conducted over the last years during my employment at L3S Research Center resulted in several publications at workshops, conferences and in journals. A list of my scientific publications is provided in Appendix A. Here, I point to those publications which prominently contribute to this thesis:

- Nicola Henze and Daniel Krause: Scalable Matchmaking for a Semantic Web Service based Architecture - Workshop on Semantics for Web Ser-

vices, December 4, 2006, Zurich, Switzerland, collocated with ECOWS 2006 (used in Chapter 3)

- Nicola Henze and Daniel Krause: User Profiling and Privacy Protection for a Web Service Oriented Semantic Web. 14th Workshop on Adaptivity and User Modeling in Interactive Systems, Hildesheim, October 9-11 2006 (used in Chapter 3)

- Nicola Henze and Daniel Krause: Personalized Access to Web Services in the Semantic Web. 3rd International Semantic Web User Interaction Workshop, November 6, 2006, Athens, Georgia, USA, collocated with ISWC 2006 (used in Chapter 3)

- Anna Averbakh, Daniel Krause, Dimitrios Skoutas: Exploiting User Feedback to Improve Semantic Web Service Discovery. 8th International Semantic Web Conference, 25-29 October 2009, Washington DC, USA (used in Section 3.3.1)

- Anna Averbakh, Daniel Krause, Dimitrios Skoutas: Recommend me a Service: Personalized Semantic Web Service Matchmaking. 17th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2009 - Workshop-Woche: Lernen-Wissen-Adaption, September 21-23, 2009, Darmstadt, Germany (used in Section 3.3.1)

- Fabian Abel, Nicola Henze, Daniel Krause, Daniel Plappert: User Modeling and User Profile Exchange for Semantic Web Applications, 16th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2008 - Workshop-Woche: Lernen-Wissen-Adaption, October 6-8, 2008, Würzburg, Germany (used in Section 4.2)

- Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, Daniel Olmedilla: Enabling Advanced and Context-Dependent Access Control in RDF Stores. 6th International Semantic Web Conference, November 11-15, 2007, Busan, Korea (used in Section 4.2.5)

- Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, Daniel Olmedilla: A User Interface to Define and Adjust Policies for Dynamic User Models, 5th International Conference on Web Information Systems and Technologies, March 23-26, 2009, Lisboa, Portugal (used in Section 4.2.6)

- Fabian Abel, Ig Ibert Bittencourt, Nicola Henze, Daniel Krause, Julita Vassileva: A Rule-Based Recommender System for Online Discussion Forums. 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, July 28-August 1, 2008, Hannover, Germany (used in Section 5.1)

- Fabian Abel, Ig Ibert Bittencourt, Evandro Costa, Nicola Henze, Daniel Krause, Julita Vassileva: Recommendations in Online Discussion Forums for E-Learning Systems. IEEE Transactions on Learning Technologies, IEEE Computer Society, 2010 (used in Section 5.1)
- Fabian Abel, Ingo Brunkhorst, Nicola Henze, Daniel Krause, Kashif Mushtaq, Peyman Nasirifard and Kai Tomaschewski: Personal Reader Agent: Personalized Access to Configurable Web Services. 14th Workshop on Adaptivity and User Modeling in Interactive Systems, Hildesheim, October 9-11 2006 (used in Section 5.2)

Chapter 2

Requirements for a Generic Personalization Architecture

"If I have 3 million customers on the Web, I should have 3 million stores on the Web"

The statement of Jeff Bezos, founder of Amazon.com, outlines that personalization has emerged from the ivory tower of research to industry and real world applications. Studies have been conducted that show the benefits of personalization for the sales rate of online stores [Schafer et al., 1999], users satisfaction of applications [Liang et al., 2007], and usage time of services [B. Smyth, 2002].

However, personalization is sparsely used in current real-world applications. Our hypothesis is that today's personalization is strongly focused on a specific application or domain, and hence using personalization in a new application is an expensive task. We inspect current state-of-the art solutions for generic personalization techniques like recommender systems, adaptive hypermedia, and rule-based personalization.

In the second part of this chapter, we substantiate our findings from literature by conducting a questionnaire among personalization experts. We asked these experts what they consider as main reasons why personalization is not used more often and identified technical obstacles of creating a personalized application. As outcome of the literature review and the analysis of the questionnaire, we summarize requirements for implementing personalization infrastructures to simplify the creation of personalized applications and propose guidelines how to support the use of personalization.

2.1 Related Work on Generic Personalization

Generic personalization, namely personalization which can be applied independently from a specific application or domain, shall simplify the process of integrating personalization functionality in new applications regardless of the application's context. This might, for example, be achieved by picking personalization algorithms which are reusable and domain-independent. As candidates for discovering these generic algorithms, we selected personalization algorithms that have been used in different application domains:

- **Recommender algorithms:** Collaborative recommender systems [Adomavicius and Tuzhilin, 2005] do not need any domain knowledge as they merely use information about user interaction. Success in various application fields makes this a perfect candidate field for generic personalization.
- **Adaptive hypermedia algorithms:** Adaptive hypermedia systems like AHA! [Bra and Calvi, 1998] have been designed to be domain-independent and provide generic methods to specify adaptivity in a hypermedia graph.
- **Rule-based personalization approaches:** Even though rule-based personalization is used in the areas of recommender systems and adaptive hypermedia, rules can be utilized in various personalization-related tasks, like protection of confidential profile information (like policies) or to describe the behavior of an adaptive system (like reactive rules).

In the following, we will inspect these candidates in more detail regarding their possible usage in generic settings.

2.1.1 Recommender Systems

Recommender systems aim at supporting users in discovery of interesting items, like books, websites, social contacts and so on. The area received great attention over the last years by both, research and business. The importance of recommender systems can be estimated when considering that the online video rental company Netflix issued a price of 1 million dollars to those who managed to outperform their own recommender algorithm by 10%[1].

Recommender systems can be distinguished broadly into two classes: content-based and collaborative recommender systems. Content-based recommender systems rely on a detailed database describing the properties of the available items. Users are represented as vectors containing their preferences according to the properties of the item database. The recommender algorithms use various measurements to find a good match between the preferences of a user and the properties of the items.

[1] http://www.netflixprize.com/

In comparison, collaborative recommender systems, which are also known as collaborative filtering systems, observe the attitude of the users towards items. This data is often available without additional effort: in a shop, for example, the assumption is drawn that users who bought an item are interested in this item. Then, the sales logs can be used to infer which users are interested in some specific item. The recommender algorithm searches similar users for a given user (according to similar buying behavior) and creates a list of most popular items among these users. These items are then used as base for the recommendations. For a more detailed survey on collaborative filtering techniques, we refer to the work of Su et al. [Su and Khoshgoftaar, 2009]. For a more detailed taxonomy of recommender systems, we refer to the work of Adomavicius et al. [Adomavicius and Tuzhilin, 2005] and Montaner et al. [Montaner et al., 2003].

In this thesis, we focus on using recommender algorithms as generic personalization components. Both recommender approaches, content-based and collaborative, can be used in different application domains: the content-based recommender considers users and items in a vector-space, defined by the properties, while collaborative recommender systems predict items that similar user liked, regardless of the domain[2].

First, we will show hybrid recommender systems that can utilize various kinds of input data to generate recommendations. We have a closer look at work from Berkovsky that discusses the use of recommender systems to generate cross-domain recommendations, i.e. based on input data from one domain recommend items from another domain. Finally, we will discuss known drawbacks of recommender systems and their relevance for generic personalization.

2.1.1.1 Cross-domain Recommender Systems

Berkovsky et al. [Berkovsky et al., 2007] conducted an interesting experiment on cross-domain user profiles. They divided a movie rating database into separate databases based on the genre of the rated movie. They used different collaborative recommender strategies to generate movie recommendations:

- **Standard collaborative filtering** This collaborative recommender operates on the entire movie database and does not take any genre information into account.
- **Local** The local recommender only takes information on one specific genre into account to generate recommendations.
- **Remote-Average** The remote-average strategy applies the local strategy and takes it as one input parameter. Then, the local strategy is applied on other genre databases that the movie belongs to. This means that

[2]N.B. There are also content-based recommender systems available that are not domain-independent.

there is one genre specific rating created for each genre of a movie. These values are finally averages to calculate an overall recommendation score.

The results of the paper outline that the standard collaborative recommender delivers the highest mean average error (MAE), while the local strategy delivers a significantly lower MAE. The remote-average strategy outperforms the local strategy when the movie ratings are very sparse. The experiment shows that personalization functionality can benefit from using external data from other application domains.

2.1.1.2 Generic Hybrid Recommender Systems

Content-based as well as collaborative recommender algorithms suffer from various problems[3] [Balabanović and Shoham, 1997, Lee, 2001]: collaborative recommender systems for example cannot generate recommendations for new users as the system has no knowledge about them. The same holds for new items that have not yet been rated. Another serious issue is the lack for adaptability of collaborative recommender systems.

In comparison, content-based recommender algorithms rely on features that need to be extracted and make recommendations expensive if the required information needs to be hand-crafted. A second problem is over-specialization: the recommender estimates preferences of a user based on ratings of the items. Items are recommended that fit best to these preferences. These items are mostly those which are very similar to items that the user already knows.

Hybrid recommender systems combine different recommender strategies, like content-based and collaborative recommender, to overcome the aforementioned problems. Burke [Burke, 2002] presented six approaches to combine recommender algorithms:

- **Weighted** The final score of different recommender algorithms is averaged.
- **Mixed** Displays results of different recommender algorithms in the user interface.
- **Switching** Among different recommender algorithms, the best matching is chosen.
- **Feature combination** Input data is mixed. An interesting approach is presented by Berkovsky et al. [Berkovsky et al., 2006] who transform a user profile based on user ratings into a content based profile. They used genre information of the movies that a user rated to derive which genre a user is interested in.

[3] http://www.readwriteweb.com/archives/5_problems_of_recommender_systems.php

- **Cascade** The first recommender creates a candidate set that is refined by the next recommender and so on.
- **Feature Augmentation** the results of one recommender are used as (additional) input data for a second recommender.
- **Meta-level** The input model of one recommender algorithm is used as input model of another recommender.

While hybrid recommender systems flexibly combine single recommendation strategies, selecting the best hybrid recommender for a specific application scenario is a domain- and application-specific problem. Thus, implementing personalization by hybrid recommender systems still bears a high manual effort to optimize the recommendation quality.

2.1.2 Adaptive Hypermedia

Adaptive Hypermedia is based on a well-known principle form knowledge structuring and organization, namely hypertext: in 1945, Bush presented Memex [Bush, 1945], the memory extender, which offered the functionality of storing and scrolling documents. Associations could be added to a document that references another document. This structure of documents and links between documents was taken up by Berners-Lee in his Mesh proposal[4]. This finally lead to the definition of the World Wide Web, which soon became the largest hypertext of the world. By increasing bandwidth and storage, the WWW turns from a hypertext to hypermedia, which embeds multimedia documents, like images, videos or audio files into a hypertext. With the growing size of a hypermedia graph, users cannot find content they are looking for or tend to lose the overview of the graph, a problem called *lost in space* [Conklin, 1987]. Techniques, like graphical browsers, could help to to get a better overview on the graph, but delivering users with the information parts they need requires a personalization of the hypergraph.

Adaptive hypermedia systems (AHS) [Brusilovsky, 1996] tackle this problem by adapting the hypermedia graph. Several techniques for adaptation are know which can be grouped into two classes, adaptive presentation and adaptive navigation support. Adaptive presentation focuses on the nodes of the hypermedia graph and generally annotates, structures and omits parts of the content of a hypermedia document while adaptive navigation support focuses on the links of a hypermedia graph and provides guidance, maps of the graph or link annotations. One well known AHS is De Bra's Adaptive Hypermedia Architecture (AHA!) [Bra and Calvi, 1998], an open, multi-purpose AHS. To use AHA!, first an author needs to define a user profile which contains a set of boolean values representing the user knowledge. Second, the author needs

[4] http://www.w3.org/History/1989/proposal.html

to annotate hypermedia pages to define rules based on conditions that a user needs to fulfill in order to visit a complete page, a paragraph of it, or click a link, and the knowledge that a user receives after visiting the page. The conditions and user knowledge use and modify the variables from the user profile. Finally, the AHA! engine adapts the hypermedia graph based on the user profile and the conditions. The main drawback of such a system is that the usefulness and expressivity is strongly coupled to the effort undertaken by the author to model a specific corpus and create a fine-grained user profile. The created rules are domain-specific and cannot be reused in another corpus without adjustment.

This problem of an adaptive hypermedia system relying on a well-defined information corpus is also called the open corpus problem of adaptive hypermedia [Brusilovsky and Henze, 2007]. For some domains, like educational hypermedia, there exist solutions for application-independent personalization. Brusilovsky and Henze [Brusilovsky and Henze, 2007] propose several techniques, like keyword-based text similarity, meta-data based similarity calculation and community-based approaches to find edges between open corpus documents and hence automatically build a hypermedia graph. However, the conducted research focuses on the educational domain and might not be applicable in other domains.

2.1.3 Rules for Personalization

Rules are by their nature very generic. We showcase three different areas where rules are successfully used in personalization, namely rules for: a) access control, b) description of the behavior of a (personalized) application and c) rules for the generation of recommendations.

2.1.3.1 Rules for Access Control

According to [Bonatti and Olmedilla, 2007] policies are rules with the purpose of describing the behavior of a system. Therefore, rule-based policy systems can be used to describe the behavior of a system regarding protecting of disclosing user profile data. Existing policy engines like Protune [Bonatti and Olmedilla, 2005a] offer advantages in comparison to a domain specific access component: instead implementing code that describes access restrictions, policies can be defined by a user without having programming knowledge. A policy database can be replaced or extended without changing the application. As Protune is a declarative language, Policies are in an easy-to-read format. The following example policy[5] allows access to emails whose subject is "payment" if the current user is an Enron employee:

[5]from http://skydev.l3s.uni-hannover.de/gf/project/protune/wiki/?pagename=RDF+policy

```
allow(access(X, Y, Z),
      [ rdfTriple(User, employer, Enron),
        rdfTriple(X, type, email),
        rdfTriple(X, subject, payment) ], []) :-
   currentUser(User).
```

Policies can be used to simplify the negotiation process. If a website for example needs a user name, an address and a credit card number, the user is often first asked about her user name, in a second step she is asked about her address and finally about the credit card information. If the user is not willing or able to provide the credit card information all the previous input data is wasted. If she will only give her credit card information to members of the BBB[6], she has no option to tell this the website. If the user and the website would define their needs about data and the requirements to provide confidential data into a policy engine, the engine can immediately decide whether there is a solution to fulfill the requirements of both, the user and the website.

2.1.3.2 Rules for Modeling the Behavior of a Personalized Application

Reactive rules detect events and react on these events. These rules can be used to describe the event-based behavior of a (personalized) application. An example for a reactive rule is the calculation of a shop's discount [Berstel et al., 2007]:

- If the customer is a new customer, grant 5% discount
- If the total amount of the shopping basket is greater 100, grant 10% discount

An intuitive formalism for expressing reactive rules are the so-called Event Condition Action (ECA) rules. ECA rules can be read as ON *events* IF *condition* DO *action*. In ECA notation the discount example would be expressed as follows:

```
ON customer clicks checkout
IF customer is new customer
DO price=price*0.95

ON customer clicks checkout
IF price>100
DO price=price*0.9
```

Such rules can hence be used to separate the business logic from the application code. Changes in the business logics can be modeled by domain experts instead of programmers. Reactive rule languages like XChange [Bailey et al., 2005] can be used to model, execute and query these ECA rules.

[6] http://www.bbb.org/

2.1.3.3 Rule-based Recommendations

Lin et al. present the ASARM algorithm [Lin et al., 2002], which uses association rules to provide recommendations. Association rules require (sales) transaction as input data stating which user bought, watched, or visited which items within a specific period of time, namely a session. ASARM transforms a user-item rating matrix into transactions by ordering positive and negative ratings. For each user, two transactions are created, namely one containing all positively rated items and the other containing all negatively rated items. From the transactions, association rules are learned that follow two patterns: a) user related rules, for example if $user_x$ likes an item, $user_y$ will also like this item and b) item related rules, for example if a user likes $item_x$ then the user will also like $item_y$.

Association rules can be considered as domain-independent because they take no underlying semantics of the transactions' domain into account: solely based on co-occurrence rules are formed. Domain independence is outlined by various application scenarios of association rules. Fu et al. [Fu et al., 2000], for example, apply association rules in the SurfLen system to analyze a user's web navigation history.

Zhang et al. [Zhang and Chang, 2005], claim that association rules will deliver only a limited amount of recommendations and that the rule mining process needs to be precomputed. They use different kinds of rules (like sequential rules) and rule mining approaches to build a general rule database. These rules are weighted according to their support and confidence values and are applied all together.

2.1.4 Discussion

In this section we shortly described possible candidates for generic personalization algorithms from the three areas recommender systems, adaptive hypermedia and rule-based personalization.

For the recommender systems, we analyzed approaches for cross-domain recommendations and hybrid recommender systems: Hybrid approaches outline the flexibility of combining different recommender algorithms while cross-domain recommendations show the potential of reuse of user profile information. Collaborative recommender systems are candidates for generic personalization as they do not rely on domain knowledge but purely on the behavior of the users. However, drawbacks like the new-user problem might render them useless in settings were predictions for new items are essential. Hybrid recommender solve those issues but require a domain-dependent optimization and tuning.

We have seen in the analysis of adaptive hypermedia that they provide a good framework for modeling adaptive systems. The drawback is that they

depend on the domain-knowledge, which needs to be provided by a domain expert. Approaches to overcome the open-corpus problem exist, but are focused to the E-Learning domain. In this thesis we do not focus on E-Learning and hence cannot make use of these generic adaptive hypermedia algorithms without adaptation effort.

We showcased the successful usage of rules in different areas of personalization, like privacy protection as so-called policy and reactive rules for the description of the behavior of an adaptive system. Rules by their abstract nature offer the advantage of domain-independence and predictable behavior.

We have seen that several approach do exist that offer generic personalization. In our opinion, an urgent issue is to combine these approaches in a flexible manner: a personalization framework, which offers different generic personalization algorithms and allows for a simple plug-and-play combination and exchange of the single algorithms does not yet exist.

2.2 Questionnaire

To substantiate our impression of the needs for a generic personalization framework, we designed a questionnaire that should reveal the opinions and ideas of personalization experts how to foster the stronger usage of personalization.

From own usage and implementation experiences, discussions with end-users, and literature research, we collected an initial set of possible reasons why an application is deliberately not personalized. We grouped these reasons by the three shareholders of the personalized application, namely the *user* who interacts with the application, the *programmer* who implements the application and the *manager* who needs to maximize the profit of an application.

We consider the distinction by shareholders as important as most of the personalization experts play multiple roles: For example, a user of a personalized application might be mainly interested in the functionality and the benefit that personalization offers while a manager focuses on the costs and the programmer has the additional effort in mind that implementation of personalization functionality costs. We will therefore ask the participants to answer questions from different shareholder's perspectives and compare these perspectives with each other.

Based on the interests of the different shareholders, our hypothesis is that the following reasons are most important for not using personalization:

1. *From a user's perspective:*

    - *Personalization delivers wrong results,* e.g. recommended items are not relevant for a user.

- *Personalization complicates the workflow*, e.g. users have to manually re-enable options that the personalization algorithm disabled to simplify the menu structure (see Microsoft's Smart Menus [Jameson, 2003]).
- *Uncontrollable behavior:* personalization is often considered as an unadjustable black box, lacking of scrutability. For example, the adaptive video recorder TiVo draws wrong conclusion about the sexual interests of the user and hence records the wrong titles [Zaslow, 2002].
- *Missing awareness:* the advantage of personalization functionality might be not obvious to the end-user.

2. *From a programmer's perspective:*
   - *High implementation effort*, i.e. existing personalization functionality needs to be reimplemented mostly from scratch to fit domain-specific settings.
   - *Personalization is just an excuse for a poor user interface*: Jakob Nielsen stated[7] that personalization is often used to overcome the fact that websites are poorly designed and recommends to run usability studies and optimize the interfaces instead of using personalization.
3. *From a manager's perspective:*
   - *High costs*: Adding personalization to an existing application comes along with a high financial investment that needs to charge back.
   - *Uncontrollable behavior*: As personalization adapts content by observing user behavior, it is hard to be controlled. A popular example is the revenge of the gay community against Pat Robertson, a TV evangelist by using Amazons recommendations to link to explicit material[8].

To verify whether the community of personalization experts agrees on these reasons, we designed a questionnaire and distributed it with the conference material of the Adaptive Hypermedia Conference 2008. We will describe the layout and purpose of the questionnaire in detail in next sections. The complete questionnaire is attached to the thesis in Appendix B.

2.2.1 Layout of the Questionnaire

Based on the identified shareholders and hypotheses our questionnaire contains 25 questions. These questions were assigned to four major blocks:

1. Experiences from a user's perspective,

[7] http://www.useit.com/alertbox/981004.html
[8] http://news.cnet.com/2100-1023-976435.html

2. experiences from a developer's perspective,
3. future perspectives on personalization, and
4. open questions

Deliberately, we omitted a separate block for the management shareholders as the majority of the participants have a research oriented background. We incorporated the management related issues into the blocks of the users and developers. The content and design rationale of the blocks are described in the next four paragraphs.

2.2.1.1 Experiences from a user's perspective

The first part of the questionnaire aims at ascertaining the participants' usage background of personalization techniques and their perception of today's usage frequency of personalization (Question 3 and 4). Question 1 and 6 shall reveal the general attitude of the participants towards personalization. If participants do not like personalization in general, it might be because they have particular personalization techniques in mind that are not satisfying for most of the participants. For example, one of Microsoft's first attempts to introduce personalization in a mainstream software product, namely the Smart Menus, were not accepted by the users [Weld et al., 2003] and might have cause a negative attitude towards personalization of several Microsoft customers.

Questions 2b and 5 ask the participants about the advantages and disadvantages of personalization, giving the possible reasons that we have identified.

2.2.1.2 Experiences from a developer's perspective

The second part of the questionnaire asks the participants about their personalization experience from a developer's point-of-view. Question 7-10 focus on the experience of the programmer in terms of general programming experience and experience in implementing personalization. Question 11 focuses on technical and non-technical reasons why, if applicable, they did not use personalization in their own applications. Question 12 finally asks for a short description of their own developed personalized applications and whether they reused code or created reusable code for providing personalization.

2.2.1.3 Future perspectives on personalization

The third section of the questionnaire focusses on getting advice from the participants how they estimate the future of reusability and interoperability for personalization. The first three questions (13-15) focus on the interoperability aspect and ask the participant if interoperability is applicable and useful in

general, and which techniques like Web Services, XML interface, etc. would support interoperability best.

Questions 16-21 focus on reusability of personalization functionality. First, the users are asked about their general attitude towards reusability in personalization. Then the participants shall declare which components of an adaptive system they consider to be reusable and to which degree. We therefore offered the following levels of reusability:

- *Data*, i. e. usage of a unified data structures, like XML.
- *Algorithm*, i. e. reimplementation of existing algorithms.
- *Code template*, i. e. adaption of existing programming code.
- *Code library*, i. e. use of programming code without modifications.
- *Web Service*, i. e. the usage of existing Web Services.

Finally, we asked the participants which level of reusability offers the greatest advantage for creating adaptive systems.

2.2.1.4 Open questions

The open questions in block four have the purpose to address general issues about the future of personalization. Namely, what are the hot topics, techniques and challenges for the future of personalization beyond reusability and interoperability.

2.2.2 Evaluation

We designed this questionnaire to get an overview of the personalization expert's opinions. To get a reasonable amount of participants we distributed the questionnaire among the conference proceedings of the Adaptive Hypermedia Conference[9] 2008, that took place in Hannover from 29th July to 1st August. During the opening ceremony and the conference we asked the participants to fill the questionnaire. Overall, from the 130 participants of the AH conference, 24 filled and returned the questionnaire.

We will briefly explain our measurements, followed by the analysis of the questionnaire and finally draw conclusions for a personalization infrastructure.

2.2.2.1 Measurements

In the following sections, we present the results of the evaluation of the questionnaire. To find dependencies and relationship among different questions

[9] http://www.ah2008.org

(for example to compare the different shareholders), we used association rules [Agrawal et al., 1993].

To find associations between two answers, we constrained the valid associations rules by several measures. Assume an association rule stating that participants who marked answer a of question X will also mark answer b of a given question Y, is formally expressed by $(X.a \rightarrow Y.b)$. Let $\#Y.b/\#Y$ be the percentage of participants who gave answer b for question B. Then, the requirements and the underlying purpose of the requirement, that the rules have to fulfil, are:

Requirement	Purpose of the requirement
The confidence of the rule must be at least 60%.	Remove rules with a too low confidence.
The confidence of the rule must be 20% higher than the occurrence rate of answer b for question Y.	This requirement ensures that a high confidence is not generated purely because of a popular answer in the rule's head.
The occurrence rate of answer b for question Y is lower than 50%.	If more than 50% of all participants give the same answer, the answer is popular in general and it is hard to assume a relationship to another answer.
The coverage of answer Y.b in Y of the rule is higher than the percental occurrence of answer Y.b in Y	The requirement ensures that rules find those user groups that give a specific answer over-proportionally frequent.

Table 2.1: Requirements for the association rules

A list of identified association rules (R1-R248), that fulfil the constraints, is given in Appendix C. We will refer to these rules within the next sections.

2.2.3 Experiences from a user's perspective

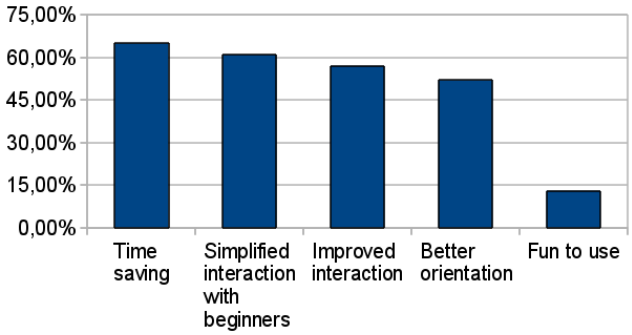


Figure 2.1: Benefits of personalization

All of the 24 participants consider personalization as useful in general (ques-

tion 3). The most important advantages of personalization are saving of time, a simplified interaction for beginners, improved interaction possibilities, and a better orientation (see Figure 2.1).

Figure 2.2 depicts the satisfaction of the participants regarding the kind of personalization which is offered by current applications. While nearly half of the participants (45%) are satisfied, the majority is not yet fully satisfied. Association rules show that users who are not satisfied with existing personalized applications are especially dissatisfied with the adaption of content (see R1 in Appendix C). In comparison, participants which are satisfied with currently offered personalization, consider device adaptation as useful (see R5). A possible reason for the satisfaction might be that device adaptation works properly today while adaptation of content does not.

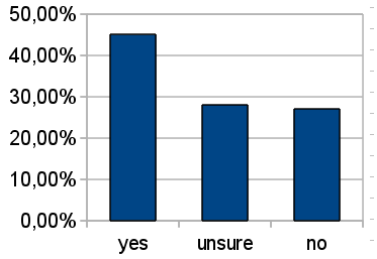


Figure 2.2: User's satisfaction of personalization offered by current systems

We tried to get a more detailed view on the participant's satisfaction based on the personalization techniques they used. Figure 2.3 depicts the satisfaction and value separated by the type of personalization, like recommendations, device adaption, etc. Important to note is that for all strategies, the participants consider the value of the personalization strategy higher than their satisfaction, which again gives information about user's satisfaction with currently available personalized applications.

Participants, who are not satisfied with currently available device adaptation are mostly well experienced programmers with about 10 years of experience in this field (see R30). These participants are also not satisfied with adaptive presentation (see R41) and adaption of content (see R42). Still, it is remarkable is that they are very interested in a reusable device adaptation component (see R33).

The relatively low values of satisfaction – which is especially remarkable as all participants are experts in the are of personalization – may be a reason for the usage of personalization in today's applications: The participants estimate that 22% of currently available applications are personalized and 95% of the participants agree that more applications can benefit from personalization.

We asked the participants about possible reasons why personalization is not used more often. The main reasons given are unclear functionality, privacy con-

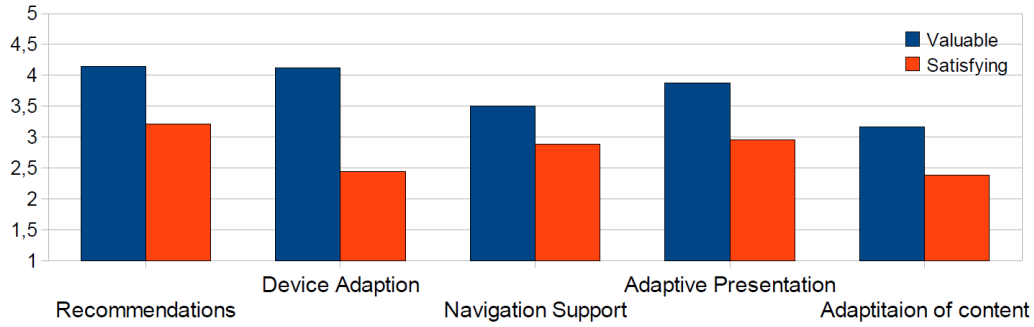


Figure 2.3: User's satisfaction of personalization technique from low (=1) to high (=5)

cerns, that the results of personalization are not satisfying, and missing transparency (see Figure 2.4). Participants that consider missing transparency as problem also criticize a lack of best practices for implementing personalization (see R142).

User who are satisfied with personalization offered by today's applications (see R2) and users who consider better feedback as an advantage of personalization (see R121), consider slow adjustment of the personalized systems as main problem. Participants who consider recommendations as useful criticize mostly that personalization suffers from unclear functionality (see R8), while participants that are not satisfied by recommendations offered by existing applications see privacy issues as main problem (see R12).

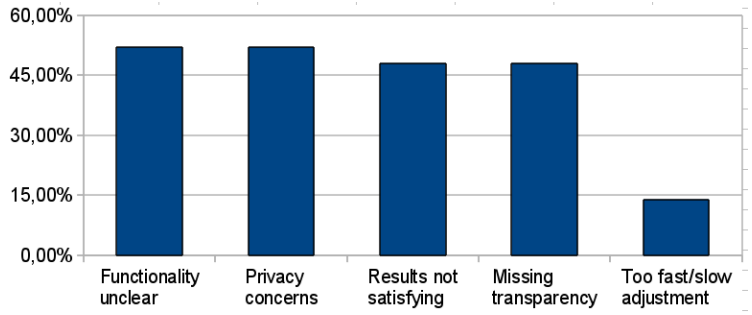


Figure 2.4: Reasons for not using personalization

2.2.4 Experiences from a developer's perspective

It is remarkable that the participants in general are very experienced in the area of personalization: 38% of the participants have more than 10 years of experience in developing personalized systems. In average, every participant created 5.6 software systems. From these applications 55% were personalized,

while the participants claim that 83% of them could benefit from personalization.

Similar as from the user's perspective, there is again a gap between actual usage and usefulness of personalization. We divided possible reasons for not implementing personalization in own applications into technical (see Figure 2.5) and pragmatic (see Figure 2.6) reasons. The participants agree that the main technical obstacle is high implementation effort that is amplified by a pragmatic reason, namely a low return on investment.

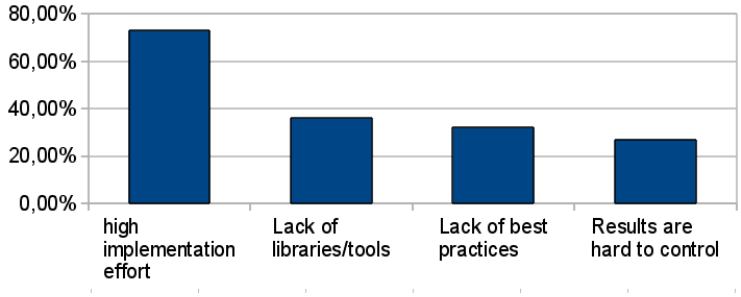


Figure 2.5: Technical reasons for not using personalization in own applications

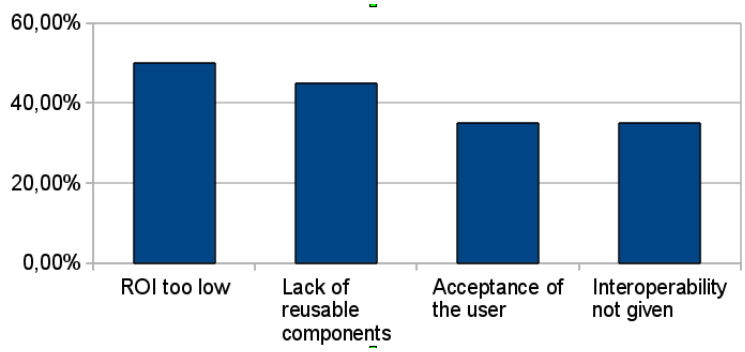


Figure 2.6: Pragmatic reasons for not using personalization in own applications

Overall, programmers acknowledge the benefit of personalization but consider a too low effort-benefit ratio as reason to not use personalization more often. Interestingly, the low effort-benefit ratio is mentioned more frequently by users who consider recommendations as useful (see R11) and might be an indication for the lack of reusable recommender tools. Programmers with experience of more than 10 years point out that missing libraries and tools are the main reason why personalization is not used more often (see R154).

Possible approaches for improving the usage of personalization are given in the figures as well: Solving the lack of reusable components, libraries, tools and/or best practices are considered by the participants as promising strategies.

2.2.5 Reusability and Interoperability of Personalization

Reusability and interoperability may offer important directions for a standardized and hence more simple use of personalization in future applications. We asked the participants about their opinion regarding the importance and feasibility of reusability and interoperability in the area of personalization.

70% of the participants believe that both, reusability and interoperability are techniques that could be incorporated in the area of personalization. And more than 70% agree that reusability and interoperability are valuable and can increase the usage of personalization.

We asked the participants what techniques they consider as most promising for enabling interoperability and reusability in personalized applications. Web Services (62%) and Semantic Web Services (50%) are considered as main techniques for interoperable personalized applications. In comparison, reuse of data (53%) and Web Services (58%) are considered to have the highest impact for providing reusable personalization while – from a programmer's point of view – code libraries (55%), Web Services (50%) and reuse of data (45%) are the preferred techniques. Interestingly, especially participants who consider personalization as useful for time saving see Web Services as most promising for reusability (see R119 and R120). Web Services are also most promising for experienced programmers who created ten or more applications (see R162). Participants who consider recommendations as useful would be most satisfied with the reuse of data (see R9).

As a personalized system is composed of different components, its potential for being interoperable and/or reusable may vary. We asked the participants which component of an adaptive system can be made generic (see Figure 2.7). On a scale from impossible (=1) to possible (=5), all components receive a score higher than 3 which expresses that the participants agree that all components of a personalized system can be made generic.

Participants that never used personalization in their own applications consider reusability of user modeling as very important (see R169). It might indicate that providing generic user modeling could foster the usage of personalization in own applications.

Programmers that see a lack of results/effects of a personalized system wish to have code libraries for reusability (see R180). This might indicate that those programmers in general would use personalization, but that they are not willing to invest in implementing own personalization algorithms from which benefit they are not yet fully convinced.

According to Figure 2.7 the two components of user event detection and user modeling can be considered as most promising for being made generic.

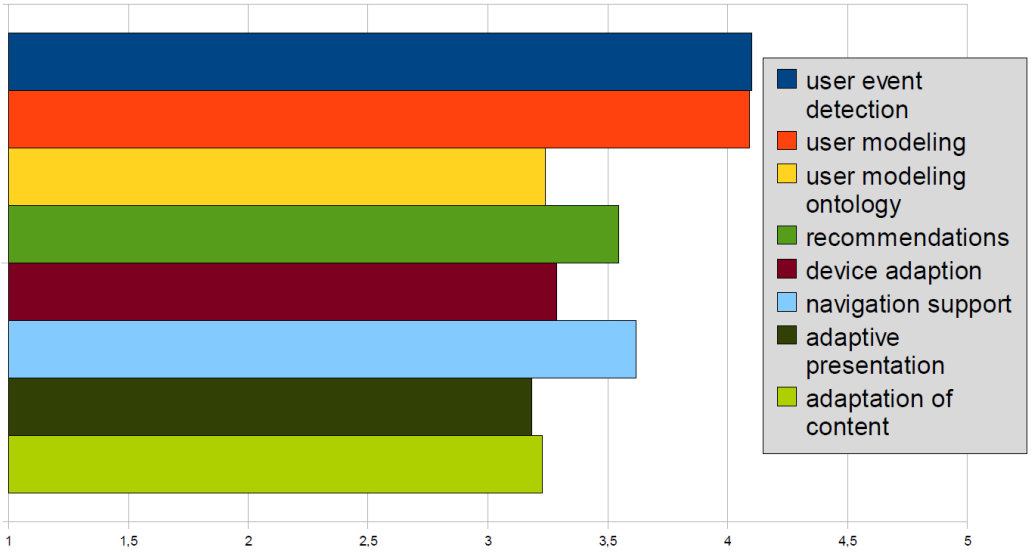


Figure 2.7: Which components of an adaptive system can be made generic? Scale from impossible (=1) to possible (=5)

2.2.6 Future Perspectives on Personalization

The fourth part of the questionnaire consists of free text questions to receive a feedback about the future perspectives of personalization beyond reusability and interoperability. We do deliberately not give an quantitative overview as we compared the given answers and tried to combine them by finding descriptive classes.

Regarding challenges for personalization participants consider the following topics as important:

- Standardization, reusability, interoperability, transparency, authoring tools,
- awareness in industry,
- privacy, trust, and
- proof the value in applications.

While techniques like standardization, reusability and interoperability focus on easing the use of personalization for the programmer, most of the points aim at making the user and industry more aware of personalization: Making the purpose and usage of personal data transparent to the user as well as showing the benefits of personalization (e.g. by good examples of personalized applications and demonstrators) will increase the acceptance on the user side.

For simplifying personalization, the participants propose these solutions:

- Multiagents, decoupling components,
- tutorials and best practices,
- visual tools, and
- educating people.

Most of these strategies focus on involving the user more in the personalization process. Tutorials, educating the users as well as visual tools for creating and adjusting personalization are good techniques for increasing the awareness and visibility of personalization.

Personalization will be influenced considerably by the following trends:

- Coping with short term changing needs of users, detect what a user wants.
- Mobile applications.
- Context detection (and usage).
- Move towards Semantic Web.
- Exploit Web 2.0, social network data, collaborative filtering.
- Combination of social network aspects, semantics, and adaptive techniques.
- Personalized add-ons: Personalization as add-on feature without altering the original application.

2.3 Requirements

Based on the results of the questionnaire, we identified the following characteristics for a promising personalization platform:

- **Usage of Web Services:** Using Web Services offers frameworks to connect to various available applications and APIs on the Web and allows other applications to access single components of the framework in a flexible manner.
- **Reusable personalization modules:** Personalization techniques like recommendations are considered to be reusable. To decrease the costs of implementing personalization, programmers shall be assisted by providing a tool box, containing important generic personalization algorithms.
- **Generic User Modeling:** Applications based on user models suffer heavily from the new user problem. The framework shall provide shared user modeling functionality that is able to combine knowledge about a user gathered from different applications.

- **Generic Event Detection:** Participants consider the components *user observation* and *event detection* as most promising to be made generic. Techniques based on web log analysis do not rely on domain knowledge of the particular web site. These techniques are a strong evidence for that assumption. The personalization framework shall offer an event detection mechanism that is able to: a) extract events from the user interaction as well as b) identify the usage context of the user to identify possible tasks of a user.
- **User Centric Design:** Studies have shown [Kobsa, 2007] that the majority of the users is willing to contribute personal data if the data is: a) kept confidential and b) the disclosure results in a benefit for the user (e.g. the user gets better product recommendations). To motivate users contributing personal data, the framework needs to take scrutability and privacy into account. Users need to be able to inspect and modify their own user data as well as define what application is allowed to access which part of the user profile. The users must have full control over their data at any point of time.

2.4 Conclusion

In this chapter we searched for possible reasons and solutions for our observation that personalization is sparsely used in today's real-world applications. We looked to related work of generic personalization algorithms which simplify the usage of personalization. We focussed on the areas of recommender systems, especially on collaborative algorithms and hybrid recommender systems, adaptive hypermedia and rule-based approaches for access control, policies for the behavior description of a system and association rules. Our analysis outline that mature generic personalization techniques exist but have not been used in a generic manner: Techniques, like collaborative recommender algorithms and hybrid recommender systems, are not yet provided in a framework offering personalization functionality as external plug-and-play component.

To underline the needs for a generic personalization framework, we designed a questionnaire that should reveal the opinions and ideas of personalization experts how to foster the stronger usage of personalization. The questionnaire reveals that the participants agree that personalization is useful in general and that the benefits of personalization are valuable. The satisfaction values of currently available personalized applications outline that personalization is already at an advanced level and satisfies a reasonable amount of users. However, the participants see the potential and need for further improvement on both, the quality of personalization techniques as well as the quantity of applications that use personalization.

The participants identified gaps on the user's and programmer's side of us-

ing personalization. This leads to the situation that personalization is used much less in today's applications than it is considered as useful. The main reasons are that users are not fully satisfied with currently available personalized applications while programmers see a high implementation effort and limited improvements. Both together result in a low return on investment (ROI), making the use of personalization unattractive for the management.

In the questionnaire, we focused on asking the participants to name solutions that will lead to a higher usage of personalization. The participants identified promising techniques for decreasing the implementation costs for personalization like reusability, generic personalization components as well as the use of standardized interfaces. It is remarkable that the participants consider reusability and interoperability of all adaptive components as possible and consider Web Services as most promising approach. Concluding, from a technical point of view, reusability and interoperability are the most important future directions for personalization.

It is further mentionable that even the group of participants with a strong technical background named a large number of non-technical approaches to make personalization more scrutable for the user. The participants recommended to take the user into the focus when designing a personalized application: The personalization process shall be more transparent and visible for the user, advantages of integrating personalization into an application shall be expressed more explicitly for the user. These trends show that personalization needs to be seen in a larger context. It is not enough to personalize based on previous knowledge gathered by a single application. Personalization should also take the possibly quickly changing usage context into account as well as exploit Web 2.0 and social network data, like friend relationships or characteristics of a group of users, to overcome problems like slow adjustment or weak performance.

Chapter 3

A Framework for Generic Personalization

The conducted literature research and the survey shows that experts in the area of personalization desire a Web Service-based personalization platform, which provides interoperable and reusable personalization functionality. In this chapter, we model and implement a framework that assists application developers to create personalized Web applications. In Section 3.1, we first study related work in the area of the Semantic Web, covering service-oriented architectures, Semantic Web Services, and matchmaking, which could be used to build such a flexible framework. The Personal Reader [Abel et al., 2005, Henze and Krause, 2006], a design approach to split applications into logic parts, serves as a basis for our framework. The core idea of the newly developed Personal Reader Framework is the concept of making personalization functionality reusable by encapsulating the functionality into Web Services. These Web Services are called *Personalization Services* and are accompanied by a machine-processable semantic description of the provided functionality using Semantic Web techniques. Thus, functionality can be discovered dynamically and applied to existing applications in a plug-and-play manner.

Functionalities that are required by the majority of adaptive applications, like user authentication, or functionalities that shall operate across applications, like user modeling, can be accessed via a centralized component, called *Connector Service*. The Framework and its components are described in Section 3.2. Section 3.3 describes the personalized matchmaking of Personalization Services and the personalized portal of the Personal Reader Framework.

3.1 Related Work on Semantic Web Techniques for Generic Personalization and User Modeling

Personalization as well as user modeling are based on an efficient processing of data: on the one hand a large amount of data needs to be processed, on the other hand both fields benefit from accessing and merging different data sources in order to improve user and item profiles. While the first task is not in the scope of this thesis, we consider Semantic Web techniques as a promising approach for data federation for personalization and user modeling.

3.1.1 Introduction into the Semantic Web

The World Wide Wide is a web made for humans. HTML is used to structure information in a human-visualizable format. Machines can hardly access information on the Web in an automated fashion: NLP techniques, which require a high computational effort and are not error-free, are required to interpret the information on HTML Web sites. As the information, which is available on the Web, grows exponentially, the need and benefit of processing Web data by machines becomes more important. For building a Web for humans and machines, Tim Berners-Lee coined the term of the Semantic Web. He defined the vision of the Semantic Web as follows:

> "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." Tim Berners-Lee [Berners-Lee et al., 2001]

To realize the Semantic Web idea, Berners-Lee proposed a stack architecture (see Figure 3.1) where every layer builds upon and extends the previous layer. Uniform Resource Identifiers (URI) and Unicode are used to reference web objects uniquely and exchange documents over language boundaries. The extensible markup language (XML) uses the concept of elements and attributes to structure documents in a machine-processable format. XML Schema is used to define the structure of an XML document and the element and attribute names. To disambiguate element and attribute names, namespaces provide unique URI prefixes, which clearly define the validity of XML terms.

On top of structured XML documents, the Resource Description Framework (RDF) is used to add machine-processable meta-data. RDF triples consists of a subject, a predicate, and an object and can be read as a natural language-based sentence. With RDF it is for example possible to specify the properties of an instance, like X has the color red. These RDF triples act on the instance level as they add properties to objects that are accessible via a URI or relate different objects with each other by relationships. However, RDF does not contain a machine-processable semantics as there are no ontological rules how to

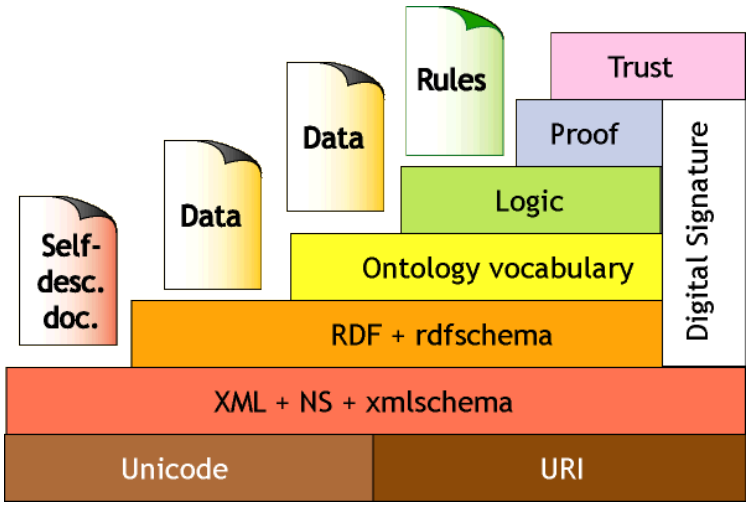


Figure 3.1: Berners-Lee's Semantic Web Stack from 2000 [Berners-Lee, 2000]

interpret the RDF statements. The RDF Schema (RDFS) layer first introduces semantics by defining classes, properties as well as hierarchical relationships. RDFS hence allows to specify that, for example, the class car is a subclass of vehicle. Given the additional information that X is a car, reasoning tools can now infer knowledge that was not explicitly given, like that instance X is not only a car but also a vehicle. The ontology layer, which is realized by the Web Ontology Language (OWL), extends the expressivity of RDFS by several new relationships, the use of XML Schema datatypes, cardinalities, and other new language constructs. Due to the expressive power of OWL, three OWL dialects have been standardized, namely OWL-Full, OWL-DL, and OWL-Lite. OWL-Full contains the entire feature set of OWL, OWL-DL contains a subset of OWL-Full which allows the creation of efficient reasoning algorithms. OWL-Lite is a subset of OWL-DL and the most limited OWL dialect. It is intended to be used for mobile environments where processing power is limited.

The upper layers of the Semantic Web stack are still in their definition phase and no W3C standard is yet published. The purpose of the rule layer is to provide reasoning mechanism that are able to infer new knowledge by exploiting the information given by ontologies as well as knowledge on instances level from different sources. A major challenge for the reasoners is to process a Web scale amount of input data. The proof layer will provide provenance data, like information source, used inference mechanism etc. to allow a client to verify how trustworthy a given information is. The trust layer aims at establishing trust between single users that finally leads to a global network of trust.

3.1.2 Service Oriented Architectures

While the Semantic Web stack defines how different techniques build upon each other to process and exchange data, nothing is said about the underlying software architecture of Semantic Web-enabled applications. *Service Oriented Architectures* (SOA) [Perrey and Lycett, 2003] are a software engineering approach to create modularized software applications, which build upon Semantic Web techniques standards, like XML. The main building block of SOA are so-called *Web Services*. A Web Service encapsulates functionality and provides a standardized interface to access the functionality. The *Web Service Definition Language*[1] (WSDL) is used to describe the interfaces syntactically by using XML Schema. For example, a WSDL document states that a web service offers a method *getPersonDetails* that requires a string *person* as input parameter. However, WSDL does not allow to link the parameter *person* to an ontological concept person stating that the name of a person shall be passed.

The *Universal Description, Discovery and Integration*[2] (UDDI) framework is a directory service for web service descriptions, which provides different discovery functionality. *White Pages* allow to search based on information about the service provider, *Yellow Pages* allow a search based on the rough purpose of the Web Service while the *Green Pages* contain the searchable WSDL descriptions of the registered Web Services.

3.1.2.1 Semantic Web Services

WSDL and UDDI are industry standards for describing and discovering Web services. However, their focus lies on specifying the structure of the service interfaces and the exchanged messages.

Thus, they address the discovery problem relying on structural, keyword-based matching, which limits their search capabilities. Other earlier works have also focused on applying Information Retrieval techniques to the service discovery problem. For example, the work presented by [Dong et al., 2004] deals with similarity search for Web services, using a clustering algorithm to group names of parameters into semantically meaningful concepts, which are then used to determine the similarity between input/output parameters. An online search engine for Web services is seekda[3], which crawls and indexes WSDL files from the Web. It allows users to search for services by entering keywords, by using tag clouds, or by browsing different facets, such as the country of the service provider, the most often used services or the most recently found ones.

To deal with the shortcomings of keyword search, several approaches have been proposed for exploiting ontologies to semantically enhance the service

[1] http://www.w3.org/TR/wsdl
[2] http://www.uddi.org/pubs/uddi_v3.htm
[3] http://seekda.com/

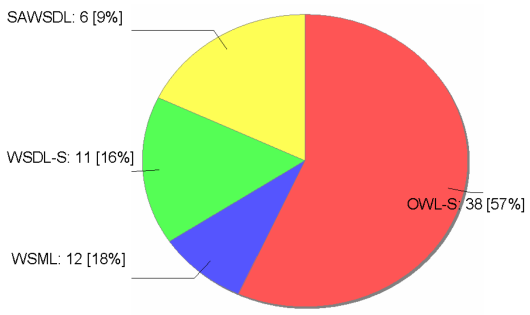


Figure 3.2: Distribution of Semantic Web Services based on service description from [Klusch and Zhing, 2008]

descriptions (SWASDL[4], WSDL-S [Akkiraju and et. al., 2005], OWL-S [Burstein and et. al., 2004], WSMO/WSML [Lausen et al., 2005]). These so-called Semantic Web services can better capture and disambiguate the service functionality, allowing for formal, logic-based matchmaking. Figure 3.2 illustrates a distribution of Semantic Web service description formats among real-world Semantic Web Services. We will focus on the two most often used formats, namely WSMO and OWL-S.

WSMO is an ontology with the four main concepts *ontologies*, *Web Services*, *Goals*, and *Mediators*. Ontology describes the domain-knowledge that a service relies on to provide the functionality. Web Services provides the semantic description of a Web Service and goals provide the vocabulary to specify the service request. Mediators define mappings between different specifications of ontologies and goals. The Web Service Modeling Language (WSML) uses WSMO and adds Description Logics and Logical Programming to describe further aspects of a Web Service.

OWL-S describes a service by four components *Service*, *Service Profile*, *Service Process Model* and *Service Grounding*. Service is an organizational class to link to the three underlying components. Service profile describes the high-level functionality while service process model describes the internal functionality of the process. This allows to distinguish for example services that have the same input and output parameters but a different algorithm to process the data. Service grounding finally contains technical invocation details like the endpoint URL. It is remarkable that in OWL-S both, a service discovery request and a service description use the same format. This is particular useful for discovering Web Services, which is also known as matchmaking.

3.1.2.2 Matchmaking of Semantic Web Services

Matchmaking describes the task of finding most appropriate Web Services for a given service request, describing the requested functionality. A logic reasoner

[4]http://www.w3.org/2002/ws/sawsdl/

is employed to infer subsumption relationships between requested and provided service parameters [Paolucci et al., 2002, Li and Horrocks, 2003]. Along this line, several matching algorithms assess the similarity between requested and offered inputs and outputs by comparing the positions of the corresponding classes in the associated domain ontology [Cardoso, 2006, Skoutas et al., 2007, Skoutas et al., 2008]. Similarly, the work in [Bellur and Kulkarni, 2007] semantically matches requested and offered parameters, modeling the matchmaking problem as one of matching bipartite graphs. In [Hau et al., 2005], OWL-S services are matched using a similarity measure for OWL objects, which is based on the ratio of common RDF triples in their descriptions. An approach for incorporating OWL-S service descriptions into UDDI is presented in [Srinivasan et al., 2004], focusing also on the efficiency of the discovery process. Efficient matchmaking and ranked retrieval of services is also studied in [Constantinescu et al., 2005].

Given that logic-based matching can often be too rigid, hybrid approaches have also been proposed. In an earlier work [Colgrave et al., 2004], the need for employing many types of matching has been discussed, proposing the integration of multiple external matching services to a UDDI registry. The selection of the external matching service to be used is based on specified policies, e.g., selecting the first available, or the most successful. If more than one matching services are invoked, again the system policies specify whether the union or the intersection of the results should be returned. OWLS-MX [Klusch et al., 2006] and WSMO-MX [Kaufer and Klusch, 2006] are hybrid matchmakers for OWL-S and WSMO services, respectively. More recently, an approach for simultaneously combining multiple matching criteria has been proposed [Skoutas et al., 2009].

On the other hand, some approaches already exist about involving the user in the process of service discovery. Ontologies and user profiles are applied in [Balke and Wagner, 2003], which are then used by techniques like query expansion or relaxation to better satisfy user requests. The work presented in [Xu et al., 2007] focuses on QoS-based Web service discovery, proposing a reputation-enhanced model. A reputation manager assigns reputation scores to the services based on user feedback regarding their performance. Then, a discovery agent uses the reputation scores for service matching, ranking and selection. The application of user preferences, expressed in the form of soft constraints, to Web service selection is considered in [Kießling and Hafenrichter, 2002], focusing on the optimization of preference queries. The approach in [Lamparter et al., 2007] uses utility functions to model service configurations and associated user preferences for optimal service selection. In [Dong et al., 2004], different types of similarity for service parameters are combined using a linear function, with manually assigned weights. Learning the weights from user feedback is proposed, but it is left as an open issue for future work.

3.1.3 Visualizing Semantic Web Data

One drawback of RDF data is that it does not contain meta-data about how to display the information, like HTML does. Therefore, solutions for visualizing Semantic Web data are required.

Currently, we can distinguish two main strategies for providing a view for Semantic Web data: the first strategy visualizes RDF documents without taking into account any particularities of the underlying domain knowledge of the RDF documents. Examples are Piggy Bank, Longwell[5] or Brownsauce[6]. These tools are, more appropriately, called RDF browsers.

The second strategy for providing Semantic Web browsing is focusing on a certain domain, which might be narrow (as in the case of DynamicView [Gao et al., 2005] or mSpace [Shadbolt et al., 2004]) or broad (Haystack [Quan and Karger, 2004] or SEAL [Hartmann and Sure, 2004]). These approaches' architectures are all based on a domain-specific fundament requiring considerable modifications for applying them in other domains. At this time there exists no approach that copes with both issues at the same time: being generic enough to handle any application domain while offering a domain optimized user interface.

3.1.4 Discussion

In this section we gave a short introduction into the Semantic Web and outlined how RDF and OWL can be used for knowledge representation. A major advantage of the Semantic Web is the clear distinction of data and meta-data, which simplifies the exchange of information and the inference of new information utilizing reasoning mechanisms.

We explored Service-oriented architectures, which split an application into loosely-coupled distributed Web Services, having a clearly defined interface. Semantic Web Services build upon the Service-oriented architecture and describe their functionality in a machine-readable format. With Semantic Web Services, new applications can be created automatically by composing existing services. Matchmaking is a technique for performing this automatic composition by discovering Semantic Web Services for a specific task.

3.2 Architecture of the Personal Reader Framework

The Personal Reader Framework (see Figure 3.3) aims at supporting programmers in the development of interoperable, personalized Semantic Web applications. Applications are split into logical parts and encapsulated in reusable

[5] http://simile.mit.edu/longwell/
[6] http://brownsauce.sourceforge.net/

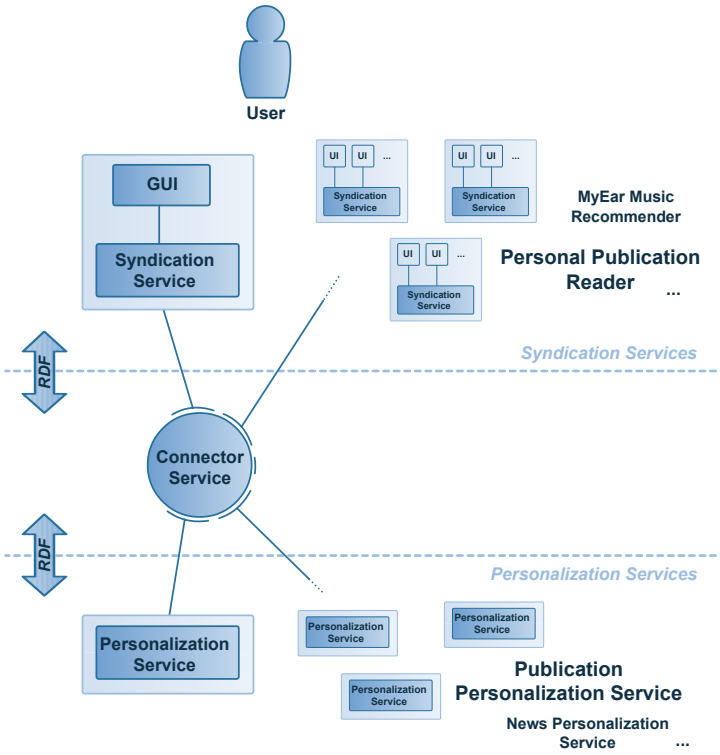


Figure 3.3: The basic Personal Reader Framework.

Web Services. The framework distinguishes three different types of services: a) Personalization Services, b) Syndication Services, and c) a Connector Service.

Personalization Services (PServices for short) provide a specific personalization functionality by accessing and processing a specific part of the Semantic Web, mostly one specific domain. For this domain, PServices contain domain-specific knowledge and offer methods to access, personalize and process the Semantic Web data. These PServices are registered at the *Connector Service* (CService for short) that maintains a directory of available PServices and their offered functionality, stored as OWL-S [Burstein and et. al., 2004] description. *Syndication Services* (SynServices for short), which contain the business logic of an application, invoke the Connector Service to discover personalization functional offered by PServices. Users enter an application by a user interface that is optimized for their device and personal preferences. The user interface is provided by the corresponding SynService. It reports user interactions to the SynService and receives and visualizes personalized data.

A more detailed description about the SynServices, CService and PServices is given in the next sections, as well as the communication between them (see Section 3.2.4).

3.2.1 Personalization Services

Applications can be enriched with personalization features in a plug-and-play manner by using Personalization Services.

Typical examples of Personalization Services range from services that simply wrap non-RDF data sources – e.g. a service that calls the Flickr API[7] considering the user's preferences and transforms the Flickr results into RDF using taxonomies like Dublin Core Metadata Element Set[8] – to services that carry out more complex tasks – e.g. a music recommender service that searches for music and filters music items based on user's preferences. This service 1) detects feeds in the music domain, 2) filters the content of the detected feeds according to the user profile and her context, and 3) aggregates the relevant items into a new feed.

In general, Personalization Services provide a personalized view on data available on the Semantic Web. To provide data, PServices perform mostly reasoning or information filtering tasks and use different kinds of data for the tasks: a) the applications context passed by the invoking SynService, b) user data from a centralized repository [Abel et al., 2008] and c) the domain-specific knowledge. Thus, applications can focus on their functionality instead of taking care about changes in domain-specific knowledge or the processing of input data.

Personalization Services are described using the Semantic Web Services standard OWL-S [Burstein and et. al., 2004] so that they can be discovered and used by other services at runtime. Therefore, the CService provides an interface to register new PServices in the framework. After registration, new PServices can be used immediately. The Personal Reader Framework provides the so-called *Configuration Ontology* to describe the input and output parameters of PServices in a standardized vocabulary.

3.2.2 Syndication Services

The Syndication Services contain the business logic of an application and interact directly with the CService and the user interfaces. A typical Personal Reader setting, that illustrates how a SynService can offer added value by combining different basic functionality, provided by PServices, is given within the *Personal Publication Reader* [Abel et al., 2005]:

Personalization Service A provides users with recommendations for scientific publications according to the users' interests. Service B offers detailed information about authors or researchers. By integrating both services via a Syndication Service users can browse publications they like within an embedded context.

[7] http://www.flickr.com/services/api/
[8] http://dublincore.org/documents/dces/

To receive (personalized) data, SynServices invoke Personalization Services, which allow a personalized access to a specific part of the Semantic Web. To invoke a PService, the SynService first creates an OWL-S based Service request. The request contains a) the Semantic description of the needed functionality, b) user-specific information that can be passed to the PService if it is executed and c) further information that can be provided to invoke the PService successfully (e.g. parameters like search keywords, etc.).

If the CService discovers appropriate PServices, a list of PService candidates is passed to the SynService. The SynService selects some PServices that shall be executed and invokes them by passing an invocation request to the CService. The CService then passes the invocation request to the PServices and receives the invocation results that are finally passed to the SynService. The Personal Reader deliberately does not allow a direct communication between PService and SynService to be able to better detect malicious services by observing the communication and to adhere to user's preferences regarding which services shall be invoked.

3.2.3 Connector Service

The *Connector Service* (*CService* for short) is an application-independent centralized component which performs and controls information exchange between the single services (mainly between PService and SynService) within the framework. Therefore all communication between PServices and SynServices is passed to the CService that forwards the messages to the corresponding services. By controlling the communication at a central point, user's restrictions on PServices are enforced. For example, users can define that only those PServices shall be invoked that are free of charge or that are trusted by a trust authority. Other pragmatic benefits of the centralized architecture are the simplified registration and discovery of Syn- and PServices and a unified access to centralized functionality.

The second task of the CService is to provide interfaces for application-independent core functionality of the Personal Reader framework. This includes interfaces for user modeling tasks, which are passed to a central user modeling service, managing lists of available PServices and SynServices and the discovery of PServices with a specific functionality.

3.2.4 Message Exchange Format

The *Configuration Ontology* defines, on the one hand, the vocabulary that is needed to describe the inputs of Web Services and, on the other hand, concepts that are required for personalization functionalities. Figure 3.4 illustrates the concepts of the Configuration Ontology.

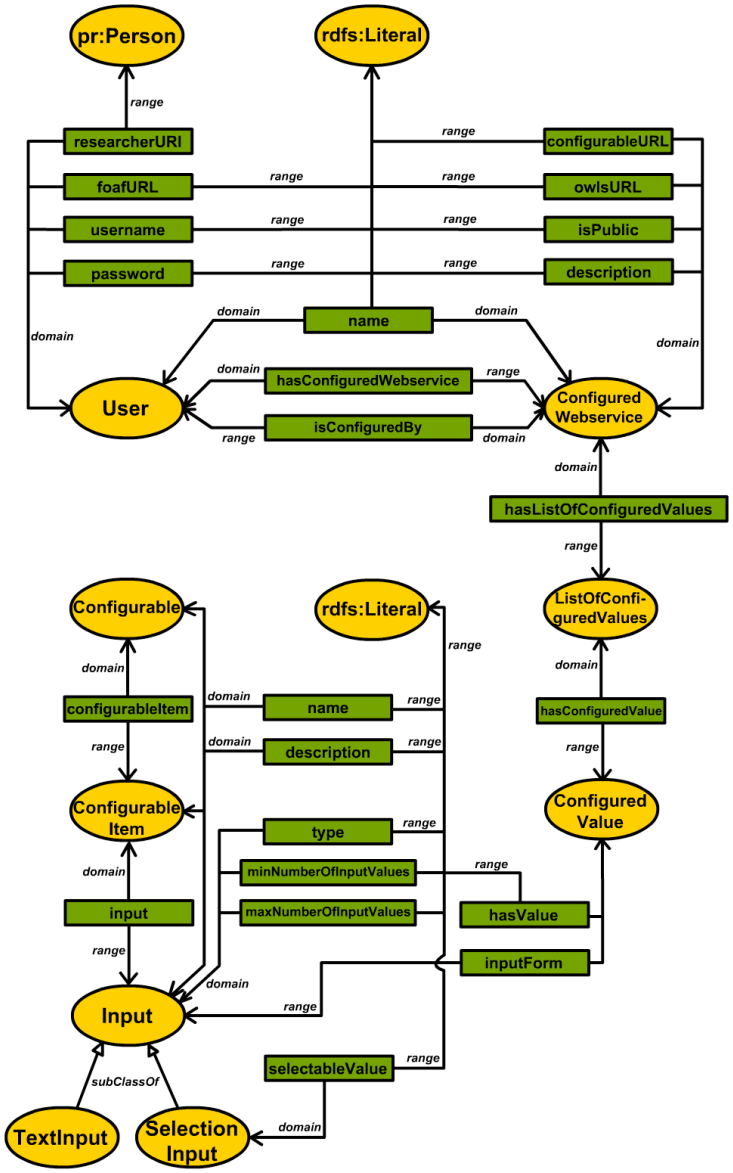


Figure 3.4: *Configuration Ontology* for describing adjustable inputs of Personalization Services.

Core Configurable Vocabulary (needed to describe a Configurable Web Service):

Configurable An instance of this class characterizes the *configurable* inputs of a Personalization Service. The `name` and a `description` of the Web Service are defined as follows:

```
(#MyEarConfigurable, name, "MyEar Configurable")
(#MyEarConfigurable, description, "Configurable things of my MyEar Music Web Service")
```

ConfigurableItem A `Configurable` consists of several `ConfigurableItems`. Example:

```
(#MyEarConfigurable, hasConfigurableItem, #DurationItem)
(#DurationItem, name, "Duration")
(#DurationItem, description, "Duration of a Song that should be
                              taken into account by my Web Service.")
```

Input Every `ConfigurableItem` has at least one `Input`. We define two special Inputs: a `SelectionInput`, which allows only predefined values, and a `TextInput`, which allows arbitrary values. For an Input a `type`, a `minNumber`- and a `maxNumberOfInputValues` have to be specified. Example:

```
(#DurationItem, input, #MinDurationInput)
(#MinDurationInput, description, "The minimum duration of a song (in minutes)")
(#MinDurationInput, type, http://www.w3.org/2001/XMLSchema#nonNegativeInteger)
(#MinDurationInput, minNumberOfInputValues, 0)
(#MinDurationInput, maxNumberOfInputValues, 1)

(#DurationItem, input, #MaxDurationInput)
...
```

User and their configured Personalization Services – concepts needed to realize personalization functionalities:

User This concept models the users of the Personal Reader. A `User` is a subclass of `foaf:Person` and is featured with a `username`, `password`, `name`, etc. and a list of `ConfiguredWebservices` (`hasConfiguredWebservice`). To link other descriptions, which characterize the user, we will use the UMService as introduced in the following chapter. Example of a user:

```
(#user1, username, "user1")
(#user1, name, "John Doe")
(#user1, foafURL, "http://www.example.com/foaf.rdf")
(#user1, hasConfiguredWebservice, #user1MyEarJazzConfigWS)
...
```

ConfiguredWebservice This concept is used to store configurations of Web Services made by a user. The properties `name` and `description` allow to describe the concrete configuration. The boolean property `isPublic` indicates whether a `ConfiguredWebservice` can be accessed and re-used by other users than the user who configured it (`isConfiguredBy`). `owlsURL` points to the OWL-S description of the Web Service that was configured by the user and `configurableURL` points to the `Configurable` description. The values that belong to the concrete configuration are listed within the `ListOfConfiguredValues`. Example:

```
(#abelFabianMyEarJazzConfigWS, name, "Jazz Music")
(#abelFabianMyEarJazzConfigWS, description, "This configuration of the MyEar Music Web
                                             Service effects the Web Service to aggregate
                                             podcasting items that are related with Jazz.")
(#abelFabianMyEarJazzConfigWS, isPublic, "true")
(#abelFabianMyEarJazzConfigWS, isConfiguredBy, #abelFabian)
(#abelFabianMyEarJazzConfigWS, owlsURL, "...MyEar/rdf/MyEarOWLS.owl")
(#abelFabianMyEarJazzConfigWS, configurableURL, #MyEarConfigurable)
(#abelFabianMyEarJazzConfigWS, hasListOfConfiguredValues, #abelFabianMyEarJazzValueList)
```

ListOfConfiguredValues This is a list of the values that are configured by a user. Each `ConfiguredValue` has a `value` (range: typed Literals) and a reference to the `Input` (`inputForm`) which defines what is applicable in general. Example:

```
(#abelFabianMyEarJazzValueList, hasConfiguredValue, #abelFabianMyEarJazzValue1)
(#abelFabianMyEarJazzValue1, value, "3")
(#abelFabianMyEarJazzValue1, inputForm, #MinDurationInput)
(#abelFabianMyEarJazzValueList, hasConfiguredValue, #abelFabianMyEarJazzValue2)
...
```

3.2.5 Conclusion

The reuse of personalization functionality and sharing of corresponding algorithms are an important requirement for the future of personalization (see also Section 2). The Personal Reader architecture enables sharing and reuse of personalization functionality across different applications by encapsulating personalization functionality into PServices. The framework uses state-of-the-art Semantic Web techniques and is due to the service based architecture extensible. In the next sections, we will have a detailed view how generic personalization functionality is provided by the framework.

3.3 Personalization in the Personal Reader Framework

The Personal Reader Framework provides mainly three building blocks for personalization:

1. Personalization functionality provided by Personalization Services.
2. Personalized configuration of the invocation of a Personalization Services.
3. Personalized discovery of Personalization Services.

While the basic concept of the personalization functionality provided be the PServices has been described in the last section, the personalized invocation of PServices is provided by the Personal Reader Agent. The Agent tries to complete PService invocation parameters automatically by searching appropriate properties from the user profile. A detailed description about the Agent will be given in chapter 5.

The personalized discovery of Personalization Services is handled in the Personal Reader by incorporating user preferences expressed as ratings when discovering PServices. The discovery is provided by a personalized matchmaking algorithm, which we will describe in detail.

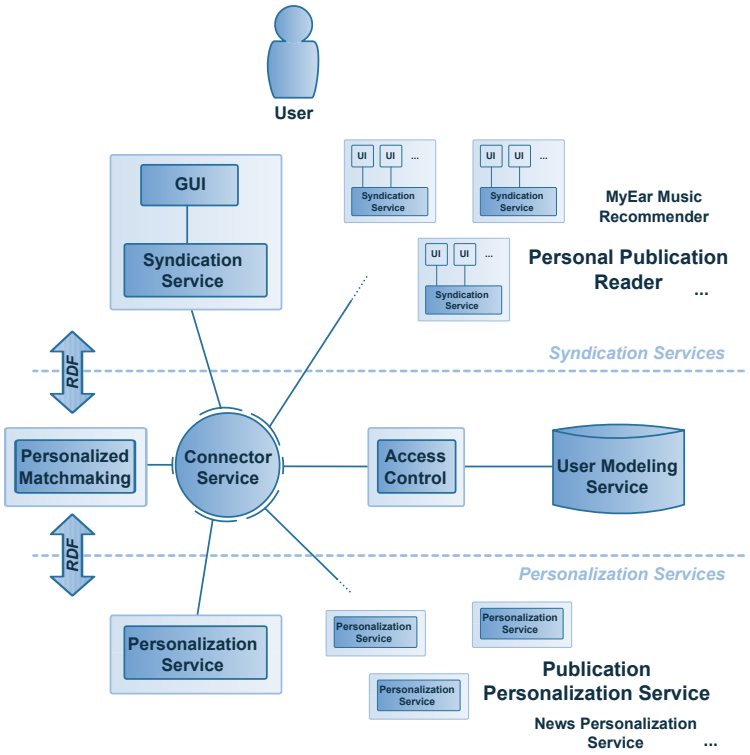


Figure 3.5: Extended Personal Reader Architecture: the personalized matchmaking component

3.3.1 Personalized Matchmaking of PServices

Personalization Services offer personalized functionality for applications. There are many settings available where different personalization strategies can be invoked to solve a problem. In the Personal Reader, we provide a meta-personalization approach that selects Personalization Services based on user preferences. Therefore, the provided functionality of each Personalization Service is described by using Semantic Web techniques. OWL-S provides an ontology to create a Web Service description that provides – among other information – input and output parameters of a Web Service. A Syndication Service can specify a service request, describing the required functionality as well as the application context information offered by the SynService.

We present a method for leveraging user feedback to improve the results of the service discovery process implemented in the Personal Reader Framework as *Personalized Matchmaking Service*: given a service request, the matchmaker searches the repository for available services and returns a ranked list of candidate matches. Then, the system allows the user posing the query to rate any of these matches, indicating how relevant or appropriate they are for this request. The provided ratings are stored in the system for future use, when the same or a similar request is issued.

Designing intuitive, easy-to-use user interfaces, can help the process of collecting user feedback. In this thesis, we do not deal with this issue; instead, our focus is on how the collected feedback is processed and integrated in the matchmaking process to improve the results of subsequent searches. Notice, that it is also possible to collect user feedback automatically, assuming that the system can track which service(s) the user actually used; however, this information would typically be incomplete, since not all relevant services are used.

3.3.1.1 Architecture of the Personalized Matchmaker

Typical service matchmaking systems are based on a unidirectional information flow. First, an application that needs a specific Web Service to perform a task creates a service request, containing the requirements that a service should fulfill. This service request is then delivered to a matchmaking component that utilizes one or more match filters to retrieve the best-matching services from a repository of Semantic Web Service descriptions. These services are finally returned to the application which invoked the matchmaker. The drawback in this scenario is that if a service is not appropriate or sufficient for any reason to perform the original task, the application has no option to inform the matchmaker about the inappropriateness of this match result.

Hence, our matchmaking architecture is extended by a feedback loop, as illustrated in Figure 3.6, enabling the matchmaking mechanism to use previously provided user feedback in order to improve the quality of the retrieved results.

Enabling this feedback loop relies on the assumption that the application users can assess the quality of retrieved Web services. This is a common principle in Web 2.0 applications, where users can rate available resources. One possibility is that users can rate services explicitly. If it is not possible or easy for the users to rate services directly, the application can still infer implicit ratings for a service through user behavior. For example, if an applications uses services to generate music recommendations, then users can be asked whether they consider the given recommendations appropriate. Based on the assumption that services delivering high quality recommendations are better matches for this task, the application can infer the relevance of a service, and pass this information as a user rating to the matchmaking service.

The user ratings are stored in Personal Reader's RDF-based user modeling service, entitled UMService (see Chapter 4). As user ratings refer to a given service request, each `Rating` instance contains the user who performed the rating, the service request, the rated service, and finally a rating score that ranges from 0 to 1 (with higher scores denoting higher rating). For example, a rating from Bob about a request X and a service Y would be stored as:

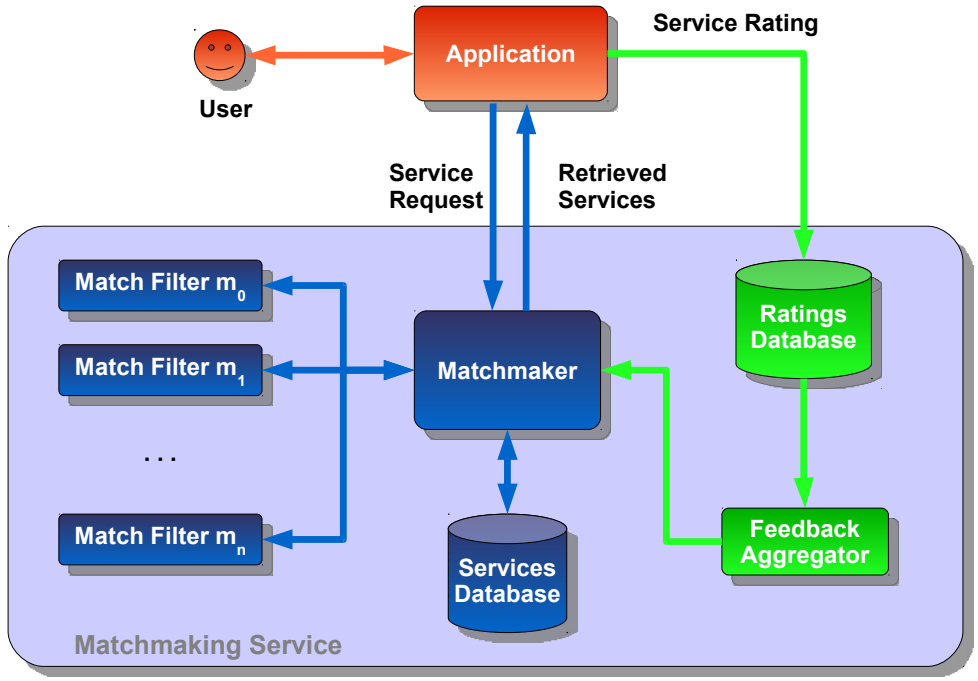


Figure 3.6: Matchmaking service with feedback component

```
<r:Rating>
  <foaf:Person rdf:about="#bob"/>
  <r:Request rdf:about="#requestX"/>
  <r:Service rdf:about="#serviceY"/>
  <r:Score rdf:datatype="&xsd;double">0.90</r:score>
</r:Rating>
```

The user feedback in form of ratings, is exploited by the user feedback component. This component aggregates previous ratings provided by different users, to determine the relevance between a service request and an actual service.

Then, given a service request, the matchmaker component combines the relevance score from the feedback component with the similarity scores calculated by the match filter(s) to assess the degree of match for each available service, and returns a ranked list of match results to the application.

3.3.1.2 Service Matchmaking

We first describe the basic service matchmaking and ranking process, without taking into account user feedback. For this task, we adopt the approach from [Skoutas et al., 2009]. The reason for this choice is that, as will be shown in the next section, it allows us to integrate user feedback in a more flexible and seamless way. In the following, we give a brief overview of how the matchmaking and ranking of services is performed.

Let R be a service request with a set of input and output parameters, denoted by R_{IN} and R_{OUT}, respectively. We focus on input and output parameters; other types of parameters can be handled in the same way. We use $R.p_j$ to refer to the j-th input parameter, where $p_j \in R_{IN}$ (similarly for output parameters). Also, assume an advertised service S with input and output parameters S_{IN} and S_{OUT}, respectively. Note that S can be a match to R, even when the cardinalities of their parameter sets differ, i.e., when a service advertisement requires less inputs or produces more outputs than requested.

The matchmaking process applies one or more matching functions to assess the degree of match among pairs of parameters. Each matching function, denoted by m_i, produces scores in the range $[0, 1]$, where 1 indicates a perfect match, while 0 indicates the lack of a match. Given a request R, a service S, and a matching function m_i, the *match instance* of S with respect to R is defined as a vector s_i such that

$$s_i[j] = \begin{cases} \max\limits_{p_k \in S_{IN}} \{m_i(S.p_k, R.p_j)\}, & \forall j : p_j \in R_{IN} \\ \max\limits_{p_k \in S_{OUT}} \{m_i(S.p_k, R.p_j)\}, & \forall j : p_j \in R_{OUT} \end{cases} \tag{3.1}$$

The match instance s_i has a total of $d = |R_{IN}| + |R_{OUT}|$ entries that correspond to the input and output parameters of the request. Intuitively, each s_i entry quantifies how well the corresponding parameter of the request R is matched by the advertisement S, under the matching criterion m_i. Clearly, an input (output) parameter of R can only match with an input (output) parameter of S.

Let $\mathcal{M}$ be a set of matching functions. Given a request R and an advertisement S, each $m_i \in \mathcal{M}$ results in a distinct match instance. We refer to the set of instances as the *match object* of the service S. In the following, we use the terms service and match object interchangeably, denoted by the same uppercase letter (e.g., S). On the other hand we reserve lowercase letters for match instances of the corresponding service (e.g., s_1, s_2, etc.). The notation $s_i \in S$ implies that the match instance s_i corresponds to the service S. Hence, a match object represents the result of the match between a service S and a request R, with each contained match instance corresponding to the result of a different match function.

Match Filter	Book	Price
M0	0.88	1.00
M1	0.93	1.00
M2	0.69	1.00
M3	0.72	1.00
M4	0.93	1.00

Table 3.1: Example of the match object for the request `book_price_service.owls` and the service `novel_price_service.owls`

As a concrete example, consider the request `book_price_service.owls` and the service `novel_price_service.owls`, both taken from the service collection OWLS-TC and matched applying the five matching filters M0–M4 of the OWLS-MX service matchmaker (see Section 3.3.1.5 for more information about OWLS-TC and OWLS-MX). The resulting match object is shown in Table 3.1.

Next, we describe how services are ranked based on their match objects. Let $\mathcal{I}$ be the set of all match instances of all services. Given two instances $u, v \in \mathcal{I}$, we say that u *dominates* v, denoted by $u \succ v$, iff u has a higher or equal degree of match in all parameters and a strictly higher degree of match in at least one parameter compared to v. Formally

$$u \succ v \quad \Leftrightarrow \quad \forall i \; u[i] \geq v[i] \; \wedge \; \exists j \; u[j] > v[j] \tag{3.2}$$

If u is neither dominated by nor dominates v, then u and v are incomparable.

Given this dominance relationship between match instances, we proceed with defining *dominance scores* that are used to rank the available service descriptions with respect to a given service request. Intuitively, a service should be ranked highly in the list if

- its instances are dominated by as few other instances as possible, and
- its instances dominate as many other instances as possible.

To satisfy these requirements, we formally define the following dominance scores, used to rank the search results for a service request.

Given a match instance u, we define the *dominated score* of u as

$$u.dds = \frac{1}{|\mathcal{M}|} \sum_{V \neq U} \sum_{v \in V} |v \succ u| \tag{3.3}$$

where $|u \succ v|$ is 1 if $u \succ v$ and 0 otherwise. Hence, $u.dds$ accounts for the instances that dominate u. Then, the dominated score of a service U is defined

as the (possibly weighted) average of the dominated scores of its instances:

$$U.dds = \frac{1}{|\mathcal{M}|} \sum_{u \in U} u.dds \tag{3.4}$$

The dominated score of a service indicates the average number of services that dominate it, i.e., a *lower* dominated score indicates a better match result.

Next, we look at the instances that a given instance dominates. Formally, given a match instance u, we define the *dominating score* of u as

$$u.dgs = \frac{1}{|\mathcal{M}|} \sum_{V \neq U} \sum_{v \in V} |u \succ v| \tag{3.5}$$

Similarly to the case above, the dominating score of a service U is then defined as the (possibly weighted) average of the dominating scores of its instances:

$$U.dgs = \frac{1}{|\mathcal{M}|} \sum_{u \in U} u.dgs \tag{3.6}$$

The dominating score of a service indicates the average number of services that it dominates, i.e., a *higher* dominating score indicates a better match result.

Finally, we define the *dominance score* of match instances and services, to combine both of the aforementioned criteria. In particular, the dominance score of a match instance u is defined as

$$u.ds = u.dgs - \lambda \cdot u.dds \tag{3.7}$$

where the parameter λ is a scaling factor. This promotes u for each instance it dominates, while penalizing it for each instance that dominates it. Then, the dominance score of a service U is defined as the (possibly weighted) average of the dominance scores of its instances:

$$U.ds = \frac{1}{M} \sum_{u \in U} u.ds \tag{3.8}$$

The ranking process comprises computing the aforementioned scores for each service, and then sorting the services in descending order of their dominance score. Efficient algorithms for this computation can be found in [Skoutas et al., 2009].

3.3.1.3 Incorporating User Feedback

As described in Section 3.3.1.1, our approach is based on the assumption that the system collects feedback from the users by allowing them to rate how

appropriate the retrieved services are with respect to their request. Assume that the collected user ratings are stored as a set $\mathcal{T} \subseteq \mathcal{U} \times \mathcal{R} \times \mathcal{S} \times F$ in the Ratings Database, where $\mathcal{U}$ is the set of all users that have provided a rating, $\mathcal{R}$ is the set of all previous service requests stored in the system, $\mathcal{S}$ is the set of all the available Semantic Web service descriptions in the repository, and $F \in [0, 1]$ denotes the user rating, i.e., how relevant a particular service was considered with respect to a given request (with higher values representing higher relevance). Thus, a tuple $T = (U, R, S, f) \in \mathcal{T}$ denotes that a user U considers the service $S \in \mathcal{S}$ to be relevant for the request $R \in \mathcal{R}$ with a score f.

To aggregate the ratings from different users into a single feedback score, different approaches can be used. For example, [Whitby et al., 2004] employs techniques to identify and filter out ratings from spam users, while [Yu et al., 2004] proposes the aging of feedback ratings, considering the more recent ratings as more relevant. It is also possible to weight differently the ratings of different users, assigning, for example, higher weights to ratings provided previously by the same user as the one currently issuing the request, or by users that are assumed to be closely related to him/her, e.g., by explicitly being included in his/her social network or being automatically selected by the system through techniques such as collaborative filtering or clustering. However, as the discussion about an optimal aggregation strategy for user ratings is orthogonal to our main focus in this paper, without loss of generality we consider in the following all the available user ratings as equally important. Therefore, we calculate the feedback value as the average of all user ratings of the corresponding service. Hence, the feedback score fb between a service request $R \in \mathcal{R}$ and a service advertisement $S \in \mathcal{S}$ is calculated as:

$$fb(R, S) = \frac{\sum_{(U,R,S,f)\in\mathcal{T}} f}{|\{(U, R, S, f) \in \mathcal{T}\}|} \tag{3.9}$$

However, it may often occur that for a given pair of a request R and a service S, no ratings (U, R, S, f) exist in the database. This may be because the request R is new, or because the service S has been recently added to the database and therefore has been rated only for a few requests. Moreover, even if some ratings exist, they may be sparse and hence not provide sufficiently reliable information for feedback. In these cases, Equation (3.9) is not appropriate for determining the feedback information for the pair (R, S). To address this issue, we generalize this method to consider not only those ratings that are directly assigned to the current service requests R, but also user ratings that are assigned to requests that are similar to R. Let $SIM(R)$ denote the set of requests which are considered to be similar to R. Then, the feedback can be calculated as:

$$fb(R,S) = \frac{\sum_{(U,Q,S,f)\in\mathcal{T}:Q\in SIM(R)} f * sim(R,Q)}{|\{(U,Q,S,f)\in\mathcal{T} : Q\in SIM(R)\}|} \tag{3.10}$$

In Equation (3.10), $sim(R,Q)$ is the match instance of Q with respect to R, calculated by a similarity measure m_i, as discussed in Section 3.3.1.2. Notice that $sim(R,Q)$ is a vector of size equal to the number of parameters of R, hence in this case $fb(R,S)$ is also such a vector, i.e., similar to a match instance. Also, Equation (3.9) can be derived as a special case of Equation (3.10), by considering $SIM(R) = \{R\}$. By weighting the given feedback by the similarity between the requests, we ensure that feedback from requests which are more similar to the considered one, is taken more into account.

A question that arises is how to select the similar requests for a given request R, i.e., how to determine the set $SIM(R)$. This choice involves a trade-off. Selecting a larger number of similar queries allows the use of more sources of information for feedback; however, if the similarity between the original request and the selected ones is not high enough, then the information from this feedback is also not highly appropriate, and may eventually introduce noise in the results. On the other hand, setting a very strict criterion for selecting similar queries, reduces the chance of finding enough feedback information. As a solution to this trade-off, we use a top-k query with constraints: given a request R, we select the top-k most similar requests from the database, given that the values of their match instances are above a specified threshold.

The process described above results in a *feedback instance* $fb(R,S)$ for the given request R and a service S. The next step is to integrate this instance to the match object of the service S, comprising the other instances obtained by the different similarity measures m_i. We investigate two different strategies for this purpose:

1. *Feedback instance as an additional match instance.* In this case we add the feedback information to the match object of the service as an additional instance (combined with the average of the previous values). That is, this method treats the feedback mechanism as an extra matchmaking function.

2. *Feedback instance integrated with match instances.* In this case we update the values of the match instances by adding the values of the feedback instance. That is, this method adjusts the results of the matchmaking functions applying the feedback information.

As a concrete example, consider the match object presented in Table 3.1. Assume that the feedback instance for the pair (`book_price_service.owls`, `novel_price_service.owls`) is

(a) Method 1

Match Filter	Book	Price
M0	0.88	1.00
M1	0.93	1.00
M2	0.69	1.00
M3	0.72	1.00
M4	0.93	1.00
AVG(M_i)+FB	1.60	2.00

(b) Method 2

Match Filter	Book	Price
M0+FB	1.65	2.00
M1+FB	1.70	2.00
M2+FB	1.46	2.00
M3+FB	1.49	2.00
M4+FB	1.70	2.00

Table 3.2: Example of the match object for the request `book_price_service.owls` and the service `novel_price_service.owls` updated using feedback information

$fb = [0.77 \;\; 1.00]$.

Then this match object will be modified as shown in Table 3.2.

3.3.1.4 Personalized Matchmaking

For the personalized matchmaking, we use a domination based matchmaking approach, as described in [Skoutas et al., 2009]. This approach uses the skyline algorithm [Kossmann et al., 2002] to combine multiple matchmaking metrics. Besides the existing matchmaker metrics $M_0 - M_4$ from the OWLS-MX matchmaker [Klusch et al., 2006], we define an additional metric rec_x, that expresses whether a service shall be recommended to a user or not.

Assume that the collected user ratings are stored as a set $\mathcal{T} \subseteq \mathcal{U} \times \mathcal{R} \times \mathcal{S} \times F$ in the ratings database, where $\mathcal{U}$ is the set of all users that have provided a rating, $\mathcal{R}$ is the set of all previous service requests stored in the system, $\mathcal{S}$ is the set of all the available Semantic Web service descriptions in the repository, and $F \in [0, 1]$ denotes the user rating, i.e., how relevant a particular service was considered with respect to a given request (with higher values representing higher relevance). Thus, a tuple $T = (U, R, S, f) \in \mathcal{T}$ denotes that a user U considers the service $S \in \mathcal{S}$ to be relevant for the request $R \in \mathcal{R}$ with a score f.

The recommendation score rec_1 of a service s_1 and a given request r_1 for a specific user u_1 can be calculated as the average of the previous ratings from the user u_1 for service s_1 in respect to request r_1:

$$rec_1(u_1, s_1, r_1) = \frac{\sum_{(u_1,s_1,r_1,f)\in\mathcal{T}} f}{|\{(u_1, s_1, r_1, f) \in \mathcal{T}|\}} \tag{3.11}$$

However, if a user specifies a request for the first time this formula is not applicable. We can overcome this new-request problem by assuming that for similar requests a user will rate services similarly.

If $SIM_r \subseteq R$ denotes a set of services requests that are considered as similar to a given service request r and $sim(r_1, r_2) \in [0, 1]$ denotes the similarity value

between r_1 and r_2, rec_2 is calculated by:

$$rec_2(u_1, s_1, r_1) = \frac{\sum_{x \in X} f * sim(r_1, r_2)}{|X|} \tag{3.12}$$

with

$$X := \{(u_1, s_1, r_2, f) \in \mathcal{T} : r_2 \in SIM_{r1}\} \tag{3.13}$$

Hence, the more similar a request r_2 is to a given request r_1, the more important is the given feedback of s_1 to r_2 for r_1.

As the amount of available Web Services grows rapidly (already today the latest OWLS test collection[9] contains more than 1000 Semantic Web Services) the user ratings - service matrix will become very sparse. Hence, the above formula will not be applicable in many cases.

To overcome the sparsity problem, we now consider also ratings from other users u_2, which are similar to the given user u_1. We consider users to be similar if they have rated services similarly. Assume that the users are represented by their rating vector, $sim(u_1, u_2)$ denotes the cosine similarity between the two rating vectors of the users u_1 and u_2. Further, SIM_u contains the set of users that are considered to be similar to user u. Then, the collaborative filtering approach as presented in [Shardanand and Maes, 1995] can be applied to rec_3 by:

$$rec_3(u_1, s_1, r_1) = \frac{\sum_{y \in Y} f * sim(u_1, u_2) * sim(r_1, r_2)}{|Y|} \tag{3.14}$$

with

$$Y := \{(u_2, s_1, r_2, f) \in \mathcal{T} : r_2 \in SIM_{r1}, u_2 \in SIM_{u1}\} \tag{3.15}$$

Hence, ratings from very similar users that rated a service s_1 in the context of a given request r_2 that is very similar to the request r_1 is considered as highly relevant for the recommendation score of s_1 in respect to r_1.

3.3.1.5 Experimental Evaluation

In this section, we evaluate the quality of our feedback-based matchmaking approach in comparison to state-of-the-art matchmaking algorithms.

[9] available at http://www.semwebcentral.org/projects/owls-tc/

Collection	# of requests	# of services	# of rel. services per req. (average)
OWL-S TC I	28	576	15.2
OWL-S TC II	28	1007	25.4

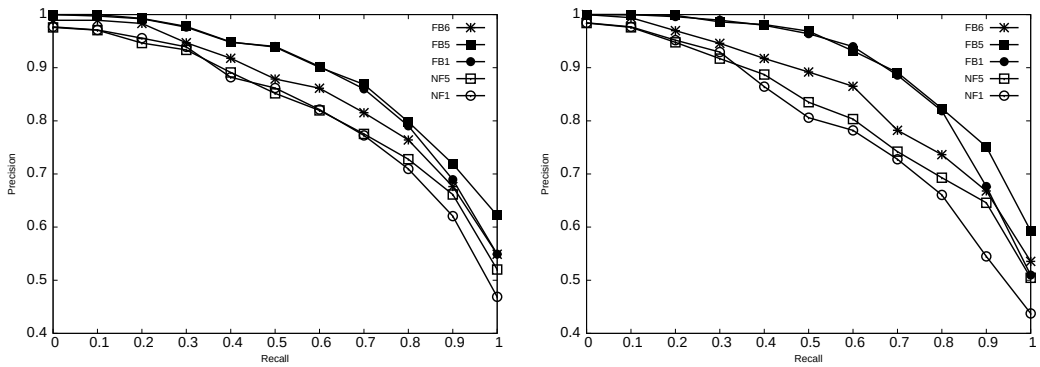
Table 3.3: Characteristics of the test collections

Experimental Setup We have implemented the feedback-based matchmaking and ranking process described in Sections 3.3.1.2 and 3.3.1.3. The implementation utilizes the OWLS-MX service matchmaker [Klusch et al., 2006], to process service requests and advertisements described in OWL-S, and to compute the pairwise similarities between parameters. In particular, OWLS-MX provides 5 different matching filters. The first performs a purely logic-based match (M0). The other four perform hybrid match, by combining the semantic-based matchmaking with the following measures: loss-of-information (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4). Notice, that for each pair (R, S) of a service request and service advertisement, OWLS-MX applies one of the filters M0–M4, and calculates a single score denoting the degree of match between R and S. We have modified this functionality to get all the individual degrees of match between the compared parameters of R and S (i.e., a vector); also, we have applied for each pair (R, S) all the similarity measures M0–M4, to get the individual match instances, as described in Section 3.3.1.2. Finally, our implementation includes also the process described in Section 3.3.1.3 for processing and using the available feedback information.

For our experiments, we have used the publicly available service retrieval test collection OWLS-TC v2[10]. This collection comes in two versions, an original one containing 576 services, and an extended one, containing 1007 services. To better assess the performance of our method, we have conducted our experiments on both versions, denoted in the following as OWLS-TC I and OWLS-TC II, respectively. The contained service descriptions are based on real-world Web services, retrieved mainly from public IBM UDDI registries, covering 7 different domains, such as economy, education, and travel. Also, the collection comprises a set of 28 sample requests. Notice that the extended version of the collection comprises one extra request, namely `EBookOrder1.owls`; however, in our experiments, we have excluded this request, so that in both cases the set of queries used for the evaluation is the same. For each request, a relevance set is provided, i.e., the list of services that are considered relevant to this request, based on human judgement. The characteristics of the two data sets are summarized in Table 3.3.

To evaluate our feedback-based mechanism, there needs to be, for each

[10]This collection is available at `http://projects.semwebcentral.org/projects/owls-tc/`. Before running the experiments we have fixed some typos that prevented some services from being processed and/or retrieved.



(a) OWLS-TC I (b) OWLS-TC II

Figure 3.7: Precision-Recall curve for the OWLS test collections

request, at least one similar request for which some services have been rated as relevant. As this was not the case with the original data set, due to the small number of provided requests, we have extended both of the aforementioned collections by creating a similar query for each of the 28 original ones. This was done by selecting a request, then selecting one or more of its input and/or output parameters, and replacing its associated class in the ontology with one that is a superclass, subclass or sibling. Then, for each of these newly created queries, some of the services in the collection were rated as relevant. To simplify this task, we have restricted our experimental study in binary ratings, i.e., the value of the user rating was either 1 or 0, based on whether the user considered the service to be relevant to the request or not. The new queries and the ratings, provided in the form of corresponding relevance sets, are made available for further use at: `http://www.l3s.de/~krause/collection.tar.gz`.

Experimental Results In the following, we evaluate the performance of our approach, including both strategies described in Section 3.3.1.3. For this purpose, we compare the retrieved results to the ones produced without taking user feedback into consideration. In particular, we have implemented and compared the following 5 methods:

- *NF1*: no feedback is used; one match instance per service is considered. The values of the match instance are the degrees of match between the request and service parameters, computed applying the Jensen-Shannon similarity measure, i.e., the filter M4 from OWLS-MX, which is shown in [Klusch et al., 2006] to slightly outperform the other measures.

- *NF5*: no feedback is used; five match instances per service are considered. The values of the match instances are the degrees of match between the

request and service parameters computed by the filters M0–M4 of OWLS-MX.

- *FB1*: feedback is used; one match instance per service is considered. The values of the match instance are the sum of the degrees of match between the request and service parameters computed by M4 and the feedback values calculated by Equation (3.10).

- *FB5*: feedback is used; five match instances per service are considered. The value of each match instance is the sum of the degrees of match between the request and service parameters computed by one of the measures M0–M4 and the feedback values calculated by Equation (3.10).

- *FB6*: feedback is used; six match instances per service are considered. The values of the first five match instances are the degrees of match between the request and service parameters computed by the filters M0–M4. The values of the sixth match instance are computed as the averages of the previous ones plus the feedback values calculated by Equation (3.10). Notice, that the reason for using also the average values of the initial instances, instead of only the feedback values, is mainly to avoid penalizing services that constitute good matches but have not been rated by users.

To measure the effectiveness of the compared approaches, we apply the following standard IR evaluation measures [Manning et al., 2008]:

- *Interpolated Recall-Precision Averages*: measures precision, i.e., percent of retrieved items that are relevant, at various recall levels, i.e., after a certain percentage of all the relevant items have been retrieved.

- *Mean Average Precision (MAP)*: average of precision values calculated after each relevant item is retrieved.

- *R-Precision (R-prec)*: measures precision after all relevant items have been retrieved.

- *bpref*: measures the number of times judged non-relevant items are retrieved before relevant ones.

- *Reciprocal Rank (R-rank)*: measures (the inverse of) the rank of the top relevant item.

- *Precision at N (P@N)*: measures the precision after N items have been retrieved.

(a) OWLS-TC I

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
FB6	0.8427	0.7772	0.8206	0.9762	0.9214	0.8357	0.7690	0.6589
FB5	0.8836	0.7884	0.8600	1.0000	0.9714	0.8857	0.7952	0.6696
FB1	0.8764	0.7962	0.8486	1.0000	0.9786	0.8786	0.7929	0.6625
NF5	0.8084	0.7543	0.7874	0.9405	0.9071	0.7964	0.7500	0.6393
NF1	0.8027	0.7503	0.7796	0.9405	0.9214	0.8143	0.7357	0.6357

(b) OWLS-TC II

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
FB6	0.8426	0.7652	0.8176	1.0000	0.9714	0.8964	0.8476	0.7875
FB5	0.9090	0.8242	0.8896	1.0000	0.9857	0.9679	0.9214	0.8536
FB1	0.8960	0.8024	0.8689	1.0000	0.9857	0.9607	0.9167	0.8411
NF5	0.8007	0.7388	0.7792	0.9643	0.9429	0.8607	0.8119	0.7536
NF1	0.7786	0.7045	0.7499	0.9643	0.9357	0.8607	0.7976	0.7268

Table 3.4: IR metrics for the OWLS test collections

Figure 3.7 plots the precision-recall curves for the 5 compared methods, for both considered test collections. Overall, the main observation is that the feedback-aware methods clearly outperform the other two ones in both test collections. The best overall method in both collections is $FB5$, because it provides two advantages: a) it utilizes user feedback, and b) it combines all the available similarity measures for matchmaking service parameters. The method $FB1$, which combines feedback information with the Jensen-Shannon hybrid filter, also demonstrates a very high accuracy. The method $FB6$, which treats the feedback information as an additional match instance, achieves lower precision, but it still outperforms the non-feedback methods. This behavior is due to the fact that although feedback is utilized, its impact is lower since it is not considered for the 5 original match instances, but only as an extra instance. Regarding $NF5$ and $NF1$, the former exhibits better performance, which is expected as it combines multiple similarity measures. Another interesting observation is that $FB5$ and $FB1$ follow the same trend as $NF5$ and $NF1$, respectively, which are their non-feedback counterparts, however having considerably higher precision values at all recall levels. Finally, for the collection OWLS-TC II, which comprises an almost double number of services, the trends are the same as before, but with the differences between the feedback-aware and the non-feedback methods being even more noticeable. Another interesting observation in this case is that after the recall level 0.8 the precision of $FB1$ drops much faster than that of $FB6$; thus, although $FB1$ has an overall higher performance than $FB6$, the latter appears to be more stable, which is due to having more instances per match object, i.e., taking into account more similarity measures.

Table 3.3.1.5 presents the results for the other IR evaluation metrics discussed above. These results again confirm the aforementioned observations.

For all the considered metrics, $FB5$ and $FB1$ perform better, followed by $FB6$.

3.3.2 Conclusion

Current state-of-the-art matchmaking algorithms generate recommendations regardless of a user's preferences. This issue becomes more serious as most modern Web 2.0 applications allow users to explicitly express their opinion by giving feedback about available resources, in the form of rating, tagging, etc. We extended the Personal Reader Framework to collect user feedback on retrieved services and incorporate it in the Semantic Web Service matchmaking process. We have proposed different methods to combine user feedback with dominance based-matchmaking algorithms in order to improve the quality of the match results. To overcome the problem of limited amount of feedback or of previously unknown requests (i.e., where no previous feedback is available for the request), we utilize information from similar requests. To compare our feedback-aware matchmaking strategies to state-of-the-art matchmaking algorithms that do not take feedback into account we used a publicly available collection of OWL-S services. Our experimental results show that user feedback is a valuable source of information for improving the matchmaking quality.

3.4 Critical Review of the Personal Reader Framework

In the introduction we defined five research questions that need to be tackled to provide support for personalization in Web Service-based environments. We will now revise these questions and verify if the proposed Personal Reader Framework can help to answer the questions. The five questions were:

1. Can the strongly-coupled personalization process of monolithic applications be divided into logic and independent services?

2. Can such personalization services be reused in various applications?

3. How shall user profiles be stored, maintained, and accessed in a Semantic Web Service-based environment?

4. Can personalization be used to orchestrate personalized applications from single Web Services?

5. Which requirements need to be fulfilled by a personalization framework and which support need to be offered to assist the programmer to create personalized applications?

Regarding question 1: the Personal Reader Framework splits an application into logical parts and encapsulates them into Web Services. An applications consists of an Syndication Service and is supplemented by Personalization Services: while SynServices encapsulate the application logics, PServices encapsulate personalization functionality into Semantic Web Services. An example for such a PService is a content-based recommender algorithm: the idea is that the SynService delivers input data, like items and their features, the algorithm then processes the data and generates recommendations which are passed back to the SynService. In the Personal Reader Framework, there exists a reasonable amount of PServices (detailed statistics will be given in Chapter 5), which simplifies the (re-)use of personalization in new applications and showcases that personalization can be externalized in various application scenarios.

Regarding question 2: the Personal Reader Framework supports application developers to reuse existing PServices. The plug-and-play concept allows existing applications to benefit from future improvements of algorithms and newly emerging PServices. PServices can be used and interpreted off the shelf and hence decrease development costs of personalized applications significantly. This motivates programmers to discover and use existing PServices during runtime. Our state-of-the-art matchmaking algorithm does not only take the global quality of a service into account when it searches for PServices, but also preferences of a user. Different real-world scenarios will be presented in Chapter 5 where PServices are successfully reused by different applications.

Regarding question 3: the UMService, which will be presented in detail in the next chapter, is a centralized component in the Personal Reader Framework and allows all Personal Reader services to store and access the profiles of the users. For the users, the advantage of keeping profile data separate from the applications in a centralized repository is that they need to maintain and update only one profile. Slow adjustment of personalization is reduced as new services can access the entire user profile if the users allow this. A simple-to-use user interface allows user to specify precisely which service is allowed to access what kind of user data. Developers have the advantage of a simplified management of user profile data as defined interfaces for accessing and storing data exist.

Regarding question 4: in the Personal Reader, applications are orchestrated according to a user's preferences by: a) allowing to fill PService invocation parameters based on user profile information and b) select PServices, that a SynService should invoke, based on user preferences. Compared to existing personalized applications, not only the data, interface or functionality is personalized, but also the composition of the application code is selected based on user preferences.

Regarding question 5: a personalization framework needs to support the entire lifecycle of a personalized application. The Personal Reader Framework provides support for the creation of PServices, SynService and entire Personal

Reader applications: Personal Reader libraries transform RDF messages into Java objects and vice versa so that programmers do not need to have a deeper understanding of Semantic Web techniques. The matchmaker simplifies the discovery of existing, reusable personalization functionality while the UMService takes care on persisting and retrieving information about the user. All central services can be invoked by simple Java methods without the need to instantiate a Web Service or performing Web Service calls. Utilizing the Personal Reader framework, an application developer can focus on creating the application logic while personalization and user modeling can be implemented with a low additional implementation effort.

3.5 Conclusion

In this chapter we presented the core components of the Semantic Web and introduced the concept of service-oriented architectures. The building blocks of a SOA application, namely Web Services are annotated by machine-readable metadata and become so-called Semantic Web Services. Semantic Web Services allow an automatic discovery of functionality with the help of matchmaking. The Personal Reader Framework building upon those Semantic Web techniques and assists programmers at the creation of personalized, service-based applications. The single building blocks of a Personal Reader application, namely PServices, for providing external personalization functionality, SynService, which encapsulate the business logic of an application and search for PServices, and the Connector service, supporting the discovery and communication with PServices, were introduced. The underlying concepts of the Personal Reader Framework, namely plug-and-play personalization and encapsulation of personalization functionality are ensured by the architecture.

We have shown and discussed that the Personal Reader contributes to the state-of-the-art in the area of personalization by encapsulating generic personalization functionality, fostering reuse of existing personalization algorithms. In the area of matchmaking, we provide a personalized matchmaking algorithm, which incorporates Web 2.0-style feedback, namely ratings, into the matchmaking process. Evaluations prove that our personalized matchmaker outperforms non-personalized state-of-the-art matchmaker.

Chapter 4

Web Service-based Generic User Modeling

In traditional desktop environments, users interact with an application over a long term. Hence, applications can create user profiles by observing the behavior of the users and due to the long-term usage users are mostly willing to adapt applications by explicitly specifying applications' options. On the Web, applications are created via a dynamic network of services that are interweaved with each other. In such setting, the number of available applications increases while users access most of such web applications only seldomly or just once. This effect is even enforced when using the Personal Reader Framework: for example, a user accesses an application by invoking a SynService, which calls two other PServices to provide the requested functionality. While the SynService can gather low-level events, like mouse clicks, it will pass only those user-specific events and observations to a PService that are required to execute the PService. However, PServices contain background knowledge that enable them to interpret user interactions in the context of the domain and thus infer knowledge about a user that would not have been possible without background knowledge. Therefore, it is important that different services can create and update a central user profile collaboratively.

When a service is accessed for the first time, it cannot rely on the users' support to provide (sensitive) user profile data. Instead, services need to retrieve and exchange existing information about a user in order to build a detailed profile about a user and avoid the cold start problem [Schein et al., 2002]. In Section 4.1 we will inspect related work in the area of generic user modeling: typical solutions for a shared user profile are User Modeling Servers [Kobsa, 2001] or approaches that use a Lingua franca, like the Generalized User Modeling Ontology [Heckmann et al., 2005]. However, both approaches require the services to refuse their own user profile storage format and adhere to a shared format. If new concepts, or facets of a user need to be described, these central vocabularies needs to be changed.

In Section 4.2 we propose the User Modeling Service (*UMService* for short), a domain-independent central storage place for cross-application user profiles, that enables services to use their own vocabulary to model a user. The UMService is a centralized web service, storing and maintaining the user profiles and providing interfaces to access and modify the profile information. The service can be accessed via the Connector Service (see Figure 4.1), which provides interfaces to the UMService to be accessed by Syn- and PServices.

A serious concern of shared user profiles are privacy issues: while a trustful application known by the user is allowed to access her bank account information, an unknown application should not be allowed to access the same data. Existing work on RDF data protection does not suit to enforce user-defined policies on RDF-based user profiles: available solutions do not handle contextual information in a proper way, as they either require a large amount of memory or unacceptably increase the response time. To address these problems we decided to enforce access control as a layer on top of RDF stores (see Section 4.2.5), which also has the positive side-effect of making our solution store-independent. For this access control system, rule-based policy languages, like Protune [Bonatti and Olmedilla, 2005a, Bonatti and Olmedilla, 2005b], can be used as they allow precisely to specify which application can operate on which data at which time. We realize a user interface that enables non-expert users to control the access to their RDF-based user profiles. For the ease-of-use we provide configurable access policy templates and embed them into the user interface. The user interface provides immediate feedback to the user, which includes information about which part of the RDF data is covered by the policy and additionally a graphical presentation about consequences that the specified policy has.

4.1 Related Work on Generic User Modeling

Jameson [Jameson, 2003] defines user modeling as a task, which fills the user profile by processing the low-level information about the user (see Figure 4.2). This low-level information is collected by the application and is simple and non-processed observations, like click events. Reasoning is used to fill the final user profile with high-level information about the user.

Please note that some authors, like Jameson, call the user profile also the user model. We use the term *user profile* for information about a user that is stored (for example, processed high-level inferences or non-processed demographic data or direct input by the user)[1]. We refer to the term *user model* if we describe the rule-set or formalism that describes how to transfer observations into high-level user profile information.

[1] we also consider observations as part of the user profile if these observations are stored and could be used in a later point of time for applying personalization or inferring high-level information

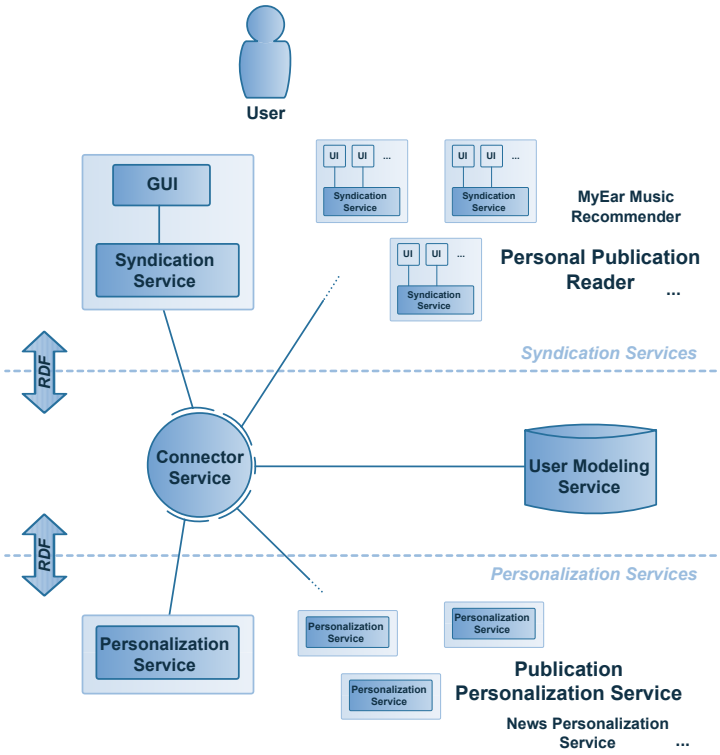


Figure 4.1: Extended Personal Reader Architecture: the user modeling component

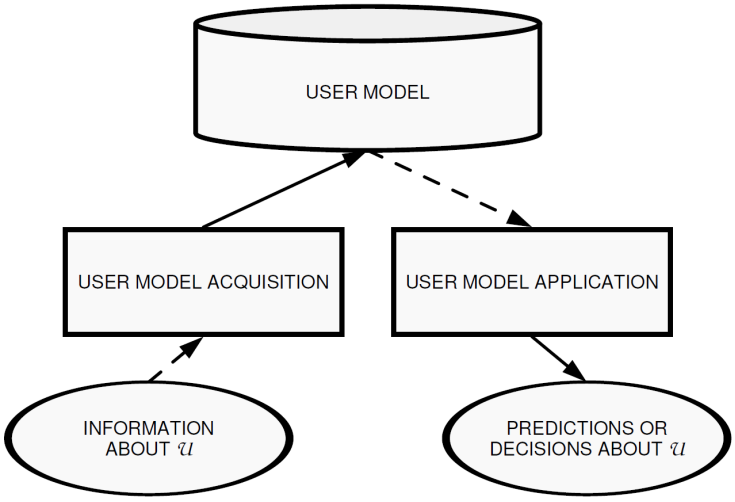


Figure 4.2: General schema of a user modeling and adaptation process from [Jameson, 2003]

We consider a user modeling system generic if all the central components of such a system are domain- and application-independent. By domain-independence we refer to the fact that the core components of a user modeling system do not provide or rely on domain-specific functionality or information.

Application-independence means that different applications can use the user modeling system for different usage settings. A generic user modeling system is composed of several central components: events can either be detected and reported generically on the user modeling system's site or be provided in a generic format by the non-generic applications. The user modeling process, the user profile itself as well as parts of the user profile application, namely the process of deriving personalized data based on the user profile (for example, the generation of recommendations), shall be generic. The non-generic application can use additional domain-knowledge to adapt the application further (for example filter the delivered recommendations by availability in a shop). In this section, we first present User Modeling Shells and User Modeling Servers that offer generic user modeling functionality. We then discuss generic user profile storage formats and finally cover related issues like shared user modeling as well as handling privacy issues. For examples of generic algorithms, that apply the user profile, we refer to presented related work for generic personalization (see Section 2.1).

4.1.1 User Modeling Shells

The first approaches to separate user modeling from the application, were coined User Modeling Shells [Kobsa, 1990] to express the interaction character of the systems. First systems, like the GUMS [Finin and Drager, 1986] or the BGP-MS [Kobsa and Pohl, 1995] provided high-level functions to query and update the user profile and maintained the user profiles apart from the application.

GUMS provides methods to add new information about a user and to query the GUMS user profile. Stereotypic user modeling [Rich, 1979] is used to complete user profile information: predefined stereotypes contain user profile properties, which are considered to be valid for a stereotypic group of users (e.g. the group of computer scientists have a high interest in math). So-called triggers describes the required observations (e.g. a user accesses the computer science faculty's website) that are sufficient to assign a user to a stereotype. Overall, GUMS focuses at modeling the long-term user profile of a user.

BGP-MS, in comparison, allows an application to report the user's actual goal or observations, aiming at the short term user context. As the shell is – from a programmer's perspective – a part of the application, it can also initiate interaction with the application's user interface component to interact with the user directly or to inform the application about important events (like newly drawn conclusions) in the user profile. Inference capability is offered by some pre-defined components that the application's developer needs to enrich by domain knowledge. User profile exchange as well as distributed user modeling were originally not covered in User Modeling Shells.

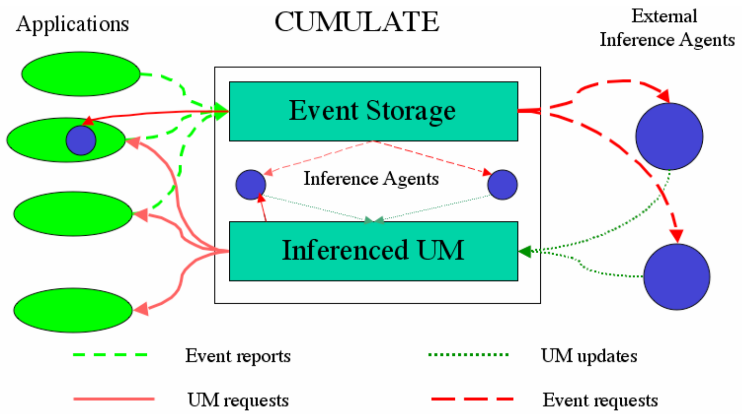


Figure 4.3: Layout of the CUMULATE server from [Brusilovsky et al., 2005a]

4.1.2 User Modeling Servers

In comparison to User Modeling Shells, User Modeling Servers are application-independent software components that provide well-defined interfaces. A detailed comparison of different User Modeling Servers is given by Fink [Fink, 2004]. We will showcase the servers: CUMULATE and PersonIs.

4.1.2.1 CUMULATE

CUMULATE [Brusilovsky et al., 2005a, Yudelson et al., 2007] is a user modeling server for the E-Learning domain and was developed by Brusilovsky et al. CUMULATE uses a topic-based overlay model to represent the knowledge level of students. Course authors therefore have to specify topics that are covered by their course and define how activities that can be performed within an E-Learning system should influence the knowledge level of a topic.

The CUMULATE server contains two independent repositories (see Figure 4.3), a so-called event storage and an inferenced user model. The E-Learning applications log low-level user activity and send them as events to the CUMULATE server that stores the events directly in the event storage. So-called inference agents then access the raw event data and try to infer from the events information about the topic-based knowledge of the user. The inference agents finally update the user profile accordingly.

4.1.2.2 PersonIs

The PersonIs [Kay et al., 2002] architecture is focussed on user control and scrutability. It is composed of the PersonIs User Model Server (see Figure 4.4), that stores the user profile information. User profile data is delivered by the applications, which are in the PersonIs architecture adaptive hypermedia

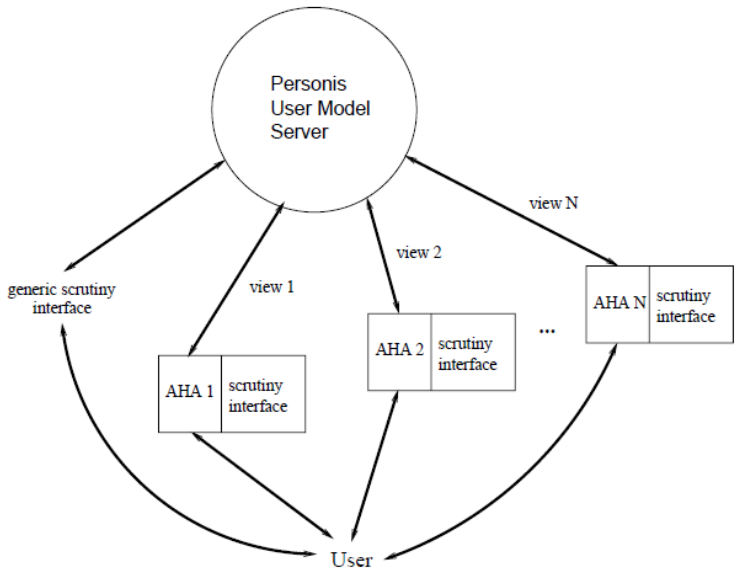


Figure 4.4: Layout of the PersonIs server from [Kay et al., 2002]

systems. It is remarkable that for every application, a separate scrutinity interface is offered that allows users to inspect and control their user profile. Views offer access to a (limited) part of the user profile provided by the PersonIs User Model Server. This ensures that only that part of the user profile is provided to that application that is needed and can be processed by the application. Views are also used to enforce user controlled access rules, hiding confidential user profile information from an application.

The storage format of the PersonIs user profile contains attribute-value pairs together with evidences. These evidences are observations and actions taken by the system's reasoners as reaction of the observation (for example activation of a stereotype). PersonIs offers two methods, *tell* and *ask* to submit observations and retrieve user profile information.

4.1.3 Generic User Profile Formats

The presented User Modeling Shells and Servers operate purely on a self-defined data format and do not describe the stored information in a semantic way. Hence, applications have to specify exactly what kind of information in which storage format they need. A search using inference to specify the required information on a semantic level is not possible.

More recent approaches, that cover a semantic description of the user profile data are Friend of a Friend (FOAF) and the Generalized User Modeling Ontology. These approaches will be described in more detail.

```
<rdf:RDF>
  <foaf:Person>
    <foaf:name>Daniel Krause</foaf:name>
    <foaf:givenname>Daniel</foaf:givenname>
    <foaf:depiction
     rdf:resource="http://www.daniel-krause.org/daniel.jpg"/>

    <foaf:knows>
      <foaf:Person>
        <foaf:name>Fabian Abel</foaf:name>
        <rdfs:seeAlso
         rdf:resource="http://www.l3s.de/~abel/foaf.rdf"/>
      </foaf:Person>
    </foaf:knows>

    <foaf:Organization>
      <foaf:name>L3S Research Center</foaf:name>
      <foaf:homepage rdf:resource="http://www.l3s.de/"/>
    </foaf:Organization>
  </foaf:Person>
</rdf:RDF>
```

Figure 4.5: Example of a FoaF file

4.1.3.1 Friend of a Friend

The Friend of a Friend[2] project has defined an RDF-based ontology to describe persons as well as their relationship. By using the FOAF ontology, users can describe personal properties, like name, email address, affiliation, as well as providing links to people they know. By following these links, a social network arises, called a FoaF network. An example of a FoaF file is given in Figure 4.5. As FoaF uses RDF, any RDF based-ontology can be used to extend the original FoaF vocabulary: by using a Geo data ontology[3], users can, for example, annotate their home or work location. With graphical browsers like the FoaF Explorer[4], users can navigate a FoaF network.

FoaF provides a simple vocabulary for defining a user profile, however it is not useful for expressing fine-grained properties. Still, it shows how Semantic Web techniques like RDF and the Linked Data paradigm[5] can build a shared and extendable user profile.

[2] http://www.foaf-project.org/
[3] http://www.w3.org/2003/01/geo/wgs84_pos#
[4] http://xml.mfd-consult.dk/foaf/explorer/
[5] http://www.w3.org/DesignIssues/LinkedData.html

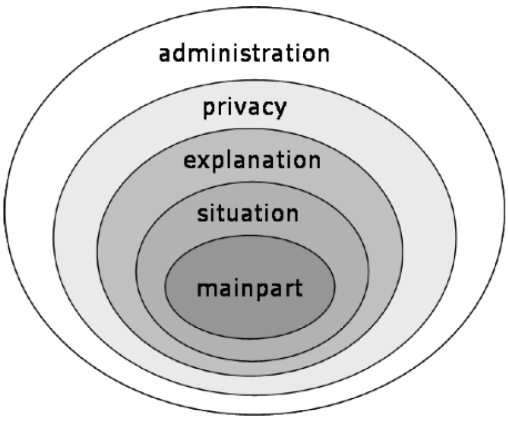


Figure 4.6: Metadata layers of SituationStatements from [Heckmann, 2005]

4.1.3.2 Generalized User Modeling Ontology

The Generalized User Modeling Ontology[6] (GUMO) [Heckmann et al., 2005] is mainly an extensible ontology that allows to express various user profile statements. GUMO allows to incorporate knowledge from various domains by refining and extending the ontology's concepts. The Web Ontology Language (OWL) was chosen as underlying ontology language for GUMO. The main concept of GUMO are *SituationalStatements*. The main information of such statements is expressed by a basic RDF triple structure, namely a subject, a predicate and an object. This basic RDF structure is extend it by the *auxiliary* and *range* concept. Such an extended RDF statement is called *mainpart* of the SituationalStatement. According to [Heckmann et al., 2005], a person's medium interest in football would be expressed by the following mainpart:

```
subject: #DanielKrause
auxiliary: hasInterest
predicate: football
range: low-medium-high
object: low
```

SituationalStatements can be enriched with further metadata that are layers around the mainpart. These layers are depicted in Figure 4.6.

The *situation* layer contains spatial and time constraints; the *explanation* layer contains an explanation for the user about how the statement was derived and who created it. The *privacy* layer implements a simple role based access control while the *administrative* layer contains internal information, like linkage between statements and a unique ID or URL to make single statements referable.

[6] an experimental version can be found at http://www.ubisworld.org

GUMO allows a fine grained and detailed description of user profile data. However, due to the definition of the ontology and the extensible nature, it is hard to maintain consistency when the ontology grows.

4.1.4 User Profile Exchange

Today, ubiquitous scenarios became reality, where users interact with different devices at different locations and times. Mobile phones have an impressive processor power, enabling Web browsing, Email exchange and the execution of arbitrary desktop applications. In such a scenario, different devices as well as different applications will create and maintain domain-specific user profiles. To transfer existing user profile information from one device to another, different solutions have been proposed.

Besides generic user modeling approaches as discusses in the previous section, there exist user modeling systems that try to enhance their locally maintained profiles by exchanging user profile information with other systems.

In this section we present three approaches for distributed user modeling: a) user profile query, b) user profile integration and c) OpenSocial.

Retrieving User Profile Information with UserQL UserQL [Heckmann, 2005] is an XML-based query language to receive user profile information. It is based on the Generalized User Modeling Ontology and uses so-called *SituationRequests* to query a user profile. Every SituationRequest is composed of *SituationalQueries* which contains three components, namely the match box, the filter box and the control box. The match box corresponds to SituationStatements as described in Section 4.1.3.2 and allows to specify properties that the SituationStatements in the query result need to fulfill. This can be used to search for all statements about a specific user or to retrieve all interests in the user profile. The filter box provides additional metadata to describe the purpose of the request and further constraints like a minimum confidence level of the returned statements. The control box allows to specify the repository that will be queried and some postprocessing options, like conflict handling, and aggregation of statements.

The weak point of UserQL is that it is tightly coupled to SituationStatements and does not adhere to standard query languages, like SQL or SPARQL.

User Profile Integration User profile integration describes the process of merging different user profiles. This problem occurs especially in the area of group recommender systems, where several users will have access to a shared medium, like music playlists at a party or the selection of a movie in the cinema. Yu et al. [Yu et al., 2006] propose a profile merging algorithm for shared watching TV. Their profile merging algorithm performs on user profiles containing

attribute value pairs between -1 (dislike) and 1 (like). First, the algorithm selects those attributes where most of the users have a similar rating (like or dislike). All other attributes are then removed from the original user profiles and the remaining user profile attributes are normalized. Then, the value for the common attributes of the merged user profile is calculated as the average of the single ratings. The merging algorithm handles contradictions very well while it cannot be applied to more complex user profiles that do not contain numerical ratings.

Heckmann [Heckmann, 2005] reduces the user profile merging task to the task of conflict resolution. Two user profiles are merged by merging the observations from the single user profiles. Then these new observations are used to fill an empty user profile. As Heckmann's architecture provides inference and conflict resolution by the user profile storage, contradicting observations can be solved so that the merged user profile will be be generated from the merged observations. This solution is very convenient as it does not require additional programming effort for providing merging functionality. All the required functionality, like conflict resolution is also needed to handle contradicting observations in a single user profile. The disadvantage of this solution is that the inference engine must be accessible by the user profile. Thus, decentralized inference engines that do process their own observation during runtime cannot be implemented in this setting. A disadvantage of the approach is scalability: it requires to keep all low-level observations, which might require large storage capacity. Further performance issues might arise when merger and process a large set of observations.

OpenSocial OpenSocial[7] provides an API that is supported by several Social Network sites, like XING[8], MySpace[9], and MeinVZ[10]. It provides standardized methods for third party applications to access user profile information in the Social Network. The user profile includes demographic information, friendship relations and the communication between the users. An application created for XING which uses the OpenSocial API to receive user data and uses OpenSocial Gadgets, a JavaScript-based rendering engine, can hence be executed without changes at any other Social Network, supporting OpenSocial.

While OpenSocial does not provide methods for aggregating or exchanging user profiles between platforms, it is up to the application to aggregate user profile data. The main advantage of OpenSocial is that it reduces the costs of porting personalized applications between different social networks. OpenSocial can be considered to be currently the most successful approach in industry for generic personalization.

[7] http://code.google.com/apis/opensocial/
[8] http://www.xing.com/
[9] http://www.myspace.com/
[10] http://www.meinvz.net/

4.1.5 Privacy Protection of User Profiles

Privacy protection is an essential requirement to gain the trust of the users and their willingness to contribute data [Kobsa, 2007]. Self-determination about how to use, change, and exchange user related information must be ensured by the user modeling systems. We require the support of machine-readability and availability of reasoners and due to the fact that any user profile data can be stored in RDF, we focus on access control systems for RDF data.

Most current RDF databases provide no or only rudimentary access control mechanisms. For example, one of today's most widespread RDF database management systems, *Sesame* [Broekstra et al., 2002], allows to define access rights only for a whole database. Hence, access to all triples stored in a Sesame repository is either allowed or prohibited. Other standard protocols to access RDF data such as the SPARQL protocol [Clark et al., 2008], do not support any access control.

Semantic policy languages (e.g., KAOS [Uszok et al., 2003], Rei [Kagal et al., 2003], PeerTrust [Gavriloaie et al., 2004] or Protune [Bonatti and Olmedilla, 2005a]) lately emerged in order to address these requirements: they provide the ability to specify complex conditions both on (i) the data in the repository to be accessed itself and (ii) external conditions such as time constraints, or even interfaces to query external packages such as other repositories. However, in the context of RDF stores, evaluating such constraints for each triple to be potentially returned is not affordable for result sets exceeding a certain size.

Filtering query results in a separate post-processing step after query execution as proposed by Cozzi et al. [Cozzi et al., 2006] is not an adequate solution for restricting access to RDF: current RDF query languages allow to arbitrarily structure the results, as shown in the following example[11].

```
CONSTRUCT {CC} newNs:isOwnedBy {User}
  FROM {User} ex:hasCreditCard {CC};
              foaf:name {Name}
  WHERE Name = 'Alice'
```

Here, post-filtering the query results is not straightforward since the result structure is not known in advance. In fact, not the results produced by the query, but rather only the data accessed in the `FROM` clause should be restricted. It could be possible to split constructs queries into (i) a select query and (ii) the generation of the returned graph (construct), therefore avoiding this problem. However, the query response time may be considerably too large since this

[11] Our examples use SeRQL [Broekstra and Kampman, 2004] syntax (and for simplicity we do not include the namespace definitions)

approach cannot make use of repository optimizations and policies are enforced after all data (allowed and not allowed) has been retrieved.

A different way to address this problem is defining *a priori* which subsets of an RDF database can be accessed by some requester. This approach is taken in [Carroll et al., 2005] which shows how Named Graphs can be used to evaluate SPARQL queries [Prud'hommeaux and Seaborne, 2008]. A framework which first applies all rules to the whole RDF database and afterwards executes the query only on the subset of it, which only contains allowed RDF triples, is proposed by [Dietzold and Auer, 2006]. TriQL.P [Bizer and Oldakowski, 2004] allows the formulation of trust-policies in order to answer graph-based queries. Those queries describe conditions under which suitable data should be considered trustworthy. However, if all requesters and the graphs they are allowed to access were known in advance, identity-based access control could also be an option to consider for access control.

We note that *a priori* solutions are not sufficient in our scenario presented above, since data access may be additionally restricted depending on externally checked, contextual conditions. Static pre-computing of Named Graphs for each possible combination of environmental factors is infeasible, since the amount of combinations can be arbitrary high; additionally, named graph creation at runtime seems to be infeasible either, since the creation process would excessively slow down the response time. Furthermore, the plug-and-play nature of the Personal Reader Framework where services dynamically change the RDF database itself by adding or removing data from the user profiles would significantly complicate managing such named graphs.

Simple rule-based policies over the RDF database are defined by [Reddivari et al., 2005]: such policies exploit graph patterns in order to identify subgraphs of the database on which actions like *read* and *update* can be executed. Other approaches also exploit RDF Schema entailment [Jain and Farkas, 2006]. However, all these approaches require to instantiate the graph patterns, i.e., to generate one graph for each policy and execute the given query on each graph, hence leading to longer response times.

Finally, many policy languages (e.g., KAOS, Rei, PeerTrust or Protune) allow in general to express access rules on the Semantic Web by means of policies. However, none of them describes how such policies can be integrated in RDF databases.

4.1.6 Discussion

In this section we presented approaches for an application independent user modeling. First application code-independent user modeling components were presented, called User Modeling Shells, like GUMS and BGP-MS, which provided first encapsulated user modeling functionality. These Shells bear the

disadvantage that they are still bound to one specific application and cannot be used simultaneously in a multi-application setting. Therefore, User Modeling Servers like CUMULATE and PersonIs were presented, that allow cross-application user modeling. Due to the required domain-knowledge these systems are intended to be used in a specific application domain and cannot be used in a generic manner. To overcome this issue, we discussed state-of-the-art Semantic Web-based generic user profile formats, like Friend-of-a-Friend and the Generalized User Modeling Ontology. Both techniques add metadata to the user profile information to make it machine understandable and interpretable. We finally presented solutions that supporting the exchange of user profiles as well as privacy protection of confidential user profile information.

We conclude that several promising approaches for generic user modeling do exist but that none of the presented related work covers all aspects of generic user modeling that we consider important, like application-independent systems, utilizing a generic user modeling format that adhere to privacy protection and allow a profile exchange.

In the area of protecting RDF-based user profiles, we could not find any solution that can be applied without modification. However, the work in the area of (access) policies is promising for implementing access control.

4.2 The User Modeling Service

The User Modeling Service (*UMService*) is a centralized service, which is implemented within the Personal Reader Framework. With store, update, and query requests, every Personal Reader service (SynServices, PServices and CService) can access the UMService. For querying the user profile, the UMService offers a simple query language that selects profile statements based on pattern-matching and a generic SERQL[12] endpoint to perform more powerful queries.

To allow services to use their own vocabulary and still be able to exchange information with other services, we defined the *User Modeling Ontology*, which is an extensible high-level ontology defined on top of the GUMO. This ontology allows to define a shared structure of the statements, enabling a common understanding of the content of the statements. We adhere to the Linked Data principle and provide mappings between GUMO and UMO so that knowledge from GUMO can be further used in our UMO.

4.2.1 The User Modeling Ontology

The *User Modeling Ontology* (*UMO* for short) defines a basic structure of the statements that are stored in the User Modeling Service. RDF has been

[12] http://www.openrdf.org/doc/sesame/users/ch06.html

```
<rdf:Description rdf:about="#HobbyStatement">
  <umo:subject rdf:resource="#John"/>
  <umo:predicate rdf:resource="#hasHobby"/>
  <umo:object rdf:resource="#sailing"/>
  <umo:ambit rdf:resource="&umo;hasInterest"/>

  <umo:scope rdf:resource="#importanceInterval"/>
  <umo:scopeValue>important</umo:scopeValue>
  <umo:identityValue>neutral</umo:identityValue>

  <umo:owner rdf:resource="#John"/>
  <umo:creator rdf:resource="#schedulerService"/>
  <umo:method rdf:resource="#questionnaire"/>
  <umo:confidence>100</umo:confidence>

  <umo:start>2008-06-01</umo:start>
  <umo:durability rdf:resource="&umo;month"/>
  <umo:replaces rdf:about="#oldHobbyStatement">
</rdf:Description>
```

Figure 4.7: An example statement expressed in the User Modeling Ontology

chosen as the underlying data model, due to its high flexibility: arbitrary RDF data referring to various ontologies can be stored, and RDF databases which allow efficient storage and access to the data are available. As the base vocabulary for our ontology, we selected Heckmann's *Generalized User Modeling Ontology* (GUMO) described in [Heckmann et al., 2005]. To adhere to the Linked Data principle and to allow the reuse of GUMO-formated user profile data, we defined mappings between UMO and GUMO. A UMO example statement is shown in Figure 4.7.

UMO consists of four segments, which contain the following attributes:

Main Segment provides the attributes *user*, *subject*, *predicate*, *object*, *ambit*, *scope*, *scopeValue*, and *identityElement*.

Explanation Segment contains *creator*, *method*, *evidence*, *confidence*, and *trust*.

Validity Segment consists of *start*, *end*, *durability*, and *retention*.

Administration Segment provides administrative attributes like *notes*, *replaces*, and *deleted*.

4.2.1.1 Main Segment

The Main Segment stores the basic statement about the user. Every statement is addressable by its own URI. *Subject*, *predicate*, and *object* represent

the reified RDF triple. The attributes *subject* and *user* can differ from each other. E.g. to model the fact that John's credit card has the number 123, we allow to create a statement whose owner (*user*) is John and whose *subject* is the credit card. The *predicate* and *object* values can be chosen freely from the application's domain-specific ontology. The *ambit* predicate relates the statement into one of six domain-independent classes of statements about the user. Possible values are:

- *hasActivity* describes statements about activities of the user, e.g. hobbies.
- *hasDone* describes statements about passed activities of the user.
- *hasPreference* describes statements about the preferences of the user.
- *hasInterest* describes statements about the interests of the user.
- *hasKnowledge* describes statements about the knowledge of the user.
- *hasConfiguration* describes statements about configurations of the user, e.g. program settings like Configurable Descriptions (see Section 3.2.4).

The *ambit* allows services to classify their own statements and especially the *predicates* of their domain ontologies into the generic UMO. This enables a basic mapping between statements of different applications and hence an exchange of user profile data across different ontologies: E.g. an application A utilizing the domain ontology OA stores a statement

S = *(u, oa:interest, dbpedia:SemanticWeb)*

with the *ambit hasInterest* in the UMService. If another application B, which utilizes a different domain ontology OB, queries the UMService for knowledge of the user (*hasKnowledge*), the UMService will not return the statement of application A. Hence, application B is aware that no appropriate data is stored in the UMService although neither application B nor the UMService itself can process the ontology OA directly. On the other hand, if application B queries for interests of the user (*hasInterest*) then it will receive statement S. Although B does not understand the full meaning of *(u, oa:interest, dbpedia:SemanticWeb)*, it can, based on the ambit *hasInterest*, still interpret that user u has interest into the object of the statement. By requesting additional information about *dbpedia:SemanticWeb*, utilizing the Link Data principle, application B is able to draw further conclusions about the particular interest of the user.

The predicates *scope*, *scopeValue*, and *identityElement* are used to describe the value of a property: *scope* describes the interval from which values can be selected. Intervals can either be numerical, e.g. from 1-10, or enumerations like *excellent, good, average, bad*. *IdentityElement* contains the neutral element to enable an automatic mapping between different intervals. The *scopeValue*

predicate finally contains the actual value of the statement chosen from the specified interval.

4.2.1.2 Explanation Segment

The Explanation Segment contains the author of the statement (*creator*) and which *method* was used to create the statement (e.g. was it a direct input of the user, or a derived information based on observations). *Evidence* contains the data that lead to the final statement. The evidence is important for a service to preserve its trustfulness. E.g. if a service A bases its assumptions on wrong data from service B, the distrust regarding this statement can be directed against service B instead of the direct creator of the wrong statement.

The *confidence* value contains the certainty of the creating service that the statement is true. In contrast, *trust* holds the percentage of agreement of the user to this statement. The difference between both values, *confidence* and *trust*, can be used to calculate the accuracy of the assumptions about a user drawn by a specific Syn- or PService.

4.2.1.3 Validity Segment

The Validity Segment defines how long – beginning from the *start* point of time – a statement shall be valid. If the validity can be defined precisely, the predicate *end* is used to indicate the end of time. This holds for statements like "John plays tennis from 6-7 pm". The validity of other statements like "John is currently in a good mood" cannot be specified precisely. Therefore, the predicates *durability* and *retention* are used: *durability* contains vague time specifications like *seconds*, *minutes*, *hours*, *years*, etc. To respect the durability of a statement, we use a linear function that decreases the *confidence* value of the statement as it becomes older. The *retention* predicate contains the point of time when a statement shall not be used any longer.

4.2.1.4 Administration Segment

The Administration Segment contains various meta data: *notes* are a free-form text field with arbitrary content, *replaces* refers to an older statement which is replaced by the current statement. Statements which shall be deleted are marked with *deleted* and are not delivered any more from this point of time. The user can decide whether statement marked as deleted should physically be removed or recovered.

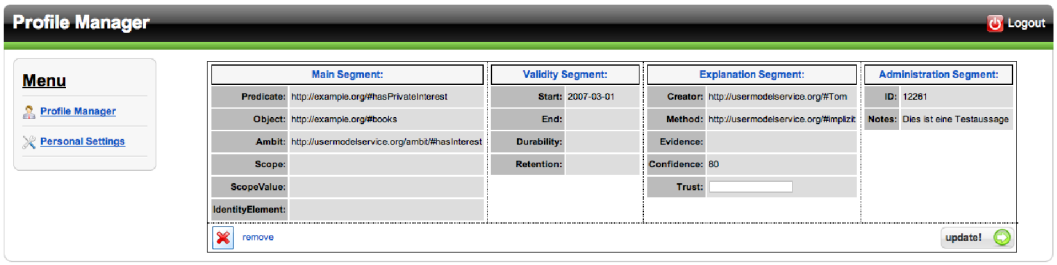


Figure 4.8: User interface to review, modify and delete the statement in the UMService

4.2.1.5 Extending the User Modeling Ontology

Due to the open nature of OWL and RDF, services can use their own vocabulary to store information in the UMService. To use own vocabulary services can use the `subClassOf` and `subPropertyOf` mechanisms to map their own vocabulary concepts to the corresponding concepts in the User Modeling Ontology.

4.2.2 User Interface

A crucial point of the UMService is the user awareness. Every user shall be able to revise her user profile and perform updates or changes when needed. To enable non-technical users to use the UMService, we provide an easy-to-use interface, which does not require any specific knowledge about RDF or policies. This user interface is divided into two parts: a) the data access and modification interface, which allows a user to access and modify the user profile data and b) the access policy editor interface, which allows users to define their access policies in a graphical fashion (see Section 4.2.6).

The data access and modification interface, called Profile Manager (see Figure 4.8) allows users to exploit and adjust their own user profile. They can change the trust value of the statement to express their agreement or disagreement with the statement. If they consider a statement as fully inappropriate, they can also remove complete statements. We decided to not let users modify single properties of the statements as this would on the one hand require a complicated user interface, which needs to describe the possible values and checks if the new statement complies to the ontology constraints. On the other hand it would also require application ontology creators to describe their ontologies in detail, which is hard to be enforced in a distributed architecture.

4.2.3 Reasoning

New information about users can be derived for the user profile by analyzing observations about a user or by combining profile information about a user

stored by different services. Additionally, background knowledge can be used to infer new information about the user. We refer to theses tasks with the term user profile reasoning.

Domain-independence of centralized components is a very important design rationale of the Personal Reader Framework as it ensures that: a) Syn- and PService from various domains can use the Personal Reader infrastructure, b) updates of central components caused by changes in the domain ontology are avoided and c) wrong reasoning caused by faulty services can be handled by access control rules. In the Personal Reader Framework, user profile reasoning is performed directly within the corresponding Syn- or PServices. SynServices can also reuse reasoning functionality offered by PServices while the overall protection of user profile data in still ensured: If a SynService is not authorized by the user to access user profile information, stored by another application, the PService, invoked by the SynService, is also not able to access the required information and hence cannot disclose any confidential user profile data.

4.2.4 Authentication and Single Sign On

User authentication requires the input of a username and password. Once authenticated, it is desirable that a user stays authenticated within the entire Personal Reader Framework and her session moves along with the user across application borders without requiring a re-authentication. It should not matter which application performed the initial authentication. Also, transmitting sensitive data like the user's password across applications is not an option. To resolve this issue, the *Identity Service* provides user session management. Once a user authenticates, a session is created and a user token is returned to the client component identifying that session. The token can safely be passed to other applications within the Personal Reader Framework as it will only have a limited validity (until the session is finished) and does not contain sensible data.

While in our settings, a basic authentication and authorization management was sufficient, the Identity Service can easily adapted to use functionality offered by Shibboleth[13] or OpenID[14].

4.2.5 Enforcing User-Defined Access Control

Users shall be fully aware of which data to share and with whom. The Personal Reader Framework offers an access control layer (see Figure 4.9 that enforces user-defined access control policies on the RDF-based user profile. The highly dynamic nature of the Personal Reader infrastructure complicates the challenge of controlling access to user profile data. Services that may request, add,

[13] http://shibboleth.internet2.edu
[14] http://openid.net

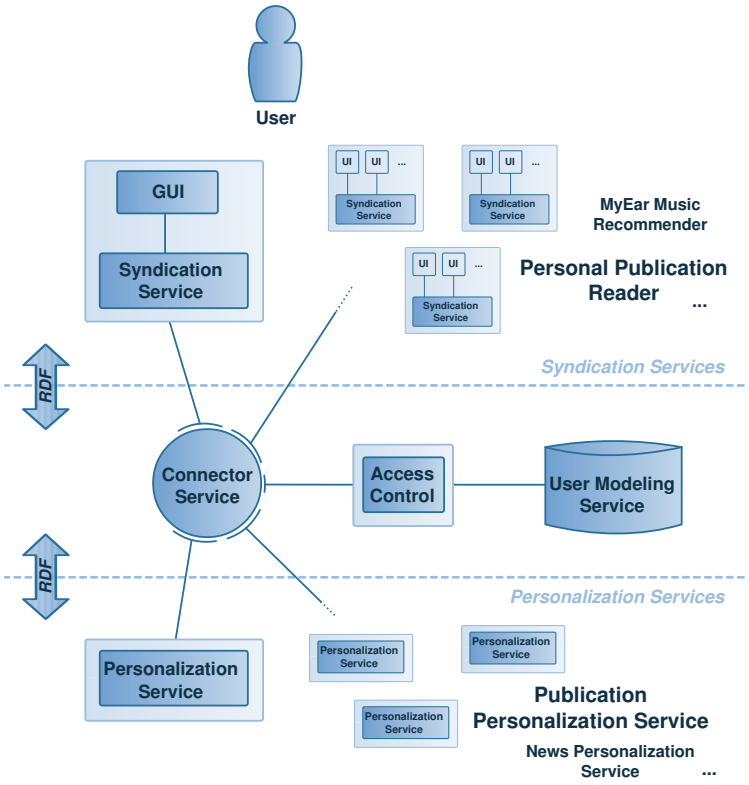


Figure 4.9: Extended Personal Reader Architecture: The access control component

or manipulate user data are not known in advance just like the data (RDF statements) and the vocabulary used to formulate these RDF statements itself. We thus need an infrastructure which allows to define and enforce access policies dynamically.

4.2.5.1 Access Control Layer

The access control layer of the User Modeling Service has to restrict the access to the data stored in the User Modeling Service. Therefore, a user should specify which web services are allowed to access which kind of data in the user profile and in which way. The environment of the access control layer is similar to a personal firewall: whenever an application tries to access a specific port, if an access rule for such application and port has been specified, the specified action (`allow` or `deny`) is performed. Otherwise the firewall asks the user how to behave. The firewall is at no time aware of which applications or ports exist in a system.

Similarly, as the framework allows to plugin new services immediately, the access control layer is not aware of which services will try to access which part of the user profile. Hence, specifying static access rules a priori like in other

access control systems is not applicable.

Our access control layer solves this issue by a deny-by-default behavior. Every Syn- or PService that tries to access an RDF statement is rejected if no existing policy is applicable. The service is informed why it was rejected and will report this to the user. Afterwards, the user can enter the user interface of the access control layer to grant or deny access. The user interface can take the context into account, which contains the statements a service tried to access, and hence supports the user in specifying policies by reducing the choices to the affected statements. By allowing users to specify also general policies we try to avoid that the user is overwhelmed by too much interaction with the access control layer. Keeping user interaction low enhances usability and at the same time avoids that users ignore repeatedly displayed confirm messages.

In the following sections, we focus for the reason on simplicity on granting read access. A similar approach can be used for write access requests.

Policies for Securing Data Securing RDF data is different from securing usual datasets. Because RDF datasets can be considered as graphs we take into account this graph structure in order to provide a definition of "security".

There are many possibilities to secure the data in the user profile, like black- or whitelisting of services for specific RDF statements by means of access control lists. We do not want to mark resources as "accessible" or not in an automatic way, because the user should keep full control on which resources (not) to grant access for. But we also want to relieve the user from marking each resource individually, so we need a more flexible solution. We think that policies provide such a flexible solution. In the following we examine how Protune policies can be applied to RDF statements and graphs.

Scenario Different services need to add, modify, or request sensitive data from the user profile data in an RDF repository within the Personal Reader Framework. Services need to store confidential contact information like email addresses or online e-commerce account information securely, in our example profile (see Figure 4.10). It is crucial that the user a) can inspect and modify the user profile as she wishes and b) has full control about which (kind of) services are allowed to access and retrieve which parts of the data stored in her profile.

We utilize the Protune [Bonatti and Olmedilla, 2005a] [Bonatti and Olmedilla, 2005b] policy language to enforce policies that control access to the single triples, for example to support John to make the phone numbers of his friends publicly available, but to hide statement S_m or maybe even statement S_{m-1}.

S_1: (John, phoneNumber, 123)
S_2: (John, hasFriend, Friend_1)
S_3: (Friend_1, phoneNumber, 234)
...
S_{m-4}: (John, hasFriend, Friend$_n$)
S_{m-3}: (Friend$_n$, phoneNumber, 345)
S_{m-2}: (John, hasFriend, Mary)
S_{m-1}: (Mary, phoneNumber, 456)
S_m : (John, loves, Mary)

Figure 4.10: John's RDF Triple based user profile

Protune Policy Templates for a User Modeling Service We need to specify prerequisites that a service has to fulfill in order to access some resource in a declarative manner. The policy language Protune allows to formulate a broad range of policies like access control policies, privacy policies, reputation-based policies, provisional policies, and business rules.

One of the main differences between Description Logics-based (DL-based) and Logic Programming-based (LP-based) policy languages can be found in the way they deal with negation: Description Logics allow to define negative information explicitly, whereas LP-based systems can deduce negative information by means of the so-called *negation as failure* inference rule. LP-based policy languages like Protune may decide whether the user should only specify allow policies (thereby relying on the negation-as-failure inference rule) or the other way around. The first approach is usually preferred, since wrongly disclosing private information is a more serious issue than not disclosing information that should be publicly available.

In our framework we need both, usual deny policies and deny-by-default policies: If a deny-by-default policy applies, the user is directed to the user interface to specify new policies; if a usual deny policy occurs the user is not informed since she already defined a policy. This feature allows us to implement in a very clean way the algorithm to be executed by the access control component, namely

```
if (a deny policy is defined) deny access
  else
   if (an allow policy is defined) allow access
    else
     deny access and ask the user
```

The access control component checks first whether a deny policy is applicable to the current access request and, if it is the case, denies access. If not, the system checks whether an allow policy is applicable. If this is not the case, access is denied and the user is asked how to proceed.

The following Protune policy applies to John's RDF-based user profile given

in the previous chapter. Its intended meaning is to allow services that belong to the user-defined group *trustedServices* to access the telephone numbers of John's friends, except Mary's number.

```
allow(access(rdfTriple(Y, phoneNumber, X))) :-
  requestingService(S),
  rdfTriple(S, memberOf, '#trustedServices'),
  rdfTriple('#john', hasFriend, Y),
  not Y = '#mary'.
```

Predicate *rdfTriple* retrieves RDF triples from some RDF repository, whereas predicate *requestingService* accesses runtime data in order to retrieve the value of the current requesting service. The rule the policy consists of can be read as a rule of a Logic Program, i.e., *allow(access(...))* is satisfied if predicate *requestingService*, all literals *rdfTriple* and the inequality are satisfied. Predicates which represent an action (i.e., *requestingService* and *rdfTriple*) are supposed to be satisfied if the action they represent has been successfully executed. The policy can therefore be read as follows: access to RDF triple *(Y, phoneNumber, X)* is allowed if the current requesting service (S) belongs to *trustedServices* and X is the phone number of someone who is a friend of John different than *Mary*.

Policy Templates for an RDF based User Profile Since expressive policies become quickly hard to read for non-technical users we defined some general purpose policies in so-called templates.

Policy types can be defined in several ways:

1. One may group targets (in our case RDF statements or parts of them), so that the user is enabled to state, *what* triples should be accessible. Examples for such a group of targeted RDF statements are:
   - Allow access to some specific phone numbers.
   - Allow access only to my own phone number.
   - Allow access only to my friends' phone numbers.
2. Policies may also be grouped according to the requester, so that the user is enabled to state *who* gets access to the triples (i.e. allow access for one service or a specific group/category of services).

Protune policies allow the usage of both kind of policy types to protect specific RDF statements, a specific group of statements or, in general, an arbitrary part of an RDF graph. So, it is possible to

- Specify RDF-predicates anywhere used in the user profile to be secured by a policy.

- Specify RDF-object/RDF-subject types anywhere used in the user profile.
- Specify RDF statements that contain information directly related to the user, like *(John, loves, Mary)*, and not just information indirectly related to the user, like *(Friend$_x$, phoneNumber, xyz)*.
- Specify meta-data predicates like requester or current time.

Our user interface allows to define policies protecting *RDF graph patterns*. When defining a policy the user must instantiate such patterns and adapt them to the given context (see Figure 4.15).

Conflict Handling in the User Interface If there is no policy defined on an RDF statement, an incoming request is denied by default and the accessing service will point the user to the user interface to define a new policy regulating the access to the RDF statement in the future. On the other hand, no user feedback is requested if a deny policy applies to the RDF statement and the current requester. Therefore, the service needs to distinguish between default denial and policy-based denial. Protune by itself uses only positive authorizations in order to avoid conflicts. For this reason we defined a deny predicate on top of Protune to enable also the definition of deny policies. However, if we allow for both positive and negative authorizations, conflicts can arise: This is the case whenever a resource is covered by both an allow and a deny policy. To avoid such situations we designed our user interface (see Section 4.2.6) in order to ensure that no conflict situations will arise or that they are solved in precedence.

When the user defines an allow policy affecting a resource that is already covered by a deny policy, the user interface will show a dialog, notifying the user that there is a conflict. If the user does not want to allow access to the resource, the allow policy will still be defined (since in our framework deny policies have by default higher priority than allow policies), otherwise the deny policy will be modified in order to exclude from its scope the resource. On the other hand when the user defines a deny policy affecting a resource that is already covered by an allow policy, the user interface will show a dialog, notifying the user that there is a conflict. If the user does not want to allow access to the resource, the deny policy will simply be added (for the same reason described above), otherwise a modified version of it will be added, which excludes from its scope the covered resource.

Finally, if the user model changes, new RDF statements can be automatically covered by existing policies. But the user has also the option to apply her policy only to RDF statements existing at policy creation time. As soon as a service adds RDF statements, the user will be asked by the user interface whether her policy should also apply to the new statements.

Policy-Based Query Expansion Our strategy to enforce access policies is to split and pre-evaluate the context-dependent conditions of the policies, i.e., the conditions which are data-dependent. Then, we modify the queries before they are sent to the database by integrating the enforcement of the other data-dependent conditions with the query processing, thereby restricting the queries in such a way that they can only include pre-filtered (and therefore allowed) RDF statements. This way, policies can hold a greater expressiveness and support both metadata and contextual conditions, while relying on the highly optimized query evaluation of the RDF store for the enforcement of metadata constraints. This approach allows to include more complex conditions without dramatically increasing the overhead produced by policy evaluation, and while relying on the underlying RDF store to evaluate RDF Schema capabilities (as discussed in [Jain and Farkas, 2006]).

RDF Queries Definition 1 uses a similar notation as in [Polleres, 2007] to describe the RDF graph. In Definition 2 we use $RDFTerm$ to denote the set $I \cup B \cup L$.

Definition 1 (RDF graph) *Let I, B and L, denote the disjoint infinite sets of IRIs, blank nodes, and literals as usual. Then, an RDF graph is a finite subset of $(I \cup B) \times I \times (I \cup B \cup L)$.*

Definition 2 (Path Expression) *Let I, B, L be as above and Var denote an infinite set of variables. Then, a path expression is a set of triples of the form (s, p, o) such that $s \in I \cup B \cup Var$, $p \in I \cup Var$ and $o \in RDFTerm \cup Var$.*

Definition 3 (Query) *A query is a triple (RF, PE, BE) where*

- *RF is either a set of variables or a path expression (result form)*
- *PE is a path expression (query pattern)*
- *BE is a set of boolean expressions representing a set of constraints in the form of (in)equality and comparison predicates (such us greater than or less than) connected by boolean connectives (AND and OR)*

Intuitively, path expressions are templates, or conjunctive queries formed by triple patterns, for matching RDF graphs which allow variables in any position (see Definition 3).

In the following we will use $vars(e)$ to refer to the set of all unbound variables occurring in a result form, path or boolean expression e. Intuitively, our definition of "query" is meant to model RDF queries having the following structure (see also Section 6.19 in [Aduna, 2005]) [15] [16]:

SELECT RF /CONSTRUCT RF
FROM PE
WHERE BE

In `SELECT` queries RF is a set of variables, modeling a projection, whereas in `CONSTRUCT` queries, RF is a path expression. The special result form RF ='*' denotes either the set of all variables occurring in PE for `SELECT` queries or a copy of PE in `CONSTRUCT` queries, respectively. An example query is provided in Figure 4.11. If no access control policy were defined, this query would return an RDF graph containing all RDF triples matching the graph pattern defined in the FROM block, i.e., the query answer would include identifier and name of a person, her phone number(s) and social connections.

Definition 4 (*disunify* function) *Given a path expression $e = (s, p, o)$ and a set of variable substitutions θ the function $disunify(e, \theta)$ returns the pair (e', BE), where e' is a new triples pattern (s', p, o') and BE is a set of boolean expressions such that*

- $\begin{cases} s' = v_s \text{ and } BE_s = \{v_s = s\} & \text{if } s \notin Var \\ s' = v_s \text{ and } BE_s = \{v_s = Value\} & \text{if } s \in Var, Value/s \in \theta \\ s' = s \text{ and } BE_s = \emptyset & \text{otherwise} \end{cases}$
- $\begin{cases} o' = v_o \text{ and } BE_o = \{v_o = o\} & \text{if } o \notin Var \\ o' = v_o \text{ and } BE_o = \{v_o = Value\} & \text{if } o \in Var, Value/o \in \theta \\ o' = o \text{ and } BE_o = \emptyset & \text{otherwise} \end{cases}$

where v_s and v_o are fresh variables and $BE = BE_s \cup BE_o$.

The disunify function is shown in Definition 4. Intuitively, the variable substitutions for the subject and object of the path expression are extracted and converted into boolean expressions. The purpose of this function is to extract variable substitutions in order to be able to reuse path expressions in the final RDF query, even if they are specified in different policies.

[15] Although our examples will use the syntax of the SeRQL query language, the results also to other languages with similar structure (e.g., SPARQL [Prud'hommeaux and Seaborne, 2008]).

[16] We focus on common *read* operations which all RDF query languages like SeRQL [Broekstra and Kampman, 2004] or SPARQL [Prud'hommeaux and Seaborne, 2008] support. Data manipulations elements, such as *insert* or *delete* operations, are proposed in some extensions such as SPARUL [Seaborne and Manjunath, 2008], but not yet part of any standard.

```
CONSTRUCT * FROM
      {Person} phoneNumber {Phone};
               hasFriend {Friend};
               loves {Name};
```

Figure 4.11: Example RDF query

No.	Policy
pol_1	ALLOW ACCESS TO (#John, hasFriend, X) AND (X, phoneNumber, Y)
pol_3	DENY ACCESS TO (#John, loves, #Mary)

Table 4.1: Example of high-level policies controlling access to RDF statements

Specifying policies on RDF data In order to restrict access to RDF statements a policy language must allow to specify graph patterns (path expressions and boolean expressions), such as one can do in an RDF query. In addition, the ability of checking contextual properties such as the ones of the requester (possibly to be certified by credentials) or current time (in case access is allowed only in a certain period of time) is desirable. Therefore, we consider a policy rule *pol* to be a rule of the form:

ALLOW/DENY ACCESS TO PE IF
CP_1 AND $\ldots CP_l$ AND
PE_1 AND $\ldots PE_m$ AND
BE_1 AND $\ldots BE_n$

where $l, m, n \geq 0$, PE and PE_i $(1 \leq i \leq m)$ are path expressions, CP_j $(1 \leq j \leq l)$ are contextual predicates (i.e., conditions related to time, location, properties of the requester, etc.) and BE_k $(1 \leq k \leq n)$ are boolean expressions. In the following we will use $H(pol)$ (resp. $H_{PE}(pol)$) to refer to the (resp. path expression in the) head of *pol*, and $B(pol)$ to refer to the (possibly empty) body of *pol*.

Notice that our policies are expressed in a high-level syntax: this way we allow them to be mapped to different existing policy languages. On the other hand it is true that the final choice of the policy language will impact the expressiveness and power of the policies which can be specified as well as the set of supported contextual predicates.

Suppose that John specified the policies presented in Table 4.1

1. Everyone can access Johns' friends' phone number(s)

2. Nobody is allowed to access the relation between John and Mary

Policy Evaluation and Query Expansion Our approach analyzes the set of RDF statements to be accessed and restricts it according to the policies in force. Contextual conditions (e.g., time constraints and conditions on properties of the requester) are evaluated by the policy engine, whereas other constraints are added to the given query and enforced during query processing.

Definition 5 (Policy applicability) *Given a path expression e, a set of policies P and a time-dependent state Σ [Bonatti and Olmedilla, 2005a], we say that a policy $pol \in P$ is applicable to e according to Σ iff e and $H_{PE}(pol)$ are unifiable and there exists a variable substitution σ'' such that*

- $\sigma' = mgu(e, H_{PE}(pol))$, *where mgu denotes the most general unifier*
- $\sigma = \sigma'\sigma''$
- $\forall cp \in B(pol), P \cup \Sigma \models \sigma cp$
- $\forall be \in B(pol)$ *such that*
  - $vars(\sigma be) \cap vars(\sigma e) = \emptyset$
  - $\forall pe \in B(pol), vars(\sigma be) \cap vars(\sigma pe) = \emptyset$

  it holds that $P \cup \Sigma \models \sigma be$

and the result of its application to e is a pair (PE, BE) such that for all pe, $disunify(pe, \theta) = (pe', BE')$

- $PE = \{pe' | pe \in B(pol), pe' \neq pe\}$
- $\widetilde{BE} = \{\sigma be | be \in B(pol) \wedge \exists pe : vars(\sigma be) \cap (vars(\sigma pe) \cup vars(\sigma e)) \neq \emptyset\}$
- $BE = BE' \cup \widetilde{BE} \cup \{X = Y | \sigma_i = X/Y \wedge (X \in Const \vee Y \in Const)\}$

In the following we will use $isAppl_{P,\Sigma}(pol, e)$ to refer to the fact that a policy *pol* belonging to a set of policies P is applicable (see Definition 5) to a path expression e according to a state Σ and $appl_{P,\Sigma}(pol, e)$ to refer to the result of such application.

Intuitively, the state Σ determines at each instance the extension of the contextual predicates. Moreover a policy *pol* is applicable to a path expression e if the triple the policy is protecting unifies with e and all contextual predicates and bound boolean expressions (or those not dependent on path expressions in the policy) are satisfied. The result of the application is a pair whose first element is the set of path expressions found in the body of the policy and whose second element is the set of all extracted boolean expressions which have not been evaluated and relate to the path expressions found.

Before we describe the query expansion algorithm, and for sake of clarity, we describe the conditions under which a query does not need to be evaluated since the result is empty.

Intuitively, a query fails if there does not exist any triple to be returned according to both the query and the applicable policies, that is if the query contains at least a path expression for which no matching triples are allowed to be accessed (disallow by default) or for which all matching triples are not allowed to be accessed (explicit disallow).

The pre-filtering algorithm is defined as follows.

Input:
a query $q = (RF, PE, BE)$
a set of policies P
a state Σ
Output:
$PE^+_{new} \equiv$ new optional path expressions (from allow policies)
$PE^-_{new} \equiv$ new optional path expressions (from disallow policies)
$BE^+_{new} \equiv$ conjunction of boolean expressions (from allow policies)
$BE^-_{new} \equiv$ conjunction of boolean expressions (from disallow policies)

$policy\_prefiltering(q, P, \Sigma)$:
$BE^+_{or} \equiv$ disjunction of boolean expressions (from allow policies)
$BE^-_{or} \equiv$ disjunction of boolean expressions (from disallow policies)
$P_{app} \equiv$ a set of applicable policies

01) $PE^+_{new} = PE^-_{new} = \emptyset$
02) $\forall e \in PE$
03) $BE^+_{or} = BE^-_{or} = \emptyset$
// check allow policies
04) $P_{app} = \{pol | pol \in P \wedge H(pol) = allow(\_) \wedge isAppl_{P,\Sigma}(pol, e)\}$
05) if $P_{app} = \emptyset$
// no triples matching e can be accessed
return query failure
06) if $\exists pol \in P_{app} : appl_{P,\Sigma}(pol, e) = (\emptyset, \emptyset)$
// all triples matching e can be accessed

else
07) $\forall pol \in P_{app}$
$appl_{P,\Sigma}(pol, e) = (PE', BE')$
08) if $PE' = \emptyset$
09) $BE^+_{or} \cup = \{\wedge_{be \in BE'} be\}$
10) else if $\exists \theta, \widetilde{PE} \in PE^+_{new} : \theta = mgu(\widetilde{PE}, PE')$
11) $BE^+_{or} \cup = \{\wedge_{be \in BE'} \theta be\}$
else
12) $PE^+_{new} \cup = PE'$

13) $BE^{+}_{or} \cup = \{\wedge_{be \in BE'} be\}$

14) $BE^{+}_{new} \cup = \left\{\vee_{be \in BE^{+}_{or}} be\right\}$

// check disallow policies

15) $P_{app} = \{pol | pol \in P \wedge H(pol) = disallow(\_) \wedge isAppl_{P,\Sigma}(pol, e)\}$

16) if $\exists pol \in P_{app} : appl_{P,\Sigma}(pol, e) = (\emptyset, \emptyset)$

// all triples matching e cannot be accessed

return query failure

17) $\forall pol \in P_{app}$

18) $appl_{P,\Sigma}(pol, e) = (PE', BE')$

19) if $PE' = \emptyset$

20) $BE^{-}_{or} \cup = \{\wedge_{be \in BE'} be\}$

21) else if $\exists \theta, \widetilde{PE} \in PE^{-}_{new} : \theta = mgu(\widetilde{PE}, PE')$

22) $BE^{-}_{or} \cup = \{\wedge_{be \in BE'} \theta be\}$

else

23) $PE^{-}_{new} \cup = PE'$

24) $BE^{-}_{or} \cup = \{\wedge_{be \in BE'} be\}$

25) $BE^{-}_{new} \cup = \left\{\vee_{be \in BE^{-}_{or}} be\right\}$

Detailed description of the algorithm: The algorithm makes no initial static addition to the path expressions contained in the query. This is stated by 1), where the variables containing additional path expression additions for allow and deny policies both start from a clean slate.

Each path expression contained in the query is evaluated in the loop, introduced in 2). Also, each path expression that probably leads to additions in the query comes with an own set of added boolean expressions. Therefore, the variables containing those boolean expressions are cleaned in 3). In 4) it is checked, if any applicable allow policies are existing that contain the path expression in their policy head / condition. If there are no such policies existing, failure is returned immediately in 5), since at least one allow policy is needed to allow for at least one result. If there is an allow policy applicable, but its evaluation leads to no extension of the query it is assumed in 6), that any result of the given query is allowed to be returned without restriction according to that one policy, since no allow policy could disclose more information. The algorithm can then directly start to evaluate the deny policies at 15).

In all other cases, the applicable policies are evaluated one by one starting at 7) and their extensions to the query is filled into the variables PE' (for added path expressions) and BE' (for added boolean expressions belonging to the added path expressions).

Now, several cases have to be distinguished: If it is detected in 8) that there was previously no addition made to the path expressions in PE', then in 9) the obtained boolean expressions are directly added to BE+or the variable, containing all boolean expressions for added path expression from allow policies and connected to the existing BE+or by AND. In 10) the path expressions are checked against all other. If the path expression to be added already exists

among the path expressions targeted for addition, its variables can be extracted and unified with the already existing path expression to reuse variables that would elsewise appear without connection to each other.

The boolean expressions to be added will be targeted for variable substitutions in 11) and the changed boolean expression string will then be added to the BE+or variable, connected with AND afterwards to the other boolean expressions to be added. If there is a path expression to be added and its not already existing in the set of path expressions to be added, in 12), this path expression is appended to the list of new path expressions and so are the new boolean expressions added to the list of boolean expressions belonging to this path expression in 13) connected by AND.

In 13) the loop that started in 7) is finished and in 14) all boolean expressions collected for the path expression are appended to the overall list of boolean expressions to be added to the query, this time connected by OR. After every path expression for allow policies was checked, now the algorithm enters the section where it looks for applicable deny policies. First, the applicable deny policies are collected in 15). If among those policies, there's at least one policy not leading to any extension of the query, the whole query fails in 16).

The reason for this is, that allow and deny parts of the newly created query are combined using a MINUS operator later. This means, that each of those parts needs to have some statements, limiting the returned results to take effect. As for the allow policies a new loop is started for each applicable policy in 17) and the policies are applied to the path expressions contained in the query in 18).

In 19), even if there is no addition for the path expressions after applying a policy, the boolean expressions obtained are added to the overall boolean list for the path expression BE-or in 20). As in 10), also in 21), if there is an additional path expression returned by the policy already contained in the path expressions to be added, the variables of the additions are unified and only the (modified) boolean expressions are added in 22). In any other case, the additional path expressions are added in 23) and so are the new boolean expressions in 24). After this, in 25) the overall boolean expression list for the whole query is extended by the addition of the list of boolean expressions BE-or obtained for the examinated path expression.

Definition 6 (Expanded query) *An expanded query is a pair* $((RF^+, (PE^+, PE_O^+), BE^+), (RF^-, (PE^-, PE_O^-), BE^-))$ *where*

- (RF^+, PE^+, BE^+) *and* (RF^-, PE^-, BE^-) *are (usual) queries*
- PE_O^+ *and* PE_O^- *are path expressions*

Intuitively, our definition of "expanded query" as formalized in Definition 6 is meant to model RDF queries having the following structure:

CONSTRUCT RF^+
FROM PE^+ [PE_O^+]
WHERE BE^+
MINUS
CONSTRUCT RF^-
FROM PE^- [PE_O^-]
WHERE BE^-

where "[" and "]" denote the optional path expression modifier (according to the SeRQL [Aduna, 2005] notation).

The extended query is constructed as follows:

Input:
1) a query $q = (RF, PE, BE)$
$PE_{new}^+ \equiv$ new optional path expressions (from allow policies)
$PE_{new}^- \equiv$ new optional path expressions (from disallow policies)
$BE_{new}^+ \equiv$ conjunction of boolean expressions (from allow policies)
$BE_{new}^- \equiv$ conjunction of boolean expressions (from disallow policies)
Output:
2) an expanded query
$q = (RF^+, (PE^+, PE_O^+), BE^+), (RF^-, (PE^-, PE_O^-), BE^-))$

3) $expandQuery(q, PE_{new}^+, PE_{new}^-, BE_{new}^+, BE_{new}^-)$
4) $RF^+ = RF^- = RF$
5) $PE^+ = PE^- = PE$
6) $PE_O^+ = PE_{new}^+$
7) $PE_O^- = PE_{new}^-$
8) $BE^+ = BE \cup \left\{ \wedge_{be \in BE_{new}^+} be \right\}$
9) $BE^- = BE \cup \left\{ \wedge_{be \in BE_{new}^-} be \right\}$

As shown in the combined CONSTRUCT query above, the resulting query consists of two parts connected by a MINUS operator. The CONSTRUCT queries are each extended by additional path expressions and boolean expressions. The first CONSTRUCT query is the original query enriched by expressions related to allow policies. The second CONSTRUCT query is built to express the limitations represented by deny policies. This algorithm extracts the new path expressions found in the body of the policy rules. It extracts their variable bindings. This is essentially important to reuse them coherently in case they appear in more than one policy rule. However, if the same path

```
  CONSTRUCT {Person} phoneNumber {Phone};
         hasFriend {Friend};
         loves {Name}
  FROM {Person} phoneNumber {Phone};
         hasFriend {Friend};
         loves {Name}
    [ Johns hasFriend {Var2} ]
  WHERE ( Var2 = Person )
MINUS
  CONSTRUCT {Person} loves {Name}
  FROM {Person} loves {Name}
  WHERE ( (Person = #John) AND (Name = #Mary) )
```

Figure 4.12: Expanded RDF query

expression is found in policies being applied to multiple from clauses, then they cannot be reused (since conditions on different expressions are connected conjunctively). After prefiltering each policy, a set of AND boolean expressions are extracted. The set of all boolean expressions from applicable allow policies to one from clause are connected by OR. The set of all boolean expressions applicable to multiple from clauses are connected by AND. From that query we have to remove the triples affected by disallow policies, which are specified in a similar fashion and added to the query using the MINUS operator.

In 4), the set of variables or set of triples used in the original query is the same in both CONSTRUCT queries (RF). This is also the case for the original path expressions (PE) (in 5)). In 6) and 7) the added path expressions are identical to the path expressions additionally built by the core query extension algorithm. 8) and 9) show that the boolean expressions are extended by the additional boolean expressions found by the algorithm.

Example 1 *Figure 4.12 shows the result of applying the above algorithm to the query in Figure 4.11 and the policies in Table 4.1.*

Architecture A key goal of our implementation is to be applicable and reusable for different settings, in which access to RDF data should be controlled. Our approach of re-writing RDF queries is based on three units, which should be adaptable to a particular setting.

RDF Query Language. Today there exist several RDF query languages like SPARQL [Prud'hommeaux and Seaborne, 2008], SeRQL [Aduna, 2005], or RDQL [RDQL, 2005]. None of them has yet prevailed in becoming a de facto standard so that the implementation has to be flexible regarding the RDF query language.

Policy language. As outlined in Section 4.1.5, there are a couple of policy

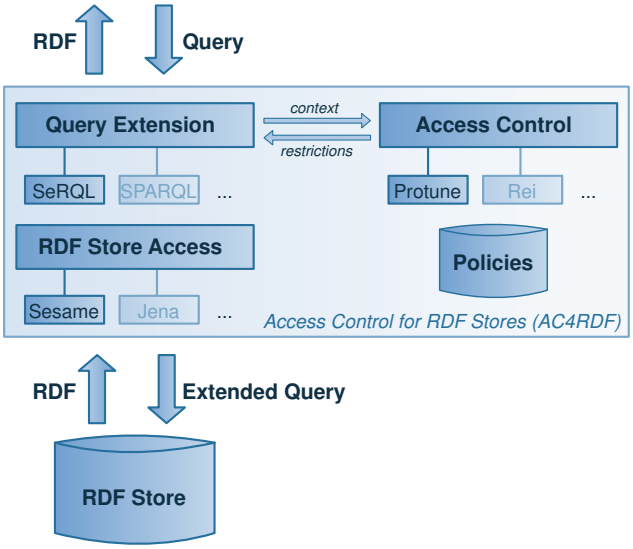


Figure 4.13: Architecture – Access Control for RDF Stores (AC4RDF)

languages and corresponding engines, which can be applied in order to specify and enforce RDF access control policies. Selecting an appropriate policy language should not influence the other components of the implementation.

RDF store. The implementation should further be independent from the way, RDF data is stored, because different stores – like Sesame [17], Kowari [18] or Jena [19] – may be preferable depending on the application scenario.

The generic architecture *Access Control for RDF Stores* (AC4RDF), which we illustrated in Figure 4.13 was designed under consideration of those requirements. It is composed of three main modules, which enable decoupling of the units mentioned above, namely: *Query Extension*, *Policy Engine* and *RDF Store Access*.

Query Extension. The main task of this core module is to rewrite a given query with the support of the policy engine in a way that only allowed RDF statements are accessed and returned. It is in charge of querying the policy engine for each FROM clause of the original query in order and expand it with the extra path expressions and constraints (cf. Section 4.2.5.1). Our initial implementation provides query extension capabilities for the *SeRQL* [Broekstra and Kampman, 2004] query language.

Policy Engine. This module is responsible for the policy evaluation. Input information (*query context*) such as the requester or disclosed credentials

[17] http://www.openrdf.org
[18] http://www.kowari.org
[19] http://jena.sourceforge.net

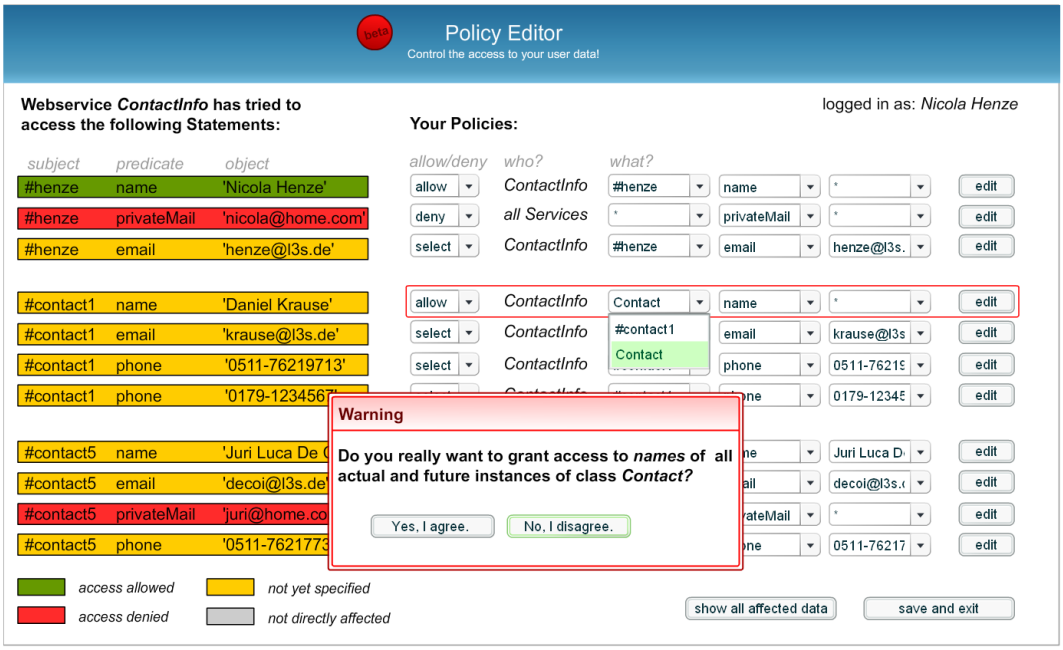


Figure 4.14: Defining Policies - Overview

may be used as well.

RDF Store Access. After extending a query the extended query can be passed to the underlying RDF repository. Since our solution is repository-independent, any store supporting SeRQL, such as *Sesame* [Broekstra et al., 2002] (which we integrated in our actual implementation), can be used. The result set returned contains only allowed statements and can be directly returned to the requester.

The three modules are interdependent (see Figure 4.13). When a query to the RDF store is received by the Query Extension module, the query language used is recognized and the access context of the query is passed to the policy engine. Based on the query, the policy engine can now process the existing policy set and generates additions for the query. Depending on the policies, these additions will be used in the query later to narrow down the result set of RDF triples to an allowed subset. These additions are passed back to the Query Extension module and are added there to the original query. The extended query is then passed to the RDF store access and executed on the underlying RDF repository.

4.2.6 User Interface for Defining Access Policies

The interface that enables users to specify Protune access policies is called *Policy Editor* and operates on top of the access control layer of the User

Modeling Service as outlined in Figure 4.9. If a service attempts to access user data for which no access policies have been defined yet, then the operation of the service fails and the user is forwarded to the Policy Editor. The interface which is shown to the user (see Figure 4.14) is adapted to the context of the failed operation. Such a context is given by the RDF statements which the service needed to access. Thus, the overview is split into a part which outlines these RDF statements, and a part which allows the specification of corresponding access policies. RDF statements are colored according to the policies affecting them (e.g. if a statement is not affected by any policy it may be colored yellow, green statements indicate that at least one service is allowed to access, etc.). Next to such statements the interface additionally shows conflicting policies by marking affected policies and RDF statements.

Warnings make the user aware of critical policies. In Figure 4.14 the user wants to allow the access to "*names*" to all instances of a class "*Contact*". But as the user may not be aware that such a policy would also disclose all future user profile entries containing a name, she is explicitly prompted for validation. If the user disagrees, she will be prompted whether the policy should be refined to cover only those name instances that are currently stored in the user profile.

In general, policies are edited using the interface depicted in Figure 4.15. This interface consists of two main parts which allow to:

1. define policies (top frame), and
2. dynamically show the effects of the policy (bottom frame).

An *expert mode* is available, which allows the user to directly enter Protune policies. Users that do not use the *expert mode* just have to instantiate a template consisting of four steps (see top right in Figure 4.15):

what The main task during creation of access policies is the specification of RDF graph patterns which identify statements that should be accessible or not. The predefined forms for defining these patterns are generated on basis of a partial RDF graph consisting of a certain RDF statement (here: *(#contact1, name, 'Daniel Krause')*) and its relation to the user *(#henze, hasContact, #contact1)*. To clarify this fact the RDF graph is presented to the user on the left hand.
To determine the options within the forms, schema information of domain ontologies is utilized. In the given example the property *name* is part of the statement from which the forms are adapted. As *name* is a subproperty of *contactDetail* both appear within the opened combo box.
By clicking on *add pattern* or *remove* the user is enabled to add/remove RDF statement patterns to/from the overall graph pattern.

allow/deny The user can either allow or deny the access to RDF statements expressly.

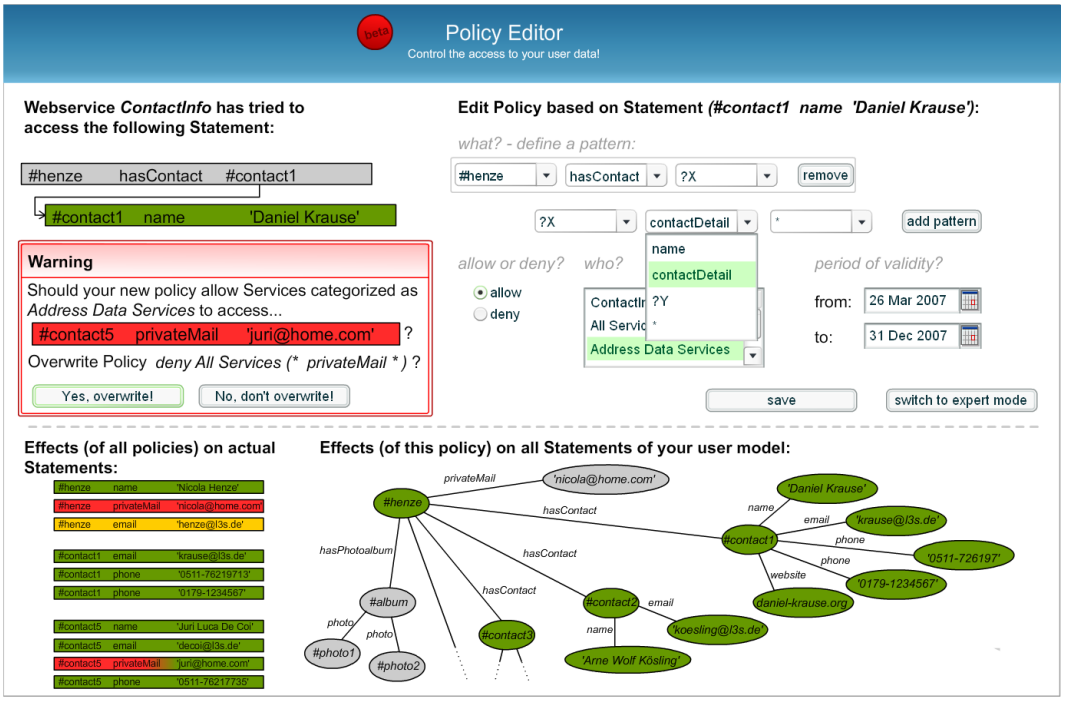


Figure 4.15: Editing a policy in a detailed view

who The policy has to be assigned to some services or category of services. For example to *ContactInfo*, the service trying to access user data, or to a category like *Address Data Services* with which *ContactInfo* is associated.

period of validity This parameter permits the temporal restriction of the policy.

According to Figure 4.15 the resulting Protune policy would be (without *period of validity*):

```
allow(access(rdfTriple(X, contactDetail, _))) :-
          requestingService(S),
          rdfTriple(S, memberOf,
          '#addressDataServices'),
          rdfTriple('#henze', hasContact, X).
```

Thus, *Address Data Services* are allowed to access all statements *(X, contactDetail, Y)* that match the RDF graph pattern *(#henze, hasContact, X), (X, contactDetail, Y)*. This policy overlaps with another policy that denies the access to statements of the form *(X, privateMail, Y)* wherefore a warning is presented to the user. This warning also lists the statements affected by this conflict: As *(#henze, privateMail, 'nicola@home.com')* does not suit, the pattern specified in Figure 4.15 *(#contact5, privateMail, 'juri@home.com')* is

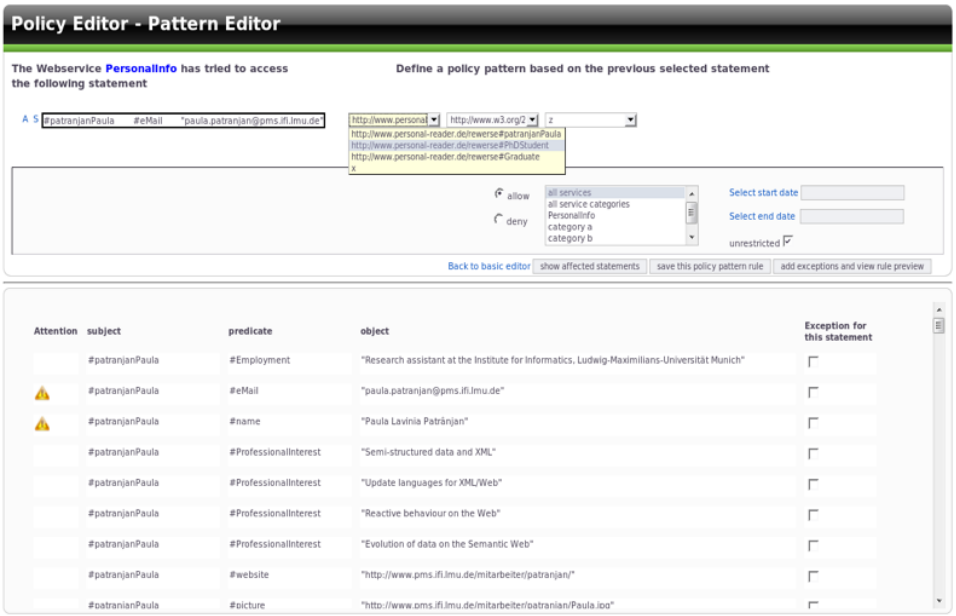


Figure 4.16: Prototype of the Policy Editor User Interface

the only covered statement. By clicking on *"Yes, overwrite!"* the deny policy would be amended with the exception:

not rdfTriple(#contact5, privateMail, 'juri@home.com').

Otherwise, by selecting *"No, do not overwrite!"* both policies would overlap. But as deny policies outrank allow policies (cf. section 4.2.5.1) the affected statement would still be protected.

Next to such warnings the Policy Editor makes the user aware of how specified policies will influence the access to RDF statements. As *name*, *email*, etc. are subproperties of *contactDetail* the above policy permits access to a big part of the user's RDF graph which is consequently shown in green (see bottom of Figure 4.15).

4.2.6.1 Evaluation of the Interface

Regarding usability issues, the main advantages of our user interface are:

- *Easy-to-use* – the users do not need to learn any policy language, policies are created by specifying simple pattern.
- *Scrutability* – users can inspect the effect of the policy immediately as the RDF data is colored either red (access not allowed) or green (access allowed).
- *Awareness of effects* – whenever a change in a policy will disclose data in

the future, it is not visualized in the current graph. Hence, users get a confirmation message to make the aware of the effects of the changes.

We evaluated the user interface for defining access control policies for RDF data by a prototype (see Figure 4.16) that supports the core functionality described in Section 4.2.6. Within our evaluation, students had to accomplish six small tasks with gradient complexity. After we read the tasks to the student in full, we took the time the student needed upon completion of the task. In all of these tasks the students had to create policies with the help of the editor's interface. After the creation, the editor generates Protune policies from the visual creation process.

Our student test group consists of five students, advanced in their study, 3 male and 2 female, coming from computer science and math. None of the students had previous knowledge of policy languages and Protune. None of the students had previously used or tested the Protune policy editor. While some students already had a basic understanding of RDF and some did not we gave a short introduction into RDF in order to make all of them aware of the graph structure and the meaning of RDF triples.

Every student conducted the tasks separately. Therefore, we gave him/her a 10 minute introduction into the Protune editor. The introduction was on a need-to-know basis and contained examples how to accomplish general tasks. We explained further issues in deeper detail, only if asked by the student. An introduction into Protune or formal policies was unnecessary, since the students did not need knowledge about Protune and policies itself in order to work with the editor.

After the introductory phase, the students had to fulfill the six tasks. Every task had to be completed after the previous one, i. e. the students received task two when they finished task one and so on. After the students have finished a task, the Policy editor was reset to an initial state. The starting state of the editor is a scenario state, in which the Policy editor shows the request for a set of RDF triples from a specific service. Those triples are based on an example dataset, we created for this scenario. We measured the time in seconds the student needed from touching the computer mouse until finishing the task.

The tasks in detail are[20]:

1. Allow the access to one specific requested RDF triple for the requesting service.

2. Allow the access to all currently requested RDF triples for the requesting service.

[20]For a users study incorporating students without a basic computer science-related background, the tasks could have been rephrased to omit the term RDF.

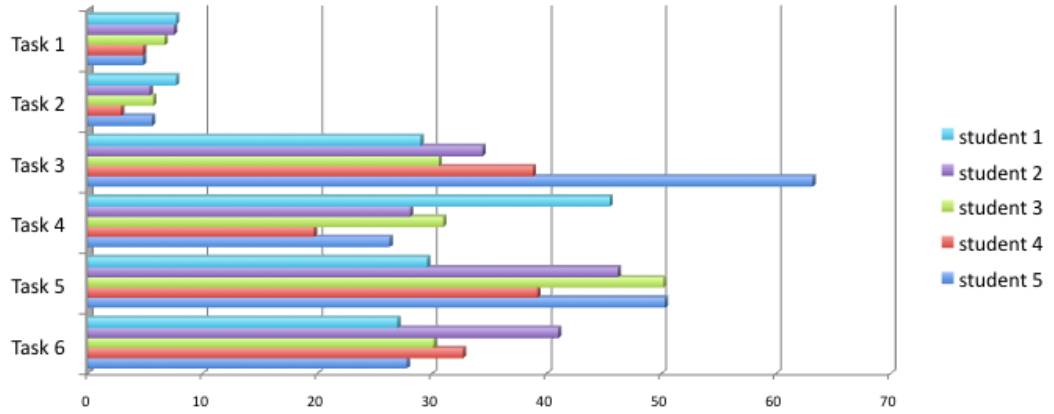


Figure 4.17: Overview of evaluation results (n=5, time measured in seconds)

3. Allow the access to all RDF triples of the user profile database that do contain a specific RDF predicate for a requesting service.

4. Allow the access to all RDF triples of the user profile database that contain a specific RDF subject; limit the access until a certain date for the requesting service.

5. Deny the access to all RDF triples of the user profile database that do contain a specific RDF subject, except of one given RDF triple.

6. Allow the access to all RDF triples of the user profile database with a specific RDF subject for a requesting service, only, if there is existing a specific RDF triple that contains this specific RDF subject as RDF object (utilizing the graph structure of RDF triples).

In Figure 4.17 the time (in seconds) is illustrated that students required in order to finish the task. The time ranged from five seconds for the most simple first task up to 50 seconds in average for the complex exercises. This was much shorter than we expected and presumably shorter than creating Protune policies by hand. Furthermore, it is remarkable that all tasks have been solved by the students. Although, the testing group was not very big, the time the students needed did not show big variance.

However, the students did also make small mistakes in solving the tasks, but corrected themselves after seconds. For example, in task 3, 3 of 5 students confused "all RDF triples" (which means "all of the user profile database") with "all requested triples", which are only the triples shown in our scenario that the service requests.

4.3 Conclusion

The advantage of the presented UMService is that it does not contain domain-specific knowledge and can be used for arbitrary applications. The User Modeling Ontology on the one hand allows the usage of a domain specific vocabulary and on the other hand allows to specify the content of the statements in a generic way. Thus, different applications can exchange user profile data with each other and are able to partially process unknown domain knowledge. All user profile statements are stored in an RDF repository, which makes them easy accessible and searchable from other applications.

We provided two user interfaces that enable users to exploit and maintain their user profiles and that allow users to specify access policies. Both interfaces were designed with the purpose to support non-technical users while using the UMService. Users do not need to have knowledge about RDF data or policies. The conducted user study reveals that our expectation regarding the performance of the interface were exceeded as users without a basic knowledge of access policies were able to specify complex policies in a very short period of time.

We described how to integrate the expressed policies into the UMService in order to provide a fine-grained access control mechanism for RDF-based user profiles. These policies may state conditions on the RDF nature and content of the RDF store as well as other external (e.g., contextual) conditions. The evaluation of the process is divided in order to pre-evaluate conditions of the policy engine not depending on the RDF store and relying on the highly optimized query evaluation of semantic databases for RDF pattern and content constraints.

Chapter 5

Applying Generic User Modeling and Personalization

In this section, we present real-world applications which were implemented using the Personal Reader Framework to prove the advantages of the Personal Reader architecture. We give a detailed description about selected Personal Reader applications, like the *Thread Recommender* (see Section 5.1) that employs the framework to generate recommendations in an E-Learning discussion board. Based on the user profile information, PServices implementing different collaborative recommender algorithms are dynamically selected during runtime. The *Personal Reader Agent* (see Section 5.2) provides a portal to store and maintain a user's invocation configuration of PServices. The *MyEar* application provides a personalized music player that uses the agent to store previous search criteria of a user and to personalize the search results based on the Personal Reader's global user profile.

Section 5.3 gives an overview on applications developed with the Personal Reader Framework. We present a timeline containing developed Personal Reader Framework components as well as Personal Reader applications. A table summarizes all known Personal Reader applications and their use of core components. Access statistics of the Personal Reader website[1] outline the visibility of the conducted research.

5.1 Thread Recommender

Current E-Learning systems focus on supporting the creation and presentation of learning materials. The communication between the learners, which is also an important factor for a successful learning experience [Bodendorf, 2009], is mostly covered by non-personalized tools like chats, wikis and discussion forums in today's E-Learning systems. Discussion forums provide unique com-

[1] http://www.personal-reader.de

munication features, which make them a perfect candidate for providing personalized communication in an E-Learning environment. Some of these properties are:

- *Asynchronous messages:* Learners can decide on their own when they access content, create own content or rate content of other learners [Schwier and Balbar, 2002]. This allows a better planning of the learning behavior than communication tools, which interrupt the learner and require immediate attention, like chats.

- *Feedback to teachers* Learner-learner communication is considered to be the most important interaction type in E-Learning [Soo and Bonk, 1998]. By observing the ongoing discussions among the learners, teachers can get an unbiased feedback about the learning process and are able to detect opaque learning content [Helic et al., 2004].

- *Motivation for the learners:* Discussion forums motivate learners in two ways: first, active discussion forums provide new content nearly every time the user accesses the forum and thus make it more attractive for a user to visit the forum regularly. Second, whenever a learner expressed her own opinion she tends to defend this opinion against others. In this way, users are turned into active participants of ongoing discussions [Thomas, 2002].

This combination of features turns discussion forums into a prominent object of research in the E-Learning area: in [Webb et al., 2004] the authors have shown that participation in discussion forums can improve the learning performance while Bradshaw et al. [Bradshaw and Hinton, 2004] state that discussion forums support collaborative learning.

Another benefit of discussion forums is their tree-like structure. While a discussion forum usually has an overall topic, user can further divide the forum into sub-forums where specific sub-topics can be discussed separately. Below these sub-forums, different discussions can be distinguished by so-called threads. This structure enables learners to browse through discussion topics quickly, and to navigate directly to relevant topics. Thus, users are less overwhelmed by unrelated information as this could happen in mailing lists where users can only decide to opt-in and receive all mails or opt-out and receive non of the mails.

Drawbacks of the structure arise when a) users start a discussion in a wrong thread, b) a topic would fit in multiple threads or c) the forum becomes so big that the structure can not be overlooked by the users immediately. In such cases, learners could possibly miss relevant information or need to spend a high time effort to find relevant information. In these situations keyword-based search, which is implemented in most of the current discussion board

systems, is not an appropriate solution as most users can hardly express their interests by keyword-based queries [Sieg et al., 2004].

A promising approach to match users and relevant threads is to use collaborative filtering techniques. The number of approaches, that are purely based on collaborative filtering, like those used in the Smart E-Learning Framework [Soonthornphisaj et al., 2006], are very limited in the E-Learning domain. The reason for this is that either the explicit ratings of the users are missing [Zaiane, 2002] or that there are not enough users in the system. The E-Learning domain is different from other domains where recommender systems perform well: as most E-Learning systems (like the Comtella-D [Webster and Vassileva, 2006b] system as well) are used to support university courses, the number of users is relatively small in comparison to other systems, like large online stores. Hence, recommender systems need to create recommendations based on a small amount of input data and might fail to generate high-quality recommendations.

Users in online communities, like forums, are not homogeneous [Kelly et al., 2002]. There are some users, who actively contribute new content while other users seldomly or even never publish own content. Those, who never publish own content may or even may not rate the content of other users. With such heterogeneous kind of input data from the users, the question arises whether a single recommendation algorithm can be appropriate to generate recommendations.

Many E-Learning systems cannot use general purpose discussion forums as they would not fit in the E-Learning systems' data structure, programming style or bear legal issues regarding the licence. We expect that most discussion forums in the E-Learning domain will be created from scratch or adapted with specific extensions and hence limit reusability of tightly integrated personalization algorithms. A promising solution is to apply the Personal Reader Framework and provide personalization functionality apart from a specific discussion board. Therefore, we propose a solution, that is loosely coupled and offers recommendation functionality as reusable PServices. Thus, different discussion forums, as well as other E-Learning systems, can benefit from the personalization features, offered by such a solution. Furthermore, by introducing personalization rules, which select the PService to be invoked based on the user profile, we make the offered functionality adjustable while applications and services can still use their existing ontologies and interfaces.

5.1.1 The Comtella-D System

Comtella Discussions (Comtella-D) [Webster and Vassileva, 2006a] is discussion tool, which has been successfully applied in different E-Learning settings, for example to discuss the social, ethical, legal and managerial issues associated with information technology or social navigation-related issues. Moreover, it

Figure 5.1: Screenshot of the Comtella application: a light color represents actively discussed threads, i.e. energy has been assigned recently to posts within the thread

represents a mechanism for motivating participation in interest-based online communities, which engages non-contributing members by modeling and visualizing the asymmetrical relations [Webster and Vassileva, 2006b] formed when reading, evaluating, or commenting other community member's contributions. It was used to support the coursework related to a 4th year undergraduate class on Ethics and IT taught in spring 2006 at the University of Saskatchewan. Access to content is restricted to registered members. Members are relatively anonymous because they are identified just by their aliases. The purpose of using Comtella-D in the class was sharing and discussing information (Internet publications, popular magazine, articles, etc.) related to the course's topics. The students had to share at least one link to an online article related to the weekly topic and summarize the article in a way that stimulates discussion. As part of their coursework, the students also had to discuss two of their colleagues' postings each week. In parallel with the students of the Ethics and IT class (4th year Computer Science students), the Comtella-D system was used in a class on Ethics and Technology offered by the Philosophy department in 2006. These students used the system as an additional resource, recommended by the instructor. The system was not related to their coursework and it was

used entirely voluntary.

Figure 5.2: Screenshot of the Comtella application: Users can increase of decrease the energy level of every post by up and down buttons

In Comtella-D, a forum is an initial theme related to a course topic (usually weekly), defined and created by the instructor. A thread is started when a student contributes a link (URL) to a paper related to the topic of the forum. The first post in a new thread contains the URL and a summary of the paper (usually half a page). Further posts in the thread are added as other students respond to/discuss the first post of the thread. Each post can be commented. A comment is usually a very specific local comment to the post rather than to the entire thread. In Comtella-D comments were used mostly by the marker to give feedback on the quality of arguments raised in the students posts. Figure 5.1 presents a thread view in Comtella-D which can be accessed by registered users to follow the discussion. For each thread, the users can view the name of the forum, a description, the number of posts and the last reply.

In addition, Comtella-D allows students to rate posts (positively and negatively) by adding or removing so-called “energy“ to or from it. A user can rate every post once, if there is free energy in the system available. To make energy distribution more valuable for the users, the system provides a limited number of energy units, depending on the level of activity in the system. Figure 5.2

shows two posts with different colors. The post with the lightest color represents the contribution of the user that received the most positive attention from the other users. In other words, the more the users give energy to the posts the lighter the color of the post gets. In total, ten different energy levels are visualized (see Figure 5.3). The sum of energy that is available within an online community measures the current level of contributions/activity in the community.

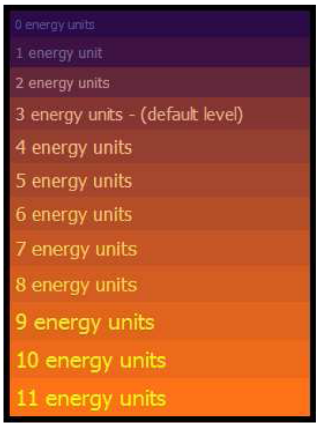


Figure 5.3: Different energy levels in the Comtella application from [Webster and Vassileva, 2006a]

With the use of energy, users who are not willing to contribute actively new content by posting or commenting, can be engaged. As the energy distribution is done by a simple mouse click and shows an immediate effect (the color changes), we assume that some of the previously passive users will at least become active in the sense that they distribute energy.

Moreover, the number of energy units in the system increases every time when a new post is created (2 new units are added), and it decays with time. In this way, the scarcity of energy in the system prevents users from overrating their colleagues' posts, and encourages them to carefully read a post before assigning energy to it. This mechanism is described in [Webster and Vassileva, 2006b].

As every week several new threads are started and popular threads attract many posts, keeping an overview of the discussion is a time consuming task. A student who does not spend the time to read all new posts could easily miss important topics of his/her interest. Hence, a recommender system is needed which points the student to relevant posts.

We determined different behavior styles among the users within the discussion forum:

- *Regularly contributing users:* These users contribute new posts regularly. Often, they discuss their opinion with other users.

- *Casual contributing users:* These users contribute only seldomly.
- *Regularly rating users:* These users do not contribute content by creating posts, but rate posts of other users regularly.
- *Casual rating users:* These users do not contribute content by creating posts, and rate posts of other users only seldomly.
- *Passive users:* These users never contribute own posts nor do they rate posts of other users.

These different user types[2] were considered when the Comtella-D System was designed to generate recommendation. Using a rule-based personalization framework as described in the following section, we can utilize collaborative recommender services to take different user groups into account.

5.1.2 Personalized Discussion Board Architecture

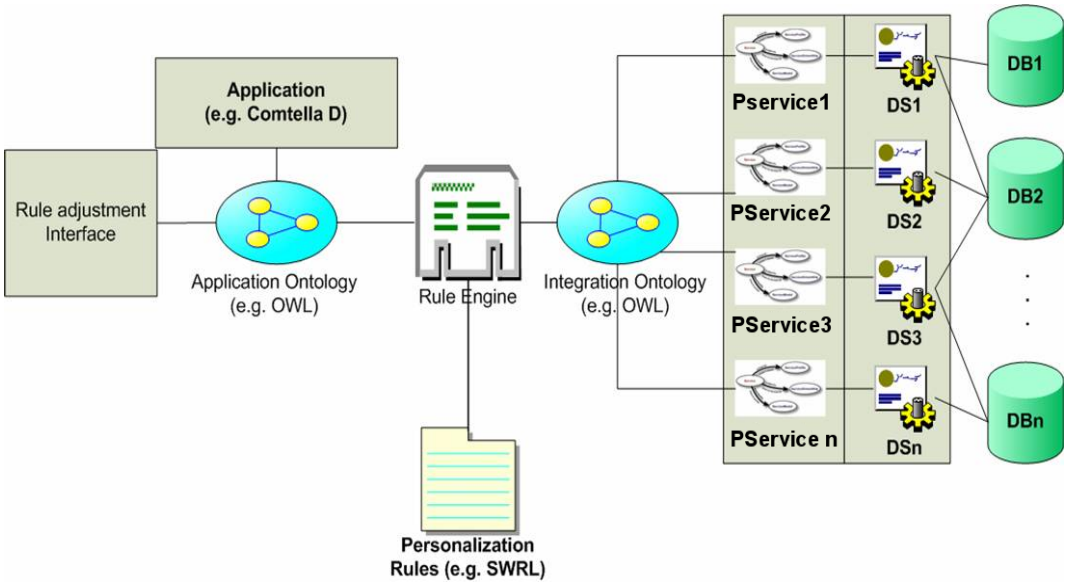


Figure 5.4: Architecture of the System – Personalization Rules map requests from the application, expressed in the Application Ontology vocabulary to the Data Source Services and their Integration Ontology

We decouple personalization algorithms, data sources, and pre- and post-processing from each other by applying the PService/SynService structure from the Personal Reader Framework. To describe the selection of the invoked PService, we allow the use of personalization rules. Furthermore, rules have be commonly used in the E-Learning environment [Dolog et al., 2004,

[2]Users who contributed posts regularly as well as rated posts by other user, will also be counted to the group of regularly contributing users. For other combinations this holds respectively.

Odeh and Ketaneh, 2007] so that E-Learning designers are used to them and are able to extend existing rules. In this architecture, rules have three main purposes that enable a flexible coupling of applications and services:

1. Rules define a clear syntactic interface by receiving requests from applications and transforming them into requests that are submitted to the PServices.
2. Rules map between applications' and PServices' ontologies and hence ensure integration on the ontology level by maintaining appropriate mappings.
3. Rules use PServices as bricks for offering complex functionality. Hence, for adjusting the functionality, it is mostly sufficient to modify or adjust the rules while there is no need to change the services.

Figure 5.4 shows the architecture of the rule-based recommender system with is based on the Personal Reader Framework as presented in Section 3.2. A description of the components of the architecture is given below.

- *DB*: DB represents databases that contain information to be used for personalization. The databases are independent from each other but can be combined by data sources if it is considered as necessary. Examples of these databases are Comtella access logs, forum posts or data provided on the Web.
- *DS*: each data source (DS) represents an encapsulated personalization algorithm. In other words, these data sources are interfaced by PServices. As a consequence of following the Personal Reader Framework, each function is separated into a distinct PServices, so that functionality can be combined and reused in a flexible manner. The development of new DS services is convenient as the Personal Reader Framework reduces the amount of code that has to be written by the programmer.
- *PServices*: PServices provide interfaces to different recommender algorithms and enrich the provided functionality of DS services by machine-readable OWL-S-based descriptions of the functionality.
- *Integration Ontology*: this ontology contains information about the users and personalization algorithms to be used by the system. For this reason, matchmaking algorithms [Klusch et al., 2006] [OWL-S/UDDIM, 2005] [Calado et al., 2009] use this ontology to discover, compose, and invoke the PServices that are used according to the user specification in the rule-based recommender interface. In addition, this ontology can be extended by the developers without causing any problems to the PServices, which have been implemented before extending the ontology. The class hierarchy

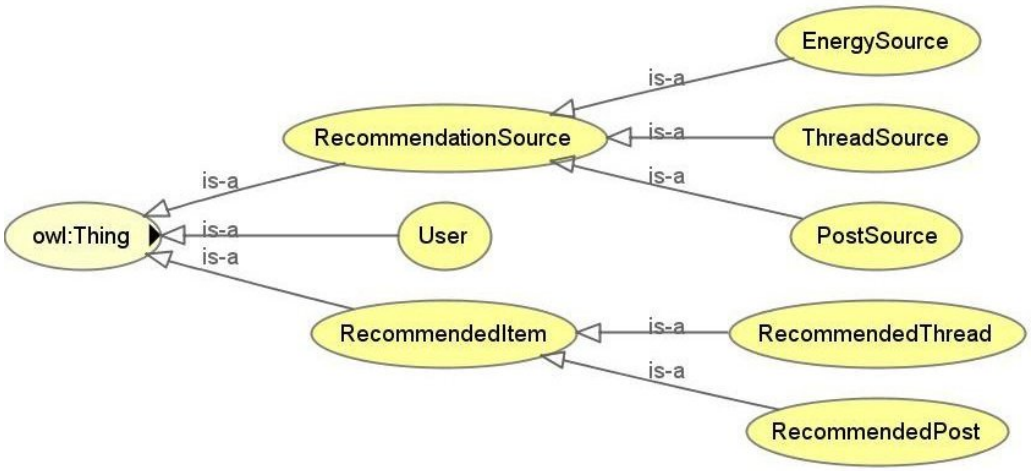


Figure 5.5: The Integration Ontology contains concepts to describe the functionality of the Data Sources. It must be fine-grained to distinguish different recommendation services from each other.

of this ontology is presented in Figure 5.5. The ontology describes three main concepts:

1. *RecommendedItem*: it represents the kind of item considered in the recommendation. In other words, based on the ontology the algorithms can recommend *Posts* or *Threads* in a forum discussion.
2. *User*: description about the users that receive the recommendation of the algorithms.
3. *RecommendationSource*: this concept defines the kind of source used in the recommendation. For example, the algorithms can take into account the post, threads, or even the *energy* (rating) of a discussion provided by a user (cf. Section 5.1.1).

- *Application*: it represents applications that can be used by the recommender architecture. In this thesis, we used Comtella-D as application.
- *Rule-adjustment Interface*: this interface is used to specify personalization rules according to the application used.
- *Application Ontology*: this ontology has the description about the configuration of the recommendation and the users. The hierarchy of the concepts of this ontology is described in Figure 5.6.

We map applications' and services' ontologies to each other to semantically combine the application with recommender PServices and to enable every component to use its own vocabulary. In the example of the Comtella-D system, the ontology is comparatively small so that a mapping was defined by hand[3].

[3]For larger mappings and the semi-automatic creation of mappings, we recommend to use the SILK

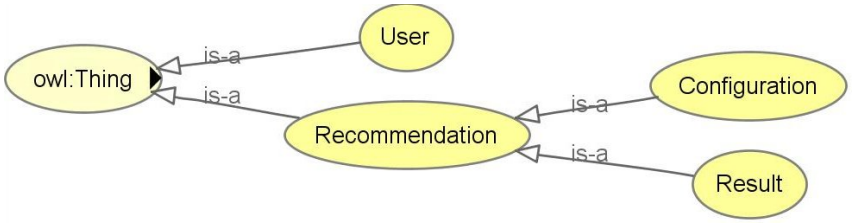


Figure 5.6: The Application Ontology contains concepts that are needed to request recommendations for the Data Sources.

5.1.3 Benefits of Using a Personalization Framework

Utilizing a personalization framework, like the Personal Reader Framework, to implement the Comtella-D Thread Recommender offered several advantages:

- *Reduce development time:* We reuse the existing recommender PServices from the Personal Reader Framework. No new recommender algorithm needed to be reimplemented. PServices were created independently from the data source. The only adjustment, which was needed, was to specify how to access the data base containing the user-thread-post relationship of the Comtella-D system. That was passed as a parameter containing an SQL query.

- *Simple exchange of recommender strategy:* For the evaluation of the effectiveness of the recommender strategies we ran several experiments. In these experiments it was necessary to replace the recommender strategies often. Due to the fact that every recommender strategy was provided by a separate Web Service, we just needed to change a single variable, namely the service URI.

- *Simple extension of experiments:* Whenever new PServices are developed within the Personal Reader, all applications can use the offered functionality. For experiments this means that it is easy to compare the performance of new algorithms with existing ones as it normally needs just a change of a parameter.

- *Future improvement of recommender strategies:* The Personal Reader provides the Personalized Matchmaker (see Section 3.3.1.2), which discovers PServices during runtime based on user feedback. Thus, even if programmers do not update their Personal Reader application, but use the matchmaker, they can immediately benefit from newly available PServices. For the following evaluation we did not use the Personalized Matchmaker but decided to use a static selection rule because user feedback required by the Personal Reader matchmaker was not available.

framework [Bizer et al., 2009].

5.1.4 Adjusting the Selection of Personalization Functionality

While the Personal Reader Framework offers the advantage of existing, configurable and reusable PServices, it is still in the responsibility of the application's developer to integrate the PServices into her own application. From several PServices with similar functionality, the best (in respect to context, available input data, etc.) service needs to be chosen. While this can be done by using our personalized matchmaker (see Section 3.3.1), an alternative is to specify the selection of the best service as a rule before the application is launched, utilizing test data.

In this section, we show such an optimization based on Comtella-D. We used a database snapshot from Comtella-D to adjust the personalization rule. This dataset was created while Comtella-D was used for a 13 week course on Ethics and IT (see Section 5.1.4.1) given in 2006 at the University of Saskatchewan. From the snapshot, we identified representative users in Section 5.1.4.2 and extracted relevant research questions to determine the selection of the best recommender PService.

5.1.4.1 Data Set

Based on the features of Comtella-D, there are different possibilities about which input data can be used by a collaborative recommender:

a) *recommendations based on explicit feedback:* we consider energy assignment done by the users as explicit feedback as users explicitly rate whether they like (add energy) or dislike (remove energy) the content. Energy assignments require free energy in the application, which is generated when user activity contribute new content to the application, and are therefore considered valuable.

b) *recommendations based on implicit feedback:* we consider the posting behavior of a user as implicit feedback, based on the assumption that a user is interested in a specific thread when she contributed a post.

For the evaluation we took a snapshot of the Comtella-D system of the *Ethics and Computer Science* course 2006. Overall, there were 110 registered users. From these users only 36 contributed actively by posting a least one message in the discussion forum. Users rated other users 183 times and posted 756 messages in 173 threads over a time period of approximately 3 months. In these three months, the lass dealt every week with a new topic.

5.1.4.2 Scenario

Assume three users A, B and C: A is a very active user, she regularly creates new posts and rates posts of other users as well. B is a user who was active

some weeks ago but did not use the system afterwards, and now requests recommendations from the system. C has used the system rarely and has contributed only two posts.

To define a personalization rule which recommends threads, we need to find a rule that takes all the different behavior patterns into account. We need to know for user A if all information that we have in our system shall be taken into account when recommendations are generated. Can we still use the possibly outdated information from user B and is C's contribution sufficient to generate recommendations?

From this scenario, we derived the following four research questions, that we will answer in this section:

1. How much training data is required to generate precise recommendations (see Section 5.1.4.3)?
2. What kind of input data (explicit or implicit) gives the best quality to recommend threads (see Section 5.1.4.4)?
3. Does the behavior of users in the discussion forum change over time (see Section 5.1.4.5)?
4. Are active users, i.e. users who have posted frequently and hence are more experienced, more reliable as source for recommendations (see Section 5.1.4.6)?

In particular, questions 1, 3 and 4 are of special interest within an E-Learning tool. E-Learning environments, like Comtella-D, are often used as a supplement for a given university course and the number of participants is small compared to other domains, where collaborative filtering techniques are used. Hence, the available amount of input data is very limited. Learners increase their knowledge level during the semester quickly. We assume that learners will also change their opinion when learning new information. Thus, old opinions and interests might be used to predict current interests. Regarding question 4, we search for domain experts and assume that these experts can be found among the most active learners.

For all of the following measurements, we used the recommender library RenkGround[4], which implemented the collaborative recommender algorithm presented in Ringo [Shardanand and Maes, 1995] and GroupLens [Herlocker et al., 1999].

5.1.4.3 Required Amount of Training Data

To determine how much training data is required to generate precise recommendations (first question) we divided our data set into weeks corresponding

[4]http://www.l3s.de/~diederich/SW/renkground-2006-09-07-1030.zip

	T1	T2	T3	T4	T5	T6	T7	T8
Training User 1	X		X	X		X		
Training User 2		X	X		X		X	
Test User	X			X		**X**		

Figure 5.7: Division of the data set into training data (week 1, containing threads T1-T4) and test data (week 2, containing threads T5-T8)

to the different topics of the lectures. Afterwards, we iterated over the weeks, selecting every week x as training set. Then, a test user was selected for whom we tried to forecast the thread in that this user will create a post in week $x+1$.

For example in Figure 5.7, week 1 (containing threads T1-T4) is used as training set to find the neighborhood of similar users for the test user. Afterwards, all posts from week 2 (containing threads T5-T8), which is considered as test data, are removed from the test user (bold cross). Finally, post recommendations for week 2 are generated from the posting behavior of the similar users in week 2. A hit is achieved if the recommendations contain the original thread (bold cross) of the test user.

To ensure that users have contributed enough input data to generate appropriate recommendations we classified the users into different classes. These classes contain sets of users who have posted at least y posts in different threads in the training period and at least one post in the test time.

To compare our results we used a non-personalized baseline algorithm. We recommend the top-k threads, based on the number of posts in the test week. This baseline algorithm seems fair as the overall data set is comparatively small and top-k lists can thus contain good recommendations for the users.

Our research hypothesis is that the more data from a user is available in the training set, the more precise the recommendation for the test set are.

The precision-recall distribution is build by iterating over all users in the class and calculating the top-k recommendations for these users. k is chosen from one to the number of all posts. For every k, the precision and recall is calculated as the average mean of all precision and recall values of all users in the class. Therefore, the recommendation system is invoked as follows: first, the posts generated in the training set are passed to the recommender system to determine the similarity between the users. Afterwards, the recommendations are calculated by passing all posts to the recommender system which were created in the test set.

Figure 5.8 displays the precision-recall distribution for the non-personalized baseline algorithm and the personalized recommendations based on users who have contributed at least 2, 3, 4, or 5 posts in the training set. While for $k <= 3$ the classes 3 to 5 perform better than class 2, class 2 performs better for $k > 4$. However, none of the different classes results in significantly better

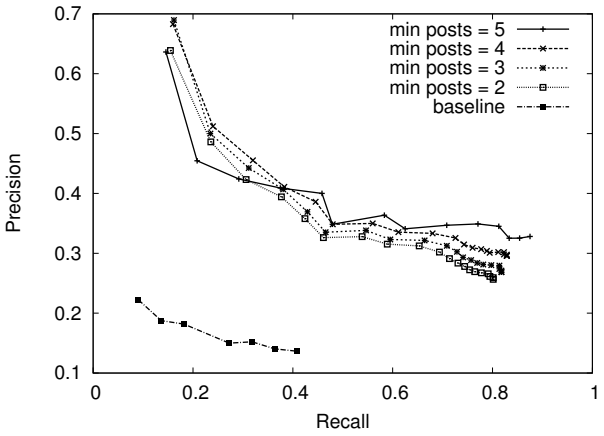


Figure 5.8: The precision-recall diagram based on implicit user feedback for users who have posted at least 2, 3, 4, or 5 times in the training set week.

results than the other classes. Furthermore, all approaches are able to retrieve not more than 80% of the threads the users have contributed to. This can be explained by the characteristics of the recommendation process: when a thread is recommended, a user who is similar to the current user must have contributed to this thread. Hence, threads which are discussed by a limited number of users are recommended rarely. This issue is known as *new item* problem in collaborative recommender systems [Burke, 2002].

Overall, the results imply that a) the non-personalized baseline algorithm is outperformed by the personalized algorithm and b) two posts in one week are sufficient to generate precise personalized recommendations while more posts do not improve the results significantly.

5.1.4.4 Implicit vs. Explicit User Feedback

In the second step we tried to deduce what kind of input data (explicit or implicit) gains the best quality regarding the recommendation of threads (second question). By explicit data we mean user ratings expressed by energy assignments[5] whereas implicit data is based on the posting behavior of a user. Analog to the classes defined in the previous section, we define classes of explicit user feedback. These classes contain users who have contributed at least x ratings (added or removed energy points to posts from other users) in the training set week and have contributed at least one rating in the test set week.

To recommend posts by using user ratings we modified the similarity function of the recommender system. Instead of comparing the similarity of user vectors containing threads a user has posted in, we use vectors containing the

[5]users can express that they like a post by adding energy to it or that the dislike a post by removing energy from it

energy distribution. Two users are considered as similar when they gave energy to the same post, hence expressing interest in the same post. We did not take into account if users added or removed energy as we interpreted every form of energy assignment as interest in a post. The recommender algorithm itself was not modified.

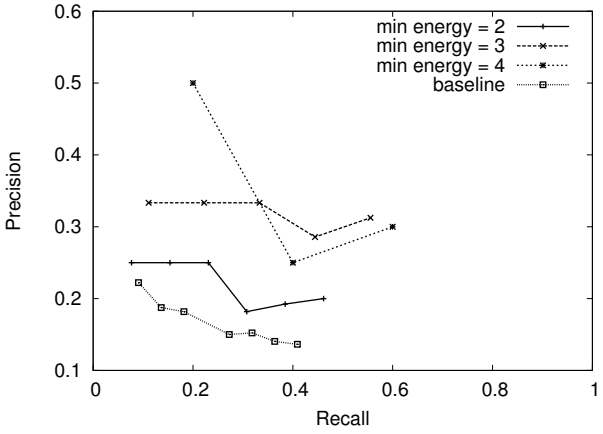


Figure 5.9: The precision-recall diagram based on explicit user feedback for users who have rated at least 2, 3, or 4 posts of other users in the training set week.

Figure 5.9 gives an overview of the precision-recall ratio of recommendations based on explicit feedback for the classes of users having rated at least 2 or 3 other users in the training set period. The class with 5 energy assignments was omitted as it did not contain enough users to deliver reliable results. The graph outlines that – like in the previous section – a comparable small amount of input data, namely two energy assignments, are sufficient to create appropriate recommendations and that increasing the amount of input data does not increase the precision or recall of the recommendations significantly. Compared to the precision-recall distribution generated by implicit user feedback, the quality of the results generated by explicit feedback, in respect of both, precision and recall, are lower.

We also considered that the smaller number of ratings in comparison to posts (in the dataset we had 183 ratings and 756 posts) might be a reason for the weaker performance. To verify this assumption we repeated the experiments by modified classes: Instead of setting only a minimum amount of feedback, we also set a maximum amount of feedback equal to the minimum amount (e.g. a class now contains those users who contributed exactly 3 posts or 3 ratings). This resulted for both, implicit and explicit feedback, in lower precision-recall values, but did not change the performance gap between implicit and explicit feedback.

To improve the overall performance, we tried to use more input data and joined explicit and implicit feedback. We used the average mean to combine

the weighted result sets of the recommendations based on explicit feedback and implicit feedback. We observed that the more we increased the weight of the explicit user feedback, the worse our recommender system performed.

Our conclusion for the given setting is that explicit feedback (energy assignments) performs worse than implicit feedback (posting behavior) and cannot be used to improve recommendation based on implicit feedback. However, if no implicit feedback is given for a specific user, explicit feedback performs better than the non-personalized baseline algorithm. Hence, explicit feedback based recommendations can be used as a fall-back if no implicit feedback is available.

Based on these results we used implicit user feedback as source for the recommendations applied in the following evaluations.

5.1.4.5 User Behavior

The Comtella-D system was strongly coupled with the timeline of the lectures. This means that the users discussed every week a new topic. The overlap between the topics was quite low so that it was not possible use the previous attitude or behavior of a student towards a specific topic to predict the future behavior. We assume that the behavior of users changes over time and over different topics, which means that the more weeks ahead recommendations are created, the more imprecise they become. Furthermore, as topics discussed in a given week should still be somewhat fresher in the memory of the students, we assume that the forecast for the next week would be more precise than forecasts for two or more weeks ahead.

To verify our assumptions, we iterated over all weeks and used them as training data. We calculate the recommendations for n weeks ahead, where $n = 1, 2, .., 7$ and compared them with the test data. Afterwards, we created the precision-recall diagram displayed in Figure 5.10.

The figure displays a result which does not comply with our assumptions: the one week ahead precision-recall values for small top-k result sets are worse than all other forecasts. Furthermore, the forecasts for more weeks ahead do not comply to any rule or trend. This means that the behavior of the users indeed change over time and topic (third question), but that the change of behavior is not predictable. External factors, like students' deadlines for assignments or projects, might also have lead to the observed unpredictable behavior. We have to remark that our dataset covers only three months of data, which cannot normalize peaks from external factors. Thus, we have only reported about the short time behavior of users. We assume that a long-lasting trend, like a learner's general aptitude (how active, diligent she is), could be predicted more precisely.

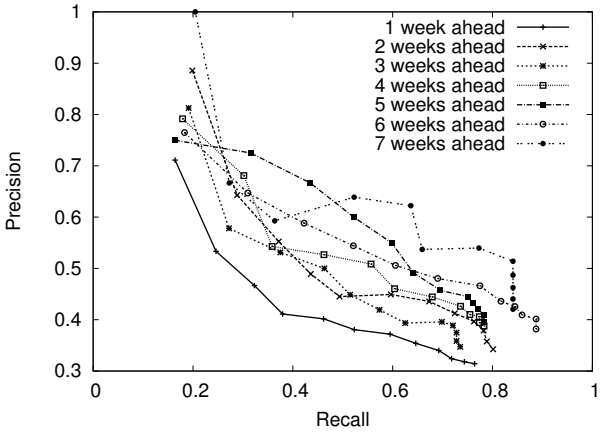


Figure 5.10: The precision-recall diagram shows prediction quality for $x+1, x+2, ..., x+7$ weeks ahead generated based on the training data of week x.

5.1.4.6 Effect of Observation Timeframe

In the previous section we have shown that the user behavior changes over the weeks making a constantly high forecast for several weeks ahead impossible. To lower this effect, we increase the input data timeframe by aggregating several weeks as training set and creating recommendations for one week ahead. We expect that aggregating several weeks of input data normalizes the behavior of a user on the one hand and increases the amount of input data one the other. Both effects should result in an increased quality of the recommendations. Figure 5.11 displays the measurement aggregating one to five weeks of input data and calculating the precision and recall of the recommendations for the following week.

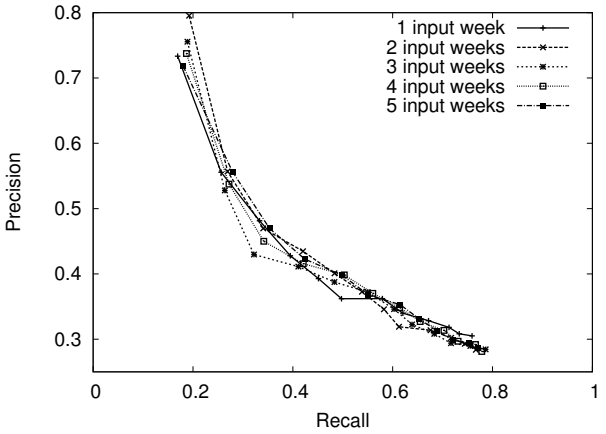


Figure 5.11: The precision-recall diagram shows prediction quality of one week ahead recommendations based on the previous 1 to 5 weeks of training data.

All reviewed input periods deliver similar results. Our expectation that more input weeks could improve the result could not be proven. This also underlines our previous observation that the changes of quality regarding precision and recall seem to follow no rule or trend. Hence, we can also answer our fourth question: active users, i.e. users who have posted frequently in Comtella-D, are not more reliable as source for recommendations than users who posted less frequently.

5.1.4.7 Personalization Rule

The results show that a small amount of input data (two posts or two energy assignments) and a small number of users – which is a typical scenario within an E-Learning application – is enough to generate precise recommendations. As we compared our algorithms against a very reasonable and often applied baseline, namely a top-k list of most popular topics, we conclude that collaborative recommender algorithms are appropriate to be used in the E-Learning domain.

Furthermore, we have shown that collaborative filtering provided by the RankGround library can be successfully applied in this E-Learning setting. More precisely, implicit user feedback, based on the posting behavior of users results in better recommendations than explicit user feedback given by the energy assignment of the users while the user behavior tends to follow no predictable trends over the weeks. A further experiment has shown that more input data does not always generate better recommendations. Thus, a flexible method to combine different recommender algorithms based in the available input data is required.

According to these observations, a personalization rule to select the optional recommender algorithm to recommend threads in the Comtella-D system is the following:

```
if
     at least two posts of the user exist
then
     create recommendation based on implicit user feedback
else if
        at least two energy assignments of the user exist
    then
        create recommendation based on explicit user feedback
    else
        use the non-personalized baseline algorithm
```

By enhancing already existing rules or adding this personalization rule to

the E-Learning environment, E-Learning systems can easily recommend relevant information/discussions to a learner.

In systems, where the number of user groups, personalization algorithms, or different kind of input data become too large to create personalization rules by hand, data mining tools like Weka[6] can be used to automatically identify the most appropriate strategies to personalize content according to a user's input data.

5.1.5 Conclusion

In this section, we outlined the advantages of discussion boards for E-Learning and specified the problems of providing personalization in such a board. We proposed an discussion-board independent architecture for flexible integration of personalization functionality in E-Learning based discussion boards utilizing the Personal Reader Framework: different generic recommender algorithms are provided as PServices and are selected during runtime based on the available user profile information.

To optimize the selection process, we used a dataset from the Comtella-D system of the University of Saskatchewan, which provides different kind of user feedback. In the evaluation, we have shown that a small amount of input data is sufficient to generate appropriate personalized recommendations and that some kind of input data are more useful for generating recommendations than others. We conclude that a careful selection of input data and corresponding personalization algorithm results in better results than using all available information of a specific user. As a result of this evaluation we provide a personalization rule, which selects the best personalization algorithm based on the available user profile information.

Using the Personal Reader Framework for providing personalization functionality for Comtella-D offered the following benefits: a) a reduced development time as some recommender algorithms could be reduced, b) a simplified exchange of recommender algorithms as they were encapsulated into PServices and c) due to the plug-and-play characteristics of the Personal Reader Framework, new recommender algorithms can easily be incorporated at any later point in time.

5.2 The Personal Reader Agent

Personalized Semantic Web applications, that provide a graphical user interface, have to cope with three user-centered issues:

- allowing users to specify their needs (customization)

[6] http://www.cs.waikato.ac.nz/ml/weka/

- optimizing result evaluation according to explicit and implicit needs of the user (adaptation; explicit needs are directly obtained during a particular interaction, implicit needs are derived from previous interactions and are interpreted and consolidated by aid of a user modeling component)
- presenting their results in a way that a) user-side applications can visualize the results and b) transparency and controllability of the result-determining processes and the adaptation steps are guaranteed.

The Personal Reader Agent is a portal to access Personal Reader applications and to personalize the invocation of PServices accordingly.

5.2.1 Usage of the Agent

First, the user accesses the Personal Reader Framework by visiting a portal website provided by the Agent. The user selects an Personal Reader application (the SynService) that shall be invoked. Then the user profile information is used to invoke the personalized matchmaker (see Section 3.3.1.2). The goal is to discover PServices that can be invoked and adapt best to the provided user data from both, the actual user's request and user profile data. N.B.: not all PServices need to receive the same user profile data, as some of them might be more trusted than others. The necessary negotiation based on the user-defined policies in the UMService and credentials of the PServices have to be executed beforehand.

Afterwards, all matching PServices will be displayed to the user who can choose which Web service(s) shall be invoked. With this selection step, it is ensured that only Web services are invoked that a user trusts, and negotiations about user profile credentials can be controlled by the user if necessary. Afterwards, PServices' customization parameters – if PServices offer them – are displayed to the user who can adjust them according to her requirements.

Every selected and customized PService is executed and returns its content, plus optionally one or several visualization templates. The visualization templates enable the SynServices to reach a high usability by providing domain–optimized visualization. The user can interact with the applications by clicking on links or completing forms in the generated user interface. As these interactions are sent back to the SynService it can adapt it's content more precise to the user's requirements and deliver more personalized content, for example displaying a higher level of detail of the relevant informations.

5.2.2 Visualization and Interface

The Agent provides the interface for searching and configuring SynServices, as described above. After PServices were selected, configured and invoked,

the SynService displays the results of all PServices which have been invoked and returned results. This separation of content collection and syndication / visualization ensures an easy processing of the PServices' output, and it allows the SynServices to adjust visualization according to user devices' capabilities and limitations, or further user preferences.

By delivering visualization templates, every PService can optimize visualization and usability, as certain domain-specific information can be taken into account for creating the user interface.

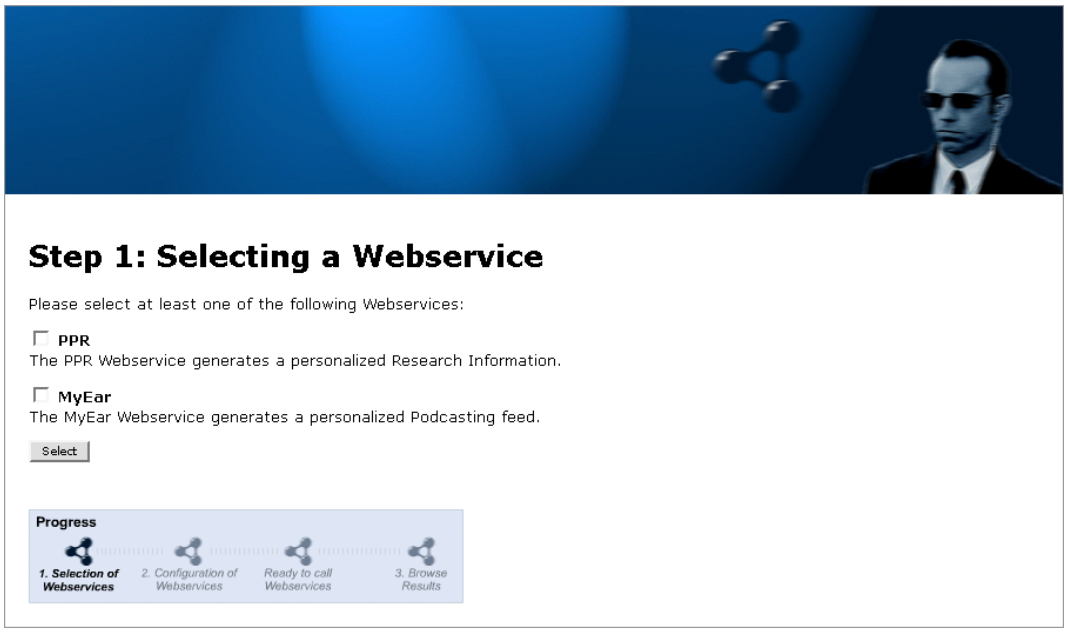


Figure 5.12: Dialog for Selecting Personalization Services

5.2.3 Scenario: MyEar Syndication Service

We use the *MyEar Syndication Service*, our Personal Music Syndicator[7], which provides recommendations for music podcast, for explaining the use of the Agent:

Assume a user who searches for podcasts in the Web. She enters a query and receives a list of appropriate podcast delivery services. She specifies which of these services she wants to launch. The user gets a list of all mandatory and optional parameters which can be used to tailor the services – the MyEar Syndication Service tries to fill all these parameters according to the information it has about the user's preferences. The user can change or simply approve these parameters, eventually the user is requested to enter information that

[7]We also created MyNews, a news aggregator with a similar usage scenario reusing services from MyEar

the MyEar Syndication Service was not able to provide. Finally, the user gets the syndicated output of all the services she launched, displayed in her personal Web interface. The appropriate visualization is chosen with respect to the currently used display device of the user.

The user can configure selected PServices. For example, the MyEar Syndication Service allows the user to specify keywords, duration and iTunes category of the podcasts she wants to listen to. The description of these customization parameters is provided by the PServices. The user profiling, which enables the automatic configuration of the PServices, is kept simple for this demonstrator: it stores the parameters the user has entered the last time she used this Web service in the UMService, and returns them as the default selection in the configuration dialog (see Figure 5.13).

The Agent can be seen as a dialog tool for application developers: whenever an application developer needs to know specific properties of a user, she normally asks the user to fill a form. The programmer will then process the data and might store it in a user profile for later use. The Agent simplifies this step by providing automatically missing parameters extracted from the global user profile, maintained by the UMService. This user profile will possibly contain a large set of standard properties as it is used for all Personal Reader applications. Only if the Agent has no information about the property, the user has to fill the property manually.

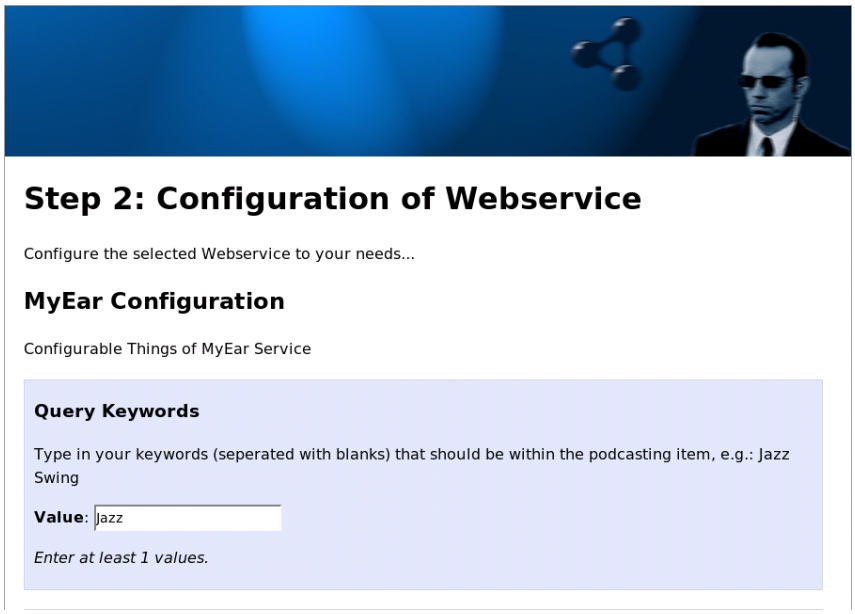


Figure 5.13: Configuration of the MyEar Syndication Service

After configuration, the MyEar Syndication Service is invoked with the specified parameters. This invocation is passed - via the connector - to the

corresponding PServices and MyEar receives the determined content (encoded as RDF document), as well as visualization templates. For MyEar, only one visualization template is currently available, which displays the RDF document on PCs within a Web browser, as can be seen in Figure 5.14.

The possibility to provide visualization templates by the PServices allows for domain-specific optimization of the user interface, which is not realizable with general-purpose RDF browsing approaches. In the case of the MyEar Syndication Service, for example drag and drop operations are available for selecting podcasts and controlling the audio together with further, music domain-specific gadgets.

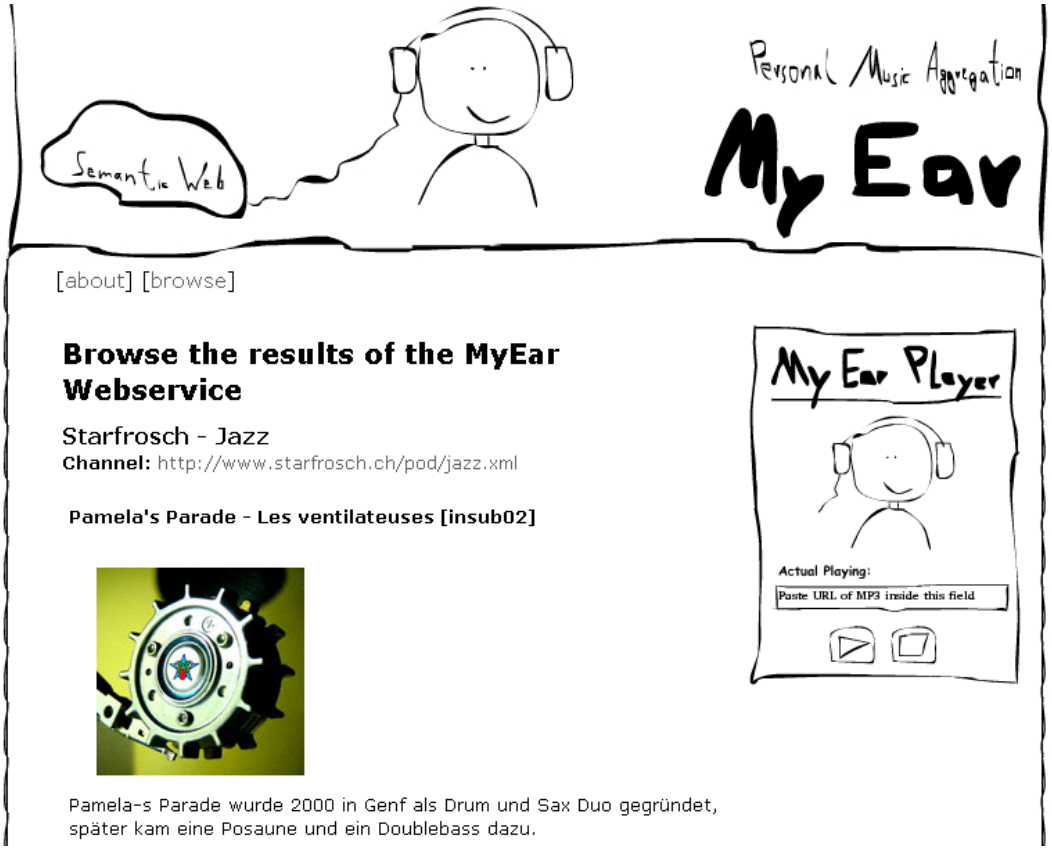


Figure 5.14: Visualization of the MyEar Syndication Service

5.2.4 Conclusion

With the Personal Reader Agent, we showcased how to provide a user-friendly interface to explore the Personal Reader Applications and to personalize the invocation of PServices. Thus, users are able to specify which PServices shall be invoked and which user profile information they shall receive. By querying

the UMService, the Agent is able to pre-fill PService's invocation parameters automatically and hence increases the ease of usages.

By utilizing the Agent's functionality a Personal Reader application does not need to take care on the user interaction required to personalize and adjust an application. Only the processing and visualization of the PService's results are still handled by the application. A usage scenario based on the *MyEar Music Syndication Service* shows the advantage of integrating the Agent into a Personal Reader application.

5.3 Usage of the Personal Reader Framework

This section gives an overview of the usage of the Personal Reader framework. First, a timeline of the Personal Reader applications as well as a table containing details of the usage of the Personal Reader components is given. Second, the usage statistics of the Personal Reader Website are evaluated.

5.3.1 Personal Reader Applications

As indicated in Figure 5.15, both, the extension of Framework functionality as well as the development of new Personal Reader applications was constantly performed over the entire considered time period. Personal Reader Framework components were developed when there was a need for it from a specific application and incorporated into existing applications. For example, the Agent and MyEar were upgraded to use the UMService after their development was finished. The continuous growth of core components based on real needs underlines the applicability of the framework in real-world applications. Table 5.1 gives an overview over the existing Personal Reader applications.

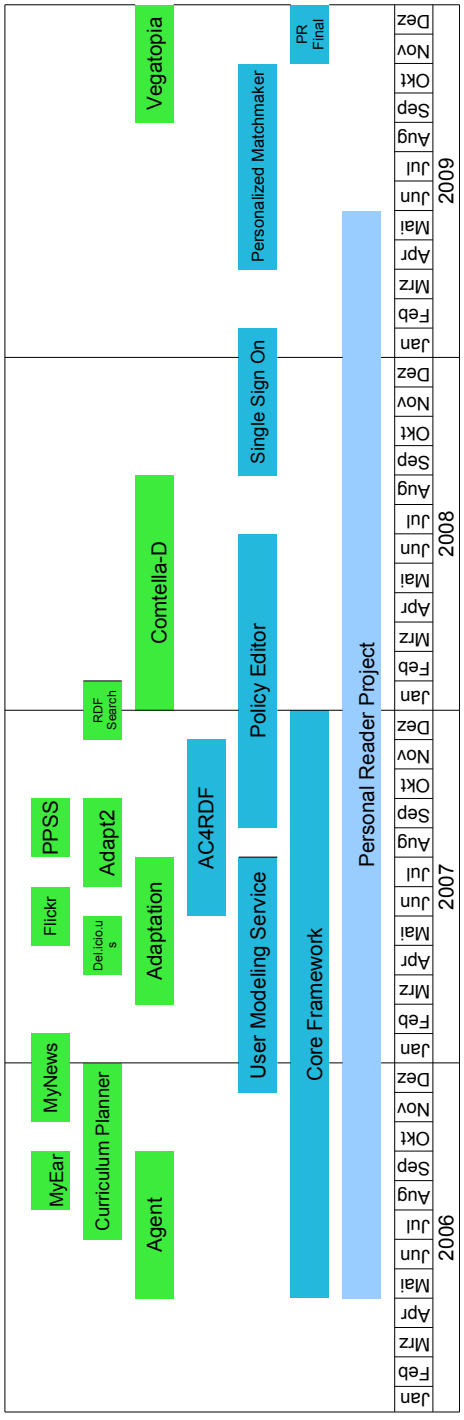


Figure 5.15: Development timeline – development of the Personal Reader Framework and the Personal Reader applications

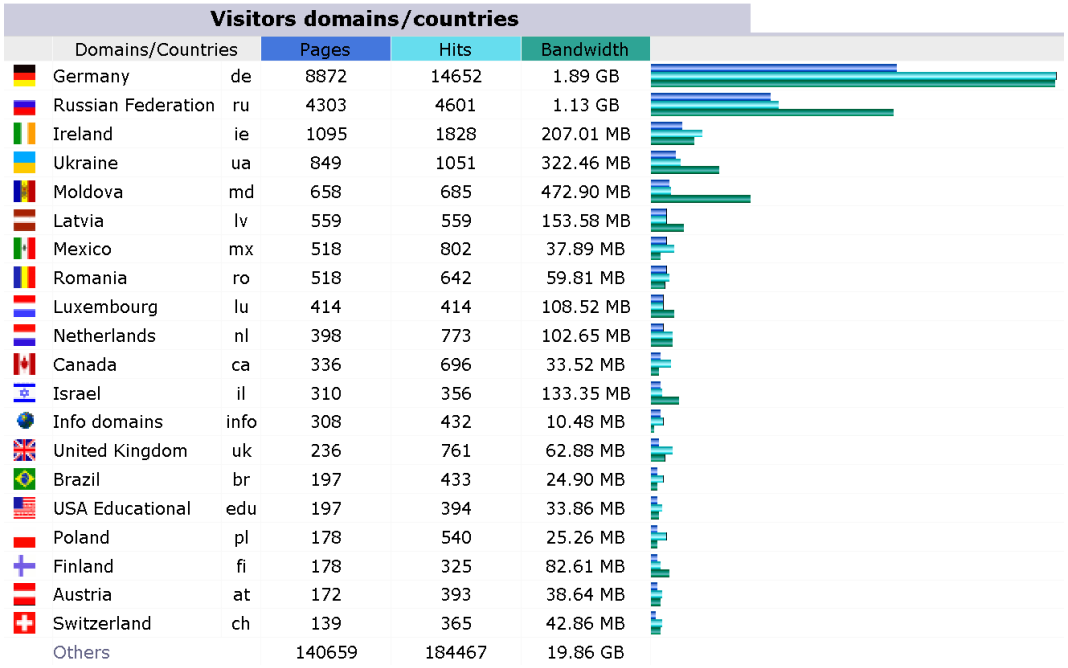
Application	**Development time**	**Description**	**Usage of the Framework**
PR Agent	05.2006 - 09.2006	Design and implementation of the Personal Reader Agent [Abel et al., 2006]	UMService to store configurations
PR Curriculum	07.2006 - 12.2006	Design and implementation of the Personal Curriculum Planner [Baldoni et al., 2006, Baldoni and Marengo, 2007], which is a service-oriented personalization system, set in an educational framework, based on a semantic annotation of courses, given at knowledge level.	-
MyEar	08.2006 - 09.2006	The development of the MyEar Music Recommender also gained a generic Personalization Service for personalizing RSS feeds.	Agent for configuration, UMService for storing listened songs
MyNews	11.2006 - 01.2007	The generic Personalization Service for personalizing RSS feeds was reused in order to realize MyNews, which enables users to browse or subscribe to personalized news feeds.	Agent for configuration
PR Deli.icio.us	04.2007 - 05.2007	The Personal Reader for Del.icio.us[a] bookmarks reused Personalization Services, which were originally developed in the context of the Personal Publication Reader[b] and the MyEar Music Recommender. Therewith, the time for developing PR Del.icio.us was minimized. [a]http://del.icio.us [b]The Personal Publication Reader [Baumgartner et al., 2005] was developed as a case study before the Personal Reader Framework was launched.	UMService for storing bookmarks
PR Flickr	05.2007 - 06.2007	The Personal Reader for Flickr[a] benefited from existing Personalization Services and the Framework infrastructure as well as the Personal Reader for Del.icio.us. [a]http://flickr.com	-
Adapt2	07.2007 - 09.2007	A Personal Reader, which connects the Personal Reader infrastructure with the *Advanced Distributed Architecture for Personalized Teaching & Training* (ADAPT2) [Brusilovsky et al., 2005b], which aims at providing personalization and adaptation services for developers of otherwise not personalized content, was developed in Summer 2007. The Personal Reader for Adapt2 made use of different Personalization Services that already existed within the Personal Reader environment, e.g. the User Mapping Service that was originally implemented for the Personal Publication Reader.	-

Application	**Development time**	**Description**	**Usage of the Framework**
PPSS	08.2007 - 09.2007	Development of the Personalized Preference Search Service[a] (PPSS) [Kärger et al., 2007] [a]http://www.personal-reader.de/PreferenceQueryGUI	-
RDF Search	11.2007 - 01.2008	Design and implementation of an RDF (Meta) Search Engine, which extends Sindice[a] [Tummarello et al., 2007] and other RDF search engines like Watson[b]. The RDF (Meta) Search Engine was – as all application listed in Figure 5.15 – realized by aid of the Personal Reader Framework and by utilizing the generic Personalization Service for personalizing RSS feeds. [a]http://sindice.com [b]http://watson.kmi.open.ac.uk	-
Comtella-D	01.2008 - 08.2008	Development of several recommender PServices for Comtella-D	Provides Recommender PService
Vegatopia	09.2009 - 12.2009	Recommendations for the Vegatopia[a] discussion board [a]http://www.vegatopia.com/	Reuses PServices from Comtella-D

Table 5.1: Overview of the Personal Reader applications

5.3.2 Usage Statistics of the Personal Reader Project

Figure 5.16 depicts the access statistics for the website of the Personal Reader Framework[8] created by AWStats[9]. This website promotes the framework itself and the various Personal Reader applications. In 2007 we promoted the website actively which resulted in more than 26.000 visitors (supplementary graphs are provided in Appendix D). In the following years the Personal Reader received continuous attention resulting in more than 1000 visits per month. This effect lasts until today without active promotion of the Website. In Figure 5.16 we analyzed the origin country of the visitors, which outlines the strong international attention that the project receives.



Visitors domains/countries

Domains/Countries		Pages	Hits	Bandwidth
Germany	de	8872	14652	1.89 GB
Russian Federation	ru	4303	4601	1.13 GB
Ireland	ie	1095	1828	207.01 MB
Ukraine	ua	849	1051	322.46 MB
Moldova	md	658	685	472.90 MB
Latvia	lv	559	559	153.58 MB
Mexico	mx	518	802	37.89 MB
Romania	ro	518	642	59.81 MB
Luxembourg	lu	414	414	108.52 MB
Netherlands	nl	398	773	102.65 MB
Canada	ca	336	696	33.52 MB
Israel	il	310	356	133.35 MB
Info domains	info	308	432	10.48 MB
United Kingdom	uk	236	761	62.88 MB
Brazil	br	197	433	24.90 MB
USA Educational	edu	197	394	33.86 MB
Poland	pl	178	540	25.26 MB
Finland	fi	178	325	82.61 MB
Austria	at	172	393	38.64 MB
Switzerland	ch	139	365	42.86 MB
Others		140659	184467	19.86 GB

Figure 5.16: Countries of visitors of the Personal Reader website from 2009

5.4 Conclusion

In this chapter, we gave a detailed description of two applications, utilizing the Personal Reader Framework, namely the Thread Recommender for the Comtella-D discussion forum and the personalized invocation of PServices of MyEar by using the Agent.

In Comtella-D, the Personal Reader Framework shortened the development time massively, as existing recommender PServices were available and could

[8] http://www.personal-reader.de
[9] http://awstats.sourceforge.net/

be reused for this recommendation task. The plug-and-play architecture of the Personal Reader allowed to develop a prototype of the Comtella-D recommender to perform the evaluation of different recommender algorithms. This evaluation revealed that a small amount of user feedback is sufficient to provide better recommendations as a top-k list of most favorite discussion board posts would do. An interesting observation of the evaluation was that taking all available user profile information for generating recommendations does not offer the highest quality. Instead, selecting high-quality user data carefully, resulted in better recommendations. This outcome was transformed into a PService selection rule which selects the recommender algorithm (and hence the PService) based on available user profile data. Finally, Comtella-D benefits from the Personal Reader infrastructure as further developed and improved recommender strategies can be easily integrated as new PServices without changing the existing applications.

The Personal Reader Agent provides an application-independent user interface that allows users to discover Personal Reader applications and to configure PServices. Application developers do no longer need to take care on receiving user information as they are provided directly by the Agent. By accessing the UMService, the Agent automatically receives user's preferred default values to minimize the interaction with the user.

An overview about the purpose-driven development process of the Personal Reader Framework is finally given. The reasonable amount of Personal Reader applications outlines the success of the framework. The constantly large number of visitors of the project's homepage from several countries is an additional indicator for the success of the Personal Reader Framework.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

Currently available personalization and user modeling functionality is strongly optimized for a specific application, making it hardly reusable. The motivation of this thesis is to present and discuss approaches for supporting the entire life-cycle of a personalized application by providing centralized functionality and offering generic personalization and user modeling components. Based on the motivation, I presented the following five research questions in the introduction:

a) Can the strongly-coupled personalization process of monolithic applications be divided into logically independent services?

b) Can such personalization services be reused in various applications?

c) How shall user profiles be stored, maintained, and accessed in a Semantic Web Service-based environment?

d) Can personalization be used to orchestrate personalized applications from single Web Services?

e) Which requirements need to be fulfilled by a personalization framework to ease the process of creating a personalized application and which support needs to be offered to assist the programmers in this process?

To answer these questions, I first conducted a literature research and evaluated the current state-of-the-art approaches for generic user modeling and personalization. From this, I derived possible obstacles why personalization and user modeling is not used more frequently in today's applications. Together with open questions about the possible future and trending topics of personalization and the questions served a input for the design of a questionnaire. The questionnaire was filled by personalization and user modeling experts at

the Adaptive Hypermedia Conference 2008 and revealed that interoperability, reusability and the usage of Web Service are key techniques to ease the process of creating personalized applications.

I picked up these techniques and implemented the Personal Reader Framework, which makes personalization functionality reusable by encapsulating generic personalization algorithms into Semantic Web Services, so-called PServices. Applications, represented in the framework as SynServices, shall discover PServices during runtime and hence be able to use personalization in a plug-and-play manner. To assist the discovery of PServices, I incorporated Web 2.0-style user feedback into the matchmaking process, turning it into a personalized matchmaker. Users were involved in the service selection process and actively improved the service selection. In this chapter, I used the concepts of the Personal Reader Framework to answer the above mentioned research questions.

The framework additionally supports developers of personalized applications by providing centralized user modeling. I developed the User Modeling Ontology to store user-related data in a central place within the Personal Reader Framework. This central repository was realized as a Web Service, called the User Modeling Service. This services allows different Personal Reader applications to exchange data with each other even if they use different vocabularies. In order to protect the RDF-based user profiles within the User Modeling Service, I developed AC4RDF, which allows to protect arbitrary RDF repositories on RDF Triple level by the use of expressive Protune policies. A user interface allows non-technical user to specify policies without the need of having knowledge of RDF or Protune.

The Personal Reader Framework was successfully used to generate recommendations in an online discussion forum. PServices, which provide recommendations based on different kinds of input data, are selected during runtime. With different experiments, I determined a selection rule which ensures that the best-performing PService was invoked. I presented the Personal Reader Agent as a central entry point into the portal, allowing to store and reuse configuration values of Personal Reader applications. The development timeline of the Personal Reader and a table of all currently available Personal Reader applications complements the thesis.

Concluding, this thesis goes beyond state-of-the-art in the following five areas:

Generic Personalization The Personal Reader Framework introduces the concept of *Personalization Services* to support personalization in a plug-and-play manner. Personalization Services encapsulate reusable personalization functionality, like recommender algorithms and offer Web Service interfaces to adjust the service's functionality. To adopt the functionality according to

a user, Personalization Services can access the User Modeling Service without any additional implementation effort from the application's programmer. The framework offers different Personalization Services for collaborative recommendations and personalized search. Several applications, that have been presented in the previous chapter, verify that the PService concept is beneficial applicable.

Generic User Modeling I presented the *User Modeling Service*, which is a generic, domain- and application-independent component. The service provides an extensible, domain-independent ontology and uses RDF as message storage format. By enhancing the ontology according to domain-specific needs, applications can still pertain their own vocabulary. The ontology and the provided methods for user profile access and mapping ensure interoperability so that different applications can easily exchange user profile data. The User Modeling Service is realized as a centralized component and acts on behalf of the user to allow the user to inspect and modify the user profile as well as protect it. The User Modeling Service was successfully integrated into MyEar.

User Profile Protection utilizing Policies The *Access Control for RDF* component allows a fine-grained protection of RDF-based user profiles. The component enforces expressive Protune policies by enhancing an RDF query by additional constraints, which exclude the protected data from the result set. Experiments show that the rewritten queries decrease performance predictably and scale well. Access protection in the Personal Reader Framework comes with a user-friendly user interface that allows the user to specify powerful access restrictions. Users are forwarded to the GUI whenever new applications try to access data or known applications try to access new data, so there is no initial configuration effort to ensure privacy in the Framework. A user study proves that users can specify and handle complex access policies utilizing the GUI.

Personalized Matchmaking State-of-the-art matchmaking of Semantic Web Services was based on the match of input and output parameters as well as pre- and postconditions defined by the Semantic Web Service description and a service request. For a matchmaker it is not possible to distinguish services with the same service description that deliver results of a different quality. I used the Web 2.0 paradigm of user generated feedback and incorporated user ratings into the matchmaking process to rank the most-popular and best-matching services first. The evaluation reveals that feedback-aware matchmaking algorithms outperformed the state-of-the-art baseline matchmakers.

Personalization and User Modeling Framework The Personal Reader Framework is the first framework that supports the development process of personalized applications by providing central functionality, like user modeling, privacy protection, and personalized matchmaking, ready-to-run personalization functionality encapsulated into PServices and an overall design architecture for personalized applications, splitting the application into SynServices and PServices. The framework realizes the recommendations given by the experts in the survey: interoperability and reusability of personalization functionality as well as storage of user profile data have been realized by usage of Semantic Web techniques.

6.2 Outlook to Future Research Directions

From the open issues mentioned in the thesis, I selected the three most promising approaches for continuing research in the area of personalization:

Enhance Application Fields of Generic User Modeling and Personalization With the framework, I have proven that some personalization algorithms, like collaborative recommender systems can be made reusable by encapsulating the functionality. Other areas, where generic personalization algorithms have a strong potential are adaptive hypermedia systems. Especially with the increasing popularity of E-Learning systems, support for simplified implementation of personalization in that area is strongly needed.

Awareness and Scrutability of Personalization and User Modeling In the Personal Reader Framework, the key techniques, which have been proposed by the participants of the questionnaire, like interoperability, reuse and Web Service-based architecture have been implemented. In the last part of the questionnaire, the participants named non-technical challenges like scrutability and increased user awareness of personalization. In the framework we tried to simplify the usage by hiding technical details from the end user. An interesting challenge for an improved interface design is to integrate explanations of the personalization process or to show a comparison of personalized and non-personalized output to the end-users.

Extended PService Composition In the thesis, we proposed a matchmaker which is able to select from a given list of available Personalization Services the best matching. The matchmaker did not take into account that a composition of several services might result in a better fit than a single service. We have foreseen this composition of single PServices in our Personal Reader architecture. However, to the best of our knowledge there is no Web Service orchestration algorithm available which does take user feedback into account.

I plan to apply the underlying idea of the personalized matchmaker algorithm to develop a personalized Web Service orchestrater.

Appendix A

Publications

List of published publications:

1. Fabian Abel, Nicola Henze, Eelco Herder, Daniel Krause: Linkage, Aggregation, Alignment and Enrichment of Public User Profiles with Mypes. In Proceedings of 6th International Conference on Semantic Systems, Graz, Austria, September 2010
2. Fabian Abel, Ernesto Diaz-Aviles, Nicola Henze, Daniel Krause, Patrick Siehndel: Analyzing the Blogosphere for Predicting the Success of Music and Movie Products. In Proceedings of International Conference on Advances in Social Networks Analysis and Mining, Odense, Denmark, August 2010
3. Fabian Abel, Nicola Henze, Daniel Krause: Optimizing Search and Ranking in Folksonomy Systems by Exploiting Context Information. Lecture Notes in Business Information Processing, Vol. 45, 2010
4. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, Daniel Olmedilla: The RDF Protune Policy Editor: Enabling Users to Protect Data in the Semantic Web. Lecture Notes in Business Information Processing, Vol. 45, 2010
5. Fabian Abel, Ig Ibert Bittencourt, Evandro Costa, Nicola Henze, Daniel Krause, Julita Vassileva: Recommendations in Online Discussion Forums for E-Learning Systems. IEEE Transactions on Learning Technologies, IEEE Computer Society, 2010
6. Fabian Abel, Nicola Henze, Eelco Herder, Daniel Krause: Interweaving Public User Profiles on the Web. In Proceedings of International Conference on User Modeling, Adaptation and Personalization, Hawaii, USA, June 2010
7. Fabian Abel, Nicola Henze, Eelco Herder, Geert-Jan Houben, Daniel Krause, Erwin Leonardi: Building Blocks for User Modeling with data

from the Social Web. In Proceedings of International Workshop on Architectures and Building Blocks of Web-Based User-Adaptive Systems at the International Conference on User Modeling, Adaptation and Personalization, Hawaii, USA, June 2010

8. Fabian Abel, Ricardo Kawase, Daniel Krause: Leveraging Multi-faceted Tagging to improve Search in Folksonomy Systems. In Proceedings of 21st ACM Conference on Hypertext and Hypermedia, Toronto, Canada, June 2010

9. Fabian Abel, Nicola Henze, Ricardo Kawase, Daniel Krause: The Impact of Multifaceted Tagging on Learning Tag Relations and Search. In Proceedings of Seventh Extended Semantic Web Conference, Heraklion, Crete, Greece, May 2010

10. Fabian Abel, Ricardo Kawase, Daniel Krause, Patrick Siehndel, Nicole Ullmann: The Art of Tagging - Interweaving spatial annotations, categories, meaningful URIs and tags. 6th International Conference on Web Information Systems and Technologies, 7-10 April 2010, Valencia, Spain

11. Anna Averbakh, Daniel Krause, Dimitrios Skoutas: Exploiting User Feedback to Improve Semantic Web Service Discovery. 8th International Semantic Web Conference, 25-29 October 2009, Washington DC, USA

12. Fabian Abel, Ricardo Kawase, Daniel Krause, Patrick Siehndel: Multifaceted Tagging in TagMe!. 8th International Semantic Web Conference, 25-29 October 2009, Washington DC, USA

13. Anna Averbakh, Daniel Krause, Dimitrios Skoutas: Recommend me a Service: Personalized Semantic Web Service Matchmaking. 17th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2009 - Workshop-Woche: Lernen-Wissen-Adaption, September 21-23, 2009, Darmstadt, Germany

14. Fabian Abel, Dominikus Heckmann, Eelco Herder, Jan Hidders, Geert-Jan Houben, Daniel Krause, Erwin Leonardi, Kees van der Slujis: Mashing up user data in the Grapple User Modeling Framework. 17th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2009 - Workshop-Woche: Lernen-Wissen-Adaption, September 21-23, 2009, Darmstadt, Germany

15. Fabian Abel, Dominikus Heckmann, Eelco Herder, Jan Hidders, Geert-Jan Houben, Daniel Krause, Erwin Leonardi, Kees van der Sluijs: A Framework for Flexible User Profile Mashups. International Workshop on Adaptation and Personalization for Web 2.0, collocated with UMAP 2009, June 22, 2009, Trento, Italy

16. Erwin Leonardi, Geert-Jan Houben, Kees van der Sluijs, Jan Hidders, Eelco Herder, Fabian Abel, Daniel Krause, Dominik Heckmann: User Profile Elicitation and Conversion in a Mashup Environment. International Workshop on Lightweight Integration on the Web, collocated with ICWE 2009, June, 2009, San Sebastian, Spain

17. Fabian Abel, Nicola Henze, Daniel Krause: Social Semantic Web at Work: Annotating and Grouping Social Media Content. Web Information Systems and Technologies. Lecture Notes in Business Information Processing, Vol. 18, 2009

18. Fabian Abel, Nicola Henze, Daniel Krause, Matthias Kriesell: On the Effect of Group Structures on Ranking Strategies in Folksonomies, Weaving Services and People on the World Wide Web, Springer, 2009

19. Fabian Abel, Nicola Henze, Daniel Krause, Mathias Kriesell: Semantic Enhancement of Social Tagging Systems. Annals of Information Systems, Special Issue on "Semantic Web and Web 2.0", Springer, 2009

20. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, Daniel Olmedilla: A User Interface to Define and Adjust Policies for Dynamic User Models, 5th International Conference on Web Information Systems and Technologies, March 23-26, 2009, Lisboa, Portugal

21. Fabian Abel, Nicola Henze, Daniel Krause: Context-aware Ranking Algorithms in Folksonomies, 5th International Conference on Web Information Systems and Technologies, March 23-26, 2009, Lisboa, Portugal

22. Fabian Abel, Matteo Baldoni, Cristina Barolgio, Nicola Henze, Daniel Krause, Viviana Patti: Context-based Ranking in Folksonomies, 20th ACM Conference on Hypertext and Hypermedia, June 29 - July 1st, 2009, Torino, Italy

23. Fabian Abel, Nicola Henze, Daniel Krause: Exploiting additional Context for Graph-based Tag Recommendations in Folksonomy Systems, IEEE / WIC / ACM Conference on Web Intelligence, December 9-12, 2008, Sydney, Australia

24. Fabian Abel, Nicola Henze, Daniel Krause: Search in Folksonomy Systems: Can groups help?, ACM 17th Conference on Information and Knowledge Management, October 26-30, 2008, Napa Valley, California, USA

25. Fabian Abel, Nicola Henze, Daniel Krause, Daniel Plappert: User Modeling and User Profile Exchange for Semantic Web Applications, 16th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2008 - Workshop-Woche: Lernen-Wissen-Adaption, October 6-8, 2008, Würzburg, Germany

26. Melanie Hartmann, Daniel Krause, Andreas Nauerz: 16th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2008 - Workshop-Woche: Lernen-Wissen-Adaption, October 6-8, 2008, Würzburg, Germany

27. Arne W. Koesling, Daniel Krause, Eelco Herder: Flexible Adaptivity in AEHS Using Policies. 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, July 28-August 1, 2008, Hannover, Germany

28. Fabian Abel, Ig Ibert Bittencourt, Nicola Henze, Daniel Krause, Julita Vassileva: A Rule-Based Recommender System for Online Discussion Forums. 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, July 28-August 1, 2008, Hannover, Germany

29. Fabian Abel, Nicola Henze, Daniel Krause: On the Effect of Group Structures on Ranking Strategies in Folksonomies. Workshop on Social Web Search and Mining, April 22, 2008, Beijing, China, collocated with WWW 2008

30. Fabian Abel, Nicola Henze, Daniel Krause: GroupMe! – Where Information meets - 17th International World Wide Web Conference, April 21-25, 2008, Beijing, China

31. Fabian Abel, Mischa Frank, Nicola Henze, Daniel Krause, Patrick Siehndel: GroupMe! – Combining Ideas of Wikis, Social Bookmarking, and Blogging, International Conference on Weblogs and Social Media, March 31-April 2, 2008, Seattle, USA

32. Fabian Abel, Nicola Henze, Daniel Krause: A Novel Approach to Social Tagging: GroupMe!, 4th International Conference on Web Information Systems and Technologies, May 4-7, 2008, Funchal/Madeira, Portugal

33. Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, Daniel Olmedilla: Enabling Advanced and Context-Dependent Access Control in RDF Stores. 6th International Semantic Web Conference, November 11-15, 2007, Busan, Korea

34. Fabian Abel, Mischa Frank, Nicola Henze, Daniel Krause, Daniel Plappert, and Patrick Siehndel: GroupMe! - Where Semantic Web meets Web 2.0. Semantic Web Challenge, 6th International Semantic Web Conference, November 11-15, 2007, Busan, Korea

35. Fabian Abel, Nicola Henze, and Daniel Krause: GroupMe! - Capturing Semantics in Social Tagging Systems. I-SEMANTICS '07, 3rd International Conference on Semantic Technologies, September 2007, Graz

36. Ingo Brunkhorst, Daniel Krause, Wassiou Sitou: 15th Workshop on Adaptivity and User Modeling in Interactive Systems. LWA 2007 - Workshop-Woche: Lernen-Wissen-Adaption, September 24-26, 2007, Halle/Saale, Germany

37. Nicola Henze and Daniel Krause: Personalized Access to Web Services in the Semantic Web. 3rd International Semantic Web User Interaction Workshop, November 6, 2006, Athens, Georgia, USA, collocated with ISWC 2006

38. Nicola Henze and Daniel Krause: User Profiling and Privacy Protection for a Web Service Oriented Semantic Web. 14th Workshop on Adaptivity and User Modeling in Interactive Systems, Hildesheim, October 9-11 2006

39. Fabian Abel, Ingo Brunkhorst, Nicola Henze, Daniel Krause, Kashif Mushtaq, Peyman Nasirifard and Kai Tomaschewski: Personal Reader Agent: Personalized Access to Configurable Web Services. 14th Workshop on Adaptivity and User Modeling in Interactive Systems, Hildesheim, October 9-11 2006

40. Nicola Henze and Daniel Krause: Scalable Matchmaking for a Semantic Web Service based Architecture - Workshop on Semantics for Web Services, December 4, 2006, Zurich, Switzerland, collocated with ECOWS 2006

Appendix B

Questionnaire

For the evaluation of the future trends in the area of personalization the following questionnaire was used. The questionnaire was given to the attendees of the 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems that took place in Hannover in 2008.

Hannover, 29 July - 1 August 2008

Questionnaire

Future Perspectives on Personalization

Dear AH2008 participants,

I am Daniel Krause and work as Phd student at L3S Research Center in Hannover. This questionnaire aims at identifying the next, important steps towards advanced, easy-to-implement and easy-to-maintain personalized systems for the Web and solicits responses from experts in the field.

All data provided by you will by used for research purposes only. The questionnaire is part of my Phd work and the results of this study will be published online at:

http://personal-reader.de/questionnaire/

The evaluation will be published by 20th August 2008.

Please return the questionnaire at the welcome desk.

Thank you very much for your filling out this questionnaire!

Daniel Krause

I Experiences from a user perspective

yes no

1. Are you satisfied with personalization offered by current personalized applications (like Amazon, AHA!, Last.fm, etc.)? □ □ □ □ □

2. Which kind of personalization was offered by the systems that you have used? Do you consider the personalization functionality as *valuable* and have you been *satisfied* with the result?

Kind of personalization functionality	Valuable? high low	Satisfying? high low	If you were not satisfied, can you give a reason why?
recommendations e.g. book recommendations in Amazon	□□□□□	□□□□□	
device adaptation e.g mobile versions of websites	□□□□□	□□□□□	
navigation support e.g personalized links to relevant sites	□□□□□	□□□□□	
adaptive presentation e.g order item according to user's needs	□□□□□	□□□□□	
adaptation of content e.g. omitting text details from news	□□□□□	□□□□□	

yes no

3. a) Do you agree that personalization is useful in general? □ □ □ □ □

   b) From a user's perspective: For which purposes do you consider personalization as most useful? Please select at most 3 items.

   □ simplified interaction for beginners
   □ time saving
   □ better feedback from the system
   □ improved interaction
   □ better orientation
   □ fun to use
   □ improved interfaces
   □ _____

100% 50% 0%

4. How many of the applications that you have used were personalized? □□□□□□□□□□□

5. Based on the previous question, how many of the applications should be personalized?
   □ much more □ some more □ just right □ some less □ much less

6. The personalization potential of today's applications is rarely used. What are the main reasons for this? Please select at most 3 items.

   □ functionality unclear
   □ results not satisfying
   □ missing transparency
   □ missing controllability
   □ too fast adjustment
   □ privacy concerns of the users
   □ slow adjustment
   □ _____

II Experiences from a developer's perspective

7. How many years of experience in developing personalized systems (web-applications, applications, prototypes, exploitables, etc.) do you have? _____

8. How many systems (personalized and non-personalized) have you created? _____

9. How many of the applications that you have created were personalized? 100% 50% 0% □□□□□□□□□□□

10. How many of the applications that you have created could benefit from personalization? □□□□□□□□□□□

11. a) From a developer's/designer's/manager's perspective: What are the reasons for not implementing personalization? Please select at most 3 items.

□ return on investment too low
□ lack of results/effects
□ acceptance of the users is critical
□ lack of reusable components
□ lack of software engineering support
□ uncontrollable system behavior
□ interoperability to existing systems is not given
□ _____

b) What are the main technical problems for implementing personalization?

□ lack of libraries/tools
□ implementation effort too high
□ results are hard to control
□ lack of best practices / common approaches
□ _____

12. If you have realized personalization in your applications, please give us some details in the following tables:

a) Name and short description of the application:

b) Description of the implemented personalization functionality:

c) Did you reuse existing personalization algorithms (includes pseudo-code)?
□ yes, an existing algorithm without modifications
□ yes, an existing algorithm with modifications
□ no, a newly developed algorithm
□ _____

d) Did you reuse existing personalization code (exclusive pseudo-code)?
□ yes, a *coding template*[1]
□ yes, a *web service*
□ yes, a *coding library*[2]
□ no
□ _____

e) What were the main challenges for implementing personalization functionality?

f) Reusability of code for personalization functionality of this application is possible □ □ □ □ □ not possible

[1] A coding template is a snippet of code which can be inserted, e.g. by the IDE, and afterwards is edited by the programmer
[2] A coding library is encapsulated code, e.g. a whole class, which can be used via interfaces

a) Name and short description of the application:

b) Description of the implemented personalization functionality:

c) Did you reuse existing personalization algorithms?
□ yes, an existing algorithm without modifications
□ yes, an existing algorithm with modifications
□ no, a newly developed algorithm
□ _____

d) Did you reuse existing personalization code?
□ yes, a *coding template*
□ yes, a *web service*
□ yes, a *coding library*
□ no
□ _____

e) What were the main challenges for implementing personalization functionality?

f) Reusability of code for personalization functionality of this application is	possible				not possible
	□	□	□	□	□

III Future perspectives on personalization

	yes				no
13. Do you think that it is possible to establish interoperability between personalized applications?	□	□	□	□	□
14. Would interoperability between personalized applications increase the number of personalized applications?	□	□	□	□	□

15. Which techniques fit best to improve interoperability between personalized applications? Please select at most 2 items.

□ Web Services □ Semantic Web Services □ RSS/RDF interfaces
□ Ontologies □ other XML interfaces □ _____

	yes				no
16. Do you think that it is possible to create reusable personalization functionality?	□	□	□	□	□
17. Would reusable personalization functionality increase the number of personalized applications?	□	□	□	□	□

18. Please explain your answer of question 17: Why do you/don't you think that personalization can benefit from reusability?

19. How important do you consider the reusability of the following components:

	Importance high low
user event detection	□ □ □ □ □
user modeling	□ □ □ □ □
user modeling ontology	□ □ □ □ □
recommendations	□ □ □ □ □
device adaptation	□ □ □ □ □
navigation support	□ □ □ □ □
adaptive presentation	□ □ □ □ □
adaptation of content	□ □ □ □ □
other: _____	□ □ □ □ □

20. a) Which of the following components of a personalization system can be made reusable in which way? Please fill in your ratings in each cell. (Score ranges from 1 to 5: 1=impossible, 5=possible)

	data	algorithm	code template	code library	web service
user event detection					
user modeling					
user modeling ontology					
recommendations					
device adaptation					
navigation support					
adaptive presentation					
adaptation of content					
other: _____					

b) Can you give a reason why components are not reusable?

	Reason
user event detection	
user modeling	
user modeling ontology	
recommendations	
device adaptation	
navigation support	
adaptive presentation	
adaptation of content	
other: _____	

21. a) Which of the reusability levels (data, algorithm, code template, ...) bear the highest impact? Please select at most 2 items.

□ data □ algorithm □ code template □ code library □ web service

b) Which strategies would you prefer for reusing personalization functionality? Please select at most 2 items.

□ data □ algorithm □ code template □ code library □ web service

IV Open questions

22. What do you consider as the biggest challenges for making personalization reusable?

23. What other techniques can be used to simplify the usage of personalization?

24. What do you consider as the biggest challenges for personalization?

25. What do you think are the most promising future trends in the area of personalization?

Appendix C

Association Rules

R1.	Q1.2 → Q2.Adaption of Content.Satisfying.2			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R2.	Q1.4 → Q6 slow adjustment.1			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R3.	Q1.4 → Q15 RSS/RDF interfaces.1			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,50	Coverage: 0,58
R4.	Q1.4 → Q19 user modeling.4			
	Confidence: 0,67	Support: 0,25	#Y(Y.b)/#Y 0,38	Coverage: 0,67
R5.	Q1.5 → Q2.DeviceAdaption.Valuable.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R6.	Q2.Recommendations.Valuable.3 → Q2.NavigationSupport.Valuable.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R7.	Q2.Recommendations.Valuable.3 → Q19 user modeling ontology.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R8.	Q2.Recommendations.Valuable.4 → Q6 functionality unclear.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R9.	Q2.Recommendations.Valuable.4 → Q21a data.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R10.	Q2.Recommendations.Valuable.5 → Q3b improved interaction.1			
	Confidence: 0,64	Support: 0,29	#Y(Y.b)/#Y 0,42	Coverage: 0,70
R11.	Q2.Recommendations.Valuable.5 → Q11a return on investment too low.1			
	Confidence: 0,73	Support: 0,33	#Y(Y.b)/#Y 0,42	Coverage: 0,80

R12.	Q2.Recommendations.Satisfying.2 → Q6 privacy concerns of the users.1			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R13.	Q2.Recommendations.Satisfying.4 → Q19 device adaptation.3			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R14.	Q2.Recommendations.Satisfying.4 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,25	Coverage: 0,67
R15.	Q2.Recommendations.Satisfying.5 → Q9.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R16.	Q2.Recommendations.Satisfying.5 → Q13.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R17.	Q2.Recommendations.Satisfying.5 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R18.	Q2.Recommendations.Satisfying.5 → Q19 recommendations.4			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R19.	Q2.Recommendations.Satisfying.5 → Q19 adaptation of content.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R20.	Q2.DeviceAdaption.Valuable.4 → Q2.Adaption of Content.Valuable.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R21.	Q2.DeviceAdaption.Valuable.5 → Q3b improved interaction.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R22.	Q2.DeviceAdaption.Valuable.5 → Q6 missing controllability.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R23.	Q2.DeviceAdaption.Valuable.5 → Q6 privacy concerns of the users.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R24.	Q2.DeviceAdaption.Valuable.5 → Q11a uncontrollable system behavior.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,38	Coverage: 0,56
R25.	Q2.DeviceAdaption.Valuable.5 → Q14.5			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R26.	Q2.DeviceAdaption.Valuable.5 → Q15 other XML interfaces.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,33	Coverage: 0,62
R27.	Q2.DeviceAdaption.Valuable.5 → Q19 user event detection.5			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60

R28.	Q2.DeviceAdaption.Valuable.5 → Q21a code library.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,29	Coverage: 0,71
R29.	Q2.DeviceAdaption.Valuable.5 → Q21b code library.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R30.	Q2.DeviceAdaption.Satisfying.1 → Q7.10			
	Confidence: 1,00	Support: 0,17	#Y(Y.b)/#Y 0,25	Coverage: 0,67
R31.	Q2.DeviceAdaption.Satisfying.1 → Q8.10			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,21	Coverage: 0,60
R32.	Q2.DeviceAdaption.Satisfying.1 → Q15 other XML interfaces.1			
	Confidence: 1,00	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R33.	Q2.DeviceAdaption.Satisfying.1 → Q19 device adaptation.4			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R34.	Q2.DeviceAdaption.Satisfying.1 → Q21a code library.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R35.	Q2.DeviceAdaption.Satisfying.2 → Q2.Adaption of Content.Valuable.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R36.	Q2.DeviceAdaption.Satisfying.2 → Q15 Semantic Web Services.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R37.	Q2.DeviceAdaption.Satisfying.3 → Q11a uncontrollable system behavior.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R38.	Q2.DeviceAdaption.Satisfying.3 → Q19 user modeling.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R39.	Q2.DeviceAdaption.Satisfying.5 → Q19 navigation support.5			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R40.	Q2.DeviceAdaption.Satisfying.5 → Q19 adaptation of content.5			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R41.	Q2.NavigationSupport.Valuable.1 → Q2.AdaptivePresentation.Valuable.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R42.	Q2.NavigationSupport.Valuable.1 → Q2.Adaption of Content.Satisfying.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R43.	Q2.NavigationSupport.Valuable.1 → Q7.10			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33

R44.	Q2.NavigationSupport.Valuable.1 → Q11b implementation effort too high.1			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R45.	Q2.NavigationSupport.Valuable.1 → Q17.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R46.	Q2.NavigationSupport.Valuable.1 → Q19 user modeling ontology.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R47.	Q2.NavigationSupport.Valuable.1 → Q21b code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R48.	Q2.NavigationSupport.Valuable.2 → Q9.2			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R49.	Q2.NavigationSupport.Valuable.3 → Q2.Adaption of Content.Satisfying.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R50.	Q2.NavigationSupport.Valuable.3 → Q5.4			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R51.	Q2.NavigationSupport.Valuable.3 → Q11b lack of best practices.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R52.	Q2.NavigationSupport.Valuable.3 → Q14.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R53.	Q2.NavigationSupport.Valuable.3 → Q17.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R54.	Q2.NavigationSupport.Valuable.3 → Q19 user modeling ontology.4			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R55.	Q2.NavigationSupport.Valuable.3 → Q19 adaptive presentation.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R56.	Q2.NavigationSupport.Valuable.4 → Q2.NavigationSupport.Satisfying.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,29	Coverage: 0,57
R57.	Q2.NavigationSupport.Valuable.4 → Q19 recommendations.3			
	Confidence: 0,83	Support: 0,21	#Y(Y.b)/#Y 0,33	Coverage: 0,62
R58.	Q2.NavigationSupport.Valuable.5 → Q2.AdaptivePresentation.Valuable.5			
	Confidence: 0,83	Support: 0,21	#Y(Y.b)/#Y 0,25	Coverage: 0,83
R59.	Q2.NavigationSupport.Valuable.5 → Q3b improved interaction.1			
	Confidence: 0,83	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50

R60.	Q2.NavigationSupport.Valuable.5 → Q4.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,29	Coverage: 0,57
R61.	Q2.NavigationSupport.Valuable.5 → Q11b lack of libraries/tools.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R62.	Q2.NavigationSupport.Valuable.5 → Q11b implementation effort too high.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,29	Coverage: 0,57
R63.	Q2.NavigationSupport.Valuable.5 → Q15 other XML interfaces.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R64.	Q2.NavigationSupport.Satisfying.1 → Q8.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R65.	Q2.NavigationSupport.Satisfying.1 → Q15 Ontologies.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R66.	Q2.NavigationSupport.Satisfying.1 → Q19 navigation support.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R67.	Q2.NavigationSupport.Satisfying.1 → Q19 adaptive presentation.3			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R68.	Q2.NavigationSupport.Satisfying.1 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R69.	Q2.NavigationSupport.Satisfying.2 → Q2.Adaption of Content.Satisfying.2			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R70.	Q2.NavigationSupport.Satisfying.4 → Q2.AdaptivePresentation.Satisfying.4			
	Confidence: 0,86	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R71.	Q2.NavigationSupport.Satisfying.4 → Q19 recommendations.3			
	Confidence: 0,86	Support: 0,25	#Y(Y.b)/#Y 0,33	Coverage: 0,75
R72.	Q2.AdaptivePresentation.Valuable.1 → Q7.10			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R73.	Q2.AdaptivePresentation.Valuable.1 → Q21b code template.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R74.	Q2.AdaptivePresentation.Valuable.4 → Q15 RSS/RDF interfaces.1			
	Confidence: 0,77	Support: 0,42	#Y(Y.b)/#Y 0,50	Coverage: 0,83
R75.	Q2.AdaptivePresentation.Valuable.5 → Q11b lack of libraries/tools.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50

R76.	Q2.AdaptivePresentation.Valuable.5 → Q11b implementation effort too high.1			
	Confidence: 0,83	Support: 0,21	#Y(Y.b)/#Y 0,29	Coverage: 0,71
R77.	Q2.AdaptivePresentation.Valuable.5 → Q15 other XML interfaces.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R78.	Q2.AdaptivePresentation.Valuable.5 → Q21a code library.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,29	Coverage: 0,57
R79.	Q2.AdaptivePresentation.Satisfying.1 → Q2.Adaption of Content.Satisfying.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R80.	Q2.AdaptivePresentation.Satisfying.1 → Q3b improved interfaces.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R81.	Q2.AdaptivePresentation.Satisfying.1 → Q7.10			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R82.	Q2.AdaptivePresentation.Satisfying.1 → Q8.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R83.	Q2.AdaptivePresentation.Satisfying.1 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R84.	Q2.AdaptivePresentation.Satisfying.2 → Q2.Adaption of Content.Satisfying.2			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R85.	Q2.AdaptivePresentation.Satisfying.2 → Q21b data.1			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R86.	Q2.AdaptivePresentation.Satisfying.3 → Q15 Ontologies.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R87.	Q2.AdaptivePresentation.Satisfying.3 → Q16.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R88.	Q2.AdaptivePresentation.Satisfying.3 → Q19 adaptation of content.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R89.	Q2.AdaptivePresentation.Satisfying.3 → Q21a code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R90.	Q2.AdaptivePresentation.Satisfying.3 → Q21b algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R91.	Q2.AdaptivePresentation.Satisfying.3 → Q21b code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40

R92.	Q2.Adaption of Content.Valuable.1 → Q2.Adaption of Content.Satisfying.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R93.	Q2.Adaption of Content.Valuable.1 → Q11b lack of best practices.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R94.	Q2.Adaption of Content.Valuable.1 → Q19 adaptation of content.2			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R95.	Q2.Adaption of Content.Valuable.3 → Q2.Adaption of Content.Satisfying.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R96.	Q2.Adaption of Content.Valuable.3 → Q7.10			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R97.	Q2.Adaption of Content.Valuable.3 → Q19 user modeling.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R98.	Q2.Adaption of Content.Valuable.3 → Q19 adaptation of content.2			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R99.	Q2.Adaption of Content.Valuable.3 → Q21a algorithm.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R100.	Q2.Adaption of Content.Valuable.4 → Q6 slow adjustment.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R101.	Q2.Adaption of Content.Valuable.4 → Q11a return on investment too low.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R102.	Q2.Adaption of Content.Valuable.4 → Q19 user event detection.5			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R103.	Q2.Adaption of Content.Valuable.4 → Q19 user modeling ontology.4			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,29	Coverage: 0,71
R104.	Q2.Adaption of Content.Valuable.5 → Q19 adaptive presentation.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R105.	Q2.Adaption of Content.Valuable.5 → Q19 adaptation of content.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R106.	Q2.Adaption of Content.Satisfying.1 → Q5.4			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R107.	Q2.Adaption of Content.Satisfying.1 → Q19 recommendations.4			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43

R108.	Q2.Adaption of Content.Satisfying.2 → Q3b improved interaction.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R109.	Q2.Adaption of Content.Satisfying.2 → Q6 privacy concerns of the users.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R110.	Q2.Adaption of Content.Satisfying.2 → Q15 other XML interfaces.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,33	Coverage: 0,62
R111.	Q2.Adaption of Content.Satisfying.2 → Q16.5			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R112.	Q2.Adaption of Content.Satisfying.2 → Q19 navigation support.4			
	Confidence: 0,88	Support: 0,29	#Y(Y.b)/#Y 0,50	Coverage: 0,58
R113.	Q2.Adaption of Content.Satisfying.3 → Q7.10			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R114.	Q2.Adaption of Content.Satisfying.3 → Q19 user modeling.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R115.	Q2.Adaption of Content.Satisfying.3 → Q19 adaptation of content.2			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R116.	Q2.Adaption of Content.Satisfying.3 → Q21a algorithm.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R117.	Q2.Adaption of Content.Satisfying.4 → Q7.6			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R118.	Q2.Adaption of Content.Satisfying.4 → Q19 adaptation of content.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R119.	Q3b time saving.1 → Q21a web service.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R120.	Q3b time saving.1 → Q21b web service.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R121.	Q3b better feedback from the system.1 → Q6 slow adjustment.1			
	Confidence: 0,67	Support: 0,33	#Y(Y.b)/#Y 0,46	Coverage: 0,73
R122.	Q3b improved interfaces.1 → Q7.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R123.	Q3b improved interfaces.1 → Q9.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50

R124.	Q3b improved interfaces.1 → Q14.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R125.	Q3b improved interfaces.1 → Q17.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R126.	Q3b improved interfaces.1 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R127.	Q3b improved interfaces.1 → Q19 user modeling.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R128.	Q3b improved interfaces.1 → Q19 device adaptation.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R129.	Q3b improved interfaces.1 → Q19 adaptive presentation.3			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R130.	Q3b improved interaction.1 → Q6 privacy concerns of the users.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,42	Coverage: 0,70
R131.	Q4.2 → Q6 missing controllability.1			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,42	Coverage: 0,70
R132.	Q4.2 → Q10.10			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,50	Coverage: 0,58
R133.	Q4.2 → Q19 navigation support.4			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,50	Coverage: 0,58
R134.	Q4.2 → Q21b web service.1			
	Confidence: 0,67	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R135.	Q4.3 → Q5.4			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R136.	Q4.3 → Q21a code library.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R137.	Q4.4 → Q19 user modeling ontology.4			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R138.	Q4.4 → Q19 adaptive presentation.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R139.	Q4.4 → Q19 adaptation of content.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50

R140.	Q5.4 → Q21b code library.1			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R141.	Q6 functionality unclear.1 → Q21b code library.1			
	Confidence: 0,64	Support: 0,29	#Y(Y.b)/#Y 0,42	Coverage: 0,70
R142.	Q6 missing transparency.1 → Q11b lack of best practices / common approaches.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R143.	Q6 missing transparency.1 → Q16.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R144.	Q7.3 → Q8.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R145.	Q7.3 → Q11b lack of best practices / common approaches.1			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,25	Coverage: 0,50
R146.	Q7.3 → Q13.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R147.	Q7.3 → Q14.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R148.	Q7.3 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R149.	Q7.3 → Q19 adaptive presentation.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R150.	Q7.3 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R151.	Q7.4 → Q9.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R152.	Q7.4 → Q19 adaptation of content.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R153.	Q7.6 → Q19 adaptation of content.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R154.	Q7.10 → Q11b lack of libraries/tools.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50
R155.	Q7.10 → Q15 other XML interfaces.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50

R156.	Q7.12 → Q8.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R157.	Q8.3 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R158.	Q8.4 → Q11b lack of best practices / common approaches.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R159.	Q8.4 → Q19 adaptation of content.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R160.	Q8.5 → Q19 user modeling.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R161.	Q8.5 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R162.	Q8.10 → Q15 Semantic Web Services.1			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R163.	Q9.2 → Q19 user event detection.3			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,25	Coverage: 0,50
R164.	Q9.5 → Q16.4			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R165.	Q9.7 → Q19 user modeling ontology.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R166.	Q9.7 → Q21b algorithm.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R167.	Q9.7 → Q21b code template.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R168.	Q9.10 → Q11b lack of libraries/tools.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R169.	Q9.10 → Q19 user modeling.4			
	Confidence: 1,00	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R170.	Q9.10 → Q19 adaptive presentation.3			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R171.	Q10.8 → Q13.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40

R172.	Q10.8 → Q14.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R173.	Q10.8 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R174.	Q10.8 → Q19 user modeling.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R175.	Q10.8 → Q19 adaptation of content.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R176.	Q10.9 → Q11a lack of reusable components.1			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R177.	Q10.9 → Q16.4			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,25	Coverage: 0,50
R178.	Q10.10 → Q16.5			
	Confidence: 0,67	Support: 0,33	#Y(Y.b)/#Y 0,46	Coverage: 0,73
R179.	Q11a lack of results/effects.1 → Q21a code library.1			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,29	Coverage: 0,71
R180.	Q11a lack of results/effects.1 → Q21b code library.1			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R181.	Q11a uncontrollable system behavior.1 → Q14.5			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R182.	Q11b lack of libraries/tools.1 → Q19 user event detection.5			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R183.	Q11b lack of libraries/tools.1 → Q19 navigation support.4			
	Confidence: 0,88	Support: 0,29	#Y(Y.b)/#Y 0,50	Coverage: 0,58
R184.	Q11b lack of libraries/tools.1 → Q21a web service.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R185.	Q11b lack of libraries/tools.1 → Q21b code library.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R186.	Q11b lack of libraries/tools.1 → Q21b web service.1			
	Confidence: 0,75	Support: 0,25	#Y(Y.b)/#Y 0,46	Coverage: 0,55
R187.	Q11b implementation effort too high.1 → Q21b code library.1			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50

R188.	Q11b results are hard to control.1 → Q14.5			
	Confidence: 0,69	Support: 0,46	#Y(Y.b)/#Y 0,46	Coverage: 1,00
R189.	Q11b lack of best practices / common approaches.1 → Q19 user modeling.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R190.	Q13.5 → Q14.5			
	Confidence: 0,67	Support: 0,33	#Y(Y.b)/#Y 0,46	Coverage: 0,73
R191.	Q14.2 → Q19 user modeling ontology.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R192.	Q14.2 → Q21b algorithm.1			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R193.	Q14.3 → Q17.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R194.	Q14.3 → Q19 user event detection.3			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,25	Coverage: 0,50
R195.	Q14.3 → Q19 user modeling.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R196.	Q14.3 → Q19 device adaptation.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R197.	Q14.4 → Q17.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,21	Coverage: 0,80
R198.	Q14.5 → Q16.5			
	Confidence: 0,73	Support: 0,33	#Y(Y.b)/#Y 0,46	Coverage: 0,73
R199.	Q14.5 → Q19 user event detection.5			
	Confidence: 0,64	Support: 0,29	#Y(Y.b)/#Y 0,42	Coverage: 0,70
R200.	Q15 Semantic Web Services.1 → Q21a data.1			
	Confidence: 0,67	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R201.	Q15 RSS/RDF interfaces.1 → Q21b web service.1			
	Confidence: 0,67	Support: 0,33	#Y(Y.b)/#Y 0,46	Coverage: 0,73
R202.	Q15 Ontologies.1 → Q19 user modeling ontology.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R203.	Q15 Ontologies.1 → Q19 navigation support.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67

R204.	Q15 Ontologies.1 → Q19 adaptation of content.4			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R205.	Q15 Ontologies.1 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R206.	Q15 Ontologies.1 → Q21a code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R207.	Q15 Ontologies.1 → Q21b algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R208.	Q15 Ontologies.1 → Q21b code template.1			
	Confidence: 0,78	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R209.	Q15 other XML interfaces.1 → Q19 device adaptation.4			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,29	Coverage: 0,71
R210.	Q16.2 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R211.	Q16.2 → Q19 navigation support.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,08	Coverage: 1,00
R212.	Q16.2 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R213.	Q16.3 → Q19 adaptation of content.2			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R214.	Q17.2 → Q19 user event detection.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R215.	Q17.2 → Q19 user modeling.3			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R216.	Q17.2 → Q19 device adaptation.2			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,12	Coverage: 1,00
R217.	Q17.2 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R218.	Q17.3 → Q19 user modeling ontology.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R219.	Q17.4 → Q19 adaptive presentation.3			
	Confidence: 0,80	Support: 0,17	#Y(Y.b)/#Y 0,33	Coverage: 0,50

R220.	Q19 user event detection.3 → Q19 user modeling.4			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,38	Coverage: 0,44
R221.	Q19 user event detection.4 → Q19 adaptation of content.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R222.	Q19 user modeling.3 → Q19 adaptive presentation.3			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R223.	Q19 user modeling.3 → Q21a code library.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,29	Coverage: 0,43
R224.	Q19 user modeling ontology.1 → Q19 recommendations.3			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R225.	Q19 user modeling ontology.2 → Q19 adaptation of content.3			
	Confidence: 1,00	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R226.	Q19 user modeling ontology.3 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,17	#Y(Y.b)/#Y 0,25	Coverage: 0,67
R227.	Q19 user modeling ontology.4 → Q19 device adaptation.3			
	Confidence: 0,71	Support: 0,21	#Y(Y.b)/#Y 0,33	Coverage: 0,62
R228.	Q19 recommendations.2 → Q19 device adaptation.3			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,33	Coverage: 0,38
R229.	Q19 device adaptation.2 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R230.	Q19 device adaptation.3 → Q21a data.1			
	Confidence: 0,62	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R231.	Q19 navigation support.2 → Q21a algorithm.1			
	Confidence: 1,00	Support: 0,12	#Y(Y.b)/#Y 0,25	Coverage: 0,50
R232.	Q19 navigation support.2 → Q21a code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R233.	Q19 navigation support.2 → Q21b code template.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,21	Coverage: 0,40
R234.	Q19 navigation support.4 → Q21a web service.1			
	Confidence: 0,75	Support: 0,38	#Y(Y.b)/#Y 0,46	Coverage: 0,82
R235.	Q19 navigation support.4 → Q21b web service.1			
	Confidence: 0,75	Support: 0,38	#Y(Y.b)/#Y 0,46	Coverage: 0,82

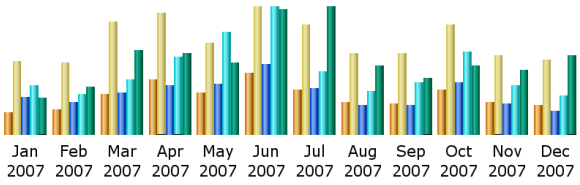
R236.	Q19 navigation support.5 → Q19 adaptive presentation.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,12	Coverage: 0,67
R237.	Q19 navigation support.5 → Q19 adaptation of content.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R238.	Q19 adaptive presentation.5 → Q19 adaptation of content.5			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R239.	Q19 adaptive presentation.5 → Q21a algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,25	Coverage: 0,33
R240.	Q19 adaptive presentation.5 → Q21b algorithm.1			
	Confidence: 0,67	Support: 0,08	#Y(Y.b)/#Y 0,17	Coverage: 0,50
R241.	Q19 adaptation of content.2 → Q21b code library.1			
	Confidence: 0,83	Support: 0,21	#Y(Y.b)/#Y 0,42	Coverage: 0,50
R242.	Q21a data.1 → Q21a web service.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R243.	Q21a data.1 → Q21b data.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,38	Coverage: 0,78
R244.	Q21a data.1 → Q21b web service.1			
	Confidence: 0,70	Support: 0,29	#Y(Y.b)/#Y 0,46	Coverage: 0,64
R245.	Q21a code template.1 → Q21b code template.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,21	Coverage: 0,60
R246.	Q21a code library.1 → Q21b code library.1			
	Confidence: 0,86	Support: 0,25	#Y(Y.b)/#Y 0,42	Coverage: 0,60
R247.	Q21a web service.1 → Q21b web service.1			
	Confidence: 0,82	Support: 0,38	#Y(Y.b)/#Y 0,46	Coverage: 0,82
R248.	Q21b algorithm.1 → Q21b code template.1			
	Confidence: 0,75	Support: 0,12	#Y(Y.b)/#Y 0,21	Coverage: 0,60

Appendix D

Web Usage Statistics

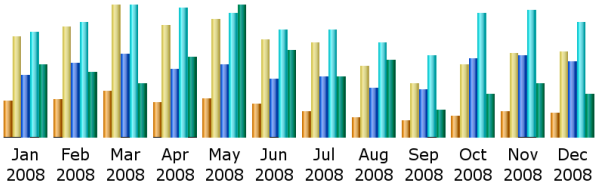
The following diagrams depict the access statistics for the website of the Personal Reader framework[1] created by AWStats[2]

[1] http://www.personal-reader.de
[2] http://awstats.sourceforge.net/



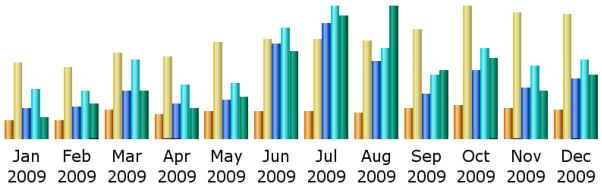
Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2007	1291	4227	11697	15401	635.31 MB
Feb 2007	1411	4110	10082	12726	825.50 MB
Mar 2007	2309	6513	13018	16830	1.44 GB
Apr 2007	3196	6999	15165	24054	1.41 GB
May 2007	2418	5340	15535	31847	1.24 GB
Jun 2007	3577	7394	22050	39785	2.15 GB
Jul 2007	2605	6362	14265	19866	2.19 GB
Aug 2007	1819	4735	9004	13425	1.19 GB
Sep 2007	1755	4732	8995	16126	977.72 MB
Oct 2007	2590	6353	16054	25848	1.17 GB
Nov 2007	1817	4538	9309	15463	1.10 GB
Dec 2007	1653	4318	7314	12322	1.35 GB
Total	26441	65621	152488	243693	15.63 GB

Figure D.1: Web usage statistics of the Personal Reader website from 2007, evaluated at 09th July 2010.



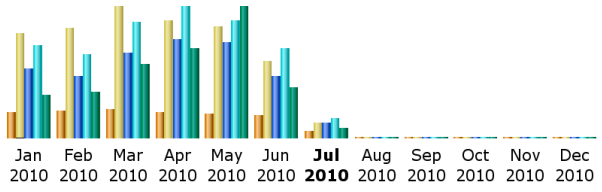
Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2008	1495	4150	7189	12117	1.18 GB
Feb 2008	1556	4532	8557	13279	1.07 GB
Mar 2008	1881	5437	9501	15265	892.19 MB
Apr 2008	1428	4633	7839	14933	1.31 GB
May 2008	1594	4840	8452	14306	2.15 GB
Jun 2008	1378	4000	6719	12543	1.42 GB
Jul 2008	1059	3906	7122	12447	1012.56 MB
Aug 2008	804	2924	5719	10940	1.26 GB
Sep 2008	686	2194	5474	9473	465.26 MB
Oct 2008	891	2987	9081	14262	717.24 MB
Nov 2008	1038	3502	9461	14664	884.62 MB
Dec 2008	996	3535	8804	13367	729.39 MB
Total	14806	46640	93918	157596	12.98 GB

Figure D.2: Web usage statistics of the Personal Reader website from 2008, evaluated at 09th July 2010.



Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2009	785	3260	7044	11562	718.87 MB
Feb 2009	785	3075	7354	11245	1.09 GB
Mar 2009	1226	3651	11062	18372	1.53 GB
Apr 2009	1031	3518	8197	12456	988.37 MB
May 2009	1154	4120	8913	12753	1.32 GB
Jun 2009	1133	4237	21947	25864	2.80 GB
Jul 2009	1181	4273	26766	30769	3.90 GB
Aug 2009	1098	4192	17903	20935	4.22 GB
Sep 2009	1260	4695	10373	14984	2.17 GB
Oct 2009	1388	5656	15883	20925	2.57 GB
Nov 2009	1285	5401	11842	17021	1.51 GB
Dec 2009	1236	5338	13810	18283	2.02 GB
Total	13562	51416	161094	215169	24.79 GB

Figure D.3: Web usage statistics of the Personal Reader website from 2009, evaluated at 09th July 2010.



Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2010	1541	6265	16816	22426	3.65 GB
Feb 2010	1670	6677	15143	20618	3.92 GB
Mar 2010	1763	7936	20625	28102	6.28 GB
Apr 2010	1543	7078	24090	32001	7.70 GB
May 2010	1462	6771	23178	28686	11.18 GB
Jun 2010	1407	4661	15260	21914	4.26 GB
Jul 2010	434	932	3675	4873	820.14 MB
Aug 2010	0	0	0	0	0
Sep 2010	0	0	0	0	0
Oct 2010	0	0	0	0	0
Nov 2010	0	0	0	0	0
Dec 2010	0	0	0	0	0
Total	9820	40320	118787	158620	37.79 GB

Figure D.4: Web usage statistics of the Personal Reader website from 2010, evaluated at 09th July 2010.

Bibliography

[Abel et al., 2005] Abel, F., Baumgartner, R., Brooks, A., Enzi, C., Gottlob, G., Henze, N., Herzog, M., Kriesell, M., Nejdl, W., and Tomaschewski, K. (2005). The personal publication reader, semantic web challenge 2005. In *4th International Semantic Web Conference*.

[Abel et al., 2006] Abel, F., Brunkhorst, I., Henze, N., Krause, D., Mushtaq, K., Nasirifard, P., and Tomaschweski, K. (2006). Personal reader agent: Personalized access to configurable web services. Technical report, Distributed Systems Institute, Semantic Web Group, University of Hannover.

[Abel et al., 2008] Abel, F., Henze, N., Krause, D., and Plappert, D. (2008). User modeling and user profile exchange for semantic web applications. In *16th Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS 2008), Würzburg, Germany*.

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.

[Aduna, 2005] Aduna, B. (2005). The SeRQL query language (revision 1.2). `http://www.openrdf.org/doc/sesame/users/ch06.html`.

[Agrawal et al., 1993] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA. ACM.

[Akkiraju and et. al., 2005] Akkiraju, R. and et. al. (2005). Web Service Semantics - WSDL-S. In *W3C Member Submission*.

[B. Smyth, 2002] B. Smyth, P. C. (2002). Personalized adaptive navigation for mobile portals. In *Proceedings of ECAI 2002*.

[Bailey et al., 2005] Bailey, J., Bry, F., Eckert, M., and Patranjan, P.-L. (2005). Flavours of xchange, a rule-based reactive language for the (semantic) web. In Adi, A., Stoutenburg, S., and Tabet, S., editors, *Rules*

and Rule Markup Languages for the Semantic Web, volume 3791 of *Lecture Notes in Computer Science*, pages 187–192. Springer Berlin / Heidelberg.

[Balabanović and Shoham, 1997] Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72.

[Baldoni et al., 2006] Baldoni, M., Baroglio, C., Brunkhorst, I., Henze, N., Marengo, E., and Patti, V. (2006). A Personalization Service for Curriculum Planning. In Herder, E. and Heckmann, D., editors, *Proc. of the 14th Workshop on Adaptivity and User Modeling in Interactive Systems, ABIS 2006*, pages 17–20, Hildesheim, Germany.

[Baldoni and Marengo, 2007] Baldoni, M. and Marengo, E. (2007). Curriculum Model Checking: Declarative Representation and Verification of Properties. In Duval, E. and Klamma, R., editors, *Proc. of EC-TEL 2007 - Second European Conference on Technology Enhanced Learning*, LNCS. Springer.

[Balke and Wagner, 2003] Balke, W.-T. and Wagner, M. (2003). Cooperative Discovery for User-Centered Web Service Provisioning. In *ICWS*, pages 191–197.

[Baumgartner et al., 2005] Baumgartner, R., Henze, N., and Herzog, M. (2005). The personal publication reader: Illustrating web data extraction, personalization and reasoning for the semantic web. In Gómez-Pérez, A. and Euzenat, J., editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 515–530. Springer.

[Bellur and Kulkarni, 2007] Bellur, U. and Kulkarni, R. (2007). Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. In *ICWS*, pages 86–93.

[Berkovsky et al., 2006] Berkovsky, S., Kuflik, T., and Ricci, F. (2006). Cross-technique mediation of user models. In *4th Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 21–30.

[Berkovsky et al., 2007] Berkovsky, S., Kuflik, T., and Ricci, F. (2007). Cross-domain mediation in collaborative filtering. In [Conati et al., 2007], pages 355–359.

[Berners-Lee, 2000] Berners-Lee, T. (2000). Semantic web - xml2000.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American Magazine*.

[Berstel et al., 2007] Berstel, B., Bry, F., Eckert, M., and lavinia Patranjan, P. (2007). Reactive rules on the web. In *In Reasoning Web, Int. Summer School, LNCS*. Springer.

[Bizer and Oldakowski, 2004] Bizer, C. and Oldakowski, R. (2004). Using context- and content-based trust policies on the Semantic Web. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 228–229, New York, NY, USA. ACM Press.

[Bizer et al., 2009] Bizer, C., Volz, J., Kobilarov, G., and Gaedke, M. (2009). Silk - a link discovery framework for the web of data. In *World Wide Web Conference 2009*.

[Blom, 2000] Blom, J. (2000). Personalization: a taxonomy. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 313–314, New York, NY, USA. ACM.

[Bodendorf, 2009] Bodendorf, F. (2009). Controlling communication processes in e-learning scenarios. In *Web-based Education*, Phuket, Thailand.

[Bonatti and Olmedilla, 2007] Bonatti, P. and Olmedilla, D. (2007). Rule-based policy representation and reasoning for the semantic web. In Antoniou, G., Aßmann, U., Baroglio, C., Decker, S., Henze, N., Patranjan, P.-L., and Tolksdorf, R., editors, *Reasoning Web*, volume 4636 of *Lecture Notes in Computer Science*, pages 240–268. Springer Berlin / Heidelberg.

[Bonatti and Olmedilla, 2005a] Bonatti, P. A. and Olmedilla, D. (2005a). Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden. IEEE Computer Society.

[Bonatti and Olmedilla, 2005b] Bonatti, P. A. and Olmedilla, D. (2005b). Policy language specification. Technical report, Working Group I2, EU NoE REWERSE.

[Bra and Calvi, 1998] Bra, P. D. and Calvi, L. (1998). Aha! an open adaptive hypermedia architecture. *The New Review of Hypermedia and Multimedia*, 4:115–140.

[Bradshaw and Hinton, 2004] Bradshaw, J. and Hinton, L. (2004). Benefits of an online discussion list in a traditional distance education course. In *Turkish Online Journal of Distance Education*, volume 5(3).

[Broekstra and Kampman, 2004] Broekstra, J. and Kampman, A. (2004). SeRQL: An RDF query and transformation language. *Semantic Web and Peer-to-Peer*.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *ISWC*, pages 54–68.

[Brusilovsky, 1996] Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6:87–129. 10.1007/BF00143964.

[Brusilovsky and Henze, 2007] Brusilovsky, P. and Henze, N. (2007). Open corpus adaptive educational hypermedia. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 671–696. Springer.

[Brusilovsky et al., 2005a] Brusilovsky, P., Sosnovsky, S., and Shcherbinina, O. (2005a). User modeling in a distributed e-learning architecture. In Ardissono, L., Brna, P., and Mitrovic, A., editors, *User Modeling 2005*, volume 3538 of *Lecture Notes in Computer Science*, pages 387–391. Springer Berlin / Heidelberg.

[Brusilovsky et al., 2005b] Brusilovsky, P., Sosnovsky, S., and Yudelson, M. (2005b). Ontology-based framework for user model interoperability in distributed learning environments. In *E-Learn 2005*.

[Burke, 2002] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

[Burstein and et. al., 2004] Burstein, M. and et. al. (2004). OWL-S: Semantic Markup for Web Services. In *W3C Member Submission*.

[Bush, 1945] Bush, V. (1945). As We May Think. *Atlantic Monthly*, 176(1):641–649.

[Calado et al., 2009] Calado, I., Barros, H., and Bittencourt, I. I. (2009). An approach for semantic web services automatic discovery and composition with similarity metrics. In *Symposium on Applied Computing - SAC ACM, in Agent-Oriented Software Engineering Methodologies and Systems Track*.

[Cardoso, 2006] Cardoso, J. (2006). Discovering Semantic Web Services with and without a Common Ontology Commitment. In *IEEE SCW*, pages 183–190.

[Carroll et al., 2005] Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA. ACM Press.

[Clark et al., 2008] Clark, K. G., Feigenbaum, L., and Torres, E. (2008). SPARQL Protocol for RDF. W3c recommendation, W3C.

[Colgrave et al., 2004] Colgrave, J., Akkiraju, R., and Goodwin, R. (2004). External Matching in UDDI. In *ICWS*, page 226.

[Conati et al., 2007] Conati, C., McCoy, K. F., and Paliouras, G., editors (2007). *User Modeling 2007, 11th International Conference, UM 2007, Corfu, Greece, June 25-29, 2007, Proceedings*, volume 4511 of *Lecture Notes in Computer Science*. Springer.

[Conklin, 1987] Conklin, J. (1987). Hypertext: An introduction and survey. *Computer*, 20:17–41.

[Constantinescu et al., 2005] Constantinescu, I., Binder, W., and Faltings, B. (2005). Flexible and Efficient Matchmaking and Ranking in Service Directories. In *ICWS*, pages 5–12.

[Cozzi et al., 2006] Cozzi, A., Farrell, S., Lau, T., Smith, B. A., Drews, C., Lin, J., Stachel, B., and Moran, T. P. (2006). Activity management as a web service. *IBM Systems Journal*, 45(4):695–712.

[Dietzold and Auer, 2006] Dietzold, S. and Auer, S. (2006). Access control on rdf triple stores from a semantic wiki perspective. In *Scripting for the Semantic Web Workshop at 3rd European Semantic Web Conference (ESWC)*.

[Dolog et al., 2004] Dolog, P., Henze, N., Nejdl, W., and Sintek, M. (2004). Personalization in distributed e-learning environments. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 170–179, New York, NY, USA. ACM.

[Dong et al., 2004] Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E., and Zhang, J. (2004). Similarity Search for Web Services. In *VLDB*, pages 372–383.

[Finin and Drager, 1986] Finin, T. and Drager, D. (1986). Gums: a general user modeling system. In *HLT '86: Proceedings of the workshop on Strategic computing natural language*, pages 224–230, Morristown, NJ, USA. Association for Computational Linguistics.

[Fink, 2004] Fink, J. (2004). *User Modeling Servers: Requirements, Design, and Evaluation*. IOS Press, Inc.

[Fu et al., 2000] Fu, X., Budzik, J., and Hammond, K. J. (2000). Mining navigation history for recommendation. In *In Intelligent User Interfaces*, pages 106–112. ACM Press.

[Gao et al., 2005] Gao, Z., Qu, Y., Zhai, Y., and Deng, J. (2005). Dynamicview: Distribution, evolution and visualization of research areas in computer science. In *Proceeding of International Semantic Web Conference*.

[Gavriloaie et al., 2004] Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K. E., and Winslett, M. (2004). No registration needed: How to use declarative policies and negotiation to access sensitive resources on the Semantic Web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume

3053 of *Lecture Notes in Computer Science*, pages 342–356, Heraklion, Crete, Greece. Springer.

[Hartmann and Sure, 2004] Hartmann, J. and Sure, Y. (2004). An infrastructure for scalable, reliable semantic portals. *IEEE Intelligent Systems*, 19(3):58–65.

[Hau et al., 2005] Hau, J., Lee, W., and Darlington, J. (2005). A Semantic Similarity Measure for Semantic Web Services. In *Web Service Semantics Workshop at WWW*.

[Heckmann, 2005] Heckmann, D. (2005). *Ubiquitous User Modeling*. PhD thesis, Department of Computer Science, Saarland University, Germany.

[Heckmann et al., 2005] Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., and von Wilamowitz-Moellendorff, M. (2005). Gumo - the general user model ontology. In *Proceedings of the 10th International Conference on User Modeling*, pages 428–432, Edinburgh, UK. LNAI 3538: Springer, Berlin Heidelberg.

[Helic et al., 2004] Helic, D., Maurer, H., and Scerbakov, N. (2004). Discussion forums as learning resources in web-based education. *Advanced technology for learning*, 1(1):8–15.

[Henze and Krause, 2006] Henze, N. and Krause, D. (2006). Personalized access to web services in the semantic web. In *SWUI 2006 - 3rd International Semantic Web User Interaction Workshop*, Athens, Georgia, USA.

[Herlocker et al., 1999] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Theoretical Models, pages 230–237.

[Jain and Farkas, 2006] Jain, A. and Farkas, C. (2006). Secure resource description framework: an access control model. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 121–129, New York, NY, USA. ACM Press.

[Jameson, 2003] Jameson, A. (2003). Adaptive interfaces and agents. In Jacko, J. A. and Sears, A., editors, *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pages 305–330. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA.

[Kagal et al., 2003] Kagal, L., Finin, T. W., and Joshi, A. (2003). A policy language for a pervasive computing environment. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, Lake Como, Italy. IEEE Computer Society.

[Kärger et al., 2007] Kärger, P., Abel, F., Herder, E., Olmedilla, D., and Siberski, W. (2007). Exploiting preference queries for searching learning resources. In *EC-TEL*, pages 143–157.

[Kaufer and Klusch, 2006] Kaufer, F. and Klusch, M. (2006). WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In *ECOWS*, pages 161–170.

[Kay et al., 2002] Kay, J., Kummerfeld, B., and Lauder, P. (2002). Personis: a server for user models. In Bra, P. D., Brusilovsky, P., and Conejo, R., editors, *Proceedings of AH 2002, 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *Lecture Notes in Computer Science*, pages 203–212. Springer-Verlag (Berlin, Heidelberg).

[Kelly et al., 2002] Kelly, S. U., Sung, C., and Farnham, S. (2002). Designing for improved social responsibility, user participation and content in on-line communities. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 391–398, New York, NY, USA. ACM.

[Kießling and Hafenrichter, 2002] Kießling, W. and Hafenrichter, B. (2002). Optimizing Preference Queries for Personalized Web Services. In *Communications, Internet, and Information Technology*, pages 461–466.

[Klusch et al., 2006] Klusch, M., Fries, B., and Sycara, K. P. (2006). Automated Semantic Web service discovery with OWLS-MX. In *AAMAS*, pages 915–922.

[Klusch and Zhing, 2008] Klusch, M. and Zhing, X. (2008). Deployed semantic services for the common user of the web: A reality check. pages 347 –353.

[Kobsa, 1990] Kobsa, A. (1990). Modeling the user's conceptual knowledge in bgp-ms, a user modeling shell system. *Comput. Intell.*, 6(4):193–208.

[Kobsa, 2001] Kobsa, A. (2001). Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63.

[Kobsa, 2007] Kobsa, A. (2007). Privacy-enhanced web personalization. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 628–670. Springer.

[Kobsa and Pohl, 1995] Kobsa, A. and Pohl, W. (1995). The user modeling shell system bgp-ms.

[Kossmann et al., 2002] Kossmann, D., Ramsak, F., and Rost, S. (2002). Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment.

[Lamparter et al., 2007] Lamparter, S., Ankolekar, A., Studer, R., and Grimm, S. (2007). Preference-based selection of highly configurable web services. In *WWW*, pages 1013–1022.

[Lausen et al., 2005] Lausen, H., Polleres, A., , and (eds.), D. R. (2005). Web service modeling ontology (wsmo). In *W3C Member Submission*.

[Lee, 2001] Lee, W. S. (2001). Collaborative learning for recommender systems.

[Li and Horrocks, 2003] Li, L. and Horrocks, I. (2003). A Software Framework for Matchmaking based on Semantic Web Technology. In *WWW*, pages 331–339.

[Liang et al., 2007] Liang, T.-P., Lai, H.-J., and Ku, Y.-C. (2007). Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *J. Manage. Inf. Syst.*, 23:45–70.

[Licklider et al., 1968] Licklider, Robert, Licklider, J. C. R., and Taylor, R. W. (1968). The computer as a communication device. *Science and Technology*, 76:21–31.

[Lin et al., 2002] Lin, W., Alvarez, S. A., and Ruiz, C. (2002). Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery*, 6:83–105. 10.1023/A:1013284820704.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *An Introduction to Information Retrieval*. Cambridge University Press.

[Mehta and Hofmann, 2008] Mehta, B. and Hofmann, T. (2008). A survey of attack-resistant collaborative filtering algorithms. *IEEE Data Eng. Bull.*, 31(2):14–22.

[Montaner et al., 2003] Montaner, M., López, B., and De La Rosa, J. L. (2003). A taxonomy of recommender agents on theinternet. *Artif. Intell. Rev.*, 19(4):285–330.

[Odeh and Ketaneh, 2007] Odeh, S. and Ketaneh, E. (2007). Collaborative working e-learning environments supported by rule-based e-tutor. In *International Journal of Online Engineering*.

[OWL-S/UDDIM, 2005] OWL-S/UDDIM (2005). Owl-s/uddim. http://projects.semwebcentral.org/projects/owl-s-uddi-mm/. Last access on August of 2008.

[Paolucci et al., 2002] Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic Matching of Web Services Capabilities. In *ISWC*, pages 333–347.

[Perrey and Lycett, 2003] Perrey, R. and Lycett, M. (2003). Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, pages 116 – 119.

[Polleres, 2007] Polleres, A. (2007). From SPARQL to rules (and back). In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 787–796, New York, NY, USA. ACM Press.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

[Quan and Karger, 2004] Quan, D. and Karger, D. (2004). How to make a semantic web browser. In *Proceedings of the 13th International Conference on World Wide Web*, pages 255–265.

[RDQL, 2005] RDQL (2005). RDQL - query language for RDF, Jena. `http://jena.sourceforge.net/RDQL/`.

[Reddivari et al., 2005] Reddivari, P., Finin, T., and Joshi, A. (2005). Policy based access control for a RDF store. In *Proceedings of the Policy Management for the Web Workshop*, A WWW 2005 Workshop, pages 78–83. W3C.

[Rich, 1979] Rich, E. (1979). User modeling via stereotypes. *Cognitive Science*, 3:329–354.

[Rossi et al., 2001] Rossi, G., Schwabe, D., and Guimarães, R. (2001). Designing personalized web applications. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 275–284, New York, NY, USA. ACM.

[Schafer et al., 1999] Schafer, J. B., Konstan, J., and Riedi, J. (1999). Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA. ACM.

[Schein et al., 2002] Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, New York, NY, USA. ACM.

[Schwier and Balbar, 2002] Schwier, R. A. and Balbar, S. (2002). The interplay of content and community in synchronous and asynchronous communication: Virtual communication in a graduate seminar. *Canadian Journal of Learning and Technology*, 28(2).

[Seaborne and Manjunath, 2008] Seaborne, A. and Manjunath, G. (2008). Sparql/update - a language for updating rdf graphs. Technical report, Hewlett-Packard Development Company.

[Shadbolt et al., 2004] Shadbolt, N. R., Gibbins, N., Glaser, H., Harris, S., and schraefel, m. c. (2004). CS AKTive space or how we stopped worrying and learned to love the semantic web. *IEEE Intelligent Systems*, 19(3).

[Shardanand and Maes, 1995] Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217.

[Sieg et al., 2004] Sieg, A., Mobasher, B., and Burke, R. (2004). Inferring users information context: Integrating user profiles and concept hierarchies. In *Proceedings of the 2004 Meeting of the International Federation of Classification Societies*, Chicago, IL, USA.

[Skoutas et al., 2008] Skoutas, D., Sacharidis, D., Kantere, V., and Sellis, T. (2008). Efficient Semantic Web Service Discovery in Centralized and P2P Environments. In *ISWC*, pages 583–598.

[Skoutas et al., 2009] Skoutas, D., Sacharidis, D., Simitsis, A., Kantere, V., and Sellis, T. (2009). Top-k Dominant Web Services under Multi-criteria Matching. In *EDBT*, pages 898–909.

[Skoutas et al., 2007] Skoutas, D., Simitsis, A., and Sellis, T. K. (2007). A Ranking Mechanism for Semantic Web Service Discovery. In *IEEE SCW*, pages 41–48.

[Soo and Bonk, 1998] Soo, K.-S. and Bonk, C. J. (1998). Interaction: What does it mean in online distance education? *ED-MEDIA/ED-TELECOM 98 World Conference on Educational Multimedia and Hypermedia & World Conference on Educational Telecommunications*.

[Soonthornphisaj et al., 2006] Soonthornphisaj, N., Rojsattarat, E., and Yimngam, S. (2006). Smart e-learning using recommender system. In *International Conference on Intelligent Computing*.

[Srinivasan et al., 2004] Srinivasan, N., Paolucci, M., and Sycara, K. P. (2004). An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In *SWSWPC*, pages 96–110.

[Su and Khoshgoftaar, 2009] Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:2–2.

[Thomas, 2002] Thomas, M. (2002). Learning within incoherent structures: the space of online discussion forums. *Journal of Computer Assisted Learning*, 18:351–366.

[Tummarello et al., 2007] Tummarello, G., Oren, E., and Delbru, R. (2007). Sindice.com: Weaving the open linked data. In Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L. J. B., Golbeck, J., Mika, P., Maynard, D., Schreiber, G., and Cudr-Mauroux, P., editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 547–560, Berlin, Heidelberg. Springer Verlag.

[Uszok et al., 2003] Uszok, A., Bradshaw, J. M., Jeffers, R., Suri, N., Hayes, P. J., Breedy, M. R., Bunch, L., Johnson, M., Kulkarni, S., and Lott, J. (2003). KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY*, page 93.

[Webb et al., 2004] Webb, E., Jones, A., Barker, P., and van Schaik, P. (2004). Using e-learning dialogues in higher education. *Innovations in Education and Teaching International*, 41(1).

[Webster and Vassileva, 2006a] Webster, A. and Vassileva, J. (2006a). Linking in lurkers: The comtella discussion forum. In *Workshop on the Social Navigation and Community based Adaptation Technologies*.

[Webster and Vassileva, 2006b] Webster, A. and Vassileva, J. (2006b). Visualizing personal relations in online communities. In *AH*, pages 223–233.

[Weld et al., 2003] Weld, D. S., Anderson, C., Domingos, P., Etzioni, O., Gajos, K., Lau, T., and Wolf, S. (2003). Automatically personalizing user interfaces. In *In IJCAI03*, pages 1613–1619.

[Whitby et al., 2004] Whitby, A., Josang, A., and Indulska, J. (2004). Filtering Out Unfair Ratings in Bayesian Reputation Systems. In *AAMAS*.

[Xu et al., 2007] Xu, Z., Martin, P., Powley, W., and Zulkernine, F. (2007). Reputation-Enhanced QoS-based Web Services Discovery. In *ICWS*, pages 249–256.

[Yu et al., 2004] Yu, B., Singh, M. P., and Sycara, K. (2004). Developing trust in large-scale peer-to-peer systems. In *IEEE Symposium on Multi-Agent Security and Survivability*, pages 1–10.

[Yu et al., 2006] Yu, Z., Zhou, X., Hao, Y., and Gu, J. (2006). Tv program recommendation for multiple viewers based on user profile merging. *User Modeling and User-Adapted Interaction*, 16:63–82. 10.1007/s11257-006-9005-6.

[Yudelson et al., 2007] Yudelson, M., Brusilovsky, P., and Zadorozhny, V. (2007). A user modeling server for contemporary adaptive hypermedia: An evaluation of the push approach to evidence propagation. In [Conati et al., 2007], pages 27–36.

[Zaiane, 2002] Zaiane, O. (2002). Building a recommender agent for e-learning systems. In *Proceedings of the International Conference on Computers in Education*, pages 55–56.

[Zaslow, 2002] Zaslow, J. (2002). If tivo thinks you are gay, here's how to set it straight — amazon.com knows you, too, based on what you buy; why all the cartoons? *The Wall Street Journal*.

[Zhang and Chang, 2005] Zhang, F. and Chang, H.-Y. (2005). On a hybrid rule based recommender system. In *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 194–198, Washington, DC, USA. IEEE Computer Society.

List of Figures

List of Tables

Wissenschaftlicher Werdegang

Persönliche Angaben

Name	Daniel Krause
Geburtsdatum	20. Oktober 1981
Geburtsort	Hannover, Deutschland

Lebenslauf

2001	Abitur am Johannes-Kepler-Gymnasium, Garbsen
2002-2005	Studium der Angewandten Informatik an der Universität Hannover mit Abschluss Bachelor of Science
2005-2006	Studium der Informatik an der Universität Hannover mit Abschluss Master of Science
2006-2011	Wissenschaftlicher Mitarbeiter am Forschungszentrum L3S, Leibniz Universität Hannover