Web Search with Variable User Model

P. Gurský, T. Horváth, J. Jirásek, S. Krajči, R. Novotný, V. Vaneková, P. Vojtáš

P.J. Šafárik University, Charles University, Czech Academy of Science
{peter.gursky, tomas.horvath, jozef.jirasek, stanislav.krajci, robert.novotny,
veronika.vanekova}@upjs.sk, peter.vojtas@mff.cuni.cz

Abstract. We propose a middleware system for web search adaptable to user preference querying as well as user independent fulltext search. We cover also induction of user preferences and effective query answering. A prototype of a new annotation tool is described. The system employs a formal model of user preferences based on fuzzy logic. Experimental implementation of this system integrates several independent software tools.

1. Introduction and Motivation

To make web accessible for automatic processing, we need systems which understand the whole process from wrapping and annotating page, up to the user querying. We present a model of a middleware system permitting users to search objects of one domain but from heterogeneous sources. This system consists of several integrated software tools. It supports different approaches to solve the main task of such a system – return to user the best objects relative to his/her preferences.

The schema of parts of the system is depicted in Figure 1. We exclude a web source download part which is not integrated in our system so far (in experiments we use either manually downloaded pages or simply wrapped pages). Before we start searching for the best objects, we need to unify the information from heterogeneous sources to one common format. We use a domain ontology [14], i.e. a database of objects and attribute values in OWL format. Common HTML pages and also data structured sources can be converted to instances of domain ontology by semantic annotation (OMIN tool described in chapter 2.1).

If the domain ontology is filled with necessary data, the system can answer user queries. We combine two different approaches where query answering can be user dependent or user independent.

The user independent part is based on tf-idf vector model of fulltext search with indexes stored in a database (JDBSearch tool). This tool is further described in chapter 2.2. It works with plain text, not the domain ontology. The same key word query will lead to the same answer for any user. This part of the system can be also used to find objects similar to some concrete suitable object.

In the user dependent approach (detailed in chapter 2.3), the same query can produce different answers for different users. Our user dependent model (chapter 2.3.3) is based on modeling preferences by fuzzy sets and fuzzy logic, arising from [17]. We propose a model which integrates user preference components on all layers of the system (both inductive and deductive/querying tasks).

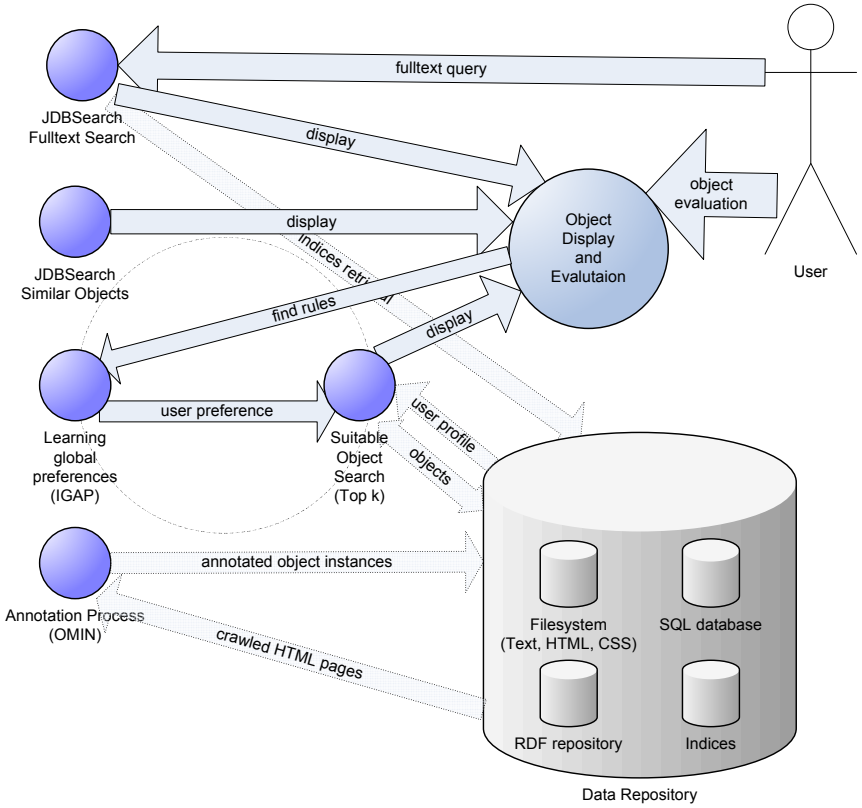

Figure 1. Data flow schema

Let us consider the following example: Imagine a user looking for a hotel which has *good price*, *good distance from an airport* and has *good equipment in rooms*. Concepts like *good price* are naturally vague and imprecise, they can be different for every user. For instance one user prefers cheap hotels (student), second prefers expensive hotels (manager) and the other one prefers middle price (professor). The natural meaning of this ordering is that the user determines the relations "better" or "worse" between two values of given property. For each such property, user has a notion about the ordering of objects based on real value of property from an attribute domain. We call these orderings of particular attribute domains the user local preferences (see chapter 2.3.1).

Local preferences are not enough to say for any two objects which one is better than the other. We often find two hotels, one of which has better price and another has better equipment. We need to combine the values of objects obtained by user local preferences in some way. This feature is modeled e.g. in multicriterial decision making. The combination of user local preferences that leads to overall values of

objects for a concrete user, is called global preference (see chapter 2.3.2). It can be represented as a set of IF-THEN rules or weighted average.

Having both local and global preferences, we face the deductive task of effective retrieval of top-*k* answers (the Top-*k* tool is described in chapter 2.3.5). Searching of suitable objects for user, based on local and global preferences, is the typical user dependent query answering.

Preferences can be obtained in several ways. Our intelligent user profile incorporates various operating modes. First, when a new user wants to find suitable objects, we give him/her a set of objects, representative samples, and their properties (i.e., in our example some hotels with their attributes). Object evaluation part evokes learning user fuzzy aggregation function (IGAP tool). Using this aggregation and query optimization (Top-*k* tool) we retrieve the answer. This cycle of object evaluation, aggregation learning and top-*k* objects finding can be repeated several times. The display and evaluation part is able to couple text search with evaluate-IGAP-Top-k part.

2. Models, Methods and Tools

In this chapter we describe models, methods and tools which drive the information from the web to the user.

2.1. OMIN – An Annotation Tool

Semantic annotation represents a specific sort of metadata extracted from documents or web sources. In the later, we distinguish between web sources which are dominantly text oriented, sources with richer structure (e.g. tables) and sources with dominantly visual information (Flash-based content).

There are many approaches to semantic annotation. Generally it is possible to reuse existing systems, such as Lixto ([1, 13]). However, we have done some independent preliminary explorations in this topic. The first method is based on the proposition that many web pages representing detailed product information (for example detailed information about particular hotel) are generated from the web template engines. Therefore the HTML sources of two pages, which have been generated from the same template, are very similar. We have observed that the differences between sources usually fall into following categories: either they are differences in the element structure, i.e. one page contains a HTML subtree not appearing in the other page. An example of this is a hotel with more information provided (additional properties or extended data). The other category encompasses the differences in HTML attributes. However, these attributes usually contain irrelevant data, typically styling information, element identifiers, or hyperlinks. The last category contains the differences in the HTML node values, which can be considered as the possible candidates for the extraction. Providing the sufficient number of different pages which were generated from one template can minimize number of mismatches (e. g. differences which do not correspond to the attribute values) and maximize the number of matches (since potential candidates will be discovered only in the differences).

The Figure 2 shows an example of differences. The areas in red rectangles represent the differences in contents of corresponding HTML elements (since we have used a classical *diff* algorithm, red-highlighting denotes the exact positions of differences).

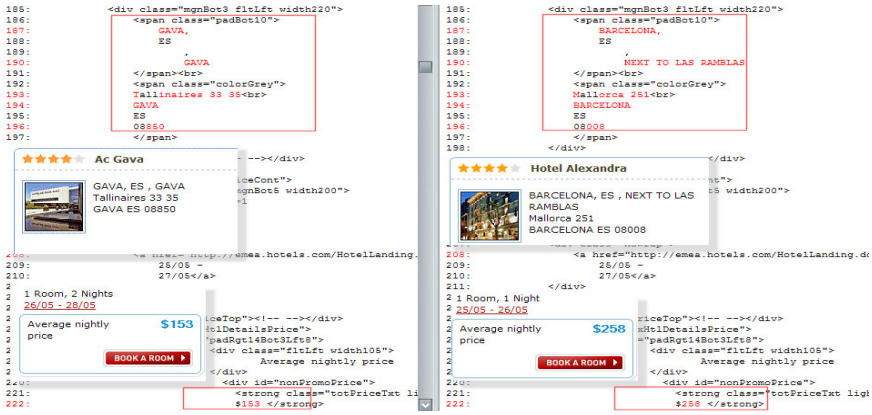

Figure2. Differences and corresponding HTML areas

However, after the discovery of potential candidates it is necessary to map these values to the corresponding attributes. In this case, one solution could be mapping of attributes to regular expressions which describe the possible values of a given attribute. Another proposal is to find nearest HTML elements and try to utilize their visual presentational form – e. g. CSS or HTML formatting instructions, since it is more probable that the attribute labels will be presented in the emphasized form – be it a bolder text or different forward or background color. Such visually important elements can stand out as potential candidates for attribute values. This technique can be used also to improve the efficiency of potential candidate discovery (even in cases when there are no explicit attribute labels). It could be seen that both price and hotel name is visually emphasized (this is achieved by CSS rules) and therefore potentially more important than the other differences occurring in the document.

Human only readable sources, like flash animations, represent information that cannot be retrieved directly from the web page source. We plan to solve this problem by OCR processing.

2.2. Text-based Approach

In user independent querying we use well known vector model [2,6]. The adopted vector model requires defining a number of fundamental notions. In this model, a document is defined as a vector in m-dimensional vector space (m denotes the number of unique terms in the text of all documents). This vector is comprised of the weights of the document terms.

The weight of each of these elements can be calculated by using the inversion document frequency formula idf_j and tf_{ij} the number of occurrences of the term t_j in the i-th document as

$$w_{ij} = tf_{ij} \,.\, idf_j \tag{1}$$

To start indexing source documents, it is usually needed to remove all meta-information hidden to user (HTML tags). Having plain text sources we can create tf-idf index. Our implementation of the index is based on the relational database as in [11, 12]. This method has multiple advantages: well-known and well-defined structure (based on the database tables) and the relatively high-speed of current relational databases.

The matching of the question on the document collection can be ordered by cosine measure. We can map each of the query types on a particular SQL query. Executing this query against the database can give us the sorted resulting list of documents, including their relevancy.

We can easily use this index also to retrieve similar documents when we put on the input not only few words but whole document. This feature underbid an easy way (by one click) to find more good objects based on a single one object.

2.3. User-dependent Search

In the user-dependent search different users can obtain different answers for the same question. Searching is based on user preferences that can be created during current and previous sessions as well.

2.3.1. Detecting Local Preferences

To learn local preferences, we give to user a representative sample of hotels (see table 1) with several attributes. The attributes in our example are: distance from the city center, price of the accommodation and equipment of rooms (without equipments, TV, internet or both). The user classifies every hotel into one of the three classes (categories): poor, good and excellent according to the relevance of the hotel to him. In real world we use 7 classes of Likert scale.

Table 1. Representative sample of hotels evaluated by the user

Hotel	distance	price	equipment	evaluation
Apple	100 m	99 $	none	poor
Danube	1300 m	120 $	TV	good
Cherry	500 m	99 $	internet	good
Iris	1100 m	35 $	internet, TV	excellent
Lemon	500 m	149 $	none	poor
Linden	1200 m	60 $	internet, TV	excellent
Oak	500 m	149 $	internet, TV	good
Pear	500 m	99 $	TV	good
Poplar	100 m	99 $	internet, TV	good
Rhine	500 m	99 $	none	poor
Rose	500 m	99 $	internet, TV	excellent
Spruce	300 m	40 $	internet	good
Themse	100 m	149 $	internet, TV	poor
Tulip	800 m	45 $	internet, TV	Excellent

The local preferences can be detected in several ways. In this chapter we discuss statistical models. Note that "equipment" is a text attribute so it needs no ordering detection. We focus on distance and price.

The task is to find attribute domain ordering which makes user object evaluation monotone according to this ordering. The first model of ordering detection (local preferences) is QUIN [3] – a system of qualitative data mining which discovers trends in data.

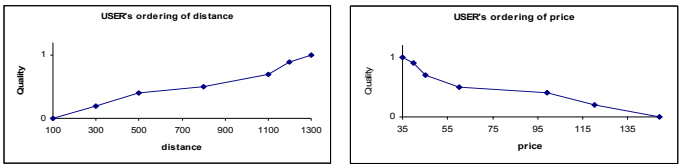

Figure 3.Local preference ordering

The final result is obtained by combination of several statistical methods (see Figure 3) and shows that the price has negative influence and the distance has positive influence on the user evaluation/ranking. In other words, the user classification decreases with increasing price of a hotel and it increases with increasing the distance of a hotel from an airport.

2.3.2. Learning Global Preferences

We learn user's global preferences by the method of Ordinal Classification with Monotonicity Constraints described in [9,10]. The IGAP tool is based on Inductive Logic Programming ILP system ALEPH [15], the well-known relational data mining model [4].

In the learning process we compute the user's global preferences with help of his/her local preferences. We use ILP because our local preferences (orderings) can be represented well in this framework by definite clauses. For illustration, consider that we have discretized values of distance to three classes: near, middle and far. The different orderings of these classes for different users can be:

$near \leq middle \leq far$ (non-decreasing)
$near \leq far \leq middle$
$far \leq middle \leq near$ (non-increasing)
$far \leq near \leq middle$
$middle \leq near \leq far$ (the *middle* values are "better" than *near* or *far*)
$middle \leq far \leq near$

A usual aggregation function can be easily simulated by monotone classification rules in the sense of many valued logic. The results of our approach (the user's global preferences) computed from the data in our illustrative example are the following classification rules about the relevance of hotels to user preferences:

- evaluation = excellent IF distance ≥ 500 AND price ≤ 99 AND services = {TV, internet}
- evaluation = good IF (distance ≥ 500 AND services = {TV})

- evaluation = good IF (price ≤ 99 AND services = {internet})

From our results we can obtain also additional information about crisp attributes (equipment) The meaning of any classification rule is as follows: If the attributes of an object *x* fulfill the expressions on the right side of the rule (body) then the overall value of *x* is at least the same as on the left side of the rule (head). We can, of course, assign the explicit values to vague concepts like excellent or good. During the simulation of aggregation function computing, we can simply test the validity of requirements of the rules from the strongest rule to weaker ones. When we find the rule that holds, we can say that the overall value of the object is the value on the left side of the rule. The rules are sorted, therefore we always rank the object with the highest possible value.

We can see that the ordered meaning of the classification is preserved in our results: hotels classified in the grade "excellent" by the user also fulfills requirements for "good" and "poor" hotels, and "good" hotels fulfills requirements for "poor" ones (e.g. the hotel with 800 m distance from airport and 45$ price equipped with internet and TV is "at least as appropriate as" the hotel 300 m far from the airport and 40$ price equipped just with internet) according to the local preferences (farther and cheaper is better).

2.3.3. Model of User Preference Ontology

As we have seen in the previous chapters, it is possible to learn user local preferences from the evaluation of a sample set of objects. Objects (hotels) are resources from domain ontology and therefore it is convenient to store user preferences in a separate ontology.

User preference ontology is created from domain ontology. For example if we are interested in hotel prices with descending ordering, we name this instance of user local preference "cheap" because the ordering returns cheap hotels first. For the same property and ascending ordering, we name the instance "expensive".

Many users can share the same local preference, for instance users that prefer cheap hotels or hotels with internet. Different combinations of local preferences yield some typical user types. An example of such user type can be user preferring cheap hotels with internet, far from the airport.

Interesting question is how we can determine the type of some new user. We can let the user specify his/her local preferences to every attribute via some user-friendly interface. If the user does not specify any preferences, we can compare his/her available information (like personal data) with other users, find similar users and choose their prevailing preferences as a default. Another possibility is to provide a representative set of hotels to the user, let him/her rank these hotels and learn preferences from ranking, as described in previous two chapters.

The user preference ontology contains finite number of orderings and therefore also finite number of user types. But even users from the same type can have different global preference. By applying the global preference (monotone aggregation function) on these values we compute the relevance of every hotel to user as in [5]. This function can be unique for every user, but very often similar for similar user types.

2.3.4. Fuzzy RDF Based on Fuzzy Description Logic

In this chapter we analyze the model of fuzzy RDF/OWL based on fuzzy description logic. The terms like *cheap*, *expensive* or *near* represent fuzzy sets as in [17]. Our model of fuzzy RDF includes such fuzzy sets stored as RDF triples. We get RDF triples <*hotel_id*, *ordering_type*, *relevance*> where *hotel_id* comes from the domain ontology and *ordering_type* from user preference ontology. Relevance is RDF literal with a number from [0, 1]. We will use fuzzy RDF as middle level in the process of retrieving best objects from domain ontology.

Let $cheap_U$ be a local preference of user U for hotel price defined by non-increasing membership function:

$$cheap_U(x) = \frac{(\max - x \cdot currency)}{(\max - \min)}$$

The following RDF triples are created for hotels Apple and Danube from Table 1 and ordering type *cheap*. We get the relevance via the function from *cheap* as follows:

$$cheap_U(99\$) = \frac{149 - 1 \cdot 99}{149 - 35} = \frac{50}{114} \cong 0.44$$

$$cheap_U(120\$) = \frac{149 - 1 \cdot 120}{149 - 35} = \frac{29}{114} \cong 0.25$$

The value of "currency" is the exchange rate which is equal to 1 for given prices in US dollars. The values of max and min are maximal and minimal price values from Table 1, i.e. 149 and 35. We use the evaluations to create following RDF triples:

<Apple, $cheap_U$, 0.44>

<Danube, $cheap_U$, 0.25>

One important feature of our model is that we can prepare fuzzy RDF independently from user global preference. This is because we can adapt to user by adjusting his aggregation function @. This makes the processing of data more effective, because we do not need to order data for every user query. We already have the data pre-ordered and we just combine the relevancies from fuzzy RDF into one result.

Example. We have developed also a formal model of fuzzy description logic **f-$\mathcal{EL}$** as the motivation for creating a model for fuzzy RDF. We describe it by an example. It consists of:

- Crisp properties N_R = {*price, distance_from_airport*} where *price* and *distance_from_airport* are RDF predicates from domain ontology.
- Fuzzy concepts N_C = {$cheap_U$, $close_U$} where $cheap_U$ and $close_U$ are fuzzy functions explicitly figured by user preference ontology.
- Instances N_I = {*Apple, Danube*, 99, 120,…}

In Herbrand-like interpretation $\mathcal{H}$ we have:

$Apple^{\mathcal{H}}$ = Apple, $Danube^{\mathcal{H}}$ = Danube, $99^{\mathcal{H}}$ = 99, $120^{\mathcal{H}}$ = 120

$cheap_U{}^{\mathcal{H}}(99) = 0.44$, $cheap_U{}^{\mathcal{H}}(120) = 0.25$

$price^{\mathcal{H}}$ = {(Apple, 99), (Danube, 120)}

For simplification we overload our concept $cheap_U$ also for hotels (usually we must create new concepts in this case e.g. $cheap\_hotel_U$):

$cheap_U{}^{\mathcal{H}}(x) = (\exists\ price.cheap)^{\mathcal{H}}(x)$ then

$$cheap_U{}^{\mathcal{H}}(\text{Apple}) = \sup\{cheap_U{}^{\mathcal{H}}(\text{y}): (\text{Apple,y}) \in price^{\mathcal{H}}\}=0.44$$
$$cheap_U{}^{\mathcal{H}}(\text{Danube})=\sup\{cheap_U{}^{\mathcal{H}}(\text{y}): (\text{Danube,y}) \in price^{\mathcal{H}}\}=0.25$$

The supremum affects the case when we have more different prices for one hotel. Then we take the biggest result.

Let aggregation function for an user U be $@(cheap_U, close_U) = (3 \times close_U + 2 \times cheap_U)/5$. If the value of $close_U{}^{\mathcal{H}}(\text{Apple})$ is 0.33, then the overall relevance of hotel Apple will be:

$$(@(cheap_U, close_U))^{\mathcal{H}}(\text{Apple}) = @^{\bullet}(cheap_U{}^{\mathcal{H}}(\text{Apple}), close_U{}^{\mathcal{H}}(\text{Apple})) = (3 \times close_U{}^{\mathcal{H}}(\text{Apple}) + 2 \times cheap_U{}^{\mathcal{H}}(\text{Apple}))/5 = (3 \times 0.33 + 2 \times 0.44)/5 = 0.37.$$

Consider the type of user preferring cheap hotels. There can be many users that prefer cheap hotels. We want to share the same instance $cheap_U$ for them. However, their notion of *cheapness* can be slightly different. For example one user can say that cheap hotels have price lower than $30, while for some other user cheap hotel ends at $50. We can easily adjust the overall evaluation of objects to reflect these individual requirements (for details see [8]).

2.3.5. Relevant Object Search

Now we have orderings of properties from learning of local preferences, rules from learning of global preferences and prepared ordered data stored in fuzzy RDF. Our last task is to find top k objects to user. We use the extension of middleware search of Ronald Fagin [5]. The main idea is to browse only necessary data until the system is sure that it has top-k objects already. Thus we do not need to calculate with whole data from domain ontology. Retrieving only k best objects can save time and is usually sufficient for user. Saving of time grows with the number of objects stored in domain otology and also with the number of properties we consider.

The model in [5] considers two kinds of accesses to the ordered properties stored in possibly distributed lists: sorted and random access. The sorted access gets the next best object from the list after each access, so we can retrieve data ordered from the best to the worst. The random access asks for the value of a particular property it includes for a concrete object. Each random access requires searching of the value of concrete object. In the case of sorted access the results are prepared immediately (values can be preordered, in fuzzy RDF in our case, to the several orderings before user comes) and, moreover, data can be sent in blocks. Because of this feature in our system we prefer mainly the algorithms, which use only sorted access.

In our application we use 3P-NRA (3 Phased No Random Access) algorithm, which is an improvement of NRA [5] algorithm (see also [7]).

As we found in our experiments (with our top-k tool), using the techniques of top-k search can significantly save number of needed accesses and accordingly the time needed especially in huge sets of objects.

3. Conclusions

In this paper we have described a model of system enabling users to search objects from the same domain and heterogeneous sources. Data are collected from various sources are processed to vector index and to a domain ontology. The system

implements both user independent search and personalized search for users with different preferences. Our theoretical model integrates all parts of the system from collected data in classical RDF form to the user query answering. We do not have a model for web resource downloading and ontology annotation part.

Presented model was experimentally implemented[1] and integration was tested using Cocoon and Spring Framework [16].

Our approach is used also in the Slovak project *NAZOU – Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources* to find relevant job offers for the user according to his/her preferences. The domain of this project is different from the one used in this paper, thus we have another domain for testing purposes. We can constitute that our model is domain-independent.

Acknowledgements Partially supported by Czech projects MSM 0021620838, 1ET100300419 and 1ET100300517 and Slovak projects VEGA 1/3129/06 and NAZOU.

References

[1] Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto VLDB 2001
[2] Baeza-Yates, R., Ribeiro-Neto, B. A.: Modern Information Retrieval. ACM Press / Addison-Wesley, 1999.
[3] Bratko, I., Šuc, D.: Learning qualitative models. AI Magazine 24 (2003) 107-119
[4] Džeroski, S., Lavrač, N.: An introduction to inductive logic programming. Relational data mining, S. Džeroski, N. Lavrač eds., Springer 2001, 48-73.
[5] Fagin, R.: Combining fuzzy information from multiple systems, J. Comput. System Sci. (1999) 58:83-99
[6] Grossman, D. A., Frieder, O.: Information Retrieval: Algorithms and Heuristics, Kluwer Acamedic Publishers, 2000.
[7] Gurský, P.: Towards better semantics in the multifeature querying. Proc. Dateso 2006
[8] Gurský P., Horváth T., Jirásek J., Krajči S., Novotný R., Vaneková V., Vojtáš P.: Knowledge Processing for Web Search – An Integrated Model, Proceedings of IDC 2007.
[9] Horváth, T., Krajči, S., Lencses, R., Vojtáš, P.: An ILP model for a graded classification problem. J. KYBERNETIKA 40 (2004), No. 3, (2004), p:317–332.
[10] Horváth, T., Vojtáš, P.: Ordinal Classification with Monotonicity Constraints. In. Proc. ICDM 2006, Leipzig, Germany: LNAI 4065, Springer, 2006, pp. 217 – 225.
[11] Lencses, R.: Indexing for Information Retrieval System supported with Relational Database, Sofsem'05 Communications, Vojtáš et al. (ed.) Bratislava 2004, 81-90
[12] Lencses, R.: Dopytovanie v systéme zameranom na získavanie informácií s podporou relačnej databázy, Proc. Datakon 04, Masaryk University, Brno, 2004, p. 271-280
[13] Lixto. [online] http://www.lixto.com/downloads/
[14] McGuinness, D. L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation, 2004.
[15] Srinavasan, A.: The Aleph Manual. Technical Report, Comp.Lab., Oxford University
[16] The Spring Framework. http://www.springframework.org/
[17] Vojtáš, P.: Fuzzy logic programming, Fuzzy Sets and Systems, 124(3) (2001) 361-370

[1] An experimental implementation of our system is available on URL http://x64.ics.upjs.sk:8080/nazou-wid/app (with user interface in Slovak).