Computing and Informatics, Vol. 28, 2009, 429–452

COMPARING INSTANCES OF ONTOLOGICAL CONCEPTS FOR PERSONALIZED RECOMMENDATION IN LARGE INFORMATION SPACES

Anton Andrejko, Mária Bieliková

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
e-mail: {andrejko, bielik}@fiit.stuba.sk

Revised manuscript received 24 February 2009

Abstract. We present a novel method for instance comparison of ontological concepts with regard to personalized content presentation and/or navigation in large information spaces. We assume that comparing properties of documents which users found interesting leads to discovery of information about users' interests specifically when considering Semantic Web applications where documents or their parts are represented by ontological concepts. We employ the ontology structure and different similarity metrics for datatype and object properties and investigate reasons behind user interest in the presented content. Moreover, we propose and evaluate an approach to instance similarity computation for a particular user while also considering the user's individual preferences.

Keywords: Ontology, concept, instance, recursion, object property, datatype property, similarity, metrics, personalization, user model, user characteristics

Mathematics Subject Classification 2000: 68T20, 68T30

1 INTRODUCTION

The amount of available information in large information spaces (e.g., the Web) is continuously growing fast. Presently, most of the available information 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.

We consider Semantic Web applications where metadata describes semantics and ontologies are typically used for metadata representation and reasoning [7]. Although an ontology is defined as a shared conceptualization [27] 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 fields known as ontology mapping, merging and alignment.

In this process, instances can also be involved 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. Recently, several approaches have emerged that use ontological representation to store more complex objects but still do not sufficiently address the processing of ontological concept instances. For example, instance similarity could be used in several areas, e.g. clustering or ontological repository maintenance.

The analysis of the content presented to users is a suitable source of information [6] 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 [26] 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 (i.e., user's interest) is known we can acquire some characteristics by employing content analysis in combination with similarity. Since the rating varies, the similarity can be used to analyze the reasons why it is low or high. For instance, let us consider job offers 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 (e.g., one in London and one in Washington, D.C.) and have different ratings, 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 located in London) or whether he or 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 infor-

mation about user preferences can be discovered using more sophisticated heuristics.

In this paper 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.

The paper is structured as follows. In Section 2 we describe our method for comparison of instances of ontological concepts. In Section 3 we present similarity metrics employed in similarity estimation. Section 4 deals with the numerical estimation of similarity and Section 5 presents our extension to similarity estimation that takes into account user characteristics. In Section 6 we present the results of evaluation, in Section 7 we give an overview of related work. Finally, Section 8 contains our concluding remarks.

2 COMPARISON BY RECURSIVE TRAVERSING OF ONTOLOGY INSTANCES

In Semantic Web applications documents or their parts are represented via (hierarchically organized) ontological concepts, which describe a set of real objects (i.e., concept instances) [9]. In our work we use the OWL Web Ontology Language[1] for ontology representation and particularly its DL sublanguage. *Datatype* and *object* properties are used to assert specific facts about instances. Datatype properties express relations between concept instances and RDF literals and XML Schema datatypes. Object properties express relations between two instances.

We have proposed a domain independent method based on recursive evaluation of instance properties to compute their similarity. Since the method was proposed as a part of the NAZOU[2] research project [20], we performed experiments in the job offers domain. Therefore, we provide examples from the job offers domain (Figure 1) and if necessary describe specifics of other application domains.

Rectangles used in Figure 1 represent instances of the concepts. Every such instance has its unique identifier but we present only its label for clarity (e.g., *Salary*). To distinguish between object and datatype properties a dashed line is used for the 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.

The main idea of the method is based on the evaluation of common property pairs present in both instances. The rough principle of the method illustrating

[1] OWL Web Ontology Language, `http://www.w3.org/2004/OWL/`

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

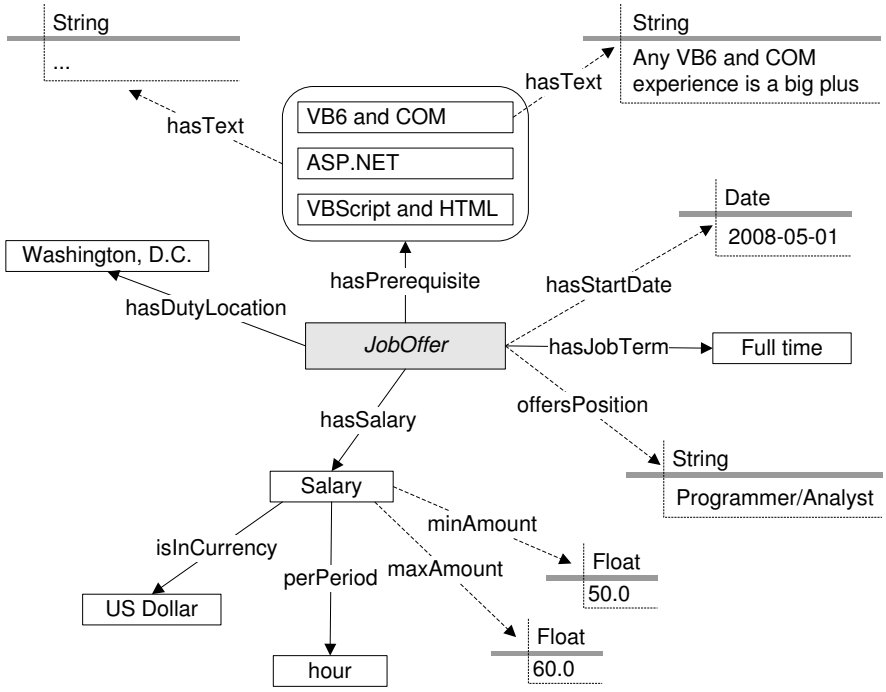

Fig. 1. A sample part of a job offer instance

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,
- single/multiple in one instance only.

When the property has a single occurrence in both instances (e.g., the *hasStartDate* property), then the similarity of related elements (*instances* in the case of object properties or *literals* 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, Washington, D.C. is connected to a job offer with the property *hasDutyLocation*. Since more than one job offer can

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
```

be located in Washington, D.C., we require all of them to refer to the same instance (e.g., Washington, D.C. *isDutyLocationOf* other job offers). Consequently, 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 to the examined property. However, loops can still occur, for example, 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* property) in an instance are the most complex case we have to address. In this case, two sets are constructed which contain elements which are 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 between the two instances. 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 in one instance only, we estimate similarity of values attached to the property as equal to 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.

3 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.

3.1 Datatype Metrics

To compute similarity between literals connected to a datatype property any string based metrics can be used[3]. To achieve better results, the 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:

$$\mathrm{sim}_{logical}(x, y) = \begin{cases} 1.0 & x = y \\ 0.0 & \text{otherwise.} \end{cases} \tag{1}$$

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:

[3] 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`

$$\mathrm{sim}_{date}(x, y) = \begin{cases} 0.0 & |x - y| \geq N \\ 1.0 - \dfrac{|x - y|}{N} & \text{otherwise,} \end{cases} \tag{2}$$

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 is. 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 resolved 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 [24] 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. For normalization of values where different units were used employing Prolog is more suitable. 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 [14] 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 get positive and also negative values are rarer than properties getting either positive or negative values only. We consider the similarity measure of two opposite numbers as equal to 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:

$$\mathrm{sim}_{numerical}(x, y) = 1.0 - \frac{||x| - |y||}{\max(|x|, |y|)}. \tag{3}$$

When comparing *strings*, a simple comparison provided by a default method in any programming language (e.g., in Java) does not give satisfactory results. Me-

thods *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.

3.2 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) [1, 25]. 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 a 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 [25]. 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 [4].

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:

$$\mathrm{sim}_{taxonomy}\left(instance_1, instance_2\right) = \frac{CommonConcepts(instace_1, instance_2)}{\max\left(\mathrm{depth}(instance_1), \mathrm{depth}(instance_2)\right)}. \quad (4)$$

Two examples are depicted in Figure 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. For the left example $\mathrm{sim}_{taxonomy}(instance_1, instance_2) = 2/4 = 0.5$, for the right example where both instances belong to the same concept in taxonomy $\mathrm{sim}_{taxonomy}(instance_1, instance_2) = 3/3 = 1.0$.

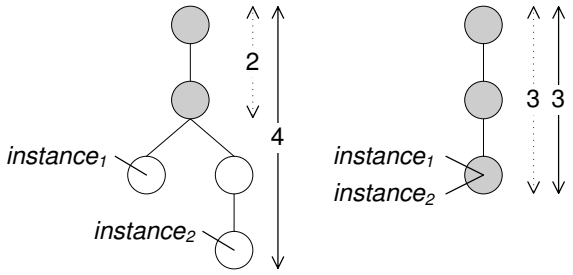

Fig. 2. Example of taxonomy distance computed for $instance_1$ and $instance_2$

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 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. Finding pairs with very low similarity measures can be prevented by using a critical threshold value. The resulting algorithm for finding relevant pairs is shown in Algorithm 2.

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 Figure 3. 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

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
    foreach item in matrix do
        if item.row = X or item.column = Y then
            set item to null
        end if
    end foreach
end while
```

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.

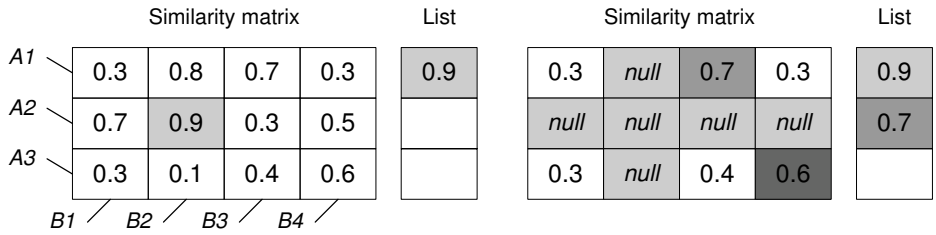

Fig. 3. Identifying relevant pairs from sets

4 SIMILARITY ESTIMATION

Similarity plays an important role in human life and in many cases people declare judgments about objects without realizing similarity. Let us assume we have two objects x and y. Formally we can define similarity between two objects as [5]:

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

There is a diversity in views at symmetry in similarity. Whereas according to [5] 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 [30]. 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. The similarity computed by our method is symmetric.

In our approach the total similarity of two instances is aggregated as the mean value of the similarities computed between elements connected to particular properties. However, other aggregation functions can be employed, such as weighted mean value. Let us compare two instances *InstA* and *InstB*. Let these instances have properties $A = \{property_1, \ldots, property_n\}$ and $B = \{property_1, \ldots, property_m\}$, respectively. 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:

$$\mathrm{sim}\,(InstA, InstB) = \frac{\sum_{i=0}^{|A \cap B|} PropertySM_i\,(elementA, elementB)}{|A \cup B|}, \tag{5}$$

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 (5) is its special case). The *General similarity measure* fulfils 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, which is computed 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, 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, which is computed as described in Section 3.2.

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:

$$\frac{LabelBSM(X,Y) + PropertyBSM(X,Y) + TaxonomyDistanceSM(X,Y)}{N}, \quad (6)$$

where *LabelBSM*, *PropertyBSM* and *TaxonomyDistanceSM* 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 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 1.

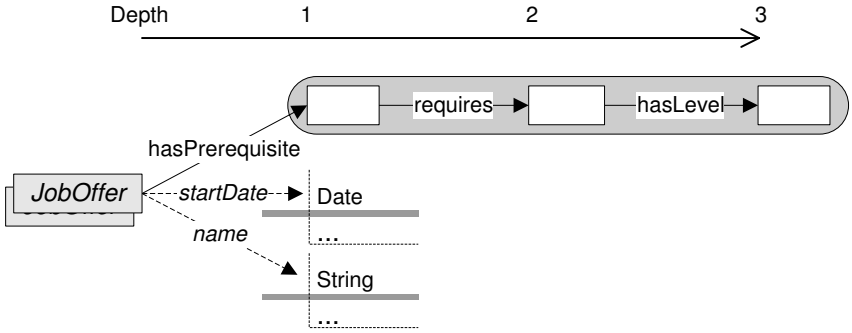

Fig. 4. Example of different depths of properties of a job offer instance

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 is closer to the root instance. The farther the 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:

$$\mathrm{sim}_{property}\,(elementA, elementB) = \frac{\sum_{i=0}^{maxLevel} \mathrm{sim}_{object}(X_i, Y_i)}{maxLevel}, \tag{7}$$

where *elementA* and *elementB* are elements connected to the examined *property*, sim_{object} is computed as defined in Equation (6) and *maxLevel* is the maximal nesting depth for the *property*. Figure 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>
```

Fig. 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:

$$\mathrm{weight}_x = \frac{1}{x}. \tag{8}$$

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

$$\mathrm{sim}_{property}\,(elementA, elementB) = \frac{\sum_{i=0}^{maxLevel} \mathrm{weight}_i \times \mathrm{sim}_{object}\,(X_i, Y_i)}{maxLevel}. \tag{9}$$

Employing weights in the similarity estimation for the object property from Figure 5 is shown in Figure 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 weights are also useful in the same way for the datatype properties. Since depth of datatype

```
<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>
```

Fig. 6. Similarity influenced by weights with regard to nesting depth

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.

5 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:

$$\mathrm{sim}\,(\mathit{InstA}, \mathit{InstB}) = \frac{\sum_{i=0}^{|A\cap B|} \mathrm{weight}_i \times \mathit{PropertySM}_i\,(\mathit{elementA}, \mathit{elementB})}{\sum \mathrm{weight}}, \qquad (10)$$

where the semantics of variables is the same as in Equation (5). 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;
- 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 with a value the user likes it will likely influence his or her rating towards 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 EXPERIMENTAL 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 then 0 is aggregated. In other 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 4. Figure 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.

The thresholds were specified experimentally for the job offer domain. We computed similarity for 55 000 properties employing the described method. Properties

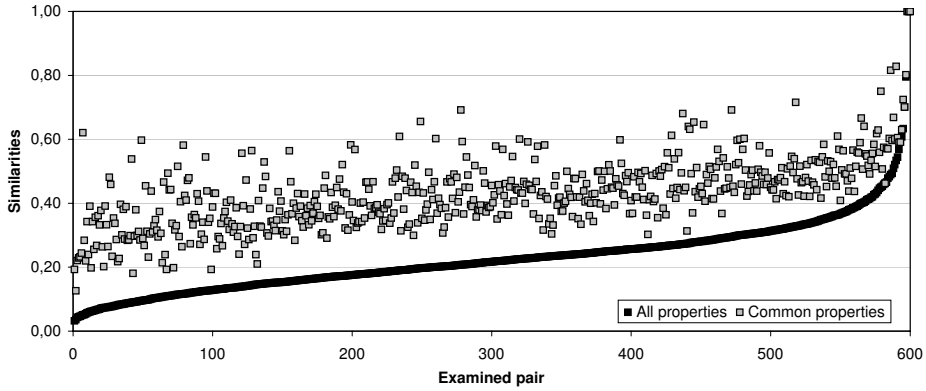

Fig. 7. Similarity computed by ConCom considering all/common properties

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 20/80 rule). We split the most influencing 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 updating 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 experimentally the positive threshold to 0.65 and the negative threshold to 0.25 but such thresholds do not produce useful properties to 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 8.

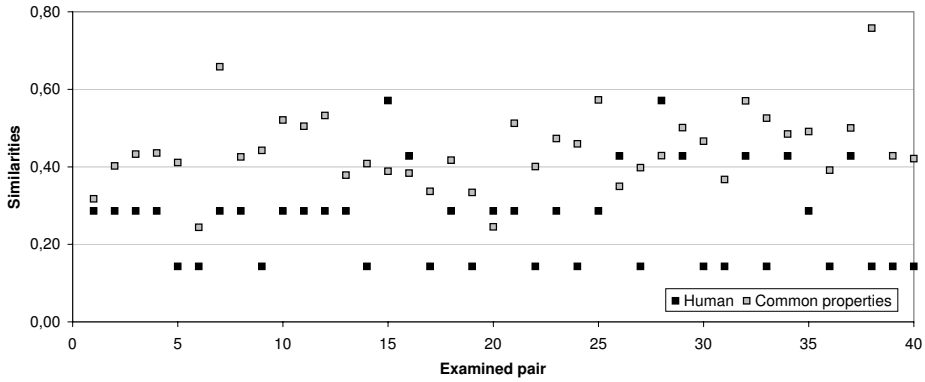

Fig. 8. Similarity estimated by a human and by ConCom for common properties

The cause of evaluation differences is likely 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, 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 and specially 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 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.

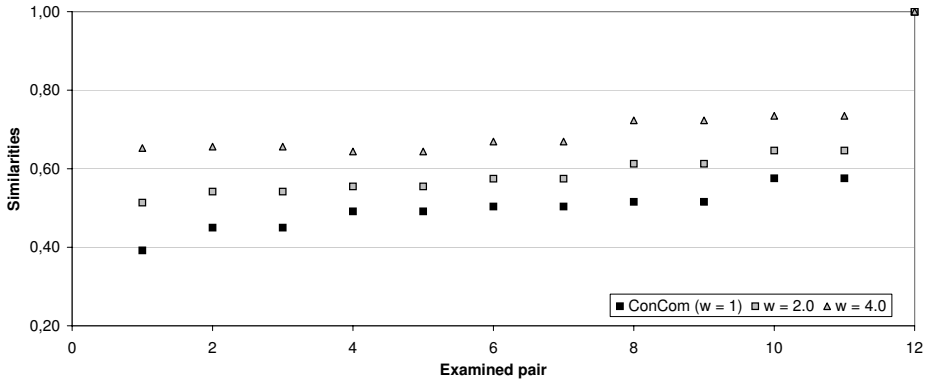

Fig. 9. Weighted similarity computed with regard to a user model

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.

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[4] that was developed as a part of the NAZOU project. The user model was acquired by the LogAnalyzer tool [3] 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 10 depicts the change in the similarity estimation caused by employing the user model (200 pairs depicted for illustration).

[4] Ontology-based UM, `http://nazou.fiit.stuba.sk/home/files/nazou_um.pdf`

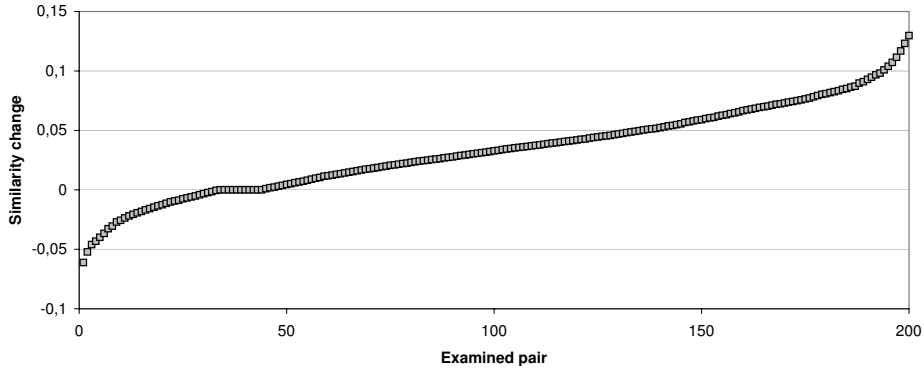

Fig. 10. Change in similarity estimation caused by the employed user model

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).

7 RELATED WORK

Mainly for the purpose of ontology management, several approaches to comparison of ontology concepts or their instances were developed. There are several overviews aimed mostly at ontology matching [12, 16, 22] and related areas (e.g., extending existing approaches with detecting mapping bugs [31]). In this section, we focus mainly on approaches that primarily consider ontology instances and deal with their similarity.

A method computing semantic similarity among instances within an ontology, which considers ontology and context layers, is described in [2]. 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 [18]. 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. Also properties assigned to the concept and their types (object or datatype) should be considered to achieve more accurate similarity enumeration since additional 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 high diversity in the concept names.

PROMPT is an algorithm for ontology merging and alignment [23] 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 [10]. 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 similarity 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 [13]. 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 afterwards converted to a degree of similarity taking into account the biggest 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. 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 [4]. 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 [17].

8 CONCLUSIONS

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 the 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 realizes 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 updating of characteristics in the user model or for acquisition of new ones.

We also performed experiments in the scientific publications domain to investigate domain independence of our method, where the computed similarity can be useful, e.g. for clustering algorithms [11], semantic annotation tools [19], context

search [21] or repository maintenance tools [8] as well as for the recommendation of similar content in recommender systems [15]. Though, the aim here was to improve semantic search using personalized navigation within ontology instances that represent metadata of large information spaces [29].

Acknowledgement

This work was partially supported by the Slovak Research and Development Agency under the contract No. APVT-20-007104, the State programme of research and development "Establishing of Information Society" under the contract No. 1025/04 and the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09.

REFERENCES

[1] Andrejko, A.—Barla, M.—Tvarožek, M.: Comparing Ontological Concepts to Evaluate Similarity. In P. Návrat, P. Bartoš, M. Bieliková, L. Hluchý, P. Vojtáš (Eds.): Tools for Acquisition, Organization and Presenting of Information and Knowledge: Research Project Workshop, 2006, pp. 71–78.

[2] Albertoni, R.—De Martino, M.: Semantic Similarity of Ontology Instances Tailored on the Application Context. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Vol. 4275, Lecture Notes in Computer Science, 2006, pp. 1020–1038.

[3] Barla, M.—Bieliková, M.: Estimation of User Characteristics using Rule-based Analysis of User Logs. In: Data Mining for User Modeling: Proceedings of Workshop held at UM2007, 2007, pp. 5–14.

[4] Bernstein, A.—Kaufmann, E.—Bürki, C.—Klein, M.: How Similar Is It? Towards Personalized Similarity Measures in Ontologies. In O. K. Ferstl, E. J. Sinz, S. Eckert, T. Isselhorst (Eds.): 7th International Conference Wirtschaftsinformatik (WI-2005), Physica-Verlag, Bamberg, Germany, 2005, pp. 1347–1366.

[5] Bisson, G.: Why and How to Define a Similarity Measure for Object Based Representation Systems. In: Towards Very Large Knowledge Bases, IOS Press, Amsterdam, 1995, pp. 236–246.

[6] Brusilovsky, P.—Corbett, A.—de Rosis, F.: User Modeling 2003: Preface. In P. Brusilovsky et al. (Eds.): 9th International Conference on User Modelling, Vol. 2702, Johnstown, USA, Lecture Notes in Computer Science, 2003.

[7] Chen, H.—Ma, J.—Wang, Y.—Wu, Z.: A Survey on Semantic E-Science Applications. Computing and Informatics, Vol. 27, 2008, No. 1, pp. 5–20.

[8] Ciglan, M.—Babík, M.—Laclavík, M.—Budinská, I.—Hluchý, L.: Corporate Memory: A Framework for Supporting Tools for Acquisition, Organization and Maintenance of Information and Knowledge. In: Proceedings of 9th International Conference ISIM '06 "Information Systems Implementation and Modelling", Brno, April, MARQ Ostrava, 2006, pp. 185–192.

[9] Ding, L.—Kolari, P.—Ding, Z.—Avancha, S.: Using Ontologies in the Semantic Web: A Survey. In R. Sharman, R. Kishore, R. Ramesh (Eds.): Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems, Vol. 14, Springer, 2007, pp. 79–113.

[10] Formica, A.—Missikoff, M.: Concept Similarity in SymOntos: An Enterprise Ontology Management Tool. Computer Journal, Vol. 45, 2002, No. 6, pp. 583–594.

[11] Frivolt, G.—Pok, O.: Comparison of Graph Clustering Approaches. In M. Bieliková (Ed.): Proceedings in IIT-SRC 2006, Slovak University of Technology, 2006, pp. 168–175.

[12] Giunchiglia, F.—Yatskevich, M.—Shvaiko, P.: Semantic Matching: Algorithms and Implementation. Journal on Data Semantics IX, Vol. 4601, Lecture Notes in Computer Science, 2007, pp. 1–38.

[13] Gurský, P.—Pázman, R.—Vojtáš, P.: Ontea: On Supporting Wide Range of Attribute Types for Top-k Search. Computing and Informatics, Vol. 28, 2009, No. 4, pp. 483–513.

[14] Heeffer, A.: Negative Numbers as an Epistemic Difficult Concept: Some Lessons From History. In: C. Tzanakis (Ed.): Proceedings of the History and Pedagogy of Mathematics Conference, July 2008, Mexico, 2008 (to be published).

[15] Horváth, T.: A Model of User Preference Learning for Content-Based Recommender Systems. Computing and Informatics, Vol. 28, 2009, No. 4, pp. 453–481.

[16] Kalfoglou, Y.—Schorlemmer, M.: Ontology Mapping: The State of the Art. The Knowledge Engineering Review, Vol. 18, 2003, No. 1, pp. 1–31.

[17] Kay, J.: Stereotypes, Student Models and Scrutability. In: ITS'00: Proceedings of the 5$^{\text{th}}$ International Conference on Intelligent Tutoring Systems. Vol. 1839, Lecture Notes in Computer Science, 2000, pp. 19–30.

[18] Kirsten, T.—Thor, A.—Rahm, E.: Instance-Based Matching of Large Life Science Ontologies. Data Integration in the Life Sciences, Vol. 4544, Lecture Notes in Computer Science, 2007, pp. 172–187.

[19] Laclavík, M.—Šeleng, M.—Ciglan, M.—Hluchý, L.: Ontea: Platform for Pattern Based Automated Semantic Annotation. Computing and Informatics, Vol. 28, 2009, No. 4, pp. 555–579.

[20] Návrat, P.—Bieliková, M.—Rozinajová, V.: Acquiring, Organising and Presenting Information and Knowledge from the Web. In: CompSysTech'06, Bulgarian Chapter of ACM, 2006.

[21] Návrat, P.—Taraba, T.: Context Search. In Y. Li et al. (Eds.): 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (Workshops), Silicon Valley, USA, Los Alamitos USA: IEEE Computer Society, 2007, pp. 99–102.

[22] Noy, N. F.: Semantic Integration: A Survey of Ontology-based Approaches. ACM SIGMOD Record, Vol. 33, No. 4, 2004, pp. 65–70.

[23] Noy, N. F.—Musen, M. A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, MIT Press/AAAI Press, 2000, pp. 450–455.

[24] Pázman, R.: Values Normalization with Logic Programming. In P. Návrat, P. Bartoš, M. Bieliková, L. Hluchý, P. Vojtáš (Eds.): Tools for acquisition, organization and presenting of information and knowledge: Research project workshop, 2007, pp. 134–141.

[25] Resnik, P.: Semantic Similarity in a Taxonomy: An Information-Based Measure and Its Application to Problems of Ambiguity in Natural Language. Journal of Artificial Intelligence Research, Vol. 11, 1999, pp. 95–130.

[26] Semančík, R.: Basic Properties of the Persona Model. Computing and Informatics, Vol. 26, 2007, No. 2, pp. 105–121.

[27] Studer, R.—Benjamins, V. R.—Fensel, D.: Knowledge Engineering: Principles and Methods. Data Knowledge Engineering, Vol. 25, 1998, No. 1–2, pp. 161–197.

[28] Tury, M.—Bieliková, M.: An Approach to Detection Ontology Changes. In: ICWE '06: Workshop proceedings of the sixth international conference on Web engineering, Palo Alto, California, ACM Press, 2006.

[29] Tvarožek, M. et al.: Improving Semantic Search via Integrated Personalized. Faceted and Visual Graph Navigation. In: SOFSEM 2008, Vol. 4910, Lecture Notes in Computer Science, 2008, pp. 778–789.

[30] Tversky, A.: Preference, Belief, and Similarity: Selected Writings. MIT Press, 2003.

[31] Wang, P.—Xu, B.: Debugging Ontology Mappings: A Static Approach. Computing and Informatics, Vol. 27, 2008, No. 1, pp. 21–36.

Anton Andrejko received his Master degree in 2004 from Technical University in Košice and his Ph. D. degree at Slovak University of Technology, Faculty of Informatics and Information Technologies in 2009. He works in the area of user modeling and personalization.

Mária Bieliková received her Master degree (with summa cum laude) in 1989 and her Ph. D. degree in 1995 both from Slovak University of Technology in Bratislava. Since 2005, she has been a Full Professor, presently at Institute of Informatics and Software Engineering, Slovak University of Technology. Her research interests include software knowledge engineering and web information systems, especially adaptive web-based systems including user modeling.