User Modeling and User Profile Exchange for Semantic Web Applications

Fabian Abel, Nicola Henze, Daniel Krause
Distributed Systems Institute, Semantic Web Group, Leibniz University of Hannover
Appelstraße 4, 30167 Hannover, Germany
{abel,henze,krause}@kbs.uni-hannover.de

Daniel Plappert
mindmatters GmbH & Co. KG
Neuer Kamp 30, 20357 Hamburg - St. Pauli, Germany
daniel.plappert@mindmatters.de

Abstract

The usage of applications, which are located on the Web, changes the usage of personalization massively: Users interact with most applications seldomly, often just once, but still can benefit from advantages of personalization. We present the User Modeling Service, a service to model a user across various applications. Our service is not restricted to a specific domain and can hence be integrated into arbitrary Web applications. To enable the reuse of user profile data from different applications, we adhere to well-known Semantic Web technologies. In addition, we developed an easy-to-use Web interface, which allows users to inspect, control, and modify their data. This Web interface is also used to create access policies, which allow users to specify on a fine-grained level which application or Web Service is allowed to access which part of the user profile.

1. Introduction

In today's Web users enjoy the benefits of personalization in different applications and for various purposes. However, all personalization techniques require sufficient descriptions of the characteristics of the user to deliver good results. There are two general strategies to gather information about the user. User data can either be received via direct, explicit input (e.g. questionnaires to determine an initial user profile) or it can be derived implicitly from the observations made when the user interacts with the applications.

While both approaches are accepted well by the users and proved to work, these approaches are inadequate for Web applications, because there is a vast amount of applications that only solve specific (possibly unusual) problems. Users access these applications only seldomly or just once. In such scenarios users, on the one hand, do not take the effort to invest time to fill a questionnaire and, on the other hand, the applications do not receive enough user feedback to model the user profile implicitly. There are solutions proposed for this problem which separate the user modeling components from the applications and create generic components. Other approaches try to exchange user profile data. The first type of applications have the problem that the user modeling system has to be aware of the application-specific vocabulary and knowledge, while the latter type of applications need a common language to be able to exchange data about users.

New issues arise as more people become aware of privacy issues regarding the storage, processing, and exchange of their personal data. In a multi-application personalization framework, users still shall be able to control which application should be able to access which part of the user profile and for which purpose (see [15]).

In this paper, we present the *User Modeling Service*, UMService for short, which is a central repository for storing, maintaining, and querying user profile data. It is embedded in the *Personal Reader Framework*[1], a Semantic Web framework for rapid development of highly flexible, Web Service based, and personalized Web applications. The Personal Reader approach addresses the issue of keeping data central while keeping central components independent of domain and application specific knowledge. Furthermore, we show how the access to RDF-based user profiles can be controlled by applying policies which can be defined via the Web interface of the UMService.

2. Related Work

The first approaches to separate user modeling from the application, like the GUMS [9] or the BGP-MS [16], man-

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

aged to maintain user profiles apart from the original application. However, they did not provide any mechanism to exchange or reuse user profile data.

Systems that allow to maintain user profiles and exchange user profile data like for example CUMULATE [18] implemented a broad domain knowledge and hence can be used only for one specific domain (in the CUMULATE case the E-Learning domain). Another system, which can be used domain independent is the Generalized User Modeling Ontology (GUMO) [11]. It is mainly a generic and extendable[2] ontology that allows to express various user profile statements. However, as the ontology grows – which is required to support new application domains – it becomes hard to ensure consistency and to find appropriate instances in the ontology to reuse.

Other generic approaches like Personis [14] and approaches for cross-domain and cross-technique user modeling [5, 4, 7] are based on a data level and do not describe the stored information in a semantic fashion. 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.

We have deliberately chosen a centralized component to model the user instead of a decentralized approach like [17]: Both approaches bear the same high-level problems like inconsistency, need for meta-data, data integration etc. However, in our opinion centralized user modeling lacks the low level problems of locating the needed data and duplication of code that is necessary for maintaining the user profiles.

3. A Distributed User Modeling Framework

The Personal Reader Framework [1, 12] (see Figure 1) enables a fast development of interoperable, personalized Semantic Web applications. Therefore, applications are split into different parts and encapsulated in reusable Web Services. The Framework distinguishes mainly three different kinds of services: a) Personalization Services, b) Syndication Services, and c) a Connector Service. The Syndication Services (*SynServices*) contain the business logic of an application and maintain the user interface. To receive (personalized) data, SynServices invoke Personalization Services (*PServices*), which allow for personalized access to a specific part of the Semantic Web. A typical Personal Reader setting could look as follows (cf. [1]): *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.*

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

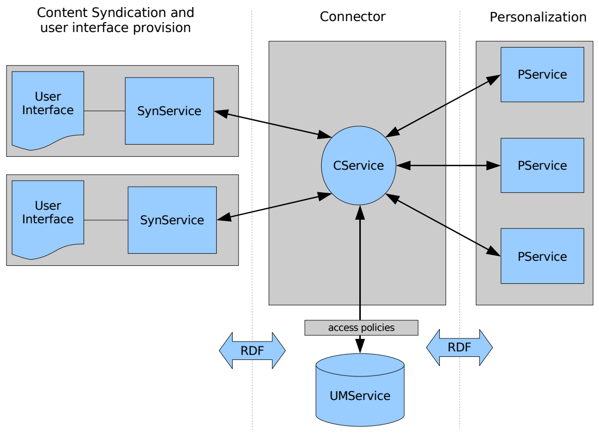


Figure 1. Architecture of the Personal Reader Framework

To find appropriate PServices, the SynService utilizes the Connector Service, which is a kind of enhanced UDDI repository. The Connector Service furthermore enforces the user's PService usage pattern. For example, users can define that only those PServices shall be selected which are free of charge or which are trusted by a trust authority. A detailed description of the RDF-based messages passed between the different services can be found at [13].

4. The User Modeling Service (UMService)

The User Modeling Service (*UMService*) is a centralized service, which is implemented within the Personal Reader Framework. With simple store, update, and query commands, every Personal Reader service (SynServices and PServices) can access the UMService.

As the Personal Reader Framework is an application independent framework, centralized components of the framework have to be domain-independent. Therefore, the UMService does not aim on modeling the user on its own, but relies on the abilities of the applications. This conforms to the nature of applications, which are certainly aware of the domain, the meaning of the user interactions, and the conclusions which can be drawn out of it.

4.1. Data Storage

As storage format we have chosen the Resource Description Framework (RDF). RDF enables to annotate information by relating statements to ontology concepts. As the base vocabulary for our ontology, called the User Modeling Ontology (UMO), we selected Heckmann's *Generalized User Modeling Ontology* (GUMO) described in [11]. An UMO example statement is shown in Figure 3.

```
<rdf:Description rdf:about="#HobbyStatement">
  <umo:subject rdf:resource="#Pete"/>
  <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="#Pete"/>
  <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 2. An example statement expressed in the User Modeling Ontology

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*.

4.1.1 Main Segment

The Main Segment stores the basic statement about the user. Every statement is addressable by its own URI, therefore *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 Pete's credit card has the number 123, we allow to create a statement whose owner (*user*) is Pete 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

The *ambit* therefore 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*. To automatically map different intervals to each other, *identityElement* contains the neutral element. The *scopeValue* predicate finally contains the actual value of the statement chosen from the specified interval.

4.1.2 Explanation Segment

The Explanation Segment contains the author of the statement (*creator*), which *method* was used to create the statement (e.g. was it a direct input of the user, or a derived information based on user interaction). Furthermore, it contains the *evidence* of the statement that describes the prerequisites that caused the 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* and hence delivers wrong data about the user, then distrust regarding this statement can precisely be mapped to service *B*.

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. Both values, *confidence* and *trust* can be used to calculate the accuracy of the statements of a specific Web Service.

4.1.3 Validity Segment

The Validity Segment defines how long – counting from *start* – a statement shall be valid. If the validity can be defined precisely, the predicate *end* is used. This holds for statements like "Pete plays tennis from 6-7 pm". The validity of other statements like "Pete 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 be deleted and hence, should not be used again.

4.1.4 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. Finally, the user can decide whether the statement should physically be removed or recovered.

4.2. Access Control

For access control we provide an enhanced SESAME[3] endpoint. With SeRQL, an RDF query language similar to SPARQL, services can access and exploit the raw RDF data stored in the UMService with the help of powerful and expressive queries.

In [2], we have shown how to protect an arbitrary RDF repository like SESAME by enforcing access policies. Therefore, we have shown how to use the Protune Policy Language (see [6]) to specify policies which protect the access to RDF data. We implemented an access control layer (AC4RDF) on top of a SeRQL endpoint, which extends the original query in a way that deny and allow rules are respected by the query. Afterwards, the original SERQL endpoint is invoked with the extended query and the result can be directly delivered to the invoker without further need of access control processing. We have shown that the policy-based query extension approach is superior to other access control approaches, e.g. in terms of expressiveness, performance, or maintainability.

Instead of more simple access rules, like pattern-based access restrictions described for example in UserQL [10],

```
a) deny(access(rdfTriple(X, _, _))) :-
      rdfTriple('X', deleted, "true").

b) deny(access(rdfTriple(X, _, _))) :-
      rdfTriple(_ , replaces, X).
```

Figure 3. Example Protune deny policies

Protune policies can exploit the full graph structure of the RDF document. Thus, for example access to a specific statement can depend on the condition if another statement is present in the user profile data store or not. Furthermore, Protune already has a large number of build-in predicates, e.g. to include time constraints or constraints concerning the invoking Web Service.

Furthermore, the Access Control Layer is used to enforce the described behavior of the UMService. For example, the two policies illustrated in Figure 3 enforce that a) no statement is accessible which is marked as deleted and b) no statement is accessible which is replaced by another statement.

4.3. User Interface

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 own access policies in a graphical fashion.

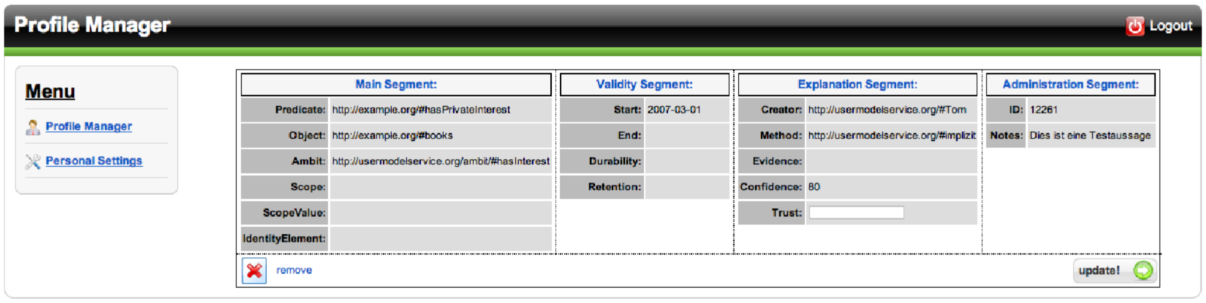
4.3.1 Data Access and Modification

The data access and modification interface (Profile Manager - see Figure 4) allows users to exploit and adjust their own user profile. Therefore, they can change the trust value of the statement to express their agreement or disagreement with the statement. Furthermore, 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 describes 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.

4.3.2 Access Policy Editor

If a service attempts to access statements from the UMService for which there are no access policies defined yet (neither allow nor deny policies), the operation of the service fails due to the deny-by-default behavior and the user is forwarded to the Policy Editor [3]. Afterwards, an overview is presented to the user which is adapted to the context of the

[3] www.openrdf.org/



Main Segment:
Predicate: http://example.org/#hasPrivateInterest
Object: http://example.org/#books
Ambit: http://usermodelservice.org/ambit/#hasInterest
Scope:
ScopeValue:
IdentityElement:

Validity Segment:
Start: 2007-03-01
End:
Durability:
Retention:

Explanation Segment:
Creator: http://usermodelservice.org/#Tom
Method: http://usermodelservice.org/#implizit
Evidence:
Confidence: 80
Trust:

Administration Segment:
ID: 12261
Notes: Dies ist eine Testaussage

Figure 4. User interface to review, modify and delete the statement in the UMService

failed operation, i.e. only those Statements are displayed to the user which were requested. This is important as the user profile can contain thousands of statements which would make it impossible to visualize the statements in an easy and efficient manner. The policy editor itself is split into a part which outlines the actual RDF statements and a part which allows the specification of corresponding access policies (see Figure 5).

The Policy Editor makes the user aware of how specified policies will influence the access to the RDF statements. Therefore, RDF statements are colored according to the effects of policies (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.).

If conflicts occur, e.g. a statement is covered by a deny and an allow rule, the user is informed about this conflict. The user can define which policy should be applicable by specializing the other policies, so that the modified policy does not longer fit to the conflicting statement. If the conflict is not resolved, the deny policy overlaps the allow policy as disclosing a private user profile information is a bigger problem than hiding a public user profile information.

For expert users, we have implemented an extended Policy Editor which allows to edit Protune policies directly. However, for most purposes this is not needed.

5. Conclusion and Future Work

We presented the User Modeling Service, a central component for maintaining user profiles in the Personal Reader Framework. The main advantage of the UMService is that it does not contain domain specific knowledge and can be used for arbitrary applications. The User Modeling Ontology allows on the one hand 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. To ensure that users have full control over their data, we extended a SESAME endpoint with AC4RDF, a mechanism to enforce access policies on top of an RDF repository.

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. Therefore, users do not need to have knowledge about RDF data or policies.

In the future, we plan to provide the UMService not only within the Personal Reader Framework, but also to exchange user profile information with other user modeling frameworks. This would on the one hand enable the User Modeling Service to access large repositories of already existing data and on the hand would enable non Personal Reader applications to use the UMService abilities.

References

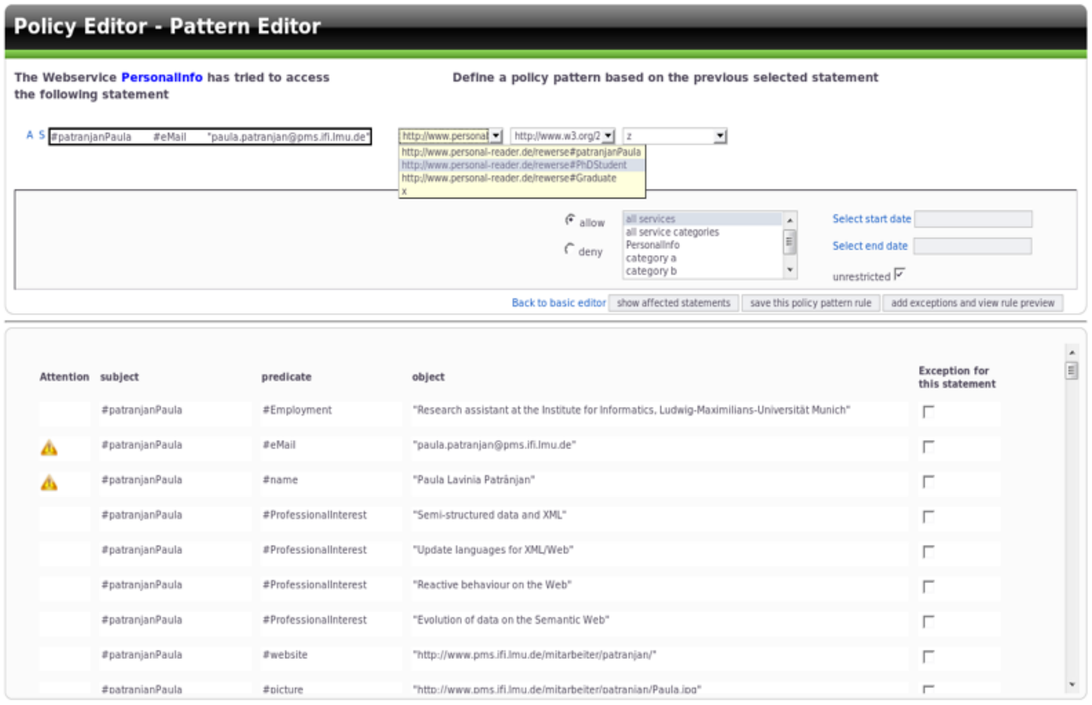
[1] F. Abel, R. Baumgartner, A. Brooks, C. Enzi, G. Gottlob, N. Henze, M. Herzog, M. Kriesell, W. Nejdl, and K. Tomaschewski. The personal publication reader, semantic web challenge 2005. In *4th International Semantic Web Conference*, nov 2005.

[2] F. Abel, J. L. D. Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *6th International Semantic Web Conference*, pages 1–14, 2007.

[3] F. Abel, J. L. D. Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla. An interface enabling users to define and adjust policies for dynamic user models. Technical report, 2007.

[4] S. Berkovsky, T. Kuflik, and F. Ricci. Cross-technique mediation of user models. In *4th Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 21–30, 2006.

[5] S. Berkovsky, T. Kuflik, and F. Ricci. Cross-domain mediation in collaborative filtering. In Conati et al. [8], pages 355–359.



Policy Editor - Pattern Editor

The Webservice PersonalInfo has tried to access the following statement

A S #patranjanPaula #eMail "paula.patranjan@pms.ifi.lmu.de"

Define a policy pattern based on the previous selected statement

http://www.personal | http://www.w3.org/2 | z

http://www.personal-reader.de/rewerse#patranjanPaula
http://www.personal-reader.de/rewerse#PhDStudent
http://www.personal-reader.de/rewerse#Graduate
x

[✓ allow] deny

all services
all service categories
PersonalInfo
category a
category b

Select start date

Select end date

unrestricted ✓

Back to basic editor
show affected statements
save this policy pattern rule
add exceptions and view rule preview

Attention	subject	predicate	object	Exception for this statement
	#patranjanPaula	#Employment	"Research assistant at the Institute for Informatics, Ludwig-Maximilians-Universität Munich"	☐
⚠	#patranjanPaula	#eMail	"paula.patranjan@pms.ifi.lmu.de"	☐
⚠	#patranjanPaula	#name	"Paula Lavinia Pătrânjan"	☐
	#patranjanPaula	#ProfessionalInterest	"Semi-structured data and XML"	☐
	#patranjanPaula	#ProfessionalInterest	"Update languages for XML/Web"	☐
	#patranjanPaula	#ProfessionalInterest	"Reactive behaviour on the Web"	☐
	#patranjanPaula	#ProfessionalInterest	"Evolution of data on the Semantic Web"	☐
	#patranjanPaula	#website	"http://www.pms.ifi.lmu.de/mitarbeiter/patranjan/"	☐
	#patranjanPaula	#picture	"http://www.pms.ifi.lmu.de/mitarbeiter/patranjan/Paula.jpg"	☐

Figure 5. The Policy Editor enables users to specify access policies

[6] P. A. Bonatti and D. Olmedilla. 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, June 2005. IEEE Computer Society.

[7] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[8] C. Conati, K. F. McCoy, and G. Paliouras, editors. *User Modeling 2007, 11th International Conference, UM 2007, Corfu, Greece, June 25-29, 2007, Proceedings*, volume 4511 of *Lecture Notes in Computer Science*. Springer, 2007.

[9] T. Finin and D. Drager. Gums: a general user modeling system. In *HLT '86: Proceedings of the workshop on Strategic computing natural language*, pages 224–230, Morristown, NJ, USA, 1986. Association for Computational Linguistics.

[10] D. Heckmann. *Ubiquitous User Modeling*. PhD thesis, Department of Computer Science, Saarland University, Germany, November 2005.

[11] D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. von Wilamowitz-Moellendorff. Gumo - the general user model ontology. In *Proceedings of the 10th International Conference on User Modeling*, pages 428–432, Edinburgh, UK, 2005. LNAI 3538: Springer, Berlin Heidelberg.

[12] N. Henze and D. Krause. Personalized access to web services in the semantic web. In *SWUI 2006 - 3rd International Semantic Web User Interaction Workshop*, Athens, Georgia, USA, nov 2006.

[13] N. Henze and D. Krause. User profiling and privacy protection for a web service oriented semantic web. In K.-D. Althoff and M. Schaaf, editors, *LWA*, volume 1/2006 of *Hildesheimer Informatik-Berichte*, pages 42–46. University of Hildesheim, Institute of Computer Science, 2006.

[14] J. Kay, B. Kummerfeld, and P. Lauder. Personis: A server for user models. In *Second Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 203–212, London, UK, 2002. Springer-Verlag.

[15] A. Kobsa. Privacy-enhanced web personalization. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 628–670. Springer, 2007.

[16] A. Kobsa and W. Pohl. The user modeling shell system bgp-ms, 1995.

[17] J. Vassileva, G. I. McCalla, and J. E. Greer. Multi-agent multi-user modeling in i-help. *User Model. User-Adapt. Interact.*, 13(1-2):179–210, 2003.

[18] M. Yudelson, P. Brusilovsky, and V. Zadorozhny. A user modeling server for contemporary adaptive hypermedia: An evaluation of the push approach to evidence propagation. In Conati et al. [8], pages 27–36.