Anton Andrejko

Novel Approaches to Acquisition and Maintenance of User Model

Dissertation thesis

Slovak University of Technology
Bratislava, Slovakia

Anton Andrejko

Novel Approaches to Acquisition and Maintenance of User Model

Dissertation thesis

This thesis is submitted
in fulfilment of the requirements
for the degree of philosophiae doctor (PhD.)
in the field of Software Engineering

Study program: Software Systems
Field of study: Software Engineering

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology

Supervisor: Prof. Mária Bieliková
January 2009

Author: Anton Andrejko
Faculty of Informatics and Information Technology
Slovak University of Technology
Ilkovičova 3
842 16 Bratislava
Slovakia

Supervisor: Prof. Mária Bieliková (Slovak University of Technology, Bratislava)
Reviewers: Assoc. Prof. Ivan Jelínek (Czech Technical University, Prague)
Assoc. Prof. Petr Šaloun (University of Ostrava, Ostrava)

Head of the Board of specialists in software engineering:
Prof. Pavol Návrat (Slovak University of Technology, Bratislava)

Keywords: adaptation, concepts similarity, personalization, ontology, semantic web, spreading activation, user, user modeling, user characteristic

ACM Subject Classification:
D.2.13 [Reusable Software] Reuse models;
H.3.3 [Information Search and Retrieval] Selection process;
H.3.4 [Systems and Software] User profiles and alert services;
H.5.4 [Hypertext/Hypermedia] User issues;
K.3.1 [Computer Uses in Education] Computer-assisted instruction

This thesis was set in Computer Modern Roman by the author using LaTeX 2_ε.

Abstract

Novel Approaches to Acquisition and Maintenance of User Model

There are many problems related to the Web — one aspect is that it is overloaded by huge amount of information where a searcher can get lost in information space, spending excessive time reading irrelevant and unproductive content, etc. These problems can be partially overcome by employing personalization (the Adaptive Web), semantics (the Semantic Web) or social relationships (the Social Web). In our research, we focus on the user modeling area and we exploit existing knowledge, mostly from the Adaptive and Semantic Web. User model in the adaptive web-based applications consists of user's characteristics that are used for personalization of layout, navigation or content. There are methods that use explicit or implicit approaches, or their combination to acquire user characteristics and keep them up to date. The result is that accurate personalization can be provided to a user.

In this work we present a contribution to the current state of the art in the user modeling area, namely we focus on creation and maintenance of the user model. We propose three novel methods for acquisition and maintenance of user characteristics in the user model. The first *method is based on generating questions* to be used for user model. Particular questions are generated according to the attributes of information concepts that are the subject of the specific application domain. The entire process of asking questions is driven by user defined rules. The second *method is based on the content analysis* and assumes that comparing attributes of documents, which were found interesting for a user, can be a source for discovering information about user's interests. We use ontology structure and different similarity metrics to compute similarity between instances of ontological concepts. Moreover, we impute reasons that might have caused user's interest in the content. The third and last *method is based on spreading activation.* If there are connections between information concepts (e.g., learning objects in an educational application) of the domain model user's characteristics can be utilized even for concepts that have not been visited yet. In such a way, more accurate and responsive information retrieval capabilities for the user become available.

The proposed methods were evaluated by means of software tools that were incorporated in research projects aimed at job offers, digital libraries and learning programming domains that have been solved at the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies at Slovak University of Technology in Bratislava in the period of 2004–2008.

Ing. Anton Andrejko

Nové prístupy k získavaniu a udržiavaniu modelu používateľa

Dizertačná práca

Dizertačná práca na získanie
vedecko-akademickej hodnosti philosophiae doctor
v odbore Softvérové inžinierstvo

Študijný program: Programové systémy
Študijný odbor: Softvérové inžinierstvo

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita

Školiteľ: Prof. Ing. Mária Bieliková, PhD.
Január 2009

Autor: Ing. Anton Andrejko
Fakulta informatiky a informačných technológií
Slovenská technická univerzita
Ilkovičova 3
842 16 Bratislava

Školiteľ: Prof. Ing. Mária Bieliková, PhD.
(Slovenská technická univerzita, Bratislava)
Oponenti: Doc. Ing. Ivan Jelínek, CSc. (České vysoké učení technické, Praha)
Doc. RNDr. Petr Šaloun, Ph.D. (Ostravská univerzita, Ostrava)

Predseda odborovej komisie pre softvérové inžnierstvo:
Prof. Ing. Pavol Návrat, PhD. (Slovenská technická univerzita, Bratislava)

Kľúčové slová: adaptácia, podobnosť konceptov, personalizácia, ontológia,
web so sémantikou, šírenie aktivácie, používateľ,
model používateľa, charakteristika používateľa

ACM klasifikácia pojmov:
D.2.13 [Znovupoužiteľný softvér] Znovupoužiteľné modely;
H.3.3 [Vyhľadávanie a získavanie informácií] Proces výberu;
H.3.4 [Systémy a softvér] Používateľský profil a služby upozorňovania;
H.5.4 [Hypertext/Hypermédiá] Problémy používateľov;
K.3.1 [Použitie počítačov vo vzdelávaní] Počítačom asistovaná výučba

Táto práca bola vysádzaná písmom Computer Modern Roman použitím LaTeX 2_ε.

Abstrakt

Nové prístupy k získavaniu a udržiavaniu modelu používateľa

S rozšírením webu súvisia viaceré problémy, napr. používateľ je preťažený veľkým množstvom informácií, stráca sa v informačnom priestore, trávi veľa času čítaním obsahu, ktorý ho nezaujíma a pod. Čiastočné riešenie týchto problémov ponúka zapojenie personalizácie (adaptívny web), sémantiky (sémantický web) alebo sociálnych vzťahov (sociálny web). V tejto práci sa zameriavame na oblasť modelovania používateľa, kde využívame najmä poznatky z výskumu adaptívneho webu a webu so sémantikou. V adaptívnych webových aplikáciách model používateľa obsahuje charakteristiky, ktoré slúžia na prispôsobovanie vzhľadu, navigácie alebo obsahu. V praxi sa používajú rôzne implicitné a explicitné prístupy (alebo ich kombinácia) na získanie charakteristík používateľa a ich udržiavanie. Následne môže byť uskutočnené prispôsobovanie individuálnemu používateľovi.

Predstavujeme príspevok k aktuálnemu stavu poznania v oblasti modelovania používateľa. Zameriavame sa najmä na vytváranie a udržiavanie modelu používateľa. Na tento účel sme navrhli tri nové metódy na získavanie a udržiavanie charakteristík používateľa v modeli používateľa. Prvá *metóda je založená na generovaní otázok.* Jednotlivé otázky sú generované na základe atribútov konceptov, ktoré sa používajú vo zvolenej aplikačnej doméne. Celý proces generovania otázok je riadený používateľom definovanými pravidlami. *Metóda založená na analýze obsahu* predpokladá, že porovnávanie atribútov dokumentov, o ktoré sa používateľ zaujíma, môže slúžiť ako zdroj informácií o používateľových záujmoch. Zavedenie výpočtu podobnosti pre jednotlivé atribúty nám umožňuje využiť sémantiku z ontologickej reprezentácie. Pri výpočte využívame štruktúru ontológie a rôzne metriky podobnosti. Navyše skúmame dôvody, ktoré mohli spôsobiť používateľov záujem o čítaný obsah. Posledná *metóda je založená na šírení aktivácie.* Ak medzi informačnými konceptmi v doméne existujú spojenia, potom jednotlivé charakteristiky môžu byť zmenené aj pre koncepty, ktoré používateľ ešte nenavštívil. Takýmto spôsobom môžeme ponúknuť používateľovi presnejšie prispôsobovanie.

Navrhnuté metódy boli overené softvérovými nástrojmi, ktoré boli zapojené do výskumných projektov zameraných na oblasť pracovných ponúk, digitálnych knižníc a na výučbu programovania riešenými na Ústave informatiky a softvérového inžinierstva Fakulty informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave v období 2004–2008.

Acknowledgement

I would like to thank my supervisor Mária Bieliková for her advice which was invaluable help throughout my studies and work on this thesis. I am also thankful for having a chance to cooperate with two great students of mine, Michal Šimún and Tomáš Klempa. The cooperation with them contributed with some great ideas to this thesis. Furthermore, I appreciated the comments which I was provided with from the reviewers, Assoc. Prof. Ivan Jelínek and Assoc. Prof. Petr Šaloun, which were an asset as well, and helped to improve quality of this work. I am also grateful to Mike Nix and his family for their patience in thoroughly proofreading this thesis. And last but not least, there are many others who already know how grateful I am for their support.

Contents

Part I

Setting the Stage

Chapter 1

Introduction

Information technologies have become a part of our lives in recent history and using services provided by the Web have become especially popular world wide. The reason why it is so accepted is because of its availability for almost everyone. The Web is the source of information for any age groups. One limitation is that not everybody is computer-literate and able to find the appropriate information effectively. Some people can find what they are looking for very easily and quickly whereas others can not function effectively in the Web environment. Even skillful Web users appreciate tools and utilities that make their search work easier as well.

Even though the Semantic Web introduces solutions to some problems, there are still many problems that limit the speed and extent of its expansion. Problems related to interoperability across various systems are caused by the existence and use of many ontology languages for knowledge representation. Therefore, many tools are being developed to address problems such as ontology mapping, ontology integration, merging and alignment. Existing solutions solve most posed problems more or less automatically.

Semantic Web technology and applications also inherit problems from its ancestor, i.e. the Web as we know it today. The amount and density of knowledge we expect to be available may cause significant problems. One problem is simple overload by huge amount of information where a searcher can get lost in information space resulting in time consuming and unproductive information searches. These data volume based unproductive search problems can be partially overcome if we take into account a user's particular search characteristics and we exploit existing search optimization approaches that are currently used in adaptive web-based applications.

The user model in adaptive web-based applications consists of identifying user's characteristics that are used for personalization of *layout*, *navigation* or *content*. There are several approaches used to acquire user characteristics

and keep them up to date. One method to acquire the user characteristics is to ask the user *explicitly* or observe the user's behavior while working with the application (*implicit feedback*). Another useful approach is to *mine* the user characteristics from logs, which can be processed on client-side and/or server-side. Another variation of this approach is an acquisition of user characteristics from logs with semantics while an estimation of the user's actions from the logs is performed aimed at the user model update. The logging (and implicit feedback in general) requires sequential processing to transform acquired information into the user characteristics. *Analyzing content* that is presented to a user is another good source of information about the user. If we know a user's given rating to displayed content (e.g., user's interest) we can acquire some characteristics by analyzing similar and different aspects of presented contents.

In this work we present a contribution to the current state of the art in the user modeling area, namely we focus on creation and maintenance of the user model. We propose three novel methods to acqusition and maintenance of the user characteristics in the user model. The method is based on *generating questions* to be used for user model updates. Particular questions are generated according to the analyzed properties of the information concepts that are the subject of the observed application domain. The entire process of asking questions is driven by user-defined rules. The second method based on the *content analysis* assumes that comparing properties of documents, which were found interesting for a user, can be a good source for discovering information about user's interests. Introducing similarity computed for individual properties allows employment of semantics from the ontology representation. We use ontology structure and different similarity metrics to compute similarity between instances of ontological concepts. Moreover, in this methodology we impute reasons that might have caused user's interest in the content. The last method is based on *spreading activation*. If there are connections between information concepts (e.g., learning objects in an educational application) of the domain model user's characteristics can be utilized to extend and extrapolate beyond the known characteristics, even for concepts that have not been visited yet. In such a way, more accurate and responsive information retrieval capabilities for the user become available.

The granularity of the methods' description being presented differs from one to another. The method based on the content analysis is described more in details since it is an individual work of the author of the thesis. The remaining methods were proposed and verified in cooperation with his master students as a part of their master thesis under his tutoring, therefore, these reflect a bit lower granularity in the description.

The proposed methods were evaluated by software tools within research projects aimed at job offers (project NAZOU[1]), digital libraries (project MAPEKUS[2]) and learning programming domains (project PeWePro[3]) that have been conducted successfully at the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies at Slovak University of Technology in Bratislava in the period of 2004–2008.

[1]NAZOU — Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources, http://nazou.fiit.stuba.sk

[2]MAPEKUS — Modeling and acquisition, processing and employing knowledge about user activities in the Internet hyperspace, http://mapekus.fiit.stuba.sk

[3]PeWePro — Adaptive web-based portal for learning programming, http://pewepro.fiit.stuba.sk

Chapter 2

Thesis objectives

The personalization process we deal with is based on the user model. The user model contains identifying characteristics that represent a real user. The more characteristics about the user are available, the more accurate personalization can be provided. However, there is one more aspect which must be considered — user characteristics' currency. The user changes (e.g., gains new knowledge or gets more experienced) over time, therefore, the user model must always reflect these changes to maintain effectiveness in enhancing the personalization.

The field of initial personal information gathering is known as *acquisition* (i.e., the user model is populated with new characteristics). The ongoing gathering of user information is termed *maintenance* (i.e., existing characteristics are kept up to date) of user characteristics, which is necessary to provide a dynamic and expanding knowledge base for further personalization. There are several methods (e.g., implicit or explicit feedback) that can be used.

In our work we propose novel approaches to automatic acquisition and maintenance of the user characteristics that employ semantics provided by ontological representation. The main objectives of this work are following:

- A typical approach to acquiring user characteristics is explicit feedback where a user is asked a question aimed at a particular characteristic. We assume that questions can be generated automatically according to the concepts in the domain model. Our goal is to propose a method of generating questions to be used for acquiring and maintenance of user characteristics for the user model.

- We assume that comparing properties of documents, which users found interesting, leads to discovery of information about users' interests. The

analysis of the content presented to users is a suitable source of information for personalized applications. Our goal is to propose a method that uses content analysis (particularly similarity estimation) to determine information suitable for estimation of user characteristics for the user model.

– In many information spaces there are well-defined connections between information concepts (e.g., learning objects in an educational application) of the domain model. We assume that, if such connections are available, a user's characteristics can be changed for the concepts, even if they have not been visited yet by the user. Our goal is to propose a method for maintenance of user characteristics in the user model based on relationships among concepts in the domain model.

– Experimental evaluation of the proposed methods includes development of software tools for the particular methods. Our goal is an evaluation of the proposed methods with regard to their domain independency.

This work is organized in 8 sections. Section 3 is presented as the basis and necessary background about the Adaptive and Semantic Web which represent a context we utilize heavily in this work. Section 4 describes main approaches and problems related to the user modeling field with the focus on acquisition and maintenance of user characteristics. In following sections are described novel methods aimed at acquisition and/or maintenance of user characteristics based on:

– generating questions according to the domain ontology concepts (Section 5);

– analyzing similar and different aspects of the documents presented to a user (Section 6); and

– spreading activation (Section 7).

Finally, Section 8 contains our concluding remarks.

Part II

Related Work and State of the Art

Chapter 3

Adaptive Semantic Web

The amount of available information in large information spaces (e.g., the Web) is continuously growing at what appears to be almost an exponential rate. Besides many well-known advantages of this expanding information resource there are some features, which cause the end-user problems and can not be underestimated. As a minimum we call attention to situations when the user is overloaded by the huge amount of the information, or can get lost in the information space, or is directed toward content he/she does not want to see, etc.

A way to improve efficiency in information acquisition is to offer a personalized approach. Improved efficiency can be achieved by employing a personalization based on user's particularities aimed at adaptation of the content, layout or navigation in the information sources (i.e., a hypermedia information space where information chunks are interconnected by links). This makes available to the information sources a new feature — *personalization* and the *user model* is its necessary element. An information source is considered as adaptive (e.g., the Adaptive Web), if it is able to dynamically adapt itself using a user model which reflects some features of the real user [20]. There is a slight difference between the terms adaptation and personalization.

Definition 3.1 *The adaptation in an adaptive application is a modification of content, style of presentation and navigation (e.g., to changing environment), whereas the personalization is an adaptation that is provided to the particular user* [14].

Presently, most of the available information in the Web is provided in a form primarily suitable for human beings, where the choice of its representation and presentation is up to individual information providers. Obviously, information representation is not uniform for all providers, which causes problems

for heterogeneous applications using various information sources. Therefore, the next stage in improving efficiency of information processing is adding semantics to the content (e.g., the Semantic Web).

There is another stage that relates to the problems which current information sources experience — *social relationships* between the users are employed to provide improved efficiency in the information processing (e.g., the Social Web), but this approach is beyond the scope of this work.

3.1 Models of adaptive web-based applications

In adaptive information sources, there are two terms used very often — *adaptivity* and *adaptability*. The meanings of these two terms are often confused or substituted. A common feature for both terms is cooperation with the user. However, in an adaptive information source a user's behavior needs to be observed to obtain necessary information for personalization. On the other hand, in an adaptable information source, the user's action is required, e.g. setting up properties. These are not mutually exclusive concepts since both approaches can be combined. For the purposes of this work, if there are at least some features, which accomplish automatic personalization, we consider such a source as adaptive [14]. Basically, adaptive applications consist of four main parts:

- *domain model*, which represents the scope for which the application is designed to be performed;
- *user model*, where all the actual user's characteristics, which are necessary for adaptation, are stored;
- *adaptation model*, which specifies a method for the adaptation to be accomplished. It is often recognized as being interwoven with the domain model. Typically, it is defined as a set of rules that impacts the appearance of the presentation, as well as it also impacts and updates the user model.
- *navigation model*, which is based on the used domain model and describes user's possible moves among the concepts within the information space [33].

Domain model contains concepts and their relationships. In the field of adaptive applications is the concept defined as follows [14]:

Definition 3.2 *The concept represents integrated parts of an information space in adaptive applications. The concept can be atomic or a composite of other concepts. The concept or a set of the concepts is presented to the user usually as a web page in adaptive web-based applications.*

Since the purpose of the adaptive applications is to provide personalization, the user model is the most important part. There are more definitions of the user model which differ according to the way the model is used. We use the following definition of the user model [50]:

Definition 3.3 *The user model represents beliefs about the user that include preferences, knowledge and attributes for a particular domain.*

We use the common term *user characteristics* to describe preferences, knowledge, attributes and other identifying features that are included in the user model. The user model can be realized as:

- *overlay model*, where all the concepts from the domain model have its own copy in the user model,
- *stereotype model*, where users are organized in groups and the same adaptation is provided for any group member. We do not consider this as a personalization to an individual user, but personalization to a group.

Sometimes we also take into account environment where the user works [14]. Information about user's environment should not be neglected, if available. For instance, if we know that the user speaks only one language, we should present any information content in this language wherever possible. Another example of recognition of a user's environment is presenting relevant information based on time criteria [15]. Also the user's location or their platform might be important for adaptation [22]. The user's environment together with the user model is called the *context model*. In particular application domains other models can also be constructed with regard to the purpose, e.g. *goal model* and *teaching model* in educational domain [29].

Adaptation can be provided according to the context when the same content is presented to all users (e.g., based on place, time) or according to the user's particularities. In applications, where the adaption is based on the user's particularities, the user model is considered as their most important part.

Adaptation in the web-based applications occurs on three levels, namely adaptation of the *content*, *presentation* or *navigation*:

– The *content* consists of the information fragments where an information fragment is related to one or more concepts. Adaptation of the content influences which information fragments are presented to the user.

– Adaptation of the *presentation* defines the appearance of the content, e.g. layout of the content, location of the navigation bar.

– Adaptation of the *navigation* influences the order of the web pages that are presented. It is important because the user does not have to see all information, but should insure that vital content is not missed.

Several methods were proposed to accomplish the adaptation. Adaptation techniques refer to methods of providing adaptation within existing adaptive hypermedia applications and adaptation methods are defined as generalizations of existing adaptation techniques [20].

In Brusilovsky's works [20, 22] we find techniques for adaptation divided in two groups. The first group aggregates techniques related to the adaptation of the *presentation* (including content). The second group contains techniques for adaptation of the *navigation* (see Figure 3.1).

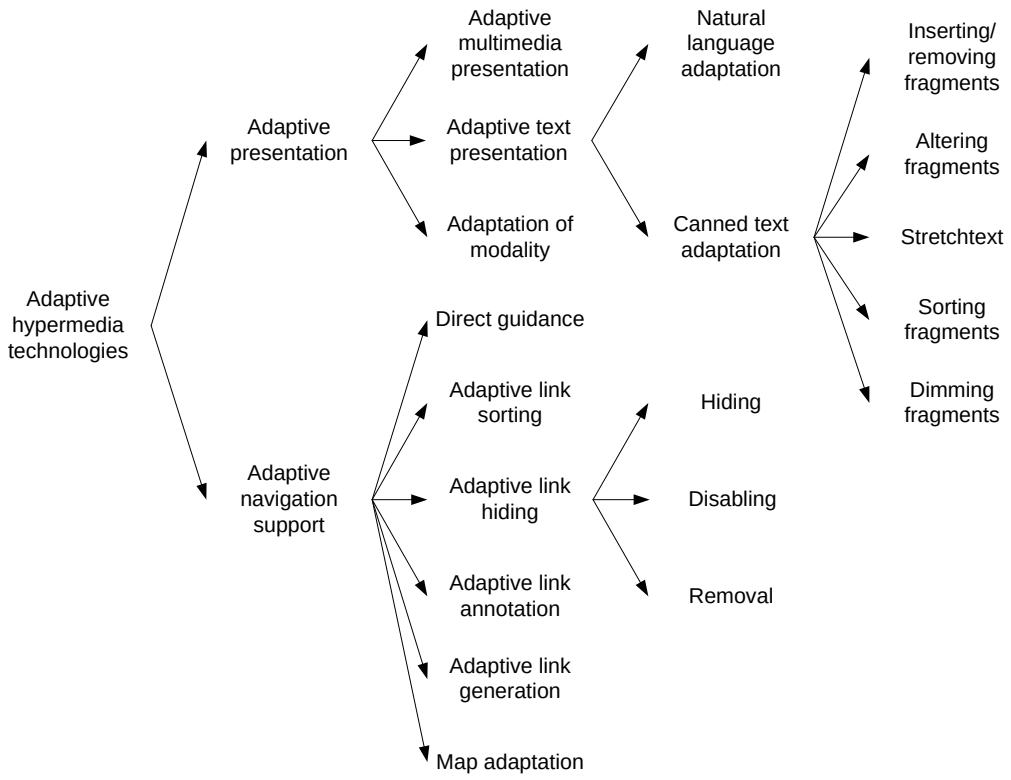

Figure 3.1: Taxonomy of adaptive hypermedia technologies [22].

A detailed description of these techniques is beyond the scope of this work and is provided in [20, 22]. For the content adaptation Brusilovsky recognizes and utilizes methods like *conditional content*, *alternative content*, or *sorting of the content*. It is possible to realize these methods by using various techniques, namely *inserting*, *removing*, *altering*, *sorting*, *dimming fragments* and *stretchtext*.

For the presentation, methods such as *local or global guidance*, *local orientation support*, *global orientation support* or *managing personalized views* are available. These methods can be realized by the following techniques: *sorting links*, *rule-based hiding links*, *annotation links*, etc. Moreover, to point out the adaptation of the navigation we briefly mention techniques related to recommendation based on group relevancy, i.e. *social navigation*, *collaborative* and *content-based filtering*. All these techniques exploit a fact that the user is a member of a group and the group reflects some features that are common for most of the group members. With regard to this aspect we can recommend a possible navigation to the individual user.

In this work we focus on adaptation based on the user model (i.e., personalization) so it is important to note that the user model must be also adapted to be continually up to date and to provide proper information about the user for further personalization. This process is depicted in Figure 3.2.

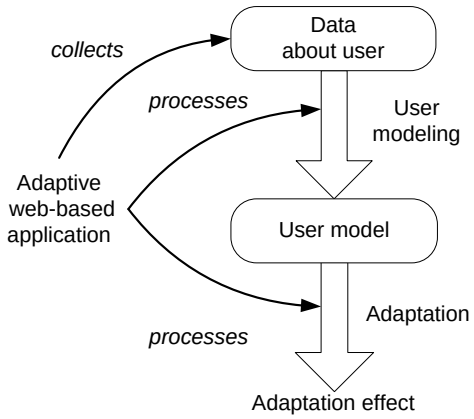

Figure 3.2: User model life cycle according to [20].

An adaptive application collects information about the user to be used for creation of the user model. Afterwards, the populated user model can also be employed in the adaptation. The process has a cyclic and iterative nature, as adaptive applications continuously acquire new data about the user (influenced by already performed adaptation and populated user model) and continuously refines the user model to better reflect reality and to provide a better source for personalization.

3.2 Adding semantics to adaptive applications

Adapting web-based applications to a user has brought an important element to the Web environment — *personalization*. The personalization helps to reduce existing problems on the Web and allows the user to work more effectively with the information that is presented.

The Web, at its current state of evolution, has become successful very rapidly. The reasons for this success include [39]:

- *simplicity* of HTML language,
- *immediate feedback* after HTML page had been designed,
- *additional benefits*, because HTML pages are not used only to present information, but also as a mean of discussion or documentation for people participating in projects,
- *low critical mass of people* needed to raise their interest to become involved.

It is important to note that even when a content presented to the user is personalized, it is still suitable only for a human because it is designed for use by humans. One has to routinely make additional efforts to make acquired information useful. This is significant especially in the case when the user is initially looking for information. After constructing a query, which can be time consuming, and for people not computer-literate is a very complicated action, the user gets a list, which is assembled based on text analysis processing (a fulltext comparison between query and data stored in databases).

To illustrate this situation, let us have a user who wants to find a job position for a programmer, thus he/she specifies "job" and "programmer" as keywords. The result acquired by a keyword-based approach obviously consists of documents containing given keywords. However, these results are not limited to job offers, e.g. a document describing benefits related to programmer's job in general can appear in results. There is no guarantee that the list with the results contains links to documents the user has been looking for and the user has to browse the content of each document to extract information.

One solution to this problem is to represent the Web content not only in a form understandable for a human, but also in a form that is easily *machine-understandable*. In [6] is the machine-understandable term considered as not very appropriate, even it is used quite often. This term makes implies that computers can really understand. Instead, authors of this work are inclined

to use the term *machine-processable* as being more descriptive of the desired characteristics.

Here emerges Tim Berners-Lee's vision known as the Semantic Web initiative [12]. This initiative tries to add semantics to knowledge to make it processable by automated tools as well as by people.

Probably the easiest way to capture the meaning of the presented content is provided by metadata (data about data), which can be added to the presented content. But this solution only partially solves problem related to the machine processing. A job offer could be described as follows:

```
<jobOffer>
  <position>Java Programmer</position>
  <company>IT Solution</company>
  <salary>10000</salary>
  <startDate>ASAP</startDate>
  <contractType>Temporary</contractType>
  <region>New York</region>
  <advantages>
    <benefit>car</benefit>
    <benefit>mobile phone</benefit>
    <benefit>health care</benefit>
  </advantages>
</jobOffer>
```

However, authors of the web page content can use their own terms to describe semantics of the presented information, with resulting variation in metadata descriptive elements which causes problems for heterogeneous applications using various information sources.

A movement from current state, where information on the Web is in a natural language, is not easy due to the decentralized origins of the data and content presented. Therefore, Tim Berners-Lee proposed a transition to the Semantic Web in steps [12] and as the progress proceeds to the next step when the prior step or level is fulfilled. These steps are known as the "layer cake" and are shown in the Figure 3.3.

The two bottom layers (Unicode, URI and XML+NS+XML Schema layer) make a syntax base for the Semantic Web languages. The Unicode layer provides a basic scheme for encoding of the international characters sets. URI is the mean for unique identification and addressing of the resources on the Web. The XML layer with namespace and schema definitions helps integrate the Semantic Web definitions with the other XML based standards. RDF allows writing statements about resources and RDF Schema (based on RDF) allows building hierarchies from these resources.

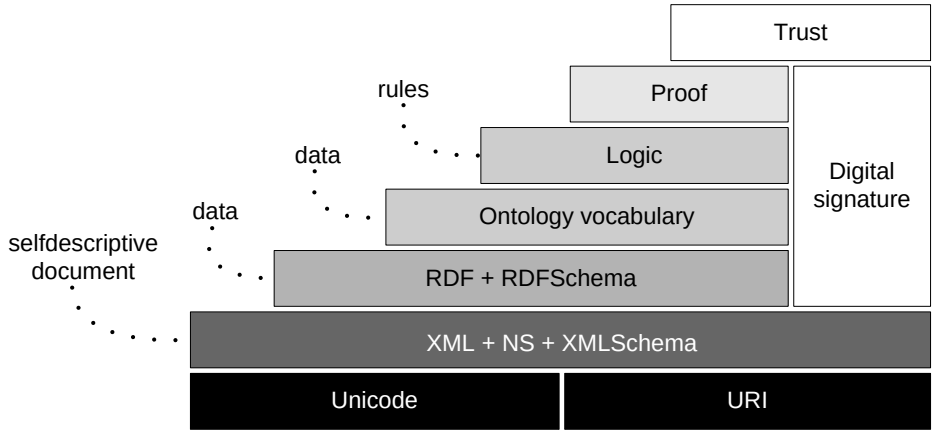

Figure 3.3: The Semantic Web layers [12].

It is recognized in this model that the RDF Schema is still not powerful enough to meet these semantic needs and more expressive ontology languages are needed to express more complex relationships. Resident in higher levels of the model are ontologies with vocabularies and relationships between concepts which make ontologies an appropriate way to represent various models in the web-based applications. The Logic, Proof and Trust layers are still being researched.

3.3 Ontology as a mean for metadata representation

The term *ontology* originates from philosophy, namely from the study of the nature of existence. Several ontology definitions have been given so far. We use Studer's et al. definition [72], which extends Gruber's definition [38].

Definition 3.4 *An ontology is a formal, explicit specification of a shared conceptualization.*

Denotation of this definition is following:

- *formal* — ontology should be machine readable,
- *explicit* — type of concepts used and constraints are defined explicitly,
- *shared* — ontology is not private, but it is accepted by a group,
- *conceptualization* — refers to an abstract simplified view of the world, which is formally represented.

In the ontology we define *classes* (general things) in the many domains of interest, *instances* (particular things), *relationships* among those things, *properties* (and property values) of those things, *functions*, *constrains* and *rules* involving those things [63]. This gives the ontology much more powerful expressiveness than the metadata approach has. Another term, which is used very often along with ontologies, is a *concept*. When dealing with ontologies the meaning of the word concept is different as defined in Section 3.1. The *concept* is sometimes used in place of *class*, where classes are a concrete representation of concepts [43]. Furthermore, the concepts can be organized hierarchically [32]. An ontological concept is defined as follows [8]:

Definition 3.5 *Ontological concept is a set or a class of individual objects that can have simple properties, often called attributes, which are typically attached to the corresponding concepts.*

A range of models with varying degree of semantic richness and complexity can exist among ontologies [63]. An ontology spectrum which provides a framework to compare information models in terms of increasing semantic richness is depicted in Figure 3.4.

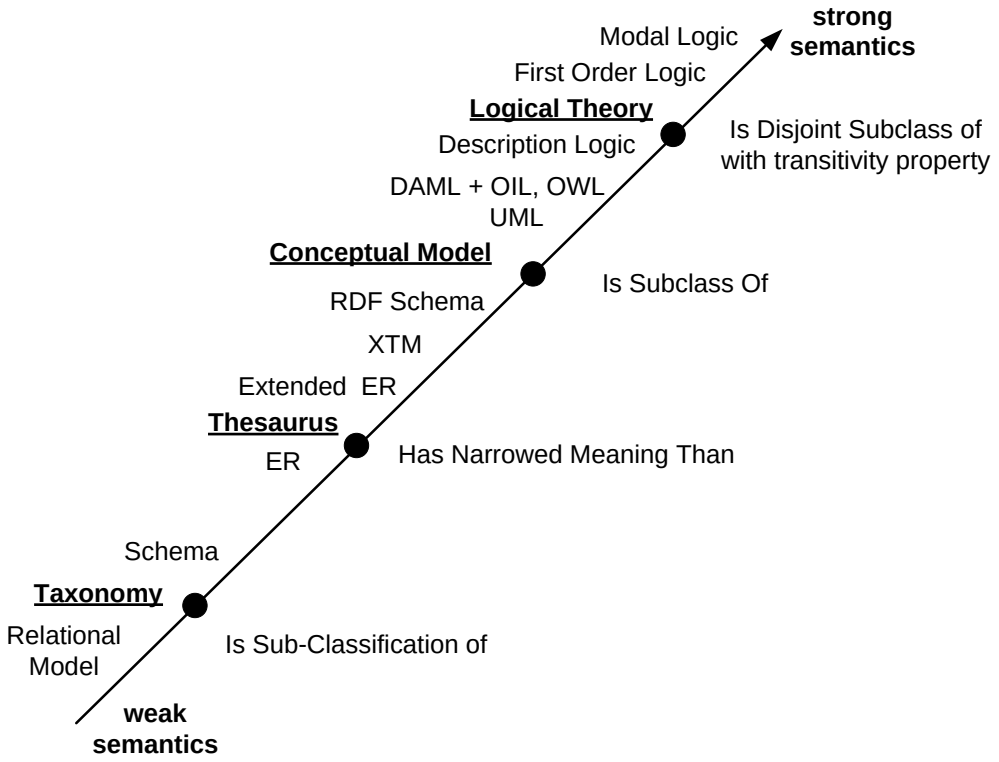

Figure 3.4: The ontology spectrum [63].

Although an ontology is defined as a shared conceptualization [72], a variety of different ontologies describing the same application domain can exist. Thus some means of ontology comparison are required for application and/or data integration. The identification of their common and different aspects is the subject of research in areas known as *ontology mapping*, *merging* and *alignment*. Mainly for the purpose of ontology management, several approaches to comparison of ontology concepts or their instances were developed. There are several overviews oriented primarily at ontology matching and related areas [36, 44, 61]. In these approaches the most important point is to estimate similarity between compared parts.

For illustration, a part of a job offer domain ontology depicting the main class (i.e., *JobOffer*) and related properties is shown in Figure 3.5. The job offer domain ontology was developed as a part of the research project NAZOU[1] [59] where we performed experiments.

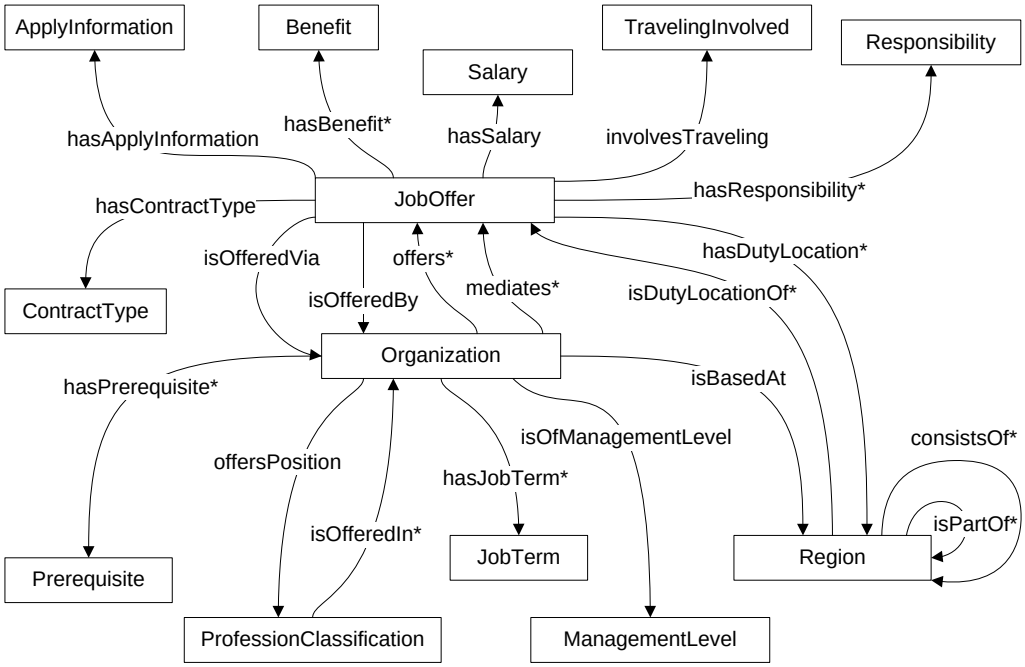

Figure 3.5: JobOffer class in job offer domain ontology.

An example of an instance representing a part of job offer in a job offer application domain is depicted in Figure 3.6. Rectangles used in Figure 3.6 represent instances of concepts. Every such an instance has its unique identifier, but we present only its label for clarity (e.g., *Salary*). *Datatype* and *object properties* are used to assert specific facts about instances. Datatype

[1] Job offer domain ontology, http://nazou.fiit.stuba.sk/home/?page=ontologies

properties express relations between concept instances and RDF literals and XML Schema datatypes. Object properties express relations between two instances. To distinguish between object and datatype properties a dashed line is used for datatype properties (e.g., *maxAmount*). *JobOffer* is the instance identifier which several properties are connected to. For simplicity, we present only a few of them, and surround multiple properties (e.g., *hasPrerequisity*) by a rounded box.

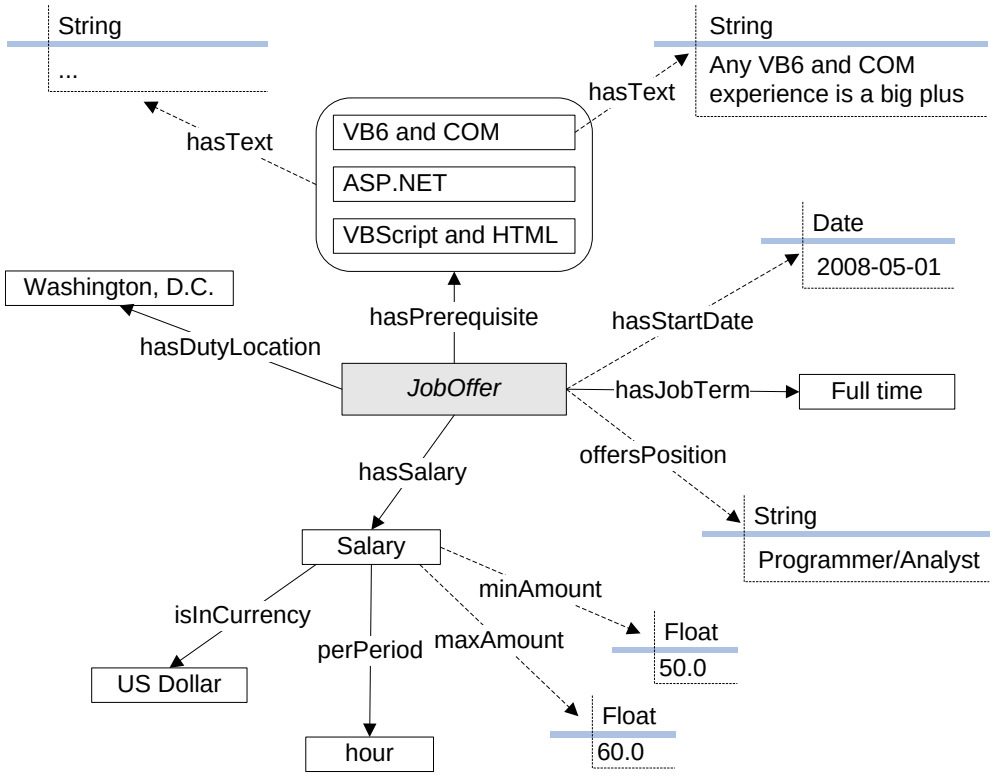

Figure 3.6: Example of an instance representing a part of job offer.

Recently, several approaches have emerged that use ontological representation to store more complex objects. Since it is up to the ontology engineer whether a real-world object is conceptualized as a class or an instance (according to the purpose of the ontology) to exploit the full power of the ontology, it is necessary to use an appropriate representation that is capable to express all concepts (i.e., general things) we have described above. There are many languages that were developed to represent ontologies (e.g., DOGMA, KIF, Ontolingua). The following ontology languages were standardized and recommend to be used on the Web by W3C[2] [56]:

[2]The World Wide Web Consortium, http://www.w3.org

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a data model for objects ("resources") and relations between them; provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantic specification for generalization hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes, such as *relations between classes* (e.g., disjointness), *cardinality* (e.g., exactly one), *equality*, *richer typing of properties*, *characteristics of properties* (e.g., symmetry) and *enumerated classes*.

Furthermore, OWL language provides three increasingly expressive sublanguages and ontology engineers should consider which sublanguage best suits their needs [56]:

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints;
- *OWL DL* supports those users who want the maximum expressiveness while retaining *computational completeness* (all conclusions are guaranteed to be computable) and *decidability* (all computations will finish in finite time). OWL DL is so named due to its correspondence with description logics; and
- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees (e.g., a class can be treated simultaneously as a collection of individuals and as an individual in its own right).

In adaptive web-based applications, the ontologies can be used as a mean to represent models. We focus particularly on the *user model* and also on *domain model* which it is interconnected with. In our work, where documents or their parts are represented via (hierarchically organized) ontological concepts, which describe a set of real objects (i.e., concept instances), we use the OWL Web Ontology Language for ontology representation and particularly its DL sublanguage.

Chapter 4

User modeling

User modeling is a natural consequence of the societal need for personalization. Approaches to the user modeling employ methods from different areas. The roots of the user model techniques are related to the Intelligent Tutoring Systems. An Intelligent Tutoring System (ITS) is a computer based program that presents educational materials in a flexible and personalized way [21]. ITS consists of three main parts: knowledge about the domain or *what* to teach, a learner specifies *who* is to be taught and a teacher strategy defines *how* to teach. If we look at ITS from the user modeling perspective the learner who represents the user of the ITS is the most important for us. An application observes students' work, tailoring feedback and providing hints along the way. By collecting information on a particular student's performance, the application can make inferences about strengths and weaknesses, and can suggest additional work.

The probability that the application's general interface and behavior designed as "one size for all" would be effective for all categories of users is rather low, thus adaptation (personalization) to a particular user is desired. In Kobsa's work we can find the term *user modeling shell systems* [52, 53] or abbreviated *shell system* or *shell*. This term describes the software tools that provide the representation of assumption about one or more user characteristics of an individual user or common characteristics of user belonging to a group, records of the user's behavior, etc.

4.1 User characteristics

A user model represents various user characteristics, which can be used to adapt the *content*, *presentation* or *navigation* of the informational materials presented. A content of the user model is described with variety of terms,

namely *attributes*, *features*, *characteristics* or *properties* are used frequently. In this work we use the term *characteristic*. Based on the differences that are not exclusively of terminology, it is obvious that the user modeling field needs to be standardized. A first attempt is known as User Modeling Meta-Ontology [79].

There are several approaches that utilize classified characteristics in the user model. One of the approaches is distinguishing the user characteristics with regard to how the characteristics change [60]:

1. *permanent characteristics*, which are independent from experience with the particular application (e.g., interests, attitudes, personality, skills, knowledge, abilities and preferences) and do not vary dynamically.

2. *variable characteristics*, which are divided into characteristics, which:

   (a) do not depend on using the application and can be changed independently, e.g. gender, age,

   (b) changes as the application is being used, e.g. working with the application raises the user's experience level.

A similar approach, taking into account how often are the user characteristics changed, divides them into [46]:

– *static group*, which covers characteristics that has been set once and do not change throughout the lifetime of the user model or are changed rarely (e.g., name, gender, date of birth).

– *dynamic group*, which characteristics are usually related to knowledge, goals, etc. and express parts of the user model where changes are made more often (e.g., user's home address). We assume, in this example, that people are moving from one place to the other one and merely seldom they live on the same address for the entire life.

These two groups also differ in the manner with which the information, which is stored in these characteristics, is acquired. The static characteristics are mostly asked in some questionnaire (e.g., gender, date of birth) whereas the dynamic characteristics are acquired by observing user's behavior throughout working with the application. For instance, we can record information about the job offers that has user previously visited. Observing user's activity can be a good source for future inferences about user's preferences or interests.

The *domain dependency* is another important distinguishing criteria [46]. It allows for dividing the user model into:

– *domain-dependent (or domain-specific) part*, which contains characteristics that are closely related to the application domain (e.g., preferred wage per month or per hour in the case of labor market domain),

– *domain-independent part*, which contains characteristics that are not related to the application domain (e.g., age or gender).

Detaching the domain-independent part of the user model allows for the building of a common user model that could be reusable in different and diverse applications. An example of dividing a user model in domain-independent and domain-specific part is shown in Figure 4.1.

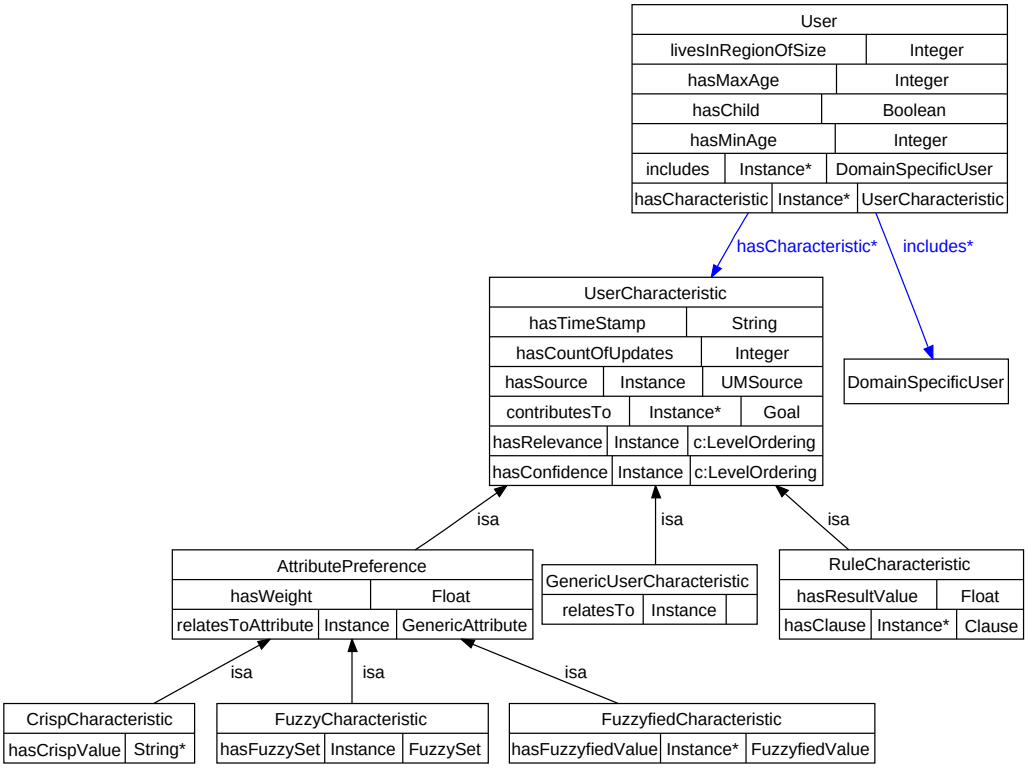

Figure 4.1: Domain-independent user model.

For some domain-dependent characteristics we can use the term *preference* and *interest*. The term *preference* expresses a user's expectations or desired outcomes for the application domain (e.g., the user prefers rather to work as a human resources manager than a technical support specialist). The term *interest* is not as conscious or focused as the preference. Interest expresses attractiveness about someone or something to the user [37] and generally is less accurate (e.g., one knows that is interested in the work with

the people and likes to manage them, but does not know if he/she wants to work as a human resources manager or consultant).

As an example of utilizing a user's characteristics in the process of adaptation, let us assume that a characteristic in the user model describes the minimum acceptable salary per month for a specific user. If there is information known that the user is not interested in job offers where the offered salary is lower (or much lower considering the fuzzy nature of the characteristic) than user's expected salary, offers that do not fulfill this condition will not be presented. The adaptive application modifies and configures information presented on behalf of the user with the help of the user model.

Another case where the user model content can be used, is the *interpretation of the user's input* [49] (which can be ambiguous, incomplete, with errors, etc.) and the personalization of the outputs (e.g., sorting of the results, number of results per page, font, colors). The more relevant characteristics describing the user that are included in the user model, the more accurate and useful is the personalization provided by the adaptive application. Knowledge of specialists who work in the particular application domain for which the user model is designed is invaluable in the process of designing user models. Their experiences and understanding of the content help to construct the user model reflecting the real user as accurately as possible.

4.2 Approaches to user modeling

There are several approaches that can be employed in the user modeling process. Some of them can be used as standalone techniques for constructing user model even though a combination of these techniques is also possible. In this section we discuss approaches (e.g., content-based and collaborative filtering, stereotype model, overlay model) which are the most commonly used and look at user modeling from different perspectives.

Content-based and collaborative filtering

Information filtering attempts to selectively reduce the potential overwhelming amount of the information that the user receives. Special types of the information filtering are content-based and collaborative filtering.

This approach evaluates the content of a document, which has been found interesting by the user and similar contents are selected with regard to the correlations among them. Content-based filtering methods depend on three main factors: the *item's content*, *ratings given by the user*, and a *filtering algorithm* [73]. Content-based filtering can exploit information only from the

document's contents (e.g., words, phrases). Algorithms for content-based filtering need large amount of the information related to the user's behavior.

The *collaborative approach*, or sometimes called social filtering, assumes that users with similar taste will likely prefer similar things. Therefore, users are divided into groups by their interests. Recommendations are provided to the particular user with regard to the ratings acquired from other users that had similar preferences in the past. Eliciting explicit feedback (rating about things) from many users is thus necessary to get these preferences and interests.

The difference between content-based and collaborative filtering is that content-based filtering is based on finding a match between content and the user's preference profile whereas collaborative filtering finds match between people with similar preferences [76]. It is important to note that these approaches are not mutually exclusive and both approaches can be used in combination in user model development.

Stereotype model

The commonly used current approach in adaptive web-based applications is using *stereotype* and *overlay user model* or in some cases a combination approach that exploits advantages of both models. Complex research about stereotypes has been done by Elaine Rich [70]:

Definition 4.1 *The stereotype is a collection of frequently occurring characteristics of users.*

User's knowledge background about the application domain differs from one user to another. The stereotype approach also takes into account that aspect. Users are associated with one or more stereotypes that commonly reflect a user's knowledge [46], social background, computer experience etc. These stereotypes can be ordered hierarchically [60] and characteristics are inherited from the superior stereotypes in the hierarchy. A user associated with the stereotype will inherit all stereotype characteristics automatically.

As an example we can consider three user stereotypes related to the user's education: elementary, secondary and higher. Let us have a user who has achieved higher (university) education — he/she is a specialist in the graduated field of science, is able to work independently, invent new solutions and bring them into the real life, etc. In the user model we will apply all these characteristics, even though a particular user does not have to cover all of them. Based on these characteristics we offer higher level job position to the user with the university education, e.g. positions like project manager

or pharmaceutical researcher. On the other hand, we do not offer positions like driver or painter that we would analytically associate with an offer to the user with a lesser elementary or secondary education.

Kobsa identified three important tasks for designers which need to be fulfilled to properly identify stereotypes [52]:

- User subgroup identification based upon similar characteristics that are relevant for designed system.
- Identification of a small number of key characteristics that allows association of a user to the particular subgroup.
- Formalization of the user group characteristics in an appropriate representation system is necessary.

Designers play an important role in building user stereotypes. They have to identify which user's characteristics should be included into the stereotype. Kay distinguishes *handcrafted* and *empirically-based approaches* to building stereotypes [48]. A designer builds a *handcrafted stereotype* based on his/her observation of the group. An *empirically-based approach* collects data about the users' actions to build stereotype. The designer also defines when the stereotype is triggered and how it will interact with the application.

Essential elements of a stereotype are *triggers*, *inferences* and *retraction facility* [49]. A trigger activates the stereotype, i.e. upon information acquired from user one or more of available stereotypes would be activated. Seldom can this information be acquired by observing the user for short interaction period of the time [48]. After assigning the stereotype inferences about the user can be developed. The user model is thus extended by inferred characteristics. In the case when characteristics obtained in the stereotype do not appear to reflect conformance to the initial stereotype assumptions by the user the retraction facility deactivates it.

A different approach to the stereotype user model is when the model consists of sets of pairs that assigns only the value "true" (i.e., if the user belongs to the stereotype) and "false" (i.e., in the opposite case). A variation on this approach can be the use of a value that reflects probability of belonging to that stereotype as well [20].

The stereotype model is prone to inaccuracy due to the need for heavy reliance on inferences, which is considered as its main drawback [52]. Even for a stereotype that has been selected and applied correctly, some inappropriate characteristics can be inferred for particular user. Here, it is important to emphasize that in this case the personalization is not performed to the benefit

of particular user, but to the selected stereotype in a collective sense. This statement is supported by another definition of the stereotype [47]:

Definition 4.2 *The stereotype is a statistically based reasoning about a group of people.*

A clear advantage of using stereotypes is visible when initializing a user model by first use of an application (know as cold start problem). The user who has provided some but not complete identifying information [26] to the adaptive application can be associated with a particular stereotype and some characteristics can then be inferred.

Overlay model

The main idea of the overlay model is that a user's knowledge of the subject is relevant subset of the domain knowledge and the overlay user model is an overlay over the domain model [20, 46, 48]. It usually uses the same representation as the domain model. An overlay model assumes an instance of itself for each user. This makes a basis for personalization of particular user in contrast to stereotypes.

Domain model in adaptive web-based applications is usually represented through concepts and their relationships. Any domain concept has a corresponding value in the overlay model which represents the user's knowledge of the concept. Estimation of the concept can be a *binary*, *qualitative* or *quantitative value* [20].

An open source general-purpose adaptive hypermedia application, AHA! (Adaptive Hypermedia for All), uses the overlay user model approach. The user model consists of the set of concepts with a corresponding value [30]. Apart from additional personal concepts stored in the user model the overlay user model in AHA! incorporates any concepts from the domain model. An appropriate value is assigned to the concept by the application. This value expresses individual user's knowledge level of this concept.

A special kind of overlay model is a *differential user model* [48]. The differential user model does not reflect the entire domain knowledge, as it is represented in the domain model, but only a part of it. This part is only the subset which the user might be interested in. The overlay model which contains only a real subset of domain knowledge and omits any incorrect knowledge is called *strict overlay model* [46]. On the other hand, the user model that includes also incorrect knowledge is called *perturbation user model* [27].

The main drawback of this approach is the necessity of initialization of the overlay model. When an application is used by a user for the first time there is no information about user's knowledge for the particular domain concept. An often used approach is a quantitative estimation of the initial value to be set at 0, or to some mean value, e.g. 50. This approach is usually used by educational adaptive applications. The way to overcome this problem is a combination of an overlay user model with a stereotype user model [78].

Shared model

An important issue related to effective usage of the user's characteristics representation is need for sharing the user model among several applications. This brings several advantages as shown in [10, 11, 16], e.g. importing and integrating data collected by other applications. An application during initial use can thus utilize the initialized data from other applications (i.e., a solution to cold start problem) what may avoid the need for the user to type the same information over again in every application. Thus, it is clear that the key advantage of the shared user model is an availability of user's characteristics discovered by other applications. Such information can be mutually shared among applications (e.g., over services).

A noticeable advantage of shared user model is that information is logically stored at one place (from technical point of view it can be replicated). This eliminates the redundancy of data storage and updates and simplifies the maintenance of the model. All changes need to be performed only once and then made available to all the relevant applications for a user. Moreover, when characteristics from various information domains are available at one place, the more complex user models can be built based on the more extensive and diverse potential information gathering possible through multiple, different applications.

There are also inherent drawbacks with sharing a user model. Each application uses its own (different) representations, terminology, granularity of stored information, requirements for higher or lower safety, etc. A promising approach for solving some of the mentioned problems is employment of mediating services for important user characteristics information from the various sources. Another problem arises when the applications using the shared user model evaluate some user's characteristics differently or reach different, conflicting conclusions about a user's interests or capabilities.

Many applications define user model as a single-unit model. In spite of this fact many models have been developed independently although through close examination we could find at least a small number of common char-

acteristics. Designing user model as an autonomous single-unit might cause reusability problems. Therefore, decomposition of the user model into the relevant parts seems to be the appropriate approach. In this approach it is necessary to identify characteristics which should belong to individual parts. This leads us to designing a general user model constructed from *domain-independent characteristics*, which reflect everything what is suitable for most of the users in different applications, and then address the remainder of the characteristics (i.e., *domain-dependent*).

To conclude, the key advantage of the shared user model is the availability of user characteristics discovered by multiple systems (e.g., in the Web environment) since user characteristics acquisition is considered to be the bottleneck of optimal personalization.

4.3 User model representation

There are several approaches to representing and storing a user model in web-based applications [2]. For user modeling it is important to analyze to what extent a particular representation is flexible for different kinds of user characteristics applied in a uniform manner together with the possibility of reasoning directed to decisions on information content presented to the user. We do not discuss representations that use proprietary formats as this would almost totally prevent the sharing and reuse of the user model.

Non-ontological representations

Markedly the most obvious is the use of a relational database to store data about the user since most applications already use this kind of data storage. In this case, the user model is represented as a set of database tables and user characteristics are mapped to attributes in the relational data model and stored values are assigned to individual user characteristics.

Using a relational database is quite straightforward, offers good performance, and several other advantages such as security, distribution ease, data recovery, etc. that result from good theoretical background of relational calculus and the maturity of its realization by database management systems. However, user models of web-based applications often contain semi-structured data as they use an overlay model, which follows the representation of the information space with various characteristics defined for concepts from the domain model. Relational databases are not primarily designed to express semi-structured data. Moreover, relational databases are not well

suited when frequent changes in data structure need to be performed, which is often the case in user modeling.

Another frequently used approach in current web-based adaptive applications is the representation of the user model by an XML based language using the file system, what results in powerful expressiveness. An example is the open source general-purpose adaptive web-based system AHA! [30]. The part of the user model which stores information about the user's name is defined in the AHA! as follows:

```
<record>
 <key>personal.name</key>
 <value>John Smith</value>
 <firsttimeupdated>false</firsttimeupdated>
</record>
```

The performance of this solution is limited by the performance of the used file system (i.e., it is effective for user models with few instances and a rich structure of user characteristics). Reusability and sharing is better than with the database approach, thanks to the platform independence of XML, while using XML has the advantage that it can be used directly in the Web environment. However, XML as a meta-language, defines only the general syntax without providing for formally defined semantics, which leads to difficulties when reasoning is used. Moreover, everyone can invent personalized names for tags; some developers store attributes as tags; some developers and applications use the attributes of tags defined by XML syntax.

Both of the above mentioned approaches offer only a way to describe user characteristics and do not offer any added value from the user modeling perspective. An ontology-based approach to user modeling offers a way to move user modeling from the low-level description of user characteristics to a higher level with additional possibilities.

Representing user model by ontology

The term *ontology* includes a whole range of various models with a wide variety of semantic richness (see Figure 3.4). In this work we focus on representing the ontology by RDF[1]/OWL[2] formalisms. An approach based on RDF and its extension OWL takes the previously mentioned XML representation (syntax) and eliminates its disadvantage by defining a vocabulary for

[1]Resource Description Framework, http://www.w3.org/RDF/
[2]Web Ontology Language, http://www.w3.org/2004/OWL/

describing properties and classes. OWL serves as a common language for automated reasoning about the content for the vision of the Semantic Web.

For illustration, shown below is a fragment representing a user's name and working experience that is a part of the ontology-based user model in the job offer domain:

```
<rdf:Description rdf:about="#name">
  <rdfs:label xml:lang="en">name</rdfs:label>
  <rdf:type rdf:resource=
   "http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Description>
<rdf:Description rdf:about="#hasExperience">
  <rdfs:label xml:lang="en">has working experience</rdfs:label>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource="http://www.fiit.sk/
    classification#ExperienceClassification"/>
  <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Description>
```

The characteristics are defined within *rdf:Description* elements, i.e. user's name and his/her working experience (*hasExperience*). The *name* characteristic is represented as a datatype property in the ontology (*rdf:type* element) and its expected value is a string. The *hasExperience* characteristic is represented as an object property and its value is an instance of the *Experience-Classification* class.

The advantages leading to using ontologies for user modeling come from the fundamentals of this formalism. Ontologies provide a common understanding of the domain to facilitate reuse and harmonization of different terminologies [50]. These characteristics support reasoning, which is considered as an important contribution to the ontology-based models. Once user characteristics are in ontological representation, the ontology and its relations, conditions and restrictions provide the basis for inferring additional user characteristics. For example, considering a user who is a programmer and works for a company that develops web-based applications using Java technologies we can infer that he/she is skillful in Java technologies.

By creating an ontology-based user model and deriving it from the domain ontology, we increase the probability that user characteristics will be shared among a range of systems of the same domain (especially on the Web,

where most ontologies are currently represented in OWL). We consider the sharing of user models as one of main advantages of using ontologies for user modeling. One of the most obvious advantages of a shared model is that one application can use the initialized data for personalization from other applications avoiding the requirement for the user entering the same information into every application (e.g., name, local settings).

As an example, let us consider a web-based application aimed at job offer acquisition. Let us project an example user whose education is in the field of information technologies with deep knowledge of the object-oriented paradigm of programming. As the user searches for a job, he/she visits another adaptive job offer application. Since the user model is shared between applications both applications have an access to information about user's education and automatically display offers seeking specialists on object-oriented design at the top of the results list.

Some authors believe that the solution to syntactical and structural differences between user models which interfere with sharing is in the use of commonly accepted user model ontology [40]. Since we agree that building common vocabularies is important and useful (we endorse the role of standards), considering a large distributed information space (e.g., the Web) we need to make a compromise between enabling diversity and looking for mappings between various models. The idea of a single commonly accepted user ontology is simply impossible to achieve in such diverse and distributed environment.

Certainly, a unified representation by ontologies can move the personalization on the Web further and give new possibilities of using user characteristics derived by other applications. Upon consideration of structural unification, a problem arises when applications using the shared user model evaluate some user characteristics differently. These characteristics can be changed constantly as the user accesses applications what can lead to unsuitable personalization in all applications using the respective characteristics. One solution to this problem is to keep track of model's changes [74]. This would allow each application to use this tracking as additional information for personalization.

4.4 Acquisition and maintenance of user characteristics

We have already mentioned that relevant and up to date characteristics describing the real user are needed in the user model to perform accurate per-

sonalization. Under the acquisition and maintenance of the user characteristics we understand the following:

Definition 4.3 *Acquisition of a user characteristic is the process when the characteristic is discovered and a value is assigned to it if it is not yet set.*

Definition 4.4 *Maintenance of a user characteristic means adjusting current value of the characteristic to reflect the user at that point in time.*

Processing of acquisition and maintenance of characteristics roughly follows these steps:

1. *Sources identification.* At this stage we need to identify sources from which we can extract information for the user model. Sources are mostly based on intuition and/or experience of the developers [53]. Their prior work and experiences are very valuable sources for the developing the user model. Designers use experiences and time-consuming research to identify all the important characteristics that ideally should be included in the user model. An extremely important part of this research is the communication with specialists in the area from the praxis which the user model is designed for.

2. *Collecting information.* There are two basic approaches to collecting information about the user [26]:

   (a) *explicit feedback*, which is based on asking the user to provide information (i.e., involving the user is necessary); whereas

   (b) *implicit feedback* exploits information acquired by observing a user's behavior and the user is not cognitively aware of the data collection during the work with the application.

3. *Analyzing and extracting information.* Information acquired by observation of the user's behavior [78] or implicit feedback requires further processing before it is stored in the user model as a characteristic.

4. *Changing the user model.* At this stage the user characteristics are stored in the user model. This process is known as:

   (a) *initialization* if there is no pre-existing characteristic in the user model; or

   (b) *actualization (or maintenance)* in the case when that characteristic already exists in the user model and only its value is changed.

Acquisition of new user characteristics can be helpful in overcoming the cold start problem. In the case of overlay user model there is no need to acquire new characteristics since the user model is filled with characteristics (possibly with assigned default values) for each respective concept from the application domain. Afterwards, the process of maintenance is employed to keep these characteristics up to date. In our proposed approach, feedback-based approaches are primarily used. Therefore, in following sections we take a closer look at the cold start problem and incorporating user feedback.

Cold start problem

We have identified two main sources of information to be used in the user model. Unfortunately, when the user starts using the application for the first time there is no information about him/her to provide successful personalization. This situation is known as a cold start problem.

When the personalization presented is not as user expects or needs to effectively use the application, it can deter the user from using the application further or returning to it in the future. We distinguish [57]:

- *new system cold start problem*, which is related to the new system where there is no initial information about any user, i.e. the user who uses the application for the first time is the first user since the application has been launched,
- *new user cold start problem* where the application has been already running for a while and there are records about the prior users of this application.

Asking the user questions would seem to be a logical solution to the cold start problem. However, the user who is presented with too many questions usually starts feeling bothered and is inclined to select random answers just to get through it as quickly as possible. Here, we point out how important it is to design clear and unobtrusive questions.

A partial solution to the problem is the provision of stereotypes where the user is assigned to the group (stereotype) after he/she has answered only a few questions. The stereotype reflects common characteristics, which are applicable for most of the group members. If the stereotype is assigned to the user, all these characteristics will appear in the user model even though some of them might not reflect that particular user.

The cold start problem could be also overcome by using default values. This approach is used very often, especially in educational applications. Most

of these type applications are dedicated to a group of students and here we can assume their previously achieved education level. The knowledge about the student's level of the concept is set as an initial value related to the particular characteristic when the user model is created.

User feedback

User feedback is used to elicit information from the user either implicitly or explicitly while he/she is working. The common example on *explicit feedback* is when the user is asked for an opinion about the presented content. Usually the user assigns a rating to the visited content from a defined scale. This evaluation feedback can be used for finding similar content, i.e. if the user likes some offer and assigns a high rating to it we can suppose that the user will also like similar ones. By explicit feedback the user can indicate either *positive* or *negative* interaction, e.g. when the user clicks on the button *"More details"* indicates a positive interaction or eventually negative interaction by clicking on the button *"Not interested"*.

When the user's work is not interrupted we are talking about *implicit feedback*. It requires collecting of evidence and making deductions before it is allowed to influence and update the user model. For example, a user reads detailed information about a content only to find it uninteresting or inapplicable after knowing all details. Implicit feedback often introduces inference errors into the user model. Explicit feedback is assigned a higher priority than implicit feedback because it actively involves the user [26], however implicit feedback is more preferred because it has little or no impact on the user's normal work activity [57].

If we focus on web-based applications, the user uses a web browser to access the content represented as web pages. During browsing the web pages the user reaches the content, which he/she does not want to read or just wants to go one or more steps back in the navigation. To accomplish this intent the user presses the *Back* button and proceeds to previous page. This provides very important information, e.g. from the pressing of the *Back* button we can infer information about user's interest in the content that is being presented. If the user spent enough long time before he pressed *Back* button we can further suppose that the presented content is of some level of interest and perform changes in the user model. By the same measure, a short duration on a web page with the Back button then being utilized can lead to the inference that the content was generally uninteresting to the user.

Usually there are limited options to eliciting personal information without asking. The commonly used approach is a questionnaire where the user has

several closed questions, i.e. prepared answers are provided and the user only needs to pick one or more presented answers that apply. The number of questions and the appropriate formulation of the question is very important to induce the user perform the feedback activities, which are not directly related to the goal, i.e. to work with the application. In the absence of easy and non-intrusive feedback tools the user could be prone to choose random answers or to only to go through the tool quickly without thoughtful consideration of the answer given. Similarly, in some circumstances the user does not know the correct answer of the questions (e.g., required knowledge is not achieved yet) or because of privacy [70] the user tries to avoid answering.

Another resource of information about the user is external sources, such as structured documents. An example of this type of document could be a structured curriculum vitae. Here, a specialized one-purpose tool is needed which is used to extract information from the document and store desired elements in the user model. The main disadvantage is developing general tools being capable of extracting information from various sources.

Observing the user's behavior or navigation in the information space is another source. The main advantage of this approach is that the user does not even have to know about it. This way we can find out details about the content activity, which web pages has the user read, when or what time has the user spent there, etc. Afterwards, we can infer user characteristics. Analysis of the user's behavior is insufficient if it is performed at the end of the session. It is necessary to do some analysis during the session [7].

At last, we should not forget the psychological aspect of answering questions. People see themselves in better light than they in fact are. In this case all the active solicitation effort is useless. Sometimes is better to work with smaller amount of information than with a larger set that is inaccurate. Therefore, the conclusions reached are that questionnaires should be unobtrusive as much as possible, so as to not discourage the user from using the application and they should be structured to guide the user to provide the answers honestly and realistically.

Inaccuracies in the user model

Information provided explicitly by the user is considered as the most confident source [47]. The problem emerges when the information about the user is inferred or is automatically acquired from the observing of the user's behavior. Thus, not only values that are assigned to the particular characteristics can be inferred, but also new characteristics. This is especially true when the stereotype model is used. The user becomes a member of the group

even though he/she probably does not have some of the global characteristics that represent the particular stereotype.

Observing a user's behavior and following inferring user's characteristics is interesting from the researcher point of view, but untrustworthy in the user's eyes. Diffidence may cause that user stops using the adaptive application because there is no user control about what kind of information are stored and applied in the user model. To avoid the problem with trust in the user model, the user should have an opportunity to check the content of the user model through some kind of user-friendly interface [48]. In the user model it is important to distinguish and also indicate for each characteristic and its value whether it was acquired directly from the user or it was inferred from indirect sources.

A special case that may sometimes appear and that causes inaccuracy in the user model is when the user allows another person to work with the application on user's behalf [49]. This will likely case the unsatisfactory personalization.

4.5 Open problems

We consider the simplification of exchanging user model data between different applications as the major advantage of using ontologies. Separating the domain-independent part of user characteristics allows building a general user model. This kind of the user model can be used in a wide-range of applications and can simplify sharing the user model.

If we focus on commonly used user models, the main drawback of the overlay user model is the necessity of its initialization. The way to overcome this problem is a combination of the overlay user model with the stereotype user model. However, the stereotype model is prone to inaccuracy due to the need for heavy reliance on inferences. Even for a stereotype that has been selected and applied correctly, some inappropriate characteristics can be inferred for particular user.

Current approaches to acquisition and maintenance of user characteristics are mostly based on user's feedback. Although these approaches are much more automatic than they were in the past in current adaptive applications there is still a substantial requirement for designer involvement, e.g. defining rules. These rules are defined for a particular application and usually they are not transferable to other applications (i.e., reusable).

The main drawback of methods for acquisition and maintenance of user characteristics is that they are not designed with regard to domain indepen-

dency and reusability. Furthermore, there is not enough attention paid to these methods, in the user modeling field, since the way the characteristics are changed in the user model is usually only supported by these methods, i.e. it is not directly visible on the application's presentation layer.

An effective user model, as a result of the user modeling process, is the critical outcome element if we want to achieve an efficient and improved personalized approach to the information content presentation. We have identified several research areas where open problems related to the user modeling exist:

- *User modeling in an open information space.* Most adaptive applications are intended to be used in a closed information space. Modeling the user when the application domain is closed and well-known is much easier. An example user model developed in a closed information space could serve similar adaptive applications for educational purposes that are structured in a similar manner in other open information spaces. If we have the well-defined information space (e.g., defined by an ontology) we can use a mapping between the existing user ontology and the domain ontology to build the new user model automatically.

- *Shared user model.* Dividing the user model into domain-independent and domain-specific parts is a promising approach to reuse available information among adaptive applications. To achieve this goal an ontological representation of the user model can be employed. Furthermore, there is a problem of finding a suitable way of mapping between the user model and domain models of applications.

- *Dynamics of the user model.* The values assigned to the characteristics in the user model change as the characteristics change during the lifetime of the model. There is a need to study and identify which factors cause the most significant changes in the user model and to propose adequate methods that will reflect these dynamic changes in the user model.

- *User model in a social environment.* The need of people to create social groups with regard to their interests in real life causes applications based on social aspect to become more popular. Here, a wide variety of information about a particular user can be discovered if one's relationships to other users are researched and analyzed.

- *Acquisition and maintenance of the user model.* To provide a successful personalization a user model populated with the characteristics that are

up to date is desired. A variety of approaches can be used, e.g. implicit and explicit feedback. It is important to note that there are other sources of information that can be used to extract information for the user model (e.g., logs). In our work we focus primarily on methods aimed at acquisition of the information to be used for the user model and its maintenance.

In this work we propose a set of novel methods for acquisition and maintenance of user characteristics in the user model. These approaches are designed with regard to reusability. We designed these methods with regard to their domain independency and also with the aim to be automatic as much as possible.

Part III

Methods for Acquisition and Maintenance of User Model

Chapter 5

Acquisition of user characteristics based on questions

> The method presented in this chapter was proposed in cooperation with Tomáš Klempa as a part of his master thesis supervised by the author of this work.

In the adaptive applications, if the overlay user model is employed for personalization purposes, there is a copy of each concept from the domain model stored in the user model. Each such a record about the concept in the user model needs to be initialized before it is used and later needs to be maintained updated. Traditional approaches to initialization and maintenance of the user model commonly consider a closed information space and are proposed for the particular application. Dynamics of changes of the information space makes commonly used approaches unsuitable in Semantic Web applications. The limitations of traditional solutions (e.g., their domain dependency) raise a need for novel methods of initialization and maintenance of the user model.

A novel method described in this chapter is suitable not only for acquisition of user characteristics but also their maintenance is possible as well. It employs an explicit approach — dynamically generated questions and answers. Motivation for the method was the M-PIRO project that uses meta-data to generate text in natural language [4]. In the course of this project a tool was developed that is capable of generating texts according to additional information about entities stored in the database.

In [55] there is an approach to planning utterances in natural language that can dynamically use context information about the environment in which a dialog is located. Similarly, in the educational system OntoAIMS [31] a *dialog agent* is used to maintain dialog aimed at achieving the goal and

a *user interface* which allows construction and modification of utterances graphically.

The proposed method uses a model of application domain represented by ontology and its concepts to generate questions in natural language (here English) aimed at acquiring user characteristics for the user model. The process of question generation is driven by defined rules. Assigning priorities to characteristics provides better control over question generation, e.g. allows suppressing generating of a question if necessary.

5.1 Principle of generating a question

An explicit way of acquiring and maintaining characteristics in the user model employs direct responses from the user. It can be accomplished by filling in forms, answering questionnaires, questions, etc. We distinguish questions:

1. that can be answered by filling their missing part; and
2. can be answered with a positive or negative answer.

We deduce that a characteristic of the user, we are interested in, is a part of the question. An example of illustrating questions and answers is provided in the Figure 5.1. It demonstrates four types of questions and related answers. Each question contains the *name of a characteristic* from the user model that is emphasized by the underlined font. The answer can be a number, text, ordinal value, etc. The name of the characteristic can be also contained in the answer.

Q: What is your name?
A: My name is ***John***.
Q: Do you have any driving experience?
A: ***Yes***, I do.
Q: How often are you willing to travel for a job?
A: ***Every day***.
Q: What is your expected salary range.
A: My expected salary is from ***25.000*** to ***40.000***.

Figure 5.1: Example on questions and related types of answers.

To be able to generate such questions we employ templates. A template is a predefined representation of a question that uses specific vocabulary elements. The author of the template defines an order of the words that can be used in the template. Instead of the name of the characteristic a special identifier is used in the template. Following context relates to the question:

– circumstances, which invoke generating of the question; and

– an object specifying which question should be generated and what about.

The circumstances of the question are handled by user defined rules and the object depends on the particular characteristic being researched for the user model. A simplified principle of the generating of a question is depicted in Figure 5.2.

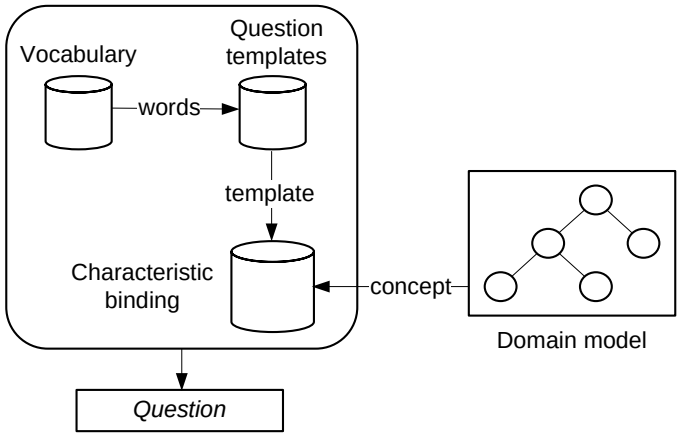

Figure 5.2: Principle of the generating of a question.

At the very beginning, a list of properties for a concept from the domain model is acquired. For instance, if job offers is the application domain, then properties of the job offer concept will be added to the list. Each property relates to a characteristic for which a question will be generated (i.e., it will appear as an instance of a particular characteristic in the overlay user model). Afterwards, properties can be used for:

– *acquisition*, if an instance of respective characteristic does not exist yet in the user model; or

– *maintenance*, if an instance of respective characteristic already exists.

Unique names and priorities assigned to a question along with its structure are stored in *Characteristic binding*. Furthermore, a binding specifies a *template* and a *noun* which will be used in the generated question as a name of the user characteristic. A suitable template for the characteristic is selected from *Question templates* according to its unique identifier. Words to be used in the question are stored in *Vocabulary*. Afterwards, a question is generated

for the user. The user is also provided with a part of an answer in the case the answer is of *ordinal value* or an *option* type.

The process of acquiring and creating of a new instance of the user characteristic differs from maintenance of existing one. In the case of creating a new instance of a characteristic (i.e., acquisition) all the concept's properties from *Characteristic binding* that do not have an instance in the user model yet are acquired. If we deal with the maintenance, properties are acquired with regard to *rules fulfillment*. In both cases, the list contains uniquely named properties that are sorted according to a predetermined priority. Employing a priority hierarchy also allows response to the cases when it is appropriate to suppress the generation of a question (e.g., gender of the user is a characteristic that needs to be acquired only once).

Vocabulary

Our first intention was to use ontology that stores words as instances where each word was specified as a part of speech. We gave up this approach because of high requirements on filling such an ontology and low efficiency of existing software support provided for ontologies thus the use of this approach would have caused a negative impact on generating questions.

We employ a vocabulary that was built in the course of the project *Text Chunking*[1] aimed at dividing a text into syntactically correlated parts of words. The vocabulary (words in English) we use is represented in the following form where each word is in a new line:

```
identifier;word;POS tag
```

The first column is a unique identifier followed by a word that will be used in the template. In the generated question the *identifier* will be replaced by *word*. The last column is the *Part Of Speech* (POS) tag. In some cases the POS tag can also express a grammatical category as well (e.g., NN is a noun in singular). The vocabulary contains approximately 17 000 words. The POS tags specify categories using the following grammatical groupings[2]:

– *number* — a grammatical category for the forms of nouns and pronouns and verbs that are used depending on the number of entities involved (singular or dual or plural),

[1] Text Chunking, http://www.cnts.ua.ac.be/conll2000/chunking/
[2] TheFreeDictionary, http://www.thefreedictionary.com

– *tense* — a grammatical category of verbs used to express distinctions of time,

– *person* — a grammatical category used in the classification of pronouns, possessive determiners, and verb forms according to whether they indicate the speaker, the addressee, or a third party.

The main advantage of using this kind of representation is that we know the semantics of particular words used by the questions. It also provides higher efficiency than could be achieved employing ontological representation of Part Of Speech. The available vocabulary contains a satisfactory amount of words. Therefore, further maintenance of vocabulary is not expected, however, it would be easy to update, considering the form it uses.

Sentence templates

We separate vocabulary from questions' templates in order to address the reusability of questions and also as an option to be able to replace vocabulary if necessary. A template uses XML representation where identifiers of words from vocabulary are used instead of individual words. The following example illustrates a template of a question named *ValueTemplate01* which can be used for questions where a value is expected:

```
<question name="ValueTemplate01">
  <pos order="1" instanceName="12456"/>
  <pos order="2" instanceName="7398"/>
  <pos order="3" instanceName="15432"/>
  <pos order="4" instanceName="n01"/>
</question>
```

The element *question* contains elements *pos* which specify words from the vocabulary to be used in the question. Each word in the vocabulary is identified by its identifier and is assigned a position (using an attribute *order*) which will be used in the question. In the above example identifiers 12456, 7398 and 15432 would be replaced by words "what", "is" and "your" respectively. Identifier *n01* would be replaced with the name of the characteristic.

We proposed several templates to be used in the domain of job offers and scientific publications. With the growing amount of available templates, the conversation with the user becomes more natural. However, the case when too many templates are provided is not very reasonable, e.g. creating one template for each question is not very effective. Therefore, templates were proposed with regard to their universality and reusability.

5.2 Binding characteristics

If we had all the necessary data (e.g., priority of a characteristic) to generate a question stored in the ontological repository we would have to access the repository to perform any change. Because of the small performance of existing (ontology) solutions, we use characteristics binding for practical reasons. This assures we access the ontological repository only to acquire characteristic's value, to create new instances of a characteristic, or to update an existing user characteristic and all other data store in the XML.

Binding characteristics provides additional information for the user model that is necessary to generate a question. Such additional information for a characteristic are:

- unique identifier;
- priority;
- template; and
- noun used in the sentence, which specifies a name of the characteristic (it does not have to be unique).

A characteristic can be represented in the ontology as a set of concepts. When binding its structure we consider following:

- namespaces;
- classes;
- datatype or object properties; and
- instances.

Simple characteristics are represented as RDF triples, i.e. subject — predicate — object, which relate to a class the characteristic belongs to, name and value of the characteristic respectively. Such a structure is easy to bind, but this is not applicable to more complex user characteristics where characteristics can have relationships to others, can create taxonomies, or a characteristic which can get an ordinal value; or which can be represented as a class or an instance.

It is more complicated considering such characteristics when we create a new or we update an existing characteristic. Figure 5.3 depicts a structure

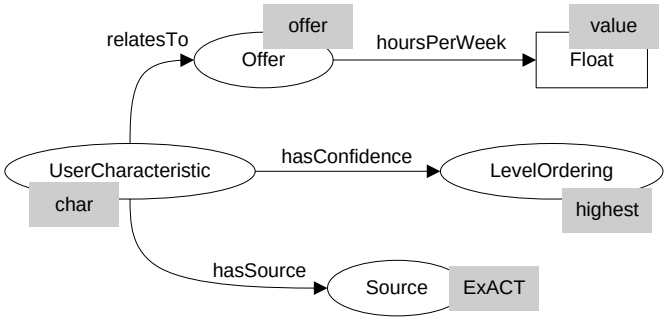

Figure 5.3: Example of a structure that represents preferred number of working hours within a week.

of a part of user model — a characteristic representing the preferred number of working hours in a week. The characteristic includes an additional property which is set when the characteristic is processed — confidence.

For illustration, we substitute full URIs identifying classes and properties in the figure with simple labels. The class *UserCharacteristic* represents a particular characteristic that is in a relationship with a job offer (*Offer*). It has a datatype property *hoursPerWeek*, which is of the floating number type. Moreover, the characteristic has a relationship expressing its confidence, which is represented as an instance of the class *LevelOrdering*, and *hasSource* that specifies the source that caused its change. Gray rectangles illustrate instances (their labels) that should be created after a user answers a question.

We propose a way of assigning names to instances in regard to the structure of concepts. When creating a new instance of a characteristic, e.g. the instance *char* of the class *UserCharacteristic* is assigned an additional suffix — timestamp. We call this way of naming instances a *pattern*. Similarly, we name the instance *offer* of the class *Offer*. On the other hand, an instance *highest* (employed to express a level of confidence) of the class *LevelOrdering* already exists in the ontology and therefore a suffix is not applied. Finally, a property *hasSource* specifies a tool that processed the characteristic as the last one. In this case, a name of the tool is used (e.g., ExACT).

To conclude, instances can be named as follows:

- *using pattern* — an instance is created according to the defined pattern with a timestamp suffix, which format is configurable;
- *adjusting desired value* — when creating an instance, an ordinal, text or numerical value can be assigned; and
- *name use* — a defined name is used to name an instance.

We use unique names (i.e., keys) to represent particular bindings which are stored in *Characteristic binding*. A name of a key consists of an *identifier* specifying a type of binding and a *postfix*, which is a unique number assigned to the key. There are several possible identifiers that can be used, namely *c* (class), *op* (object property), *dp* (datatype property), or *i* (instance).

Every key is assigned a symbolic name, namespace (in a prefix form), instance's name and its type. A part of binding data is a list of namespaces to allow binding from prefix form to full URI form. Employing keys in bindings improves its flexibility especially in the cases of changes in the domain or user ontology. In the Figure 5.4 an example is depicted from the Figure 5.3 with keys assigned to individual parts of characteristic's structure.

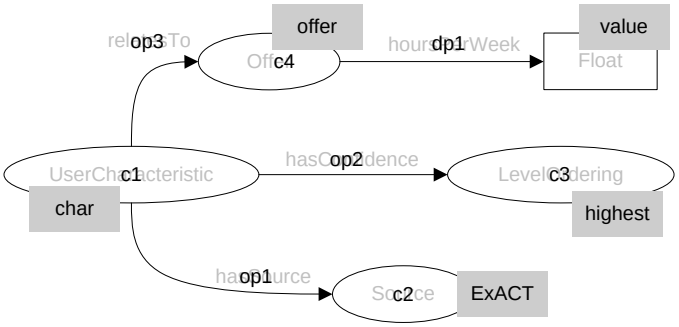

Figure 5.4: A characteristic with assigned keys.

If a characteristic contains a complex structure of concepts, i.e. relationships between classes and instances, binding would not be very effective in term of the user characteristic's structure. Therefore, only object and datatype properties are acquired. Domains and ranges for properties are acquired separately by employing specific queries.

However, in some cases, a bound property can contain more domains or ranges. Here, it is necessary to define a particular domain or range that will be used to create an instance of a specific characteristic explicitly. Every occurrence of a definition is related to the particular characteristic and has no influence on other characteristics that contain that property.

5.3 Rules for question generation

We use a rule based approach to manage characteristics. An advantage of this approach is simplicity of creating and maintenance of such rules. We employ a well known principle for rules: *IF* [condition] *THEN* [action]. Required

parts of the condition are a characteristic from the user model, operator, and parameter of the condition. A default action generates a question.

A condition is defined according to data types, i.e. literals and instances which represent a value of a characteristic. Literal values can be of several types defined according XML Schema. An instance can get one of the following values:

- *ordinal value* — it is not represented as a number, but values can be compared and mutually sorted; or
- *option* — when a value occurs that is not measurable (e.g., level of programming skills in Java).

Operators, such as *less*, *greater*, *equal* as well as special operator *instanceOf*, can be used in a condition. The meaning of first three operators is obvious. The operator *instanceOf* is employed when a value is of the type *option* and expresses a condition in relationship between an instance and a class. In the case of literals, a suitable type has to be set as the parameter of a condition (e.g., for a number a numerical value has to be set, for string a string has to be given). In the case of date type, an additional property defining a unit has to be specified and obtains values from the minute up to the year. In the case of an object property, values are identifiers of existing instances or classes from the ontology.

Well defined rules initiate generation of a question in a right time, e.g. if a characteristic is older than given time period. Another example is generating questions with regard to a tool that is responsible for creating or updates of the characteristic. Employing priority allows also reaction to the cases when it is appropriate to suppress the generation of a question (e.g., gender of the user is a characteristic that needs to be acquired only once).

5.4 Evaluation

The proposed method aimed at acquisition and maintenance of user characteristics in the user model is based on generating questions in natural language (here English). We employ concepts from the domain model in the process of question generation.

To evaluate the proposed method a software tool called Explicit Actualizer (abbreviated *ExACT*) was developed. It was implemented in Java SE 6 and uses Sesame[3] framework to access ontological models which are repre-

[3]Sesame, http://www.openrdf.org/

sented in OWL DL. Figure 5.5 depicts output from the tool, i.e. generated question with answers.

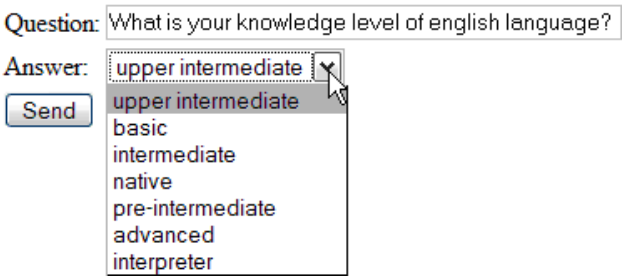

Figure 5.5: User interface for a question with multiple answers.

The method was evaluated using the job offer domain ontology narrowed to information technologies field (created in the course of the project NAZOU [59]) and ontology of scientific publications (project MAPEKUS [18]) with respecting user models (represented as ontology). During the evaluation we focused on creating new characteristics (i.e., initialization of the user model when the user accesses an application for the first time) and maintenance of existing characteristics in the user model.

In both domains, we bound selected characteristics and created templates to generate questions. A part of characteristic with binding information is shown in the following listing:

```
<properties characteristic="hoursPerWeek" priority="2">
  <property name="oa3" noun="none" sentenceTemplate="none"
    domainConcept="c1" rangeConcept="c4" increment="false"/>
  <property name="oa2" noun="none" sentenceTemplate="none"
    domainConcept="c1" rangeConcept="c3" increment="false">
    <setOptionValue name="c3" value="da4"/>
  </property>
  <property name="da1" noun="hours per week"
   sentenceTemplate="QuantityTemplate" domainConcept="c4"
   rangeConcept="none"/>
</properties>
```

A structure of a characteristic is defined within the element *properties*. Its attribute *characteristic* specifies a name of the characteristic. Datatype and object type properties related to the characteristic are described using an element *property*, which allows definition of other attributes. Namely, *name* is an identifier, *noun* specifies a part which will be included in the question, *sentenceTemplate* is a name of a question's template, *domainConcept* and

rangeConcept is the domain and range respectively. Domain and range have a default value set to *none*. Attribute *increment* is applicable if a value of datatype property is of a numeric type, e.g. number of updates for a characteristic. In the case, when attribute *increment* is *true*, the related value of datatype property is incremented whenever a characteristic is updated. An element *setOptionValue* sets a value of a type options within the range of the element *property*.

Furthermore, rules to manage generating questions were defined. Examples of proposed rules are shown in the following listing:

```
<rule type="dateTime">
  <condition parameter="gu:hasTimeStamp" operator="less"
    value="1" unit="week"/>
  <action type="ask"/>
</rule>
<rule type="ordinalValue">
  <condition parameter="c:hasConfidence" operator="less"
    value="c:_loAverage"/>
  <action type="ask"/>
</rule>
```

The templates were proposed with regard to their universality, i.e. our goal was to design templates to be used for as many questions as possible. With the growing amount of available templates, the conversation with the user becomes more natural. However, the case when too many templates are provided is not very reasonable, e.g. creating one template for each question is not very effective.

We designed 7 templates for the job offer domain. These templates allow for generation of questions addressing 22 characteristics. For the scientific publications domain we proposed 3 templates which allow generation of 5 questions. The small amount of templates (questions) in this domain is influenced by the straightforward structure of the domain.

Actualization of characteristics directly influenced approximately 10–16 statements in the user model (ontology). Since the ontology repository does not provide satisfactory performance as for instance relational repositories, we also evaluate time demands. For simpler characteristics (with 10 statements) actualization took an average of 4.7 seconds, whereas actualization of characteristics with 16 statements took 6.4 seconds on average.

We performed another experiment on a travel ontology to verify domain independency of the method. We wanted to generate a question to find out the user's opinion about accommodation (represented by class *Accommodation*) in a particular destination (class *Destination*). The result is that

the method, as it is proposed, is not suitable for characteristics where two independent instances need to be created since we expect one name of a characteristic in the question.

5.5 Discussion

One way of acquiring information for the user model is employment of the user feedback. The main problem of feedback based solutions is that necessary questions are mostly hard coded for a particular application. However, there are some solutions that use generation of the questions, such as dialog systems. We focus on approaches that can be used for feedback generation and/or employ ontologies.

Dialogue systems can provide relatively open-ended prompts, i.e. users may provide responses that are not an exact fit with what is expected [58]. In this approach rich ontological structure of the domain is crucial for building dialogue systems. Employment of ontology helps to exploit well known relationships (e.g., synonyms, hypernyms, hyponyms) to achieve more natural communication. Sentences are generated according to templates. The templates are of the three types, i.e. *property*, *entity* and *process*. The type of the template as well as the use of proper terms is driven by rules.

In [55] there is described an approach to planning utterances in natural language that can dynamically use context information about the environment in which a dialog is located. The process of planning is guided by a *chooser*. It specifies entry conditions and actions associated with different choices what is similar to our method where we use *if-then* rules.

The educational system OntoAIMS [31] is built upon the OWL-OLM (an OWL-based open learner model). OWL-OML uses a *dialog agent* to maintain dialog with regard to the prepared questions aimed at achieving the goal and a *user interface* which allows construction and modification of utterances graphically. The used dialog framework is domain independent. There are two types of questions: *yes/no questions* and *open questions*. The questions are generated according to the domain concepts and can be answered by transforming them into queries. The answer is checked by the agent whether it is supported by the domain ontology. If the OWL statement is incorrect, the user is notified that his or her question can not be answered. The goal of this approach is to improve user's knowledge in a particular domain using dialog. Our method generates questions according to a concept's properties and provides the user with prepared answer considering the type of the question.

An approach, described in [65], distinguishes *student feedback* given to a student during learning, *author feedback* given to an author during course authoring and *group feedback* given from a group of learners who study a course to an author. This approach is suitable for domains with hierarchically structured terms. The aim is to provide semantically rich feedback to authors and learners. The feedback is provided according to an ontology structure and can be of *generic* or *specific* type. The generic feedback is independent from the ontologies being used and makes this approach available for other applications. On the other hand, the specific feedback requires user's involvement (it is defined by the author). The feedback is given about completeness and correctness of the list of studied domain concepts and relations using different dialog patterns. The advantage is that the feedback is generated automatically (not hard coded) with regard to domain concepts however its purpose is to improve learning process not to be used for the user model.

The M-PIRO project uses meta-data to generate text in natural language [4]. Application domain is an electronic museum where multilingual personalized descriptions of exhibits are produced from non-linguistic information in the database. From the personalization perspective, the user is avoided from being presented repeating information that he/she has already seen. M-PIRO is build upon the ILEX natural language generation system [64]. In the course of this project a tool was developed that is capable of generating texts according to additional information about existing entities. The text generation is processed in four stages. The first stage is *content selection* where the most appropriate facts are selected to be delivered to the user with regard to stereotypes and user's history of visited content. The next stage, *document planning*, outputs an overall document structure with the desired sequence of the facts and rhetorical relations. The *micro-planning* stage specifies how a fact can be expressed as a clause, which verb to use, in what tense, etc. Here, templates are employed. The last stage, *surface realization*, is responsible for producing the final textual form of the descriptions (e.g., word forms, placing the various constituents (subject, verb, object, adverbials) in the correct order, accounting for number and gender agreement). This project served as a motivation for our method (e.g., employing templates). Our method generates questions with regard to ontological concepts.

To sum up, the proposed method can be used for acquisition and also for maintenance of the user model. The main advantage of the method is that questions are generated automatically for concepts from the domain model. Afterwards, user's answers are transformed into characteristics in

the user model. The entire process (i.e., *when* and *what* question needs to be generated) is driven by user defined rules that can be restricted by question's priority.

Chapter 6

Acquisition of user characteristics based on content analysis

The analysis of the content presented to users is a suitable source of information [25] for personalized applications where the personalization of visible aspects is usually based on user characteristics represented in a user model. To provide proper personalization the user model needs to be populated with meaningful user characteristics that are up to date, which can be obtained explicitly from the user (e.g., by filling forms) or by observing user behavior (implicit feedback), or by data mining from activity logs.

If a user's rating of the displayed content is known (i.e., user's interest) we can acquire some characteristics by employing content analysis in combination with similarity. Since the rating varies, similarity can help to analyze the reasons why it is low or high. For instance, let us consider a job offer in the information technology field. One can find hundreds of offers on the Web that advertise a position for Java programmers requiring high school education with at least three years of previous experience, knowing basics in Web technologies and providing a motivating salary. If two offers are similar in all properties except the location and have different ratings (e.g., one in London and one in Washington, D.C.), the difference in rating was likely caused by the location property. It is unimportant whether a user from Europe prefers to work in Europe (high rating for job offer in London) or whether he/she is an adventurer who wants to try an overseas job (high rating for Washington).

From different ratings given to different content we can deduce values of specific user characteristics and thus populate the user model. Higher value of interest given to the offer located in Washington, D.C. could reveal that the location is an important property for the user, while equivalent ratings given to different content could reveal that location is irrelevant for the user since

it had no influence on the ratings. While the presented examples were rather simple, more general information about user preferences can be discovered using more sophisticated heuristics.

People declare judgments about objects without realizing similarity. For the purposes of this work we define similarity between two objects formally.

Definition 6.1 *Let us assume we have two objects x and y. Then the following statements are valid for the similarity between these objects [19]:*

- $sim\,(x, y) \in [0..1]$*;*
- $sim\,(x, y) = 1 \rightarrow x = y$*: similarity of identical objects;*
- $sim\,(x, y) = 0$*: similarity of entirely different objects;*
- $sim\,(x, x) = 1$*: similarity is reflexive;*
- $sim\,(x, y) = sim\,(y, x)$*: the similarity is symmetric;*
- *similarity and distance are mutually inverse;*
- $sim\,(x, z) \leq sim\,(x, y) + sim\,(y, z)$*: the triangle inequality.*

There exists a diversity in views regarding symmetry in similarity. Whereas according to [19] the similarity is symmetric, on the other hand, Tversky claims that symmetry is the choice of similarity and apparently, direction of asymmetry is determined by the relative salience of the stimuli; the variant is more similar to the prototype than vice versa [75]. He demonstrates it on an example that judged similarity of North Korea to Red China exceeds the judged similarity of Red China to North Korea.

Another related term, which is used often, is *semantic similarity*. We distinguish *probability-based* [69] and *structure-based* [68] approaches to the semantic similarity. The probability-based approach depends on the frequency of the terms occurrence, whereas the structure-based approach considers a structure of compared objects (e.g., taxonomy). Both approaches consider means of the ontological representation for computing similarity.

Definition 6.2 *The semantic similarity is a concept where likeness of compared objects is determined considering their content while different metrics are used with regard to the meaning of the content.*

We present a method for the comparison of instances of ontological concepts aimed at the identification of common and different aspects for personalization purposes. The method exploits the advantage of ontological information

representation and computes instance similarity with regard to particular properties of concepts. In personalized applications where the user model is available, the method also supports more accurate similarity computation for individual users according to their characteristics. Our method is designed with regard to domain independency to be available for using in arbitrary domains. Inputs and outputs of the method are depicted in the Figure 6.1.

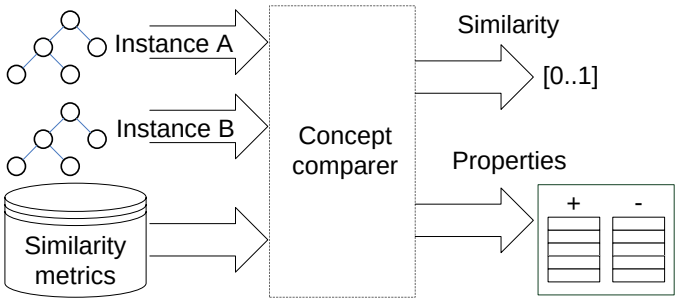

Figure 6.1: Inputs and outputs of the method.

The input of the method is two instances (Figure 3.6) and a set of similarity metrics that can be used. The output is quantitatively expressed similarity measure between instances and an output object that contains two sets of properties — a positive and a negative.

6.1 Comparison by recursive traversing of ontology instances

The main idea of the method comparing ontology instances is based on the evaluation of common property pairs present in both instances. The rough principle of the method illustrating comparison of two instances *instanceA* and *instanceB* is shown in Algorithm 1.

When comparing two instances of the concepts, properties can appear in different cardinalities:

- single in both instances;
- multiple in both instances; or
- single/multiple in one instance only.

When the property has a single occurrence in both instances, then the similarity of related elements (*instances* in the case of object properties or *literals*

Algorithm 1 Recursive method basics

```
function getSimilarity(instanceA, instanceB)
    set similarity to 0.0
    set counter to 0
    store properties for instanceA and instanceB to properties

    foreach property in properties do
        increment counter
        if property is in both instances then
            store connected elements to elementX and elementY
            add computeSimilarity(elementX, elementY) to similarity
        else
            add 0.0 to similarity
        end if
    end foreach

    return similarity/counter
end function

function computeSimilarity(elementX, elementY)
    if property is datatype then
        return getDatatypeSimilarity(elementX, elementY)
    else
        set similarity to 0.0
        add getObjectSimilarity(elementX, elementY) to similarity
        add getSimilarity(elementX, elementY) to similarity
        return mean value of similarity
    end if
end function
```

in the case of datatype properties) is evaluated using different similarity metrics. The comparison of *datatype* properties ends after a metric is used to compute the similarity measure between the related literals. For *object* properties a metric for related instances is computed (e.g., *taxonomy distance*) and further comparison is performed recursively on the respective instances until literals are reached or until there are no properties left.

When an instance is being traversed recursively, an *inverse property* can connect it to an already traversed instance. For instance, London is connected to a job offer with the property *hasDutyLocation*. Since more than one job offer can be located in London, we require all of them to refer to the same instance (e.g., London *isDutyLocationOf* other job offers). If we do not

consider inverse or symmetric properties, the algorithm will traverse them and enter an infinite loop. Therefore, we filter out inverse and symmetric properties related to the examined property. However, loops can still occur, e.g. if two different properties lead to the same instance. In such cases, the already traversed instances are omitted and further traversing stops.

Multiple occurrences of properties (e.g., *hasPrerequisite*) in an instance are the most complex case we address. In this case, two sets are constructed which contain elements connected to the examined property in the first and second instance respectively. These two sets can have different cardinalities — the problem is to identify (i.e., to match) similar elements between these two sets. We use our similarity measure to identify such element pairs, which are then compared and the computed similarity contributes to the total similarity. However, the identified pairs do not provide satisfactory results in some cases. For example, if in the first instance the *hasPrerequisite* property has the value *"Java or C programming"* and in the second instance multiple values *"Java programming"* and *"C programming"* consistent results are difficult to achieve. In our approach a pair with higher similarity according to the used similarity metric is selected (i.e., similarity with only one property's value from the second instances is considered), but more complex heuristics can be proposed and employed to identify a $1 : n$ mapping.

If single or multiple occurrence of a property occurs only in one instance, we estimate similarity of values attached to the property as equal zero. It is based on the similarity definition, i.e. the similarity equals zero if two objects are entirely different. Here, we assume that instances are entirely different in the property, since a value is assigned to the property in one instance only.

6.2 Comparison metrics

A variety of comparison metrics can be used to compute similarities between instances or literals connected to a property. We proposed two groups of metrics with regard to a property's type since they must be treated differently due to their different nature — *datatype* and *object* metrics.

Datatype metrics

To compute similarity between literals connected to a datatype property any string based metrics can be used[1]. However, to achieve better results, the

[1] A collection of methods suitable for string comparison is implemented in the open source library SimMetrics, http://www.dcs.shef.ac.uk/~sam/simmetrics.html

literal type should be considered (e.g., simple string, date, number, logical value).

If the literal type is a *logical value* there are only two possible values that can be set — *true* or *false*. Although the comparison of two logical values with any string metrics as two strings would result in a value in the range $\langle 0, 1 \rangle$, the similarity of two logical values can be either 1 or 0. Let x and y denote logical values. The similarity measure for literals of logical type is computed as follows:

$$sim_{logical}(x, y) = \begin{cases} 1.0 & x = y \\ 0.0 & otherwise \end{cases} . \tag{6.1}$$

Comparing literals of *date* and *number* type is strongly dependent on the application domain and context. In an ancient history domain a time difference of two centuries would likely be considered similar, while grocery expiration dates would not. Therefore, we propose the following similarity metrics for date similarity:

$$sim_{date}(x, y) = \begin{cases} 0.0 & |x - y| \geq N \\ 1.0 - \dfrac{|x - y|}{N} & otherwise \end{cases} , \tag{6.2}$$

where x, y denote dates and N is a number expressing precision or time period that is reasonable to be considered while comparing dates. N denotes the number of the smallest time units that are still meaningful with regard to the chosen time period. For instance, for a job offer, one year is a period that is still meaningful since typical properties with date type values are the *start date* or the *validity* of an offer. Therefore, N is set to 365 and absolute value of distance between the respective dates is also expressed in the same units (here days).

Similar problems emerge when *numbers* are compared. Specifying precision (or range) for numbers is difficult as we usually do not know what the range of compared numbers can be. Furthermore, it is likely that numbers will occur in one instance in several contexts, i.e. in the job offer domain the number of working hours (dozens) or the salary (dozens of thousands or even hundreds depending on the period) have different possible ranges and units (e.g., hours, currencies).

However, this can be solved if additional information about the range is present, such as a property specifying the type of units. This can be helpful if value normalization before comparison is performed and heuristics can be also used. In [67] an approach is described which is aimed at the normalization of values that are expressed in different units (e.g., the salary

in various currencies). The approach is based on logical programming and two implementations are provided — ASP and Prolog. Employing Prolog is more suitable for normalization of values where different units were used. In our experimental evaluation, we employed this approach to normalize numeric values in our dataset.

Another problem that should be mentioned is that a number can occur in *positive* as well as *negative* form. People tend to think in positive numbers [41] which is more natural. For instance, people do not say "I have gained minus 5 kilograms", but they automatically change statement to avoid using negative numbers to "I have lost 5 kilograms". Properties that generate positive and also negative values are rarer than properties generating either positive or negative values only. We consider the similarity measure of two opposite numbers as equal 0 because of their different nature. Similarity between two numbers is expressed according to their distance but, in particular cases, the context is also important, e.g. a difference of 2 degrees Celsius when the temperature is 20 degrees is not quite the same as when the temperature is 0 degrees when water in liquid state turns into ice. The similarity measure of two numbers x and y that are both either positive or negative and considers particularity of the job offer domain is computed as follows:

$$sim_{numerical}(x, y) = 1.0 - \frac{||x| - |y||}{max(|x|, |y|)}. \tag{6.3}$$

When comparing *strings*, a simple comparison provided by a default method in any programming language (e.g., in Java) does not give satisfactory results. Methods *equals* and *equalsIgnoreCase* are provided in Java implicitly but they return a *Boolean* value instead of a similarity measure. To compare strings we employ *Levenshtein* similarity metric from the SimMetrics library that uses decapitalised strings as input.

Object metrics

When computing the similarity of instances connected to an object property their other characteristics can be considered (e.g., the number of related properties and their types or the position in the taxonomy) [3, 69]. Since instances in ontologies can belong to multiple classes simultaneously, one way of measuring the similarity at the class level is to determine the number of common and different classes they belong to. Consequently, if two instances belong to several classes simultaneously, they are more similar than instances that have no common class (except the base class, i.e. *owl:thing*).

Taxonomy distance is a heuristic similarity measure for evaluating similarity between instances of the concepts that are connected to the object properties. Concepts on higher level in the taxonomy are more general. A natural way to estimate similarity in the taxonomy is to measure the distance between concepts to which the compared instances belong.

The distribution of the concepts' density is usually not balanced in *is-a* taxonomy [69]. Also granularity in various parts is different, i.e. the distance of links between concepts is not semantically uniform.

We assume that the closer instances are in the taxonomy the more similar they are. The *edge-counting method* computes the shortest path between related concepts and it is also known as *common-ancestor specification*. Distance is defined as the shortest path going through a common ancestor or as the general shortest path, potentially connecting two instances through common descendants/specializations [13]. These methods compute the distance in the taxonomy and do not capture differences in taxonomy granularity. We introduce a taxonomy distance metric where the similarity of instances increases based on how many common concepts they have in the taxonomy. A combination of the number of common concepts with the depth of the instances in the taxonomy is a way to consider granularity in the taxonomy structure. When using this approach there is no need for further normalization of the distance between instances to get a valid similarity measure.

Let us define a function $depth(instance)$ expressing the depth from the root concept in the taxonomy to the concept which a given *instance* belongs to. Let us define a function $CommonConcepts(instance_1, instance_2)$ that computes the number of concepts that have two instances in common in the paths leading from the root concept to concepts which $instance_1$ and $instance_2$ belong to. The similarity is computed as the number of common concepts in the taxonomy divided by the number of concepts in the higher depth:

$$sim_{tax}\left(instance_1, instance_2\right) = \frac{CommonConcepts(instace_1, instance_2)}{max\left(depth(instance_1), depth(instance_2)\right)}. \quad (6.4)$$

Two examples are depicted in Figure 6.2. The common concepts in the taxonomy are emphasized by dotted arrows while solid arrows are used to show greater depth from the root concept.

Another problem we encountered is the identification of relevant element pairs in the case of multiple occurrences of a property. Each instance connected to an object property in the ontology can have a label that could be compared employing a datatype metric. However, solving the problem using only the label is not satisfactory as labels are only optional and do not

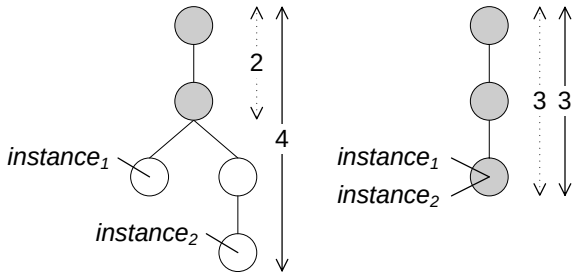

Figure 6.2: Example of taxonomy distance computed for $instance_1$ and $instance_2$. For the left example $sim_{tax}(instance_1, instance_2) = 2/4 = 0.5$, for the right example where both instances belong to the same concept in the taxonomy is similarity computed as $sim_{tax}(instance_1, instance_2) = 3/3 = 1.0$.

necessarily express semantics. Therefore, they should be used very carefully (for automatically acquired instances it is obvious that meaningful labels are not present).

To identify pairs of compared elements we construct a similarity matrix whose size is specified by the cardinalities of the respective element sets. The matrix holds similarities for each pair of elements from the sets. In the case of literals, datatype metrics are used as described above. For instances connected to object properties the recursive algorithm is employed. Afterwards, the identification of relevant pairs is performed where the number of pairs is given by the cardinality of the smaller set. The resulting algorithm for finding relevant pairs is shown in the Algorithm 2.

Algorithm 2 Finding relevant pairs

```
while count(pairs) < count(getSmallerSet(setA, setB)) do
    set maxValue to getMaxValue(matrix)
    store maxValue to List
    set coordinates of maxValue to X and Y
    foreachitem in matrix do
        if item.row = X or item.column = Y then
            set item to null
        end if
    end foreach
end while
```

Finding pairs with very low similarity measures can be prevented by using a critical threshold value. Leftover elements are handled in the same way as described above for elements connected to a property that has occurrence in one instance only. An example of finding pairs from the similarity matrix based on the described algorithm is shown in the Figure 6.3.

Figure 6.3: Identifying relevant pairs from sets.

Similarities used in the example are random numbers. In the first iteration (left) at $[A2, B2]$ is the maximal value 0.9 and it is stored in the *List*. This value corresponds to the similarity computed for a pair of elements. The value will be used in the aggregation of the total similarity and other values on that row and column are set to *null* (second row and second column). In the next iteration, the maximal value 0.7 is at $[A1, B3]$, the last coordinate is $[A3, B4]$. Element $B1$ is evaluated as a leftover one.

6.3 Similarity estimation

In our approach the total similarity between two instances of ontological concepts is defined as follows:

Definition 6.3 *The total similarity of two instances is aggregated as the mean value of the similarities computed between elements connected to particular properties. The computed similarity is symmetric.*

However, other aggregation functions can be employed such as weighted mean value. Let us compare two instances *InstA* and *InstB*. Let these instances have the following properties respectively:

$$\begin{aligned} A &= \{property_1, \ldots, property_n\}, \\ B &= \{property_1, \ldots, property_m\}. \end{aligned} \tag{6.5}$$

The way of ordering properties in the sets is not important since the comparison method treats common properties and leftover properties differently as described above. Let *PropertySM* be a similarity measure (SM) that is computed for elements connected to a common property. Then similarity measure for two instances is computed as follows:

$$sim\,(InstA, InstB) = \frac{\sum_{i=0}^{|A \cap B|} PropertySM_i\,(elementA, elementB)}{|A \cup B|}, \tag{6.6}$$

where *elementA* and *elementB* are elements (instances or literals) connected to the i-th property. Since there can be datatype or object properties, we introduce the *General similarity measure* that encapsulates all the similarity measures that are available. It is computed for elements connected to a property (e.g., *PropertySM* used in the Equation 6.6 is its special case). The *General similarity measure* fulfills the same conditions as defined for the similarity and it gets values from the range $\langle 0, 1 \rangle$.

Our method is based on the comparison of elements connected to identical properties. A special case when an instance connected to an object property is compared with a literal does not occur, as that would be against the OWL DL specification we focus on.

The *General similarity measure* is computed with regard to the property type. In the case of a datatype property the used metric depends on the corresponding literal type as described above. For object properties, the similarity measure for related instances is computed as the aggregation of the following partial similarity measures:

- *Label-based similarity measure (LabelBSM)*, which is computed for elements' labels employing string metrics if labels holding meaningful information are present (if instances were acquired automatically meaningful labels are usually not present);
- *Property-based similarity measure (PropertyBSM)*, which is computed if instances have additional properties that are used to invoke a recursive computation of the *General similarity measure*;
- *Taxonomy distance similarity measure (TaxonomyDSM)*, which is computed as described in the Section 6.2.

Each of these similarity measures has values in the range $\langle 0, 1 \rangle$. The final similarity measure $sim_{object}(X, Y)$ for two instances X and Y having the $property_i$ is computed as the mean value of the used measures:

$$sim_{object}(X, Y) = \tag{6.7}$$

$$= \frac{LabelBSM(X, Y) + PropertyBSM(X, Y) + TaxonomyDSM(X, Y)}{N},$$

where *LabelBSM*, *PropertyBSM* and *TaxonomyDSM* are the respective similarity measures described above and N is the number of similarity measures

that were employed in the individual case. For instance, if the meaningful labels are not present, the *Label-based similarity measure* is omitted and $N = 2$.

In Figure 6.4, there is an example showing two job offers that both consist of three properties only. The depicted job offers have the object property (*hasPrerequisite*) and two datatype properties. The notion of objects used in the figure is the same as used in Figure 3.6.

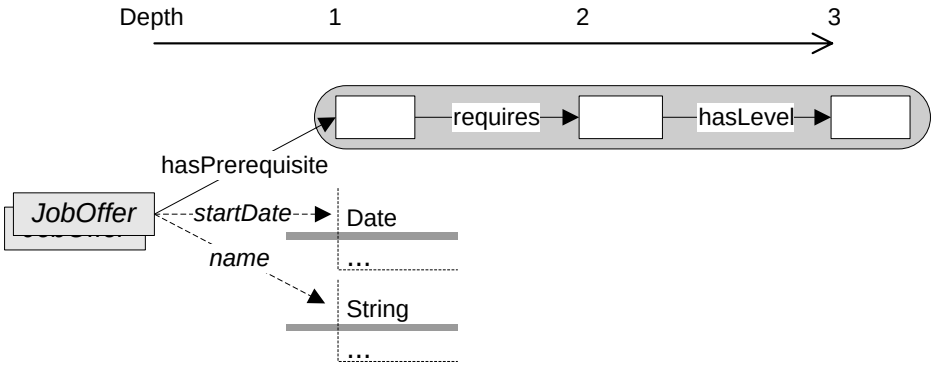

Figure 6.4: Example of different depths of properties of a job offer instance.

The gray rounded box is used to highlight all the parts of the object property which are considered while similarity is computed. Datatype properties have only one assigned value, thus highlighting was not necessary. For illustration, only properties and connected elements are shown; other elements as well as possible labels that are considered in enumeration are not depicted.

The purpose of this example is to show that particular properties can have different depths. While the largest distance of datatype properties is 1 from the root instance, in the case of an object property where other properties can be attached to the instance, the depth is 3 as depicted in the figure. In such properties more general information are closer to the root instance. The more distant instances are from the root, the more specific information they hold. If similarity is computed as a mean value of similarities computed on particular levels, high values of similarity on lower levels will significantly influence low similarity computed on the top level. The similarity measure for the entire property is computed as mean value of similarities computed on particular levels:

$$sim_{property}\left(elementA, elementB\right) = \frac{\sum_{i=0}^{maxLevel} sim_{object}\left(X_i, Y_i\right)}{maxLevel}, \qquad (6.8)$$

where *elementA* and *elementB* are elements connected to the examined *property*, sim_{object} is computed as defined in Equation 6.3 and *maxLevel* is the maximal nesting depth for the *property*. Figure 6.5 shows the similarity computed for an object property in XML notation. The element *similarity* specifies the aggregated similarity from inner elements via its *value* attribute, while its attribute *property* specifies the property for which the aggregated similarity is computed. In the example, partial similarities were 0.2, 0.3 and 1.0 respectively to the growing distance from the root. The total similarity measure computed for the property is 0.5.

```
<similarity value="0.5" property="hasPrerequisite">
<parts>
  <similarity value="0.2" property="requires"/>
  <parts>
    <similarity value="0.3" property="hasLevel"/>
    <parts>
      <similarity value="1.0"/>
    </parts>
  </parts>
</parts>
</similarity>
```

Figure 6.5: Example on computing similarity for object property.

Decreasing the weight with growing distance seems to be reasonable and we propose to employ the reciprocal function of x, where x stands for level of nesting depth. Thus, the *weight* for each level is computed as:

$$weight_x = \frac{1}{x}. \tag{6.9}$$

When using weights the similarity for a property is computed as follows:

$$sim_{property}\left(elementA, elementB\right) = \frac{\sum_{i=0}^{maxLevel} weight_i \times sim_{object}\left(X_i, Y_i\right)}{maxLevel}. \tag{6.10}$$

Employing weights in the similarity estimation for the object property from Figure 6.5 is shown in Figure 6.6 and results in the similarity measure 0.2278. The *similarity* element is extended with the *weight* attribute that specifies the weight computed based on the nesting depth.

Now, the computed similarity is closer to the similarity computed on the highest level and shows less of an influence from bottom levels. The proposed

```
<similarity value="0.2278" property="hasPrerequisite">
<parts>
  <similarity value="0.2" property="requires" weight="1.0000"/>
  <parts>
    <similarity value="0.3" property="hasLevel" weight="0.5000"/>
    <parts>
      <similarity value="1.0" weight="0,3333"/>
    </parts>
  </parts>
</parts>
</similarity>
```

Figure 6.6: Similarity influenced by weights with regard to nesting depth.

weights are also useful in the same way for the datatype properties. Since depth of datatype properties connected to the root instance always equals 1, the computed weight also equals 1 and the computed similarity measure is thus only the result of the used metric according to the literal type.

6.4 Personalized similarity and user characteristics

The aggregate of partial similarities is always the same no matter what the context is. To improve the accuracy of our similarity evaluation method with respect to individual users' preferences (if a user model is available), we introduce weights that personalize the similarity estimation which allows us to compute personalized similarity for individual users:

$$sim\left(InstA, InstB\right) = \frac{\sum_{i=0}^{|A \cap B|} weight_i \times PropertySM_i\left(elementA, elementB\right)}{\sum weight}, \quad (6.11)$$

where the semantics of variables is the same as in Equation 6.6. The *weight* variable has values in the range $\langle 1, w \rangle$ based on the match between the property and the value of the corresponding characteristic in the user model. Since we assume that the user's likes should have greater influence on the total similarity, we increase the weights of properties for which corresponding characteristics are present in the respective user model and their values match with the compared instance. The exact increase of individual weights

is the subject of experiments for any particular domain. The meaning of the proposed *weight* is as follows:

- "1" if there is no correlation between a property of the instance and a characteristic in the user model; this weight also solves problems when the user model is not available and thus has no influence on the computing of personalized similarity for a particular user;
- "w" if there is match not only between a property of the instance and a characteristic but also between their values; or
- a value between the previous two values means that there is a match between the examined property of the instance and the user model, but the related value is not identical. For instance, the values could be instances that are located on different levels in the taxonomy tree (e.g., a city belongs to the same region as the city preferred by the user in the user model but it is not that city).

For personalization purposes, our goal is not only to compute the similarity between instances but also to investigate reasons that "caused" the similarity or difference. User preferences can be deduced from implicit and explicit user feedback (e.g., rating). We assume that if the instance includes a property whose value the user likes, it will likely influence his/her rating towards the higher (or positive) values. On the other hand, properties of the content with the values that the user dislikes will influence rating towards lower (or negative) values.

Since we are interested in properties that significantly influence user rating and thus also total similarity, we introduced two threshold values that divide properties into three sets based on the computed similarities. If the similarity computed for a property is greater than the *positive threshold* then the property is assigned to the positive set, if the computed similarity is lower than the *negative threshold* the property is assigned to the negative set.

6.5 Evaluation

Experimental evaluation was performed using the software tool called Concept Comparer (abbreviated *ConCom*), which was implemented in Java and uses the Sesame framework to access ontological models represented in OWL DL. For datatype properties, the *Levenshtein method* was used while for object properties the proposed *taxonomy distance* was employed. The evaluation was performed on the job offer ontology developed in the course of

the research project NAZOU. The smallest dataset contains 100 job offers mostly from the information technologies field. *ConCom* can work in two modes which are configurable from the command line. In the first mode, the total similarity is computed for all properties, i.e. if a property occurs in one instance only the 0 is aggregated. In another case, only properties that are common for both instances are considered, thus other properties are ignored and do not influence the total similarity.

Experiment 1: All properties vs. Common properties

The aim of the experiment was to compare results computed in two ways and to specify positive and negative thresholds. In the experiment, the similarity for all possible 10 000 instance pairs was computed. The experiment showed that results satisfy all criteria required for similarity as defined in Section 6.3. Figure 6.7 depicts a sample of 600 instance pairs for which similarity was enumerated in both operating modes of *ConCom* — the computed values are ordered by similarities computed for all properties.

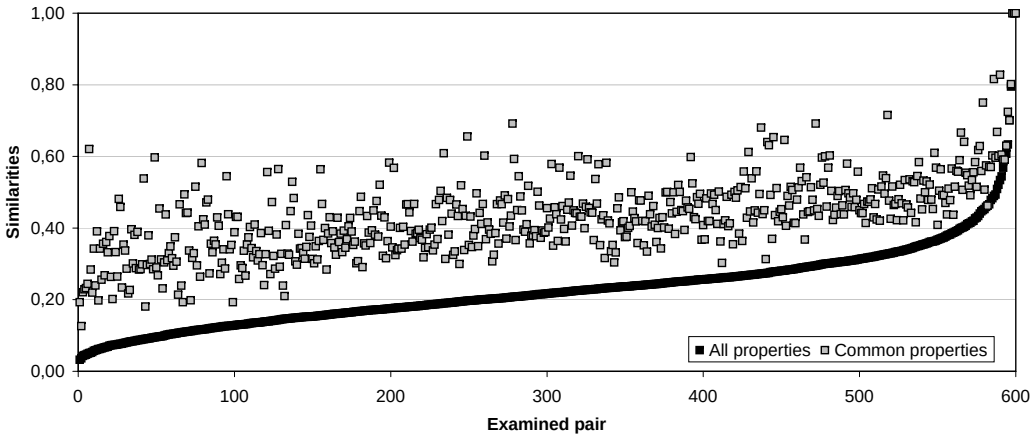

Figure 6.7: Similarity computed by ConCom considering all/common properties.

The thresholds were specified experimentally for the job offer domain. We computed similarity for 55 000 properties employing the described method. Properties with similarity equal to 0.0 or 1.0 were not considered to eliminate identities and properties with no occurrence in both instances. The rest of the properties was ordered according to the computed similarity measure and using the *Pareto principle* (also known as 80/20 rule). We split the most influential 20% in half to select 10% of the highest and 10% of the lowest values. This way, the positive threshold was set to 0.65 and the negative

threshold to 0.25. The domain dependence of thresholds is the subject of further experiments.

The properties classified by this method can be transformed into user characteristics and used for populating or updating existing user models. Since the transformation of properties into user characteristics as well as their update in the user model is beyond the scope of this paper, the presented method only prepares inputs for further processing. Using both the positive and negative set of properties in combination with user feedback for user characteristics updates in the user model would improve user characteristics estimation.

Using only common properties in our experiments with job offers resulted in a narrow range of similarity values — in 89% of the cases the computed similarities were in the range 0.30 to 0.75. We set the positive threshold experimentally to 0.65 and the negative threshold to 0.25, but such thresholds do not produce useful properties that could be used for user characteristics discovery. Therefore, similarity computed for all properties must be used to acquire properties based on thresholds.

Experiment 2: ConCom vs. Human

The aim of the experiment was to find out which type of the similarity computed by *ComCom* better mimics similarity assessed by human users. A sample of 300 job offer pairs was used with 30 randomly selected sample pairs that were presented to the user twice in order to verify evaluation consistency. The user assessed similarity on a scale from 0 to 7, specifying that offers had nothing in common (rating 0) to equivalent offers (rating 7). Afterwards, the acquired values were normalized to the similarity interval.

Similarity computed for common properties was used for comparison with human estimation since its values more accurately mimic human assigned similarity values. This could have been caused by the fact that human users can more easily evaluate a lower amount of (common) properties. For illustration, the result for a set of 40 randomly selected offer pairs is depicted in Figure 6.8.

It is theorized that the cause of evaluation differences is probably derived from specific preferences of the human user who usually makes decisions based on the properties he/she considers important. It is likely that another user would evaluate the same sample differently.

We have not found significant differences between the two evaluations of the same instance pairs. In 70% of the cases, the pairs were assigned identical similarity values, in 16.67% of cases the difference was one point on the scale,

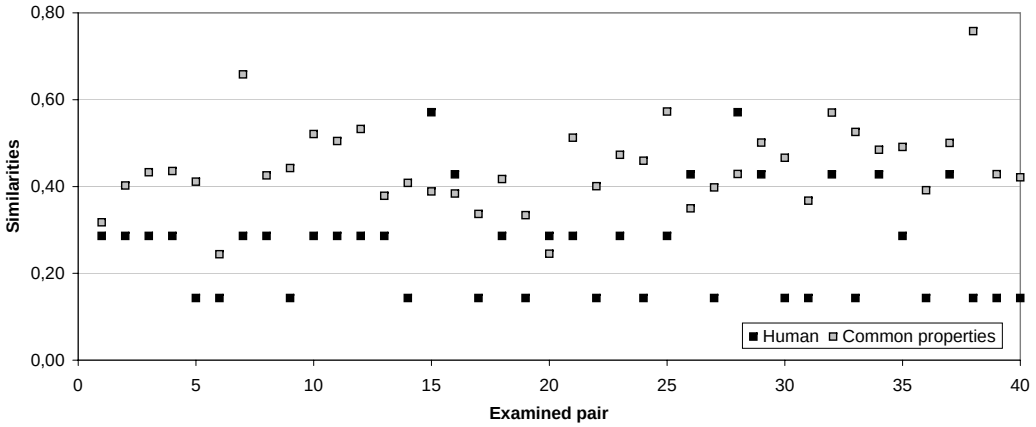

Figure 6.8: Similarity estimated by a human and by ConCom for common properties.

in 10% of cases it was two points while only in 3.33% of cases it was three points. This shows that users do not necessarily evaluate the same content in the same way if an adequately large scale is provided, especially if there is some time delay between evaluations.

Consequently, for further experiments where the user model was involved we used similarity computed only for common properties.

Experiment 3: Adjusting weight for personalized similarity

We assume that a user's likes or preferences stored in the user model influence personal similarity perception. Therefore, if the user model is available, its characteristics should have a notable influence on the total estimated similarity. The goal for this experiment was to identify the upper weight bound to be used in the computation of personalized similarity.

Figure 6.9 depicts similarities computed with respect to a given user model, which consisted of only one characteristic (*hasDutyLocation*). The job offers used in the experiment contained the *hasDutyLocation* object property and its value was the same as in the user model.

The growth of the similarity estimation is not linear as it depends on the number of properties the job offers consist of. The differences were in the range 0.06 to 0.12 for the upper bound w set to 2.0 and from 0.15 to 0.26 for $w = 4.0$, and varied based on the number of properties. Job offers used in the experiment had an average of 16 properties. Our experiment shows that using doubled weights causes a significant improvement in the similarity evaluation and is a worthy selection.

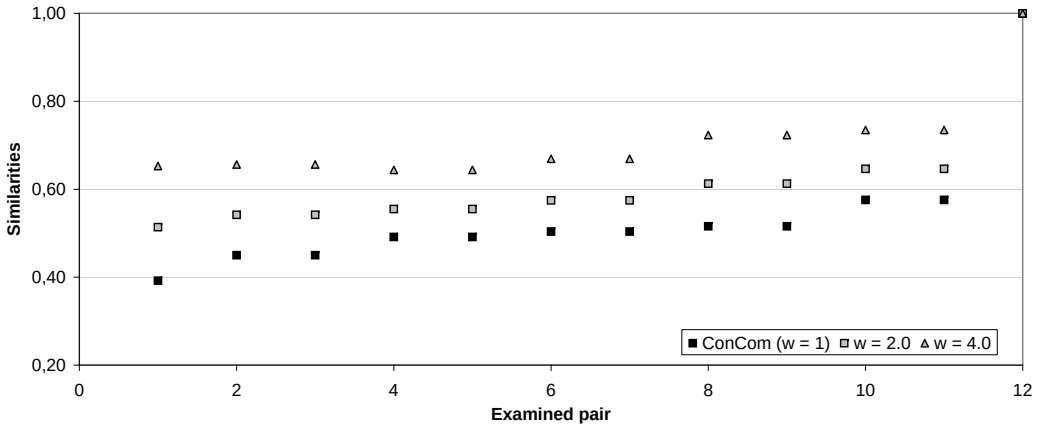

Figure 6.9: Weighted similarity computed with regard to a user model.

Experiment 4: Personalized similarity

The aim of the last experiment was to investigate how the user model influences similarity computation and accuracy. In the experiment a user model with three characteristics was used — *hasDutyLocation*, *offersPosition* and *hoursPerWeek*. We use an overlay ontology-based user model[2] that was developed as a part of project NAZOU. The user model was acquired by the Log-Analyzer tool [9] and contained both characteristics and preferences. A preference indicates that the related property is important for the user but there is no specific value assigned to it. For preferences we consider the *weight* as half of the upper bound employed in the computation of personalized similarity ($weight = w/2$). Doubled weights were used for characteristics as described in the previous experiment. If a property occurs in the user model both as a characteristic and as a preference, the final weight is computed as the sum of their weights, i.e. $weight = w + w/2$.

The experiment was performed on the sample of 10 000 job offer pairs. Figure 6.10 depicts the change in the similarity estimation caused by employing the user model (200 pairs depicted for illustration).

The employed user model influences the computed similarity in two ways. If the compared properties are similar (i.e., for high values of the similarity measure) the personalized similarity increases towards higher values (a positive change in the figure), while if they are different, the personalized similarity decreases to lower values (a negative change).

[2]Ontology-based User Model, http://nazou.fiit.stuba.sk/home/files/nazou_um.pdf

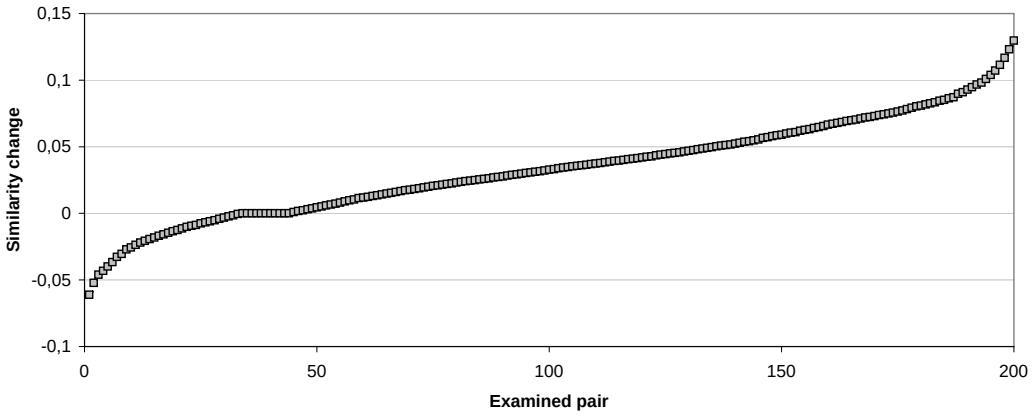

Figure 6.10: Change in similarity estimation caused by the employed user model.

6.6 Discussion

We described a method for the comparison of instances of ontological concepts based on the recursive traversing of an instance's structure. The final similarity is the aggregate result of the individual similarities computed for particular properties while their type is considered to select a suitable similarity metric for each property. The introduction of similarity metrics for properties allows us to take advantage of semantics provided by ontological representation, which allowed us to extend similarity with personalized weights reflecting users' individuality.

In the following text we compare proposed method to existing approaches. A method computing semantic similarity among instances within an ontology, which considers ontology and context layers, is described in [1]. The data layer estimates similarity by considering simple or more complex types, such as integer and string. Distinguishing only numbers and strings in datatype properties is not satisfactory (e.g., boolean values need to be treated carefully). We propose datatype similarity metrics that deal with all types of datatype properties as defined by XML Schema. The ontology layer exploits relations between entities in the ontology and the context layer assesses the similarity according to how the entities are used in some external contexts. Considering the context, there are three operations which influence semantic similarity — *cardinality*, *intersection* and *similarity* of/between properties or relations. The total similarity measure is a weighted combination of *external* and *extensional* similarity. The external similarity employs structural aspects of instances in the comparison (classes and slots which instances belong to are investigated), whereas the extensional similarity performs comparison in term of the instances' properties and relations.

An approach to ontology matching based on instances is described in [51]. Its main idea is to derive similarity between concepts from the number of shared instances, since the number of instances is usually greater than the number of concepts. Moreover, using instances makes ontology matching independent from concept names and other metadata. The novelty of this approach is in using well known similarity metrics (*baseline*, *minimum*, *dice* and *kappa*) from "traditional" ontology matching in instance-based ontology matching. A combination of these metrics is used to improve the achieved matching results. The best results were achieved as a union of the *minimum* and *kappa* metrics. The drawback of this approach lies in the fact that it only considers a given number of instances of the concept. Additionally, properties assigned to the concept and their types (object or datatype) should be considered to achieve more accurate similarity enumeration since extra information about instances is provided. Another approach presented here is matching based on metadata. Concepts are matched with regard to trigram similarity of their names, though experiments showed that it is not very effective due to the high diversity in the concept names.

PROMPT is an algorithm for ontology merging and alignment [62] that guides the user in creation of a merged ontology. It starts with creating an initial list of matches based on class names where linguistic similarity metrics are employed. Afterwards, the user has two options: either to select one of the suggestions provided by PROMPT or to use the editing environment to perform one's own changes in the ontology. The next step consists of automatic operations based on the previous choice. These steps are repeated in cycles. When a conflict occurs, a list of solutions is provided. PROMPT performs the merging of concepts, properties, relations between concepts and properties, and copies parts of a hierarchy (classes including their parents etc.). We consider name matching as a drawback of the algorithm since names of the concepts do not have to necessarily carry meaning, especially when automatic approaches are used to build or populate an ontology.

A two phase method for instance comparison of tourism ontology concepts is described in [35]. The first phase performs concept preprocessing. Two graphs are built — an *inheritance graph* that organizes ontological concepts according to the generalization hierarchy and a *similarity graph* in which nodes relate to concepts and edges correspond to the degree of similarity. The similarity itself is enumerated in the second phase consisting of three steps. First, flat structural similarity is computed exploiting structural slots (*part*, *related*, *predicate*). Second, hierarchical structure is exploited by using results from the previous step and extending them by further elements according to the hierarchical relationships. In the third step, the final sim-

ilarity measure between concepts is computed as a result of combination of two previous steps. The advantage of this approach is that total similarity for more than two concepts can be expressed as one number. Furthermore, the similarity of concepts from different contexts can be computed as well. The main drawback of this method is that a similarity ontology holding similarity relations between properties and entity names from the domain ontology must be provided in order for the similarity graph to be built.

The comparison with an "ideal" instance related to a particular domain is used in a search method based on user criteria [66]. The method also supports search for instances that do not entirely satisfy the criteria of the ideal offer. The similarity of particular properties of the offer is enumerated as the distance between their values. The computed distances are later converted to a degree of similarity taking into account the largest possible distance. To distinguish between particular properties, *precision* is introduced that reflects a user's subjective tolerance. Furthermore, the user is allowed to specify for each criterion, its *importance* and whether it has to be satisfied. This approach is aimed at searching similar instances according to a user's given criteria, however, unlike our approach, it computes asymmetric similarity.

A common property of the aforementioned approaches is that they do not investigate the causes of similarity. Automated similarity enumeration mimics the human similarity measure if different strategies are used based on clusters of users [13]. Users gave reasons of their assessments which were the basis for machine learning algorithms that assigned users to specific clusters. We use an automated approach to figure out reasons of similarity, which also contributes to the *scrutability* of the user model [48].

Chapter 7

Maintenance of user characteristics based on spreading activation

> The method presented in this chapter was proposed in cooperation with Michal Šimún as a part of his master thesis and was supervised by author of this work.

While a user works with a concept[1] (e.g., job offer, learning text) characteristics for that concept (e.g., estimated interest) can be obtained. In some application domains there may be a logical connection between concepts and other parts of the domain model expressed by defined relationships. This is obvious in educational domains where prerequisites can be defined and thus their fulfillment is a necessary condition for further studies. In such domains it is also reasonable to model user characteristics for parts of the domain model that user has not visited yet.

With regard to defined connections among the concepts in the domain model, we spread a change to other related parts of the domain model, even when the user has not worked with these parts. In this chapter we describe a method for maintenance of user characteristics that uses spreading activation. Spreading activation principle originates from psychological studies of human memory operations.

The principle is based on an idea that initial energy (activation) of a selected node (in a graph) is spread to other nodes and can be summarized as follows [28]: *The network data structure consists of nodes interconnected by links. The nodes may represent objects or features of "real world" objects, and are usually labeled. The links model relationships between objects*

[1]The concept is in the meaning as defined in the Section 3.1.

or features of objects. The links may be labeled and/or weighted and usually have directions, reflecting on the relationship between the connected nodes. The spreading is iterative. The iteration consists of one or more pulses and a termination check. Each pulse is made up of three stages: preadjustment, spreading and postadjustment. The first and third phase allows control over the activation (i.e., decay or retention) whereas spreading phase consists of the flow of activation waves from one node to all other nodes connected to it.

Our method was primarily proposed for an educational domain where relationships among concepts and their fragments are better defined than for the concepts involving a job offer domain. However, the proposed method is not restricted to be used only in the educational domain.

7.1 Models of adaptive web-based educational system

In the personalization process it is desirable knowing *document's attributes* (educational content) in an *educational course* (domain model) and also *user's characteristics* (user model) [20]. Domain model specifies characteristics of an educational course including learning materials. In this section we provide examples based on learning programming which presents the basis for domain model structuring. To support educational process (independently from a domain) it is important to take into account the individual learner, i.e. personalization. Learner's characteristics are mapped on characteristics of educational materials and express the semantics of educational materials. Therefore, the domain (and user) model is using ontology representation [2]. The developed model is an extension of the model described in [17]. Beside structural changes we provide following unique features:

- relationships defining a dependence between domain model parts and required knowledge, i.e. *prerequisites* (necessary for a learner to be able master particular knowledge and skills expressed in educational documents) and *knowledge* acquired while studying; and

- parts defining *explanation*, *tests* and other types of learning objects that are useful for better structuring of the domain.

Existing domain models usually consist of mutually interconnected learning objects that represent learning materials [23]. Metadata connected to learning objects allows reasoning while taking into account learner's knowledge

with regard to the particular learning object. Having reusability of the models in mind we divide the domain model into a *knowledge item space* and a *learning object space*. The elementary structure of the domain model is depicted in Figure 7.1. Different arrows in the figure express variety of relations that can be used in a domain model.

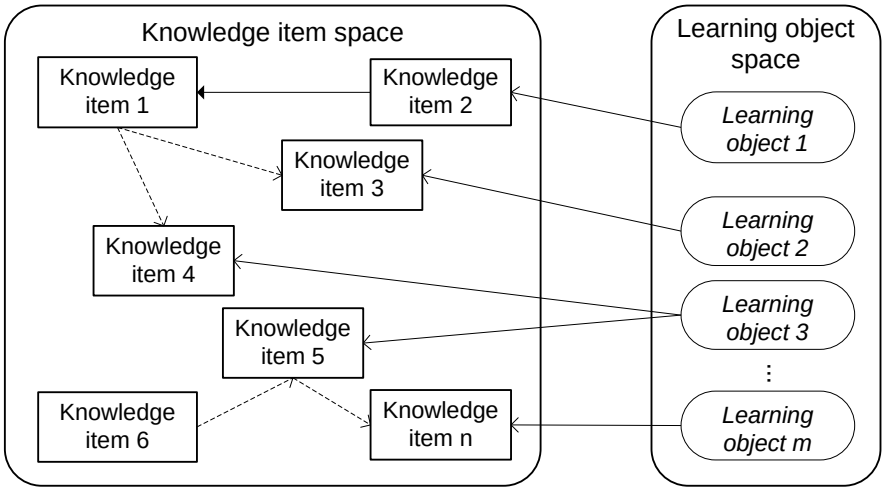

Figure 7.1: Structure of the domain model.

Learning object space

The learning object space consists of *learning objects* and *relationships* among them. This part of the domain model meets the standard view of the domain model for adaptive web-based systems. A learning object contains *identifying characteristics* (name and description) and *domain attributes*, such as difficulty level, programming language, etc. We have designed following types of learning objects:

- *course* — encapsulate the entire educational course as a unit;
- *programming learning object* — encapsulates definition of all learning objects in the learning programming domain; and
- *fragment* — defines smaller parts which a learning object can possibly consist of.

Programming learning object can be defined as *explanation*, *exercise*, *template* or *simple text* [54]. Furthermore, it can contain following fragments which are related to programming exercises that represent primarily content for learning programming in our courses:

- *definition* of the learning object of each type;
- *hint* that may help to achieve solution in the exercise;
- proper *solution* of the exercise; or
- *note* as an additional information to the example.

Programming learning objects are assigned to corresponding learning objects in a course. A structure of the educational course is specified by a hierarchy relation defined between learning objects (*content*) and relations that specifies assignment of a learning object to a knowledge item (*relate*). Hierarchical relations allow defining structure of the course (sections and subsections).

Knowledge item space

A knowledge item (KI) represents a topic or a key word that represent key terms of the domain. Its aim is a categorization of available learning objects into knowledge items according to learning goals of particular learning objects. Examples of knowledge items can be found in the Figure 7.3. We have added knowledge item level to the domain model to be able to model personalization also at the knowledge items level. Introducing knowledge items allows modeling user's characteristics on the level of knowledge items. It allows determining user's characteristics for all learning objects related to the knowledge item.

Our approach also supports changing learner's characteristics for the learning objects, even in the case they have not been visited yet, as a knowledge item can be connected to learning objects using following relations:

- *prerequisite* — specifies parts of knowledge item space which a learner has to master (at least at defined level) to be able to comprehend the knowledge item. Furthermore, a prerequisite can be specified as *required* (logical *AND*) or *optional* (*OR*) with defined level of its fulfillment;
- *contain* — defines hierarchical relations between knowledge items, i.e. parts which a knowledge item consist of can be defined; or
- *isRelatedTo* — defines logical relations between knowledge items.

The user model

Personalization in an educational course is based on recommendation of suitable learning objects to an individual user (learner) who is represented in the user model. We use an *overlay* user model that models the relation of the user to the individual parts of domain model. The developed user model is influenced by work presented in [24]. It consists of a *domain independent part* (*GenericUserCharacteristic* in the Figure 7.2) and a *domain dependent part* (*EducationalSpecificCharacteristic*). The domain-dependent part consists of records about user's visits, interests and knowledge in current educational course. An example of a domain independent characteristic is learner's name whereas a user's level of knowledge is example of domain dependent characteristics.

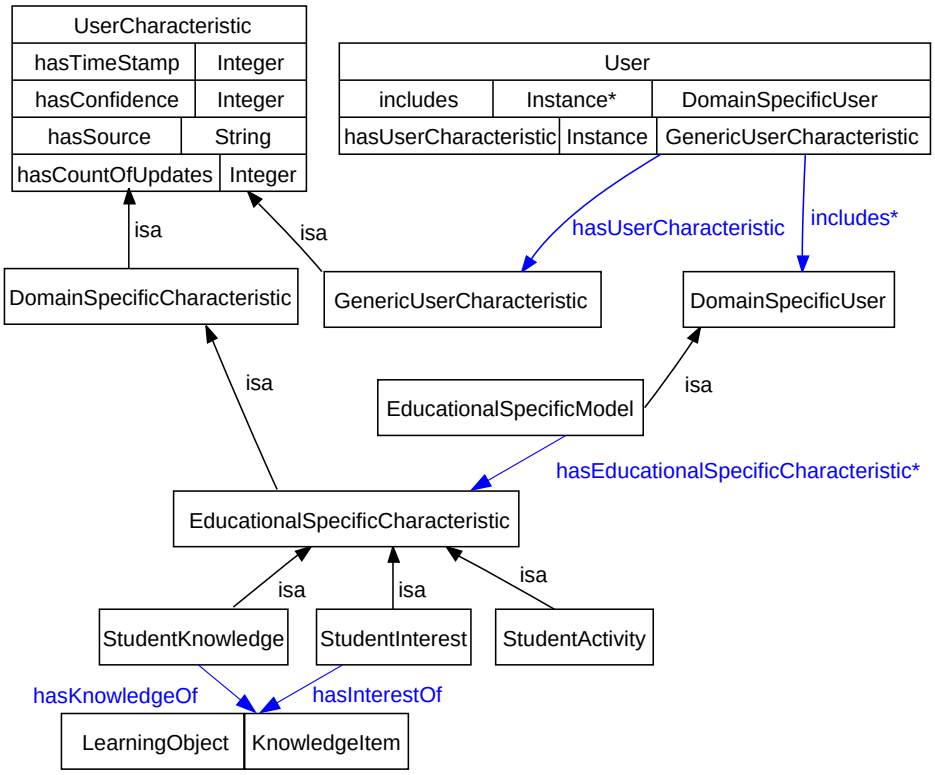

Figure 7.2: Structure of the user model.

This way we define user's relation to individual knowledge items and learning objects in the domain model. The *knowledge* characteristic consists of three parts:

– *level* is expressed as a real number from the range $\langle 0, 1\rangle$ that specifies a level at which the user has mastered that characteristic;

- *probability* expresses how likely modeled value match the real user; and
- *view* specifies the strategy which is used to adjust level and probability. A characteristic can be defined in several views and its final value is computed as a combination of used views.

For instance, the view *feedback* handles a setting of the characteristic by the user, *classified* is an evaluation of the characteristic from the activity with the learning object, and *inferred* is spreading of the characteristic from other parts of domain model.

Each characteristic in the user model has assigned a *timestamp* (time of the last change), *confidence*, *number of changes* and *source* that made a change of the characteristic (see Figure 7.2). Confidence expresses belief about match between real user characteristic and modeled characteristic.

7.2 Maintenance of the user characteristics by spreading change

While a user works with a learning object we obtain his/her characteristics for that learning object (e.g., estimated interest). Since there is a connection between learning objects and other parts of the domain model we spread a change to other related parts of the domain model even though the user has not worked with these parts yet. Modeling user's characteristics consists of the following steps:

1. Setting characteristics for actual learning object (e.g., *Example with I/O operations* in the Figure 7.3) — with regard to user's activity with the learning object we can find out user's *interest* and *knowledge* for that learning object.
2. Spreading changes of characteristic's values from the actual learning object to knowledge items (i.e., knowledge item *Display* and *Scan*).
3. Spreading changes of characteristic's values from the knowledge item (*Scan* and *Display*) to other related parts of knowledge item space (*Input/Output* and *Files*).
4. Spreading changes of characteristic's values from the knowledge items which characteristics have been changed to related learning objects (i.e., *Example with writing to file* and *Example with displaying items of array*).

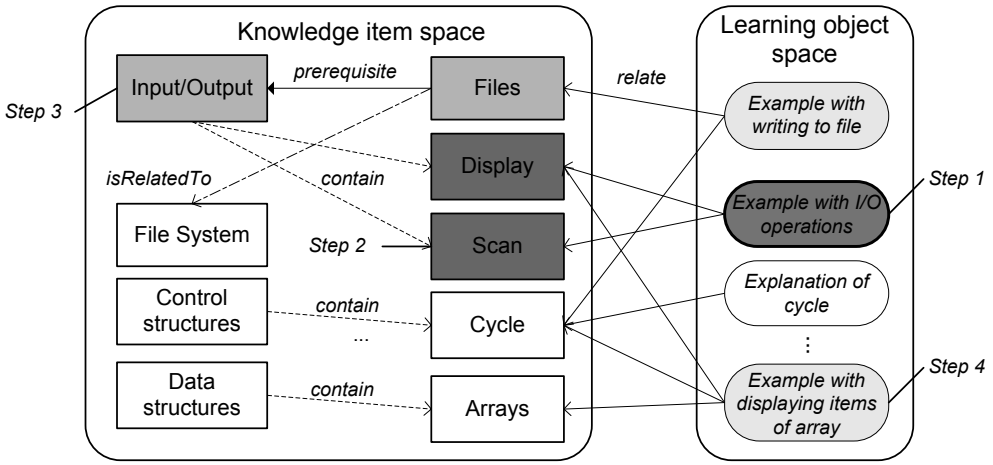

Figure 7.3: Sequence of steps in process of change of characteristic.

Step 1: Setting characteristics for actual learning object

The user model stores information about learner's activities and his/her estimated knowledge and interests. Learner's activity contains records about visited learning objects in a course. Knowledge and interest can have various assigned views while each view defines a strategy for characteristics with regard to their update. Furthermore, the *interest* and *knowledge* have assigned:

- *value*, which is numerically expressed knowledge or interest from an interval of available values (e.g., interval from 0 to 1 for knowledge, where 1 reflects level of expert whereas 0 is none); and
- *probability* of modeled value which is a measure how likely a real value of user's characteristic matches a modeled value of the characteristic.

By analyzing learner's activity we can directly change the following views for actual learning object:

- *feedback* — a user adjusts interest and knowledge in a graphical user interface directly. Selected value is set in the user model for actual learning object in feedback view with probability equal 1; and
- *classification* — every defined level of the characteristic (e.g., expert) creates a class of the classification. According to selected strategy a model of classifier is created that assigns a user into one of defined classes in regard to user's activity. Variety of classifiers can be used (e.g., content-based learning classifier).

After a characteristic's update is accomplished it is necessary to summarize values of particular views and also take into account their confidence. The final combined value of the characteristic $char\,(U, LO, combined)$ of learning object LO for user U is evaluated as follows:

$$char\,(U, LO, combined) =$$

$$= \frac{\sum_{\forall V \in Views} confidence\,(V) \times char\,(U, LO, V) \times probability\,(char\,(U, LO, V))}{\sum_{\forall V \in Views} confidence\,(V) \times probability\,(char\,(U, LO, V))}, \quad (7.1)$$

where V stands for a view from the set of *Views*, *char* is characteristic's value in the user model, notation of *confidence* and *probability* is as we defined before. Confidence of the particular views is specified in a configuration file. The probability of a characteristic is modeled by normal distribution of values.

We use two characteristics of distribution — the *mean* of the distribution expressing characteristic's value and *functional value of the mean* specified by probability of modeled characteristic's value of the user. Modeling probability allows for specifying probability for the entire range of available values. For instance, if we model knowledge and its value is 0.9 with probability of 1 then from the distribution the probability of expert knowledge (value 1) can be computed. The probability of characteristic's final value $probability\,(char\,(U, LO, combined))$ is computed as follows:

$$probability\,(char\,(U, LO, combined)) =$$

$$= \frac{\sum_{\forall V \in Views} confidence\,(V) \times f_{prob}\,(char\,(U, LO, V))}{max\left(\sum_{\forall V \in Views} confidence\,(V), 1\right)}, \quad (7.2)$$

where the meaning of *char* and *confidence* is the same as in Equation 7.1, $f_{prob}\,(char\,(U, LO, V))$ expresses functional value of characteristic's normal distribution for the computed characteristic's value and models estimation of probability of used view for computed characteristic's value.

The statement in the denominator assures that combination of views with low confidence defines a characteristic with lower probability.

Step 2: Spreading changes of characteristic's values to knowledge items

After user's characteristic is changed (its value and probability) for a learning object with defined feedback view it is needed to accomplish relevant change also for all other knowledge items which this learning object belongs to. For instance, if a user studied a learning object focused on cycles in Java it is desirable to increase knowledge for the knowledge item *Cycle*. In the Figure 7.4 for the learning object *Example on cycle* value of a characteristic with the feedback view was changed (e.g., interest).

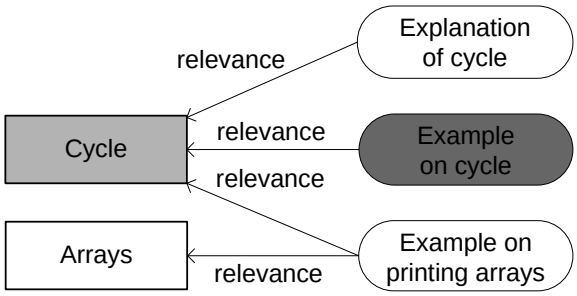

Figure 7.4: Spreading a change from learning object to knowledge item.

This learning object is a source of spreading change of the characteristic and the change is spread to related knowledge item (i.e., *Cycle*). The value of characteristic for knowledge item is computed from all related learning objects (*Explanation of cycle*, *Example on cycle*, *Example on printing arrays*). Characteristic's value for related knowledge item is computed using all related learning objects that have initialized a characteristic with the feedback view.

The characteristic's value $char\,(U, KI, feedback)$ for the user U with the *feedback* view for the knowledge item KI is influenced by weighting of learning object's characteristics related to the knowledge item:

$$char\,(U, KI, feedback) =$$

$$\frac{\sum_{\forall LO \in related\ KI} char\,(U, LO, feedback) \times probability\,(char\,(U, LO, feedback)) \times rel\,(LO, KI)}{\sum_{\forall LO \in related\ KI} probability\,(char\,(U, LO, feedback)) \times rel\,(LO, KI)}$$

where *related* describes relation between a *learning object* and KI with relevance *rel*, *char* is a characteristic's value with defined *view* and *probability* is probability of the value.

Afterwards, the probability of computed characteristic's value for the knowledge item needs to be estimated. We use the same method as we described in Step 1 while *confidence* is computed from relevance of the relation between the learning object and the knowledge item. When a characteristic is changed, combined view at the characteristic needs to be computed as described in Step 1.

Step 3: Spreading changes in knowledge item space

Since modeled knowledge items are mutually related it is desirable to take into account a change of characteristic in the knowledge item also in other related knowledge items. Therefore, spreading of this characteristic with *combined* view in the knowledge item space is accomplished. Spread user's characteristic is set in the *inferred* view in other knowledge items. Spreading a change of a characteristic for *knowledge* and *interest* is accomplished through the modeled relations between knowledge items (i.e., *isRelatedTo*, *prerequisite*, *contain*).

Spreading a user's characteristic to other knowledge items is based on spreading activation principle [28] where energy is spread from the selected node to other nodes, while with the increasing distance it fades. The energy E of the node j, which is spread from the node i, is computed as follows:

$$E_j = E_j + w_{ij} \times f_D\,(i,j) \times E_i, \tag{7.3}$$

where w_{ij} is weight of relation between the nodes, f_D expresses the distance and E_i is the energy of the node, from which the spreading comes. Afterwards, the activation for the nodes with changed energy is computed.

In our approach we spread a change of the characteristic. Energy is represented as a couple *value* and its *probability*, where

$$value = char(U, KI, combined). \tag{7.4}$$

Characteristic's value with *combined* view of user U for knowledge item KI and *probability* of this value is spread from knowledge items (nodes in the knowledge space) assigned to the actual learning object.

In the spreading activation process there are three factors that need to be considered:

- way of activation fading;
- summing of activation; and
- activation function for a node of knowledge items network.

Activation fading is accomplished by sequentially decreasing the value of the probability (decreasing global maximum of probability in normal distribution). If the node i has energy at the level of $E_i = (value, probability)$, then after the spreading activation to the node j is $E_j = (value, f_D(probability))$, where f_D is function of the distance. In the case when the energy is spread to the node j from more nodes simultaneously then the final energy is computed as follows:

$$E_j = \sum_{i\ inferTo\ j} (value_i, probability_i) \tag{7.5}$$

whereas *value* and *probability* are computed as follows:

$$value_j = \frac{\sum_{\forall i\ inferredTo\ j} value_i \times probability_i}{\sum_{\forall i\ inferredTo\ j} probability_i} \tag{7.6}$$

$$probability_j = \sum_{\forall i\ inferredTo\ j} f_{prob_i}\left(value_j\right) \tag{7.7}$$

where f_{prob_i} expresses the functional value of normal distribution for the computed value.

Activation of the node j, which energy E_j was spread into, is computed as $A_j = f\left(E_j\right)$, while f expresses linear function in the range $\langle 0, 1\rangle$. Here, activation of the node j is defined as a probability of the characteristic of the *inferred* type for the node j (i.e., its value is equal $value_j$).

We use the principle of spreading activation to spread a change of the *interest* and *knowledge* in the knowledge item space. The interest is spread through the *isRelatedTo* relation and knowledge through the *prerequisite* relation. In the following text we focus on updating of knowledge that can be changed using spreading activation and/or prerequisites.

Spreading knowledge (interest)

The spreading activation is used to change user's estimated knowledge (interest) level in a graph where relations express a relevance among knowledge items according to the knowledge (or interest). We use hierarchical relations between knowledge items to specify from which items selected knowledge item consists of (if spreading interest we use also relation *isRelationTo*). In the Figure 7.5 there is an example on spreading of the interest and decreasing intensity of the spreading is expressed by the fading color.

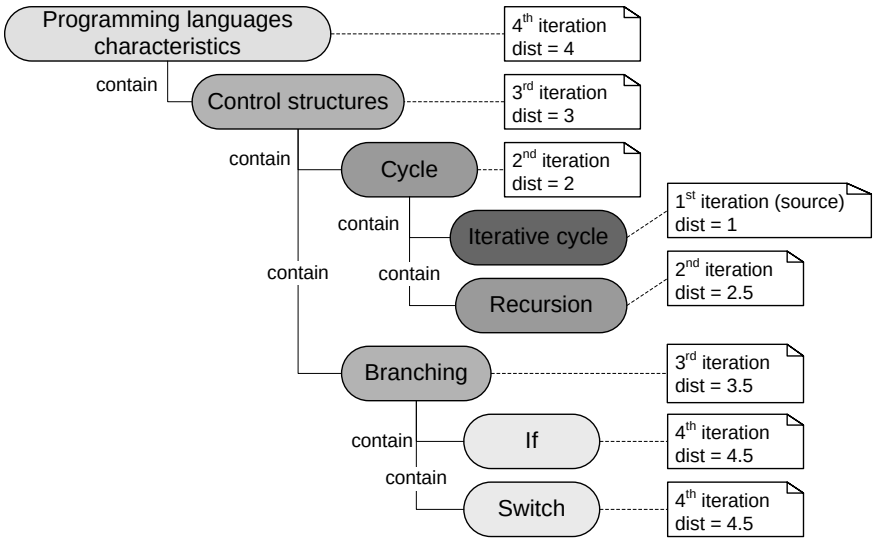

Figure 7.5: Example on spreading user's interest in the programming course.

When changing the knowledge we take into account also parts which knowledge item is related to. Therefore, the spreading activation is oriented towards the parents. For activation fading we use the exponential descending function. We use the same function for activation fading of the interest as well. In the case of spreading a change of knowledge it descends steeper because of spreading the knowledge to lower amount of nodes and the influence of a node is lower for nodes that are further.

Spreading prerequisites

Prerequisites define which knowledge items a learner has to know (at requested level) to be able study educational materials related to selected knowledge item. Thus, if the learner is an expert in some area (expressed by the knowledge item) we can infer that he/she probably knows item's prerequisites at least at required level. A change of knowledge can be spread only from the knowledge items where user's knowledge is at the expert level (equals 1) and probability is greater than defined threshold. The reason is that only in this case we can reason that user knows this knowledge item and fulfils all prerequisites. In the opposite case, we do not know what caused decreased value of knowledge whether failure in prerequisites or misunderstanding of presentation. Two types of prerequisites are possible:

- required (*and*) — any prerequisite has to be fulfilled; or
- optional (*or*) — at least one prerequisite has to be fulfilled.

If a prerequisite of the "and" type occurs, knowledge is spread to its prerequisites with energy (its value is relevance of the prerequisite) and probability (its value is probability of spreading activation). A prerequisite should achieve at least a minimal requested level of knowledge with probability to be allowed to say whether a user's level of knowledge is at expert level in this prerequisite. An example on required prerequisite is shown in the Figure 7.6. First, spreading starts in the node B which knowledge is 1.0 with probability 0.8. Since we assume that the user is an expert for item B with probability greater than defined threshold we start spreading knowledge into the node A, which knowledge has not been set yet. In the second step the value of knowledge for node A is adjusted and probability is the same as in node B.

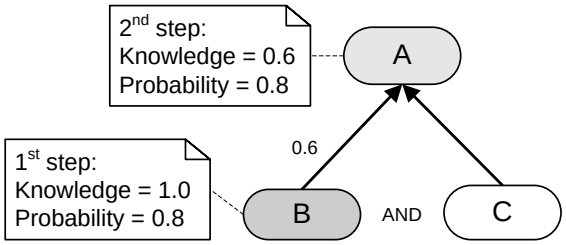

Figure 7.6: Spreading knowledge over required prerequisite.

In the case of prerequisite "or" we can not say with certainty which of the prerequisites a user used during studies. Therefore, probability spread from the node with this prerequisite is split into same parts among all prerequisites of the "or" type.

The last step in the process of spreading knowledge through the prerequisites is setting the knowledge with the inferred view in prerequisites. This process is identical with the spreading a change of interest.

Step 4: Spreading changes from knowledge items to learning objects

The process of changing user's characteristics in the domain model ends by spreading activation of a characteristic's change from the knowledge item space to the learning object space. After the change of value and probability of a characteristic for the knowledge item occurs it is necessary to process the change of the characteristic for all learning objects in the learning object space that have assigned selected knowledge item.

This way is spread a characteristic with combined view to a characteristic with inferred view from the knowledge item and this view is sequentially combined into a final characteristic with a combined view for the learning

object. The described principle of changing a characteristic is identical with spreading a characteristic in opposite direction from step 1.

A characteristic's value for a respective learning object is computed from the characteristics of all related knowledge items that have initialized the characteristic. The value of the inferred view of a characteristic for user U and learning object LO is computed from characteristics to which a learning object is assigned. Afterwards, the probability of a characteristic's value is computed for processed knowledge item.

When a described change of a characteristic is accomplished it is necessary to update a combined view at the characteristic according to equation defined in step 1.

7.3 Evaluation

For the evaluation process a simple programming course was created. It consisted of 16 learning objects containing simple text, exercise and explanation types. Knowledge items, which learning objects are assigned to, come from ACM classification[2]. The knowledge item space contains 1 476 topics and 4 keywords added specially for our course.

We developed an adaptive web-based educational application that recommends learning objects according to defined domain model. The user model is created automatically based on user's activity (i.e., number of learning object's visits, time spent by reading, interest). Ontology representation is used for the domain (and user) model. Content of the educational course is stored separately from the domain model in relational database structures.

The application is built on three architectural layers — user, application and data. The *data layer* contains learning materials (stored in relational database), semantics of the course — domain ontology and user's characteristics (both in OWL DL). An access to ontologies is provided through the Sesame framework. The *application layer* is responsible for maintenance of user's characteristics using our method and is implemented in Java SE 6. The *user layer* is implemented in JRuby[3] and it is responsible for generating views in the educational course and acquiring information about user's activities. Data about user's actions are processed at the application layer.

In the evaluation we concentrate on validation of the correctness of achieved results and time demands. We describe the evaluation on spreading interest which is performed on actual learning object with regard to user's

[2] ACM classification, http://www.acm.org/about/class/1998/
[3] JRuby, http://jruby.codehaus.org/

feedback and consequent spreading of changes in the domain model. The interest gets a value from the range −3 (the lowest user's interest) to 3 (the highest interest).

The evaluation starts with the empty user model (i.e., with no initialized characteristics). We gradually set user's interest to 3 (to observe changes of interest for actual learning object), then to 2 (to observe spreading change and combination with previous value of interest) and at last to −3 (to verify changes also for negative values). In the first experiment, the interest was adjusted for the actual learning object and afterwards it was spread to respective knowledge items, then continued spreading in the knowledge item space and, at last, spreading of interest continued to respective learning objects for changed knowledge items. In the second experiment the value of interest influenced by its initial value (i.e., from first experiment) is tested. Spreading negative interest was successfully accomplished the same way as in the first experiment.

To evaluate time demands we measured time during spreading *interest* and *knowledge* in the domain model. The results are depicted in the Figure 7.7. The most time expensive is the first experiment, since data acquired from the ontology has not been loaded to cache memory.

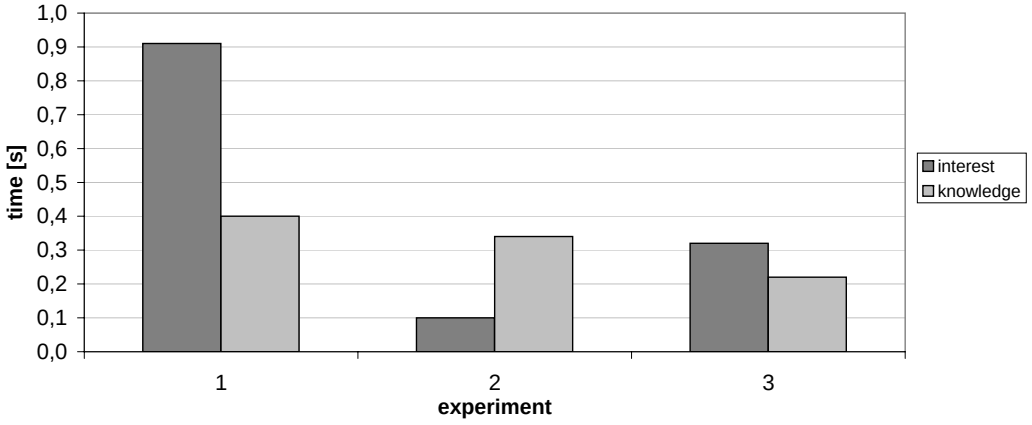

Figure 7.7: Evaluation of the time demands.

In the second experiment the spreading was performed through the same path (i.e., sources of spreading were connected to the same knowledge items) what led to decreased time demands. Spreading in the third experiment was led through a different path and acquiring additional information from the ontology caused increased time demands.

To find out dependence between time demands and number of learning objects in a course we performed another two measurements. We used our

test course (with 16 learning objects) and added other 10 and 20 learning objects with connections to all knowledge items in the course to spread changes to all new learning objects. The achieved results are depicted in Figure 7.8.

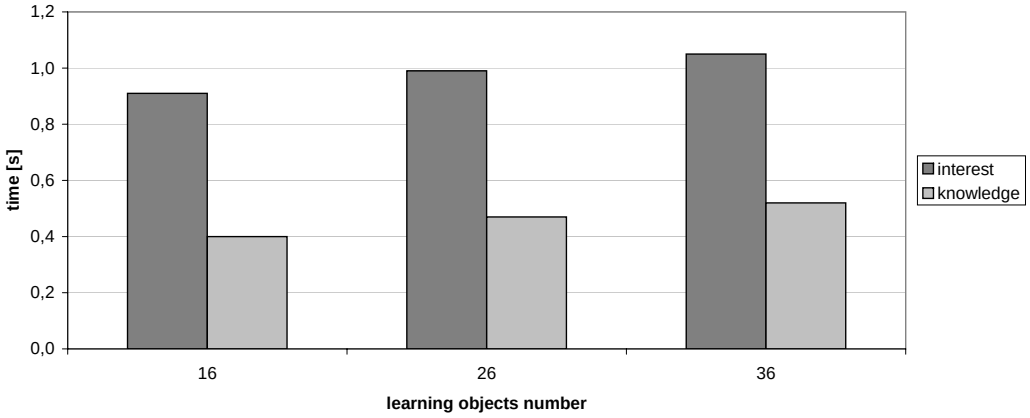

Figure 7.8: Dependency of learning objects number and time for first spreading in domain model.

After adding new learning objects, the first three steps of the spreading are not influenced. The change appears in the fourth step, where changes are spread from knowledge items to (new) learning objects.

7.4 Discussion

Personalized approach to learners provides a way to improve educational process in e-learning courses. It is based on information about learner but this information is subject of change. We described our extension of the domain model so that it enables more accurate adaptation using two related parts — *knowledge item space* and *learning object space* as well as that it can be reused across several educational applications. The domain model and the user model are represented by an ontology and different types of relationships between information contents are defined. An idea of distinguishing types of links between information contents in educational applications and to use ontology to represent links is described in [5]. The advantage of such an approach is in potential subsequent adaptive behavior that can be added to the hypermedia structure and it fosters consistency in linking practices.

There are several ways to model characteristics in a user model. Mostly used are *stereotype* (e.g., Multibook [71]) or *layered* (e.g., AHA! [30], NetCoach [77]) user models. In Multibook learners are divided into groups (stereotypes) according to a selected difficulty level of a course while each

learner is assigned to one stereotype only and is provided with studying materials belonging to the particular stereotype. The stereotype can be changed after successfully passed test. Domain model uses several relationships between information concepts to generate presentation (e.g., *superconcept*, *follows*, *problemsolution*). However, the presentation is adjusted to a stereotype not to a particular user.

In layered user model user's characteristics are modeled according to educational documents in particular course. In this case evaluation of user characteristics' for a visited document and for related parts of the course (characteristic propagation) is provided. Actualization of a characteristic for the course can be based on rules defined by author of the course [30] or on analysis of learner's activity.

An alternative approach to rule based adaptation in educational hypermedia systems is described in [45]. The proposed alternative sequencing method, instead of generating the learning path by populating a concept sequence with available learning resources based on pre-defined adaptation rules, it first generates all possible learning paths that match the learning goal in hand, and then, adaptively selects the desired one, based on the use of a decision model that estimates the suitability of learning resources for a targeted learner. The proposed methodology generates almost accurate sequences avoiding the need for defining complex rules. Unlike our approach, content to be presented is selected according to the suitability function that estimates the suitability of a learning object for a specific learner. We adjust characteristics and then provide the user/learner with the content with the highest relevance.

AHA! provides a propagation of a characteristic's change to other parts of a course via prerequisite relationships between concepts [30]. The propagation is based on author's defined rules. Defining rules for each document (or type of documents) is complicated and time consuming. Furthermore, maintenance of the rules is necessary anytime a new document is added and relations with other documents need to be updated. We use spreading activation to spread (i.e., propagate) a change of characteristics among concepts without involvement of the author.

Personal Reader is an experimental environment supporting personalized learning based on semantic web technologies [34]. The prototype allows providing, annotating and recommending learning material suitable for specific courses in an e-learning context. The user model (here learner profile schema) provides slots for information about a learner and is represented as a RDF document [42]. Provided recommendations are personalized according to the current learning progress of the user. The entire process of recommending is

driven by rules. These rules query for resources and metadata, and reason over distributed data and metadata descriptions.

Our method uses overlay user model and maintenance of user's characteristics is based on spreading activation, i.e. we employed structure of the domain model to spread changes of user characteristics. Our approach allows modeling user characteristics in the entire domain model and not only in the parts that the user has already visited, thus, it provides more accurate recommendations of learning objects to achieve more effective education.

Part IV

Outlook

Chapter 8

Contributions

Proper personalization requires a sufficiently populated user model. The user changes as he/she works with the application (e.g., gains new knowledge). Therefore, the user model should always reflect these changes to be a reliable source for personalization. There are several methods (e.g., implicit or explicit feedback) that can be used. Main contribution of this work is a proposal of three novel methods to automatic acquisition and maintenance of user characteristics that employ semantics provided by ontological representation. Another contribution of this work is its aim at the methods for acquisition and maintenance of user characteristics. Currently, there is less attention paid to these problems (in comparison to adaptive navigation and presentation) in the adaptive hypermedia field. One of the reasons is that changes of the user model are rather considered as a support for the personalization and are not directly visible on presentation layer.

Method for acquiring user characteristics based on questions

The method is aimed at acquisition and maintenance of user characteristics using questions generated in the natural language (here English). The advantage of the method is that questions are generated automatically for concepts from the domain model and afterwards answers are transformed into characteristics in the user model. The entire process (i.e., *when* and *what* question needs to be generated) is driven by user defined rules that can be restricted by question's priority.

The method was evaluated using the job offer and scientific publications domain ontology using our software prototype *ExACT* (Explicit Actualizer). The templates were proposed with regard to their universality. We designed 7 templates for the job offer domain. These templates allow generating questions for 22 characteristics. For the scientific publications domain we pro-

posed 3 templates which allow generating 5 questions. The small amount of templates (questions) in this domain is influenced by straightforward structure of the domain. The prototype uses ontology models that were developed in the course of the projects NAZOU and MAPEKUS. The projects resulted in two pilot applications. However, the prototype was not incorporated into the final pilot applications because it was implemented in the early stages of the projects using different technologies as the pilot applications. Incorporating the prototype into the pilot applications would require time consuming refactoring of the source codes.

The method as it is proposed is not suitable for processing characteristics where two independent instances need to be created since we expect in the question one name of a characteristic, e.g. we wanted to generate a question in the travel ontology to find out user's opinion about accommodation (represented by class *Accommodation*) in a particular destination (class *Destination*).

Method for acquiring user characteristics based on content analysis

The method is aimed at comparison of instances of ontological concepts and is based on the recursive traversing of an instance's structure. The final similarity is the aggregate result of the individual similarities computed for particular properties while their type is considered to select a suitable similarity metric for each property. The introduction of similarity metrics for properties allows us to take advantage of semantics provided by ontological representation, which allowed us to extend similarity with personalized weights reflecting users' individuality.

We have developed the software tool *ConCom* (Concept Comparer) that automates the proposed method. *ConCom* supports the computing of two kinds of similarity — either for all properties or only for properties that are common for both compared instances. Our experiments showed that similarity where all properties are considered is more suitable for discovering user characteristics, while similarity computed for common properties only better mimics the similarity estimated by real users.

Furthermore, we investigated reasons (properties) that influenced user evaluation of content (e.g., interest). We introduced two threshold values used to discover a user's likes and dislikes. From the personalization perspective we were only interested in the two outer sets — positive and negative items. The thresholds were set experimentally for the job offers application domain — the positive threshold to 0.65 and negative threshold to 0.25. The

identified properties can be used by other tools for actualization of characteristics in the user model or for the acquisition of new ones.

The method of the recursive evaluation implemented in the *ConCom* tool is universal and exploits ontological structure of the concept. It is based on acquiring properties and instances (literals) which are connected. Therefore, it can be used also in other application domains. However, in some cases it might be desirable to add additional metrics to achieve better results or to deal with particularities typical for processed domain.

The tool was incorporated into the pilot application that was developed in the course of the research project NAZOU. The tool was being implemented at the time of finishing the pilot application of the project. Therefore, only the part that computes the similarity (without involving the user) is incorporated and it is used to recommend similar instances to a given instance.

Method for maintenance of user characteristics based on spreading activation

The method is aimed at maintenance of user's characteristics where spreading activation is used. Information about the user is in the user model expressed as knowledge or interest about particular learning objects. We developed a prototype of an adaptive web-based educational application that recommends learning objects from the domain model. The educational course was used because relationships among learning objects and their fragments are better defined here than for instance in job offer domain. However, the proposed method use is not restricted to only an educational domain. The prototype was implemented in the course of the PeWePro project. This project is still in progress and the prototype is being incorporated along with other prototypes into a portal solution that is aimed at learning programming.

We described our extension of domain model that enables more accurate adaptation using two related parts — knowledge item spaces and learning object spaces — that can be reused across several educational applications. The course consisted of 16 learning objects containing simple text, exercise and explanation types. Knowledge items, which learning objects are assigned to, came from ACM classification. Knowledge item space contained 1 476 topics and 4 keywords added especially for our course.

Benefit of this approach is in changing user's characteristics in the entire domain model and not only in the parts that the user has already visited. Results of our method are used in the process of personalization and thus more accurate recommendation of learning objects is performed what helps to achieve more effective education.

The described methods (including implementation of the respective software tools) were proposed and evaluated in the course of the following research projects:

Project NAZOU: Tools for Acquisition, Organisation, and Maintenance of Knowledge in an Environment of Heterogeneous Information Resources (1025/2004)

Project leader: Pavol Návrat for STU

Supported by: State programme of research and development "Establishing of Information Society"

Duration: September 2004 – May 2008

Project MAPEKUS: Modeling and Acquisition, Processing and Employing Knowledge about User Activities (APVT-20-007104)

Project leader: Mária Bieliková

Supported by: Slovak Research and Development Agency

Duration: January 2005 – April 2008

Project PeWePro: Adaptive web-based portal for learning programming (KEGA 3/5187/07)

Project leader: Mária Bieliková

Supported by: Cultural and Educational Grand Agency of the Ministry of Education of Slovak Republic

Duration: January 2005 – December 2009

Models of software systems in the semantic web environment (VEGA 1/3102/06)

Project leader: Pavol Návrat

Supported by: Scientific Grant Agency of the Ministry of Education of Slovak Republic and the Slovak Academy of Sciences

Duration: January 2006 – December 2008

Bibliography

[1] Riccardo Albertoni and Monica De Martino. Semantic similarity of ontology instances tailored on the application context. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275, pages 1020–1038. Springer Berlin / Heidelberg, 2006.

[2] Anton Andrejko, Michal Barla, and Mária Bieliková. Ontology-based user modeling for web-based information systems. In Wita Wojtkowski, W. Gregory Wojtkowski, Jože Zupancic, Gabor Magyar, and Gabor Knapp, editors, *Advances in Information Systems Development, New Methods and Practice for the Networked Society*, volume 2, pages 457–468. Springer Science+Business Media, New York, 2007.

[3] Anton Andrejko, Michal Barla, and Michal Tvarožek. Comparing ontological concepts to evaluate similarity. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for Acquisition, Organisation and Presenting of Information and Knowledge: Research Project Workshop*, pages 71–78, Bystrá dolina, Nízke Tatry, 2006.

[4] Ion Androutsopoulos, Spyros Kallonis, and Vangelis Karkaletsis. Exploiting OWL ontologies in the multilingual generation of object descriptions. In *Proceedings of the 10th European Workshop on Natural Language Generation*, page 150–155, Aberdeen, Scotland, 2005.

[5] Miguel Ángel Sicilia, Elena García, Paloma Díaz, and Ignacio Aedo. Using links to describe imprecise relationships in educational contents. *International Journal for Continuing Engineering Education and Lifelong Learning*, 14(3):260–275, 2004.

[6] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts, 2004.

[7] Liliana Ardissono and Pietro Torasso. Dynamic user modeling in a web store shell. In *14th European Conference on Artificial Intelligence*, pages 621–625, Berlin, Germany, 2000.

[8] Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[9] Michal Barla and Mária Bieliková. Estimation of user characteristics using rule-based analysis of user logs. In *Data Mining for User Modeling: Proceedings of Workshop held at UM2007*, pages 5–14, 2007.

[10] Shlomo Berkovsky. Decentralized mediation of user models for a better personalization. In V. P. Wade, H. Ashman, and B. Smyth, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems, AH 2006*, volume LNCS 4018, page 404–408, Dublin, Ireland, 2006. Springer.

[11] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, 18(3):245–286, 2008.

[12] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[13] Abraham Bernstein, Esther Kaufmann, Christoph Bürki, and Mark Klein. How similar is it? towards personalized similarity measures in ontologies. In Otto K. Ferstl, Elmar J. Sinz, Sven Eckert, and Tilman Isselhorst, editors, *7th International Conference Wirtschaftsinformatik (WI-2005)*, page 1347–1366, Bamberg, Germany, 2005. Physica-Verlag.

[14] Mária Bieliková. Presentation of adaptive hypermedia on the web. In Lubos Popelínský, editor, *DATAKON 2003*, pages 72–91, Brno, Czech Republic, 2003.

[15] Mária Bieliková and Rastislav Habala. University course support by web-based adaptive e-board. In F. Jakab, L. Samuelis, I. Sivý, and M. Bučko, editors, *ICETA 2004*, page 395–402, Košice, Slovak Republic, 2004.

[16] Mária Bieliková and Jaroslav Kuruc. Sharing user models for adaptive hypermedia applications. In *ISDA 2005*, pages 506–511, Wroclaw, Poland, 2005. ACM Press.

[17] Mária Bieliková and Michal Moravčík. Modeling the content of adaptive web-based system using an ontology. In *Proceedings of 1st International Workshop on Semantic Media Adaptation and Personalization SMAP 2006*, pages 115–120, Athens, Greece, 2006. IEEE Computer Society.

[18] Mária Bieliková and Pavol Návrat. Modeling and acquisition, processing and employing knowledge about user activities in the internet hyperspace (in slovak). In *Znalosti 2007: Proceedings of the 6th annual conference*, pages 368–371, Ostrava, Czech Republic, 2007.

[19] Gilles Bisson. Why and how to define a similarity measure for object based representation systems. In *Towards Very Large Knowledge Bases*, page 236–246. IOS Press, Amsterdam, 1995.

[20] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.

[21] Peter Brusilovsky. Adaptive and intelligent technologies for web-based education. *Special Issue on Intelligent Systems and Teleteaching*, 4:19–25, 1999.

[22] Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2):87–110, 2001.

[23] Peter Brusilovsky. Developing adaptive educational hypermedia systems: From design models to authoring tools. In S. Blessing, T. Murray, and S. Ainsworth, editors, *Authoring Tools for Advanced Technology Learning Environments*, pages 377–409. Dordrecht: Kluwer Academic Publishers, 2003.

[24] Peter Brusilovsky, John Eklund, and Elmar Schwarz. Web-based education for all: A tool for developing adaptive courseware. *Computer Networks and ISDN Systems*, 30(1-7):291–300, 1998.

[25] Peter Brusilovsky and C. Tasso. Preface to special issue on user modeling for web information retrieval. *User Modeling and User-Adapted Interaction*, 14(2-3):147–157, November 02, 2004 2004.

[26] Charles Callaway and Tsvi Kuflik. Using a domain ontology to mediate between a user model and domain applications. In Peter Brusilovsky, Charles Callaway, and Andreas Nürnberger, editors, *Workshop on New Technologies for Personalized Information Access*, pages 13–22, Edinburgh, Scotland, UK, 2005.

[27] Cristina Carmona and Ricardo Conejo. A learner model in a distributed environment. In Paul De Bra and Wolfgang Nejdl, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems: Third International Conference, AH 2004*, volume 3137 of *Lecture Notes in Computer Science*, pages 353–359, Eindhoven, The Netherlands, 2004.

[28] Fabio Crestani and Puay Leng Lee. Searching the web by constrained spreading activation. *Information Processing Management*, 36(4):585–605, 2000.

[29] Alexandra Cristea and Paul De Bra. ODL education environments based on adaptivity and adaptability. In *Proceedings of the AACE E-Learn 2002*, pages 232–239, Montreal, Canada, 2002.

[30] Paul De Bra, Ad Aerts, Bart Berden, Barend de Lange, Brendan Rousseau, Tomi Santic, David Smits, and Natalia Stash. AHA! the aadaptive hypermedia architecture. In *ACM Conference on Hypertext and Hypermedia*, pages 81–84, Nottingham, UK, 2003.

[31] Ronald Denaux, Lora Aroyo, and Vania Dimitrova. An approach for ontology-based elicitation of user models to enable personalization on the semantic web. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1170–1171, Chiba, Japan, 2005. ACM Press.

[32] Li Ding, Pranam Kolari, Zhongli Ding, and Sasikanth Avancha. Using ontologies in the semantic web: A survey. In Raj Sharman, Rajiv Kishore, and Ram Ramesh, editors, *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*, volume 14, page 79–113. Springer, 2007.

[33] Peter Dolog and Mária Bieliková. Hypermedia systems modeling framework. *Computers and Infromatics*, 21(3):221–239, 2002.

[34] Peter Dolog, Nicola Henze, and Wolfgang Nejdl. The personal reader: Personalizing and enriching learning resources using semantic web technologies. In *Proc. of the AH 2004*, pages 85–94. Springer Verlag, 2004.

[35] Anna Formica and Michele Missikoff. Concept similarity in symontos: An enterprise ontology management tool. *Computer Journal*, 45(6):583–594, 2002.

[36] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. In *Journal on Data Semantics IX*, pages 1–38. Springer Berlin/Heidelberg, 2007.

[37] Gustavo González. *Towards Smart User Models for Open Environments*. PhD thesis, University of Girona, 2003.

[38] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In Nicola Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[39] Stefan Haustein and Jörg Pleumann. Easing participation in the semantic web. In *International Workshop on the Semantic Web 2002*, Hawaii, 2002.

[40] Dominik Heckmann, Tim Schwartz, Boris Brandherm, Michael Schmitz, and Margeritta von Wilamowitz-Moellendorff. GUMO — the general user model ontology. In Liliana Ardissono, editor, *Proceedings of the 10th International Conference on User Modeling (UM'2005)*, LNAI 3538, pages 428–432, Edinburgh, UK, 2005. Springer-Verlag Berlin Heidelberg.

[41] Albrecht Heeffer. Negative numbers as an epistemic difficult concept: Some lessons from history, 2008. [Online; accessed February 3rd, 2008]. http://logica.ugent.be/albrecht/thesis/HPM2008.pdf.

[42] Nicola Henze and Marc Herrlich. The personal reader: A framework for enabling personalization services on the semantic web. In *Proceedings of the Twelfth GIWorkshop on Adaptation and User Modeling in Interactive Systems (ABIS 04*, 2004.

[43] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building OWL ontologies using the protégé-OWL plugin and CO-ODE tools edition 1.0, 2004. [Online; accessed February 3rd, 2008]. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf.

[44] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: The state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[45] Pythagoras Karampiperis and Demetrios Sampson. Adaptive learning resources sequencing in educational hypermedia systems. *Educational Technology & Society*, 8:128–147, 2005.

[46] Alenka Kavcic. The role of user models in adaptive hypermedia systems. In *10th Mediterranean Electrotechnical Conference MEleCon 2000*, Lemesos, Cyprus, 2000.

[47] Judy Kay. The um toolkit for cooperative user modelling. *User Modelling and User-Adapted Interaction*, 4:149–196, 1995.

[48] Judy Kay. Stereotypes, student models and scrutability. *Lecture Notes in Computer Science*, 1839:19–30, 2000.

[49] Judy Kay. User modeling for adaptation. In Constantine Stephanidis, editor, *User Interfaces for All*, Human Factors Series, pages 271–294, Florence, Italy, 2000.

[50] Judy Kay and Andrew Lum. Ontology-based user modeling for the semantic web. In *10th International Conference on User Modeling (UM'05): Workshop 8*, pages 11–19, Edinburgh, Scotland, 2005.

[51] Toralf Kirsten, Andreas Thor, and Erhard Rahm. Instance-based matching of large life science ontologies. In *Data Integration in the Life Sciences*, volume 4544, pages 172–187. Springer Berlin/Heidelberg, 2007.

[52] Alfred Kobsa. User modeling: Recent work, prospects and hazards. In M. Schneider-Hufschmidt, T. Kühme, and U. Malinowski, editors, *Adaptive user interfaces: Principles and practice*, pages 111–128, Amsterdam, North-Holland, 1993.

[53] Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11:49–63, 2001.

[54] Radovan Kostelník and Mária Bieliková. Web-based environment using adapted sequences of programming exercises. In M. Beneš, editor, *ISIM 2003: Proceedings of Information Systems Implementation and Modelling*, pages 33–40, Brno, Czech Republic, 2003. MARQ Ostrava.

[55] Geert-Jan M. Kruijff. Context-sensitive utterance planning for CCG. In *Proceedings of the 10th European Workshop on Natural Language Generation*, page 83–90, Scotland, 2005.

[56] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language: W3c recommendation 10 february 2004, 2004. [Online; accessed July 7th, 2008]. http://www.w3.org/TR/owl-features/.

[57] Stuart Edward Middleton. *Capturing knowledge of user preferences with recomender systems*. PhD thesis, University of Southampton, 2003.

[58] David Milward and Martin Beveridge. Ontology-based dialogue systems. In *18th International Joint Conference on Artificial Intelligence (IJCAI03): 3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2003.

[59] Pavol Návrat, Mária Bieliková, and Viera Rozinajová. Acquiring, organising and presenting information and knowledge from the web. In *CompSysTech'06*. Bulgarian Chapter of ACM, 2006.

[60] István-Tibor Nébel, Barry Smith, and Ralf Paschke. A user profiling component with the aid of user ontologies. In *Workshop Learning, Teaching, Knowledge, Adaptivity*, Karlsruhe, 2003.

[61] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *ACM SIGMOD Record*, 33(4):65–70, 2004.

[62] Natalya Fridman Noy and Mark A. Musen. PROMPT: algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 450–455. MIT Press/AAAI Press, 2000.

[63] Leo Obrst. Ontologies for semantically interoperable systems. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 366–369, New Orleans, LA, USA, 2003. ACM Press.

[64] M. O'donnell, C. Mellish, J. Oberlander, and A. Knott. ILEX: an architecture for a dynamic hypertext generation system. *Natural Language Engineering*, 7(3):225–250, 2001.

[65] Harrie Passier and Johan Jeuring. Ontology based feedback generation in design-oriented e-learning systems. In *Proceedings of the IADIS International conference, e-Society*, volume II, pages 992–996, 2004.

[66] René Pázman. Ontology search with user preferences. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for acquisition, organization and presenting of information and knowledge: Research project workshop*, page 139–147, 2006.

[67] René Pázman. Values normalization with logic programming. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for acquisition, organization and presenting of information and knowledge: Research project workshop*, pages 134–141, Horský hotel Poľana, Slovakia, 2007.

[68] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[69] Philip Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

[70] Elaine Rich. User modeling via stereotypes. In *Readings in intelligent user interfaces*, pages 329–342. Morgan Kaufmann Publishers Inc., 1998.

[71] Cornelia Seeberg, Achim Steinacker, Klaus Reichenberger, Abdulmotaleb El Saddik, Stephan Fischer, and Ralf Steinmetz. From the user's needs to adaptive documents. In *Proceedings of the Integrated Design & Process Technology Conference*, pages 226–238, 2000.

[72] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.

[73] Peter Tiňo and Gabriela Polčicová. Topographic organization of user preference patterns in collaborative filtering. *Neural network world*, 13(3):311–324, 2003.

[74] Michal Tury and Mária Bieliková. An approach to detection ontology changes. In *ICWE '06: Workshop proceedings of the sixth international conference on Web engineering*, Palo Alto, California, 2006. ACM Press.

[75] Amos Tversky. *Preference, Belief, and Similarity: Selected Writings*. MIT Press, 2003.

[76] Robin van Meteren and Maarten van Someren. Using content-based filtering for recommendation. In G. Potamias, V. Moustakis, and M. van Someren, editors, *ECML/MLNET Workshop on Machine Learning and the New Information Age*, pages 47–56, Barcelona, Spain, 2000.

[77] Gerhard Weber, Hans christian Kuhl, and Stephan Weibelzahl. Developing adaptive internet based courses with the authoring system netcoach.

In *Proceedings of the Third Workshop on Adaptive Hypermedia, AH2001*, pages 226–238. Springer, 2001.

[78] Hongjing Wu, Geert-Jan Houben, and Paul De Bra. Supporting user adaptation in adaptive hypermedia applications. In *On-line Conference and Informatiewetenschap 2000*, De Doelen, Rotterdam, 2000.

[79] Michael Yudelson, Tatiana Gavrilova, and Peter Brusilovsky. Towards user modeling meta-ontology. In Liliana Ardissono, Paul Brna, and Antonija Mitrovic, editors, *User Modeling 2005: 10th International Conference, UM 2005*, pages 448–452, Edinburgh, Scotland, UK, 2005. Lecture Notes in Computer Science.

Appendix A

About the author

Anton Andrejko was born in Stropkov, Slovakia on July of 26^{th}, 1980. He received his Master degree in Industrial engineering from Technical University in Košice, Faculty of Electrical Engineering and Informatics in 2004. He is presently a PhD student at Slovak University of Technology, Faculty of Informatics and Information Technologies. He works in the field of user modeling and personalization.

In the period of October 2005 through September 2008 he worked as a researcher at Faculty of Informatics and Information Technologies and cooperated on several research projects. Within this period he published several papers, namely 1 paper in a journal, 8 papers at international conferences and workshops, chapters in 2 books and 13 papers at local or national conferences. The full list of publications is in Appendix B.

He was a member of the organizing committee for the Znalosti 2008 conference. He also participated in organizing meetings for the Personalized Web group held at Faculty of Informatics and Information Technologies in period of 2005–2008. His others interests include typography, proper publishing and digital photography.

Appendix B

Publications by author

Journals

1. Anton Andrejko and Mária Bieliková. Comparing instances of ontological concepts for personalized recommendation in large information spaces. *Computing and Informatics*. To appear.

International Conferences

2. Anton Andrejko and Mária Bieliková. Personalized Comparing Instances of Domain Ontology Concepts. In Phivos Mylonas, Manolis Wallace, and Marios Angelides, editors, *Proceedings of SMAP 2008 — 3rd International Workshop on Semantic Media Adaptation and Personalization*, pages 76–81, Prague, Czech Republic, 2008. CS IEEE Press.

3. Anton Andrejko and Mária Bieliková. Investigating similarity of ontology instances and its causes. In Vera Kurkova, Roman Neruda, and Jan Koutnik, editors, *Artificial Neural Networks — ICANN 2008*, LNCS 5164, pages 1–10, Prague, Czech Republic, 2008. Springer.

4. Michal Šimún, Anton Andrejko, and Mária Bieliková. Maintenance of learner's characteristics by spreading a change. In *Learning to Live in the Knowledge Society*, volume 281 of *IFIP International Federation for Information Processing*, pages 223–226. Springer Boston, 2008.

5. Peter Bartalos, Michal Barla, György Frivolt, Michal Tvarožek, Anton Andrejko, Mária Bieliková, and Pavol Návrat. Building ontological base for experimental evaluation of semantic web applications. In

J. van Leeuwen, G.F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plášil, editors, *SOFSEM 2007*, LNCS 4362, pages 682–692. Springer, 2007.

6. Anton Andrejko, Michal Barla, and Mária Bieliková. Ontology-based user modeling for web-based information systems. In Wita Wojtkowski, W. Gregory Wojtkowski, Jože Zupancic, Gabor Magyar, and Gabor Knapp, editors, *Advances in Information Systems Development, New Methods and Practice for the Networked Society*, volume 2, pages 457–468. Springer Science+Business Media, New York, 2007.

7. Anton Andrejko, Michal Barla, Mária Bieliková, and Michal Tvarožek. User characteristics acquisition from logs with semantics. In *ISIM 2007 Information Systems and Formal Models: 10th International Conference on Information System Implementation and Modeling*, pages 103–110, 2007.

8. Michal Šimún, Anton Andrejko, and Mária Bieliková. Ontology-based models for personalized e-leraning environment. In *ICETA 2007: 5th Internation Conference on Emerging e-Learning Technologies and Applications*, pages 335–340. Elfa, Stará Lesná, Slovak Republic, 2007.

9. Mária Bieliková, Jaroslav Kuruc, and Anton Andrejko. Learning Programming with Adaptive Web-Based Hypermedia System AHA!. In *ICETA 2005: 4th Internation Conference on Emerging e-Learning Technologies and Applications*, pages 251–256. Elfa, Košice, Slovak Republic, 2005.

Chapters in books

10. Mária Bieliková, Pavol Návrat et al. *Selected software engineering studies 1 (in Slovak): Advanced methods of designing program systems.* Edition of research texts in in-formatics and information technologies, 232 pages. Faculty of informatics and information technologies STU Bratislava, 2006, ISBN 80-227-2464-5.

11. Mária Bieliková, Pavol Návrat et al. *Selected software engineering studies 2 (in Slovak): Advanced methods of designing program systems.* Edition of research texts in in-formatics and information technologies, 226 pages. Faculty of informatics and information technologies STU Bratislava, 2006, ISBN 80-227-2531-5.

Local and National Conferences

12. Anton Andrejko and Mária Bieliková. Estimating similarity of the ontological concepts instances for the adaptive applications based on Semantic Web (in Slovak). In V. Snášel, editor, *Znalosti 2008: Proceedings of the 7th annual conference*, pages 30–41. STU, Bratislava, 2008.

13. Michal Šimún, Anton Andrejko, and Mária Bieliková. Initialization and actualization of the user model in job offers search on the Web (in Slovak). In *Znalosti 2007: Proceedings of the 6th annual conference*, pages 109–120. VŠB-Technická univerzita Ostrava, 2007.

14. Anton Andrejko and Mária Bieliková. Comparing Instances of the Ontological Concepts. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for acquisition, organization and presenting of information and knowledge (2): Research project workshop*, pages 26–35, Horský hotel Poľana, Slovakia, 2007.

15. Tomáš Klempa, Anton Andrejko, and Mária Bieliková. Maintenance of user characteristics in the user model based on generating questions according to domain ontology concepts (in Slovak). In Peter Vojtáš, editor, *ITAT 2007: Informačné technológie — Aplikácie a Teória: Zborník príspevkov prezentovaných na pracovnom seminári ITAT Poľana*, pages 3–8, Poľana, Slovakia, 2007.

16. Anton Andrejko and Mária Bieliková. Estimating similarity of the ontological concepts instances for personalization purposes (in Slovak). In F. Babič and J. Paralič, editors, *WIKT 2007 Proceedings: 2nd Workshop on Intelligent and Knowledge oriented Technologies*, pages 46–49. Technická univerzita, Košice, 2008.

17. Anton Andrejko, Michal Barla, Mária Bieliková, and Michal Tvarožek. Software tools for acquisition of user characteristics. In P. Vojtáš, T. Skopal, editors, *Datakon 2006, Proceedings of the Annual Database Conference*, pages 139–148. Brno, Czech Republic, 2006.

18. Anton Andrejko, Michal Barla, and Michal Tvarožek. Comparing ontological concepts to evaluate similarity. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for Acquisition, Organisation and Presenting of Information and Knowledge: Research Project Workshop*, pages 71–78, Bystrá dolina, Nízke Tatry, 2006.

19. Michal Tvarožek, Michal Barla, Mária Bieliková, Vladimír Grlickyý, Anton Andrejko, Roman Filkorn, and Peter Bartalos. Presentation and Personalization of Information in the Semantic Web. In Pavol Návrat, Pavol Bartoš, Mária Bieliková, Ladislav Hluchý, and Peter Vojtáš, editors, *Tools for Acguisition, Organisation and Presenting of Information and Knowledge: Research Project Workshop*, pages 201–207, Bystrá dolina, Nízke Tatry, 2006.

20. Ján Krausko, Michal Barla, Anton Andrejko. Semantic searching in job offer domain (in Slovak). In *ITAT 2006: Information Technologies — Applications and Theory: Workshop on Theory and Practice of Information Technologies*, pages 75–80, 2006.

Student Research Conferences

21. Anton Andrejko. Enumerating Similarity of Ontology Instances in Regard to User Model. In M. Bieliková, editor, *Student Research Conference 2008: 4th Student Research Conference in Informatics and Information Technologies*, pages 142–149, STU, Bratislava, 2008.

22. Anton Andrejko. Ontological Concepts Similarity Influenced by the User. In M. Bieliková, editor, *Student Research Conference 2007: 3rd Student Research Conference in Informatics and Information Technologies*, pages 172–179, STU, Bratislava, 2007.

23. Anton Andrejko. Approaches to the User Modeling. In M. Bieliková, editor, *Student Research Conference 2006: Proceedings in Informatics and Information Technologies*, pages 160–167, STU, Bratislava, 2006.

24. Anton Andrejko. Improving Adaptive Hypermedia by Adding Semantics. In M. Bieliková, editor, *Student Research Conference 2005: Proceedings in Informatics and Information Technologies*, pages 234–241, STU, Bratislava, 2005.

Appendix C

Software tools for acquisition and maintenance of the user model

C.1 Explicit Actualizer

Each record about the concept in the overlay user model needs to be initialized before it is used and later needs to be maintained updated. Traditional approaches to initialization and maintenance of the user model mostly consider a closed information space and are proposed for the particular application. We proposed a novel method for acquiring information for the user model by asking questions.

Tool description

Experimental evaluation of the method for acquisition based on questions, as described in Chapter 5, was performed using the software tool called Explicit Actualizer (abbreviated *ExACT*). The tool's specification is as follows:

Technologies used: Java, Apache Cocoon, Sesame repository

Inputs: User model, domain model, binding characteristics, templates with vocabulary, rules

Outputs: Changes of characteristics in the user model

ExACT is a tool which is responsible for acquisition of information from the user and their consequent transformation into characteristics in the user model. When the tool is lunched for the first time a list of all possible characteristics is created according to the domain model concept. The list is ordered with regard to the priority of the characteristics. As the user works

with the application he/she is asked questions with respect to the predefined rules. The questions with the higher priority are asked first.

User's answer to a question is transformed into a characteristic in the user model. In the case, when a respective characteristic already exists in the user model, its value is updated to a newly acquired value from the user. On the other hand, if the characteristic does not exist yet, a new instance is created.

Configuring to other domains

When using *ExACT* in other application domains, more attention should be paid especially to binding characteristics. Furthermore, it is necessary to specify rules that are valid for selected application domain.

C.2 Concept Comparer

Many approaches acquire user characteristics for a user model to be populated or kept up to date and this way provide a basis for successful personalization of visible aspects in adaptive web-based applications. Some information can be acquired directly from the user (e.g., the user is asked a question, fills in a form), observations of user's behaviour while working with the application, analysis of logs on the web server or analysis of the presented content. We focus on the analysis of the presented content especially on evaluation of the similarity to find common or different aspects of the content.

Tool description

Experimental evaluation of the method for acquisition based on a content analysis, as described in Chapter 6, was performed using the software tool called Concept Comparer (abbreviated *ConCom*). The tool's specification is as follows:

Technologies used: Java, Sesame repository

Inputs: URIs identifying instances in domain model, user model, similarity metrics

Outputs: Quantitatively expressed similarity measure between instances

ConCom computes similarity measure for two instances of ontological concepts given in the command line. The result is a similarity measure computed using respective similarity metrics. Furthermore, *ConCom* provides an interface that allows searching for the most similar instances to the given one. *ConCom* uses Log4J[1] logging utility and open source Similarity Measure Library — SimMetrics[2].

ConCom is not a standalone application as the tool is proposed to be included in other applications/tools, which will call its interface methods. The method of the recursive evaluation implemented in the *ConCom* tool is universal and exploits ontological structure of the concept. It is based on acquiring properties and instances (literals) which are connected. Therefore, it can be used also in other application domains. However, in some cases it might be desirable to add additional metrics to achieve better results or to deal with particularities typical for processed domain.

Configuring to other domains

When using *ConCom* in other application domains, more attention should be paid especially to inverse properties because they cause loops. Inverse properties can be identified through *owl:InverseOf* property of OWL language. However, query returns both, i.e. the property and its inverse property. Therefore, it is necessary to fill in the list of inverse properties for given domain in the configuration file to be ignored.

C.3 Student Model Updater

Learners have different knowledge, interests and goals and thus require personalized learning content. Personalization consists of two processes — estimation of user's characteristics in a user model and adjusting (or adaptation) of the educational content according to modelled user's characteristics. We proposed a novel method for maintenance of user's characteristics based on spreading a change.

Tool description

Experimental evaluation of the method for maintenance of user characteristics based on spreading, as described in Chapter 7, was performed using

[1]Log4J, http://logging.apache.org/log4j/
[2]SimMetrics, http://www.dcs.shef.ac.uk/~sam/simmetrics.html

the software tool called Student Model Updater (abbreviated *SMUter*). The tool's specification is as follows:

Technologies used: Java, XML (JAXB), Sesame repository

Inputs: User model, events reported from presentation layer — learning object reading time, feedback from currently displayed learning object (knowledge and interest)

Outputs: Changes of characteristics in user model

SMUter is a tool which responsible for changing values of user characteristics in the user model. A user characteristic related to a particular concept determines user characteristics for all learning objects which fall within the scope of the concept. That means that we are able to estimate and maintain learner's characteristics also for the learning objects which have not been visited yet.

While a user works with a learning object, *SMUter* collects his/her characteristics for that learning object (e.g., estimated interest). Since there is a connection between learning objects and other parts of the domain model, we spread a change of the characteristic to other, related parts of the domain model. *SMUter* is not a standalone application as the tool is proposed to be included in an educational application or portal.

Configuring to other domains

SMUter was implemented with regard to the needs of the project PeWePro and uses specific domain and user models from the project. An employment of the tool in other application domains would require changes in the configuration file. Furthermore, if a new application domain uses different models, appropriate modifications in the implementation are necessary.

Index