Computing and Informatics, Vol. 28, 2009, 515–553

USER PREFERENCE WEB SEARCH – EXPERIMENTS WITH A SYSTEM CONNECTING WEB AND USER

Peter Gurský, Tomáš Horváth, Jozef Jirásek, Stanislav Krajči
Róbert Novotný, Jana Pribolová, Veronika Vaneková
Peter Vojtáš

Institute of Computer Science
P. J. Šafárik University
Košice, Slovakia
&
Faculty of Mathematics and Physics
Department of Software Engineering
Charles University
Prague, Czech Republic

e-mail: {<name>.<surname>}@upjs.sk, peter.vojtas@mff.cuni.cz

Revised manuscript received 24 February 2009

Abstract. We present models, methods, implementations and experiments with a system enabling personalized web search for many users with different preferences. The system consists of a web information extraction part, a text search engine, a middleware supporting top-k answers and a user interface for querying and evaluation of search results. We integrate several tools (implementing our models and methods) into one framework connecting user with the web. The model represents user preferences with fuzzy sets and fuzzy logic, here understood as a scoring describing user satisfaction. This model can be acquired with explicit or implicit methods. Model-theoretic semantics is based on fuzzy description logic f-$\mathcal{EL}$. User preference learning is based on our model of fuzzy inductive logic programming. Our system works both for English and Slovak resources. The primary application domain are job offers and job search, however we show extension to mutual investment funds search and a possibility of extension into other application domains. Our top-k search is optimized with own heuristics and repository with special indexes. Our model was experimentally implemented, the integration was tested and is web accessible. We focus on experiments with several users and measure their satisfaction according to correlation coefficients.

Keywords: Web search, information extraction, top-k optimization, fuzzy ILP, user preference learning/mining, text indexing, natural language processing in inflective languages, fuzzy DL, fuzzy RDF, relevant objects, user preferences

Mathematics Subject Classification 2000: 68T37, 68T50, 03B80

1 INTRODUCTION AND MOTIVATION

One of the main goals of the semantic web is to enable easy and automatic access to web resources and services by middleware engines or agents. Enabling access should be achieved by annotations with ontology metadata. Our research leads to a model of a middleware system which will help users in searching for objects from heterogeneous sources but a single domain. Connection of the middleware with web resource annotation is achieved by a Web Information Extraction (WIE) model and methods implemented in our tool OMIN. Annotation is done with respect to a domain ontology. In this paper, we present different approaches to the most important aspect of such a system – retrieving the best objects according to user's preferences. This can be achieved either by key word search (tool JDBSearch, originated in [21]) or user model dependent top-k search (implemented by the Top-k tool). Input for the latter search consists of user preferences that cover evaluation of user preferred domain values of attributes and fuzzy logic programming rules, mined by our fuzzy inductive logic programming tool IGAP.

All of these methods are developed under the umbrella of the NAZOU project [24] where the main application domain is job offer and job search. Our method implementations (from now on called *tools*) are integrated into the portal-like web application (web accessible on `http://x64.ics.upjs.sk:8080/nazou/`, currently in the pilot status). Our tools are also integrated into pilot application of the whole NAZOU project.

Let us consider the following example: Imagine a user looking for a job which has *good salary*, *good location or distance from a place (of living)* and has *good work area in the domain of user expertise* and *acceptable traveling duties*. Each user (or group of users) has his/her own sense of quality (i.e. the preference) for each relevant attribute of an object. For example, one user could prefer only the highest salaries, while another would prefer reasonable minimum and all the higher salaries would be considered equally good. Good location and/or distance to work can differ from user to user. A mother with a small child would like to work close to her home. A young single programmer would like to collect experience abroad. Domain of expertise can also be considered differently. One user does not like to learn new things and is highly specialized in .NET programming, another one would like to learn new things and prefers new technologies – e.g. programming for Web 2.0 applications. Level of traveling involved can be measured in different ways by different users.

The underlying meaning of these orderings is that users determine the relations "better" or "worse" between two values of a given property like salary, location and work area. For each such property, the 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*.

Nevertheless, these user-specific local preferences usually lead to incomparable objects, e.g. one job offer has higher salary than another, but it has disadvantages in job location; another job offer with better location offers lower salary. It is necessary to combine local preferences to one overall score. This would allow to compare the whole objects. The combination of local preferences is called *global preference* and will be modeled by fuzzy aggregation operators.

Having presented the motivation and the basic notions, we can summarize the main contributions of this paper:

- User preference model based on fuzzy description logic (truth value understood as scoring and user preference degree) and ontologies.
- Methods of web information extraction (WIE), fuzzy inductive logic programming (ILP), top-k and preference acquisition.
- Implementations of WIE (OMIN), fuzzy ILP (IGAP), preference search Top-k, preference acquisition and management (UPreA). Integration of all these methods via the UPreA tool.
- Experiments with our system with several users.
- Usability models and methods with arbitrary domain.
- Presentation of pilot web application.

The paper is organized as follows: Section 2 introduces models of user preferences, based on fuzzy description logic and ontology representation. It also presents domain ontology of job offers. Section 3 describes methods for detecting user preferences and searching for relevant objects. Section 4 provides more detailed description of the system, tool implementations and integration. In the last part of the Section 4 there is general view on the overall system. Section 5 describes the system evaluation and experiments. Finally, we discuss our findings in Section 6 and we provide some plans for future research.

2 MODELS

In this chapter, we describe models we use for solution of the problem of web search. These models will be used as a starting point for design and implementation of the tools, which are described in the next section.

2.1 Basic Fuzzy Model of User Preferences

We can represent terms like "high salary", "good work area" or "close", mentioned in the introduction, as fuzzy sets. Compared to classical sets (also denoted as *crisp*

sets), fuzzy sets allow partial membership of their elements. The fuzzy membership function range is $[0, 1]$. The membership value 1 means full membership of an element in the fuzzy set, while lower values indicate partial membership and 0 means no membership. E.g., a fuzzy set membership function *high_salary(x)* will have higher value for better paid jobs. Thus we have an ordering of job offers with respect to the user preference of salary.

In practice, if the domain of attribute is ordered, user preferences usually belong to one of the following variants of trapezoidal membership function (see Figure 1). Such attributes are called *ordinal*. Typical examples are integer or decimal numbers. Many attributes have a finite range of possible values, usually strings, with no apparent ordering (other than lexical). We call them *nominal* attributes and we usually express user preference as a crisp set of suitable values. However, user can still prefer some values more than other values. It is possible to specify user's degree of preference to some values. This method is called fuzzification of crisp attribute

- lower-best function (LB) – lower attribute values are better,
- higher-best function (HB) – higher attribute values are better,
- middle-best function (MID) – middle values are preferred,
- marginal-best function (MAR) – marginal values are better than middle values.

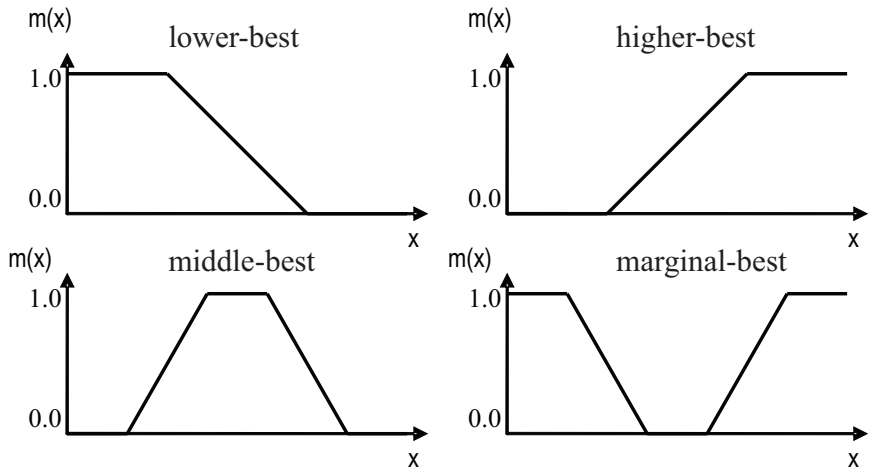


Fig. 1. Basic fuzzy set types

There are many possibilities of aggregating fuzzy local preferences to one overall value. We only require such combination function to be order preserving in all arguments. It can be represented as a composition of fuzzy connectives (conjunctions or disjunctions). Another possibility is a weighted average. In our preference model, we can specify global preferences also by monotone classification rules which can be

learned from objects ranked by the user [18]. These rules reflect global preferences in more natural and comprehensible way. The following example shows a set of classification rules:

- $good\_job(x) \geq 0.8$ IF $high\_salary(x) \geq 0.8$ AND $good\_work\_area(x) \geq 0.3$
- $good\_job(x) \geq 0.4$ IF $high\_salary(x) \geq 0.5$ AND $good\_job\_term(x) \geq 0.5$ AND $close(x) \geq 0.1$.

Every rule consists of a head (e.g. $good\_job(x) \geq 0.8$) and a body, which is a conjunction of clauses (e.g. $high\_salary(x) \geq 0.8$ AND $good\_job\_term(x) \geq 0.3$). If these conditions are fulfilled for some object, then this object has overall preference value at least the same as on the left side of the rule (head). Note that more than one rule can apply for some object. Let us consider an object job_1 with local preference values $good\_salary(job_1) = 0.91$, $good\_work\_area(job_1) = 0.4$, $good\_job\_term(job_1) = 0.7$ and $close(job_1) = 0.5$. It is easy to see that this object satisfies the conditions of both rules stated above. We infer $good\_job(job_1) \geq 0.8$ from the first rule and $good\_job(job_1) \geq 0.4$ from the second rule. The higher value 0.8 is taken as a final result. The inequality $good\_job(job_1) \geq 0.8$ implies also $good\_job(job_1) \geq 0.4$, so both rule heads are fulfilled.

As we can see from the example above, an object can satisfy condition clauses of several rules at the same time. The result value has to be such that it satisfies all corresponding rule head inequalities. The evaluation algorithm would therefore check all rules for this object, select the rules that are fulfilled by the object and finally take the highest result value, i.e. the greatest lower bound.

2.2 Fuzzy Description Logic

Our model is based on fuzzy description logic f-$\mathcal{EL}^{@}$ formally introduced in [31, 32]. This logic removes some features of both classical and fuzzy description logic (like negation, universal restriction and fuzzy roles). On the other hand, it adds an aggregation operator @. It should be also noted that we lose the ability to describe fuzziness in roles. However, our source data are crisp – we do not consider uncertainty in values. User preferences are represented as fuzzy concepts and they are the source of fuzziness in results. Thus, we gain combination of particular user preferences to a global score by his aggregation function denoted as @. An advantage of this description logic is in lower complexity of querying. Expressivity is lower than that of full fuzzy description logics (DL), but still sufficient for our task and embeddability into web languages and tools (see [32]).

Concepts and roles are the basic building blocks of every description logic. Here, roles express properties of job offers. We sometimes express RDF triples $\langle subject, predicate, object \rangle$ in the language of logic as $predicate(subject, object)$.

The alphabet of f-$\mathcal{EL}^{@}$ consists of sets $\mathcal{N}_C$ of concepts names, $\mathcal{N}_R$ role names and $\mathcal{N}_I$ instance names. The roles in f-$\mathcal{EL}$ are crisp and the concepts are fuzzy. Our language of description logic further contains a constructor $\exists$ and a finite set

of aggregation functions symbols $@_U$ for each user and/or for each group of users. Concept descriptions in f-$\mathcal{EL}$ are formed according to the following syntax rules:

$$C \to \top \mid A \mid @(C_1, \ldots, C_n) \mid \exists r.C.$$

In order to give this syntax a meaning, we have to define interpretations of our language. In f-$\mathcal{EL}$ we have interpretations parameterized by a linearly ordered set of truth values with aggregations. For a preference structure (a set of truth values $P = [0, 1]$), a P-interpretation is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \bullet^{\mathcal{I}} \rangle$ with a nonempty domain $\Delta^{\mathcal{I}}$ and fuzzy interpretation of language elements:

- $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to P$, for $A \in \mathcal{N}_C$
- $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, for $r \in \mathcal{N}_R$
- $(\exists r.C)^{\mathcal{I}} = \sup\{C^{\mathcal{I}}(y) : (x, y) \in r^{\mathcal{I}}\} \cup \{0\}$
- $(@(C_1, \ldots, C_n))^{\mathcal{I}}(x) = @^{\bullet}(C_1^{\mathcal{I}}(x), \ldots, C_n^{\mathcal{I}}(x))$.

Suppose that we have $\mathcal{N}_R = \{\mathit{Salary}, \mathit{Region}, \mathit{TravelingDuties}, \ldots\}$ where elements of this set represent the RDF predicates from the domain ontology.

$$\mathcal{N}_C = \{\mathit{high\_salary}_U, \mathit{close}_U\}$$

where $\mathit{high\_salary}_U$ and close_U are fuzzy membership functions explicitly figured by user. Fuzzy function $\mathit{high\_salary}_U$ depends on the specified salary, while close_U depends on the region. As *Region* is a crisp attribute, the user must say directly how "close" he considers some area to be. This simple approach can be improved by using GPS coordinates and determining distances automatically. Such attribute would be called *metric* attribute, then.

$$\mathcal{N}_I = \{\mathit{Java\_Developer}, \mathit{Secretarial\_Support}, 42\,000\,\mathrm{Sk/W}, \mathit{US}, \mathit{FR} \ldots\}$$

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

$$\begin{aligned}
\mathit{Java_Developer}^{\mathcal{H}} &= \text{Java Developer} \\
\mathit{Secretarial_Support}^{\mathcal{H}} &= \text{Secretarial Support} \\
42\,000\mathrm{Sk/W}^{\mathcal{H}} &= 42\,000 \\
US^{\mathcal{H}} &= US \\
FR^{\mathcal{H}} &= FR \\
\mathit{high_salary}_U^{\mathcal{H}}(42\,000) &= 0.53 \\
\mathit{close}_U^{\mathcal{H}}(US) &= 0.9 \\
\mathit{Salary}^{\mathcal{H}} &= \{(\text{Java_Developer}, 42\,000), (\text{Secretarial_Support}, ???)\} \\
\mathit{Region}^{\mathcal{H}} &= \{(\text{Java_Developer}, US), (\text{Secretarial_Support}, FR)\}
\end{aligned}$$

Note that we work with open world assumption as usual in description logics. Some values are not presented on the web, or our WIE tool was not able to recognize

it, so ??? means unknown value of salary for the Secretarial Support job offer. We overload our concept $high\_salary_U$ and $close_U$ also for job offers by using existential quantifier:

$$high\_salary_U^{\mathcal{H}}(x) = (\exists salary.high\_salary)^{\mathcal{H}}(x)$$

and subsequently

$$high\_salary_U^{\mathcal{H}}(\text{Java\_Developer}) =$$
$$\sup\{high\_salary_U^{\mathcal{H}}(y) : (\text{Java\_Developer}, y) \in Salary^{\mathcal{H}}\} \cup \{0\} = 0.53$$
$$high\_salary_U^{\mathcal{H}}(\text{Secretarial\_Support}) =$$
$$\sup\{high\_salary_U^{\mathcal{H}}(y) : (\text{Secretarial\_Support}, y) \in Salary^{\mathcal{H}}\} \cup \{0\} = 0.$$

Similarly, by several other bindings we can get $close_U^{\mathcal{H}}(\text{Java\_Developer}) = 1$.

The supremum affects the case when we have more different salaries for one job. Then we take the highest result. If there is no salary specified, the result will be 0 according to the definition. Aggregation from particular job offer attributes scoring to global score is modeled by fuzzy aggregation

$$(@(high\_salary_U, close_U))^{\mathcal{H}}(Java\_Developer) =$$
$$@^{\bullet}(high\_salary_U^{\mathcal{H}}(Java\_Developer), close_U^{\mathcal{H}}(Java\_Developer)) =$$
$$\frac{(3 \times high\_salary_U^{\mathcal{H}}(Java\_Developer) + 2 \times close_U^{\mathcal{H}}(Java\_Developer))}{5} = 0.72$$

hence a TBox can contain a statement $good\_job_U \equiv (@(high\_salary_U, close_U))^{\mathcal{H}}$.

Fuzzy description logic f-$\mathcal{EL}^{@}$ is the motivation for creating a model for fuzzy RDF. For example, a fuzzy instance $high\_salary_U^{\mathcal{H}}(\text{Java\_Developer}) = 0.53$ can be modeled by the RDF triple: $\langle \text{Java\_Developer}, high\_salary_U^{\mathcal{H}}, 0.53 \rangle$.

This is an embedding of a fuzzy logic construct into classical RDF, which needs to translate also constructions of DL, namely $\exists salary.high\_salary$ in fuzzy DL turns to composition of roles, where $\exists salary.(\exists high\_salary.\top)$ needs an aggregate max (or top-k) extending DL with a concrete domain (see [1]). It is out of the scope of this paper to describe such DL with a concrete domain and embedding of our fuzzy DL in more detail. What we claim here is an experimental implementation of both our fuzzy DL, special crisp DL with a concrete domain and a related model of RDF with extended syntax and semantics of `owl:someValuesFrom`. This f-$\mathcal{EL}^{@}$ description logic forms a model theoretic part of our semantics. In practice, methods and implementations (computational model) are based on many valued logic programming. In generalizing logic programming to many valued case we have two basic possibilities. We can consider a rule as an implication (see [33]) or a clause (see [26]). In this paper, we have chosen the first approach, especially because it has a good induction model. Rules learned can get the form $good\_job(U, JO) \longleftarrow @(high\_salary(U, JO), close(U, JO))$.

2.3 User Preference Ontology

Description logics are theoretical counterparts of OWL ontologies. Therefore we also use an ontology to represent and store our user preference model. Ontology structure is especially suitable for storing user preferences because of their rich irregular structure and possible missing values. The main ontology class *User* has the following properties that express relevant personal information:

- *hasAge*
- *hasMaritalStatus*
- *hasGender*
- *hasEducationLevel*.

Every user is then represented as an instance of class *User* with a unique URI identifier. Personal information is collected as a part of user's registration in the system. It can be used later to determine initial set of local user's preferences.

User's personal information is independent from particular domain, but preferences are domain dependent. It means that the user may have different preferences to place in different domains, e.g., flats and job offers. User instance is therefore connected with one *DomainSpecificUser* instance for every domain that we want to model, e.g., one instance for job offers and another instance for notebooks. Instances of *DomainSpecificUser* can be acquired from different sources and with different methods. Together they form a complex model of domain specific preferences which are used for finding best objects.

DomainSpecificUser has the following properties:

- *hasAttributePreference* – local preferences related to domain specific attributes
- *hasRuleCharacteristic* – part of global preferences.

Local preferences are related to fuzzy, fuzzified or crisp attributes. In the first case, *AttributePreference* instance is connected with fuzzy set, as shown in Figure 2. Fuzzy set membership function is piecewise linear and it is specified by a set of points with x, y coordinates and string labels. We consider only fuzzy sets that correspond to four basic types mentioned in Section 2.1. Thus, every fuzzy set in our ontology must have a specified type. In Figure 2, HB is an abbreviation for higher-best fuzzy membership function and LB for lower-best function, MID means middle-best and MAR means marginal-best.

All ontology visualizations in this chapter are created with the Ontoviz tool[1]. They have frame-like structure – every ontology class is represented by a frame containing the class name and a list of its attributes. Every attribute has a specified type, which can be either some primitive datatype like e.g. float, or (instances of) another class. Thus we distinguish datatype attributes and object attributes. Object attributes can also be represented by an arc labeled with attribute names. Special

[1] `http://protegewiki.stanford.edu/index.php/OntoViz`

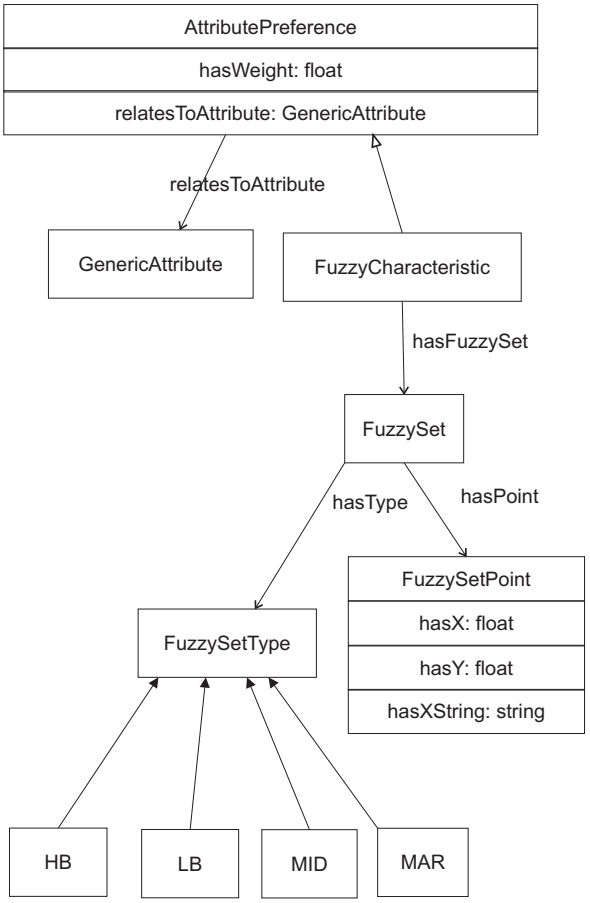


Fig. 2. Ontology representation of preference to fuzzy attributes

types of arcs include instance relationship (indicated by a full arrow head) and subclass relationship (indicated by an empty arrow head).

In the case of crisp attributes, we can express preference as a (crisp) set of preferred attribute values (Figure 3).

However, the user may still prefer some values of such attribute more that other values. It is possible to specify user's degree of preference to some values. This method is called fuzzification of crisp attribute. In the user preference ontology (see Figure 4), each fuzzified value consists of the original crisp value and its evaluation from $[0, 1]$.

As stated in Section 2.1, global preferences can be represented as monotone classification rules. Each rule consists of one or more condition clauses and a result value. An example of clause is $good\_salary(x) \geq 0.5$ where *good_salary* is a fuzzy

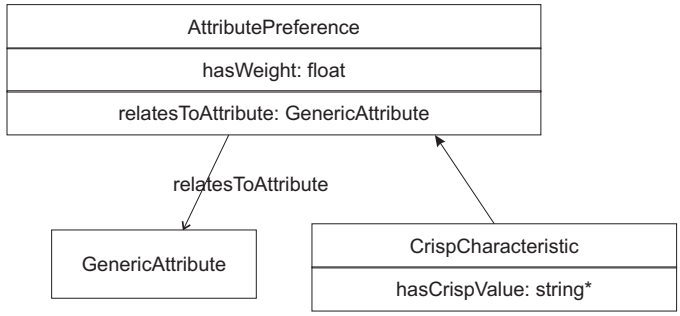


Fig. 3. Crisp attributes

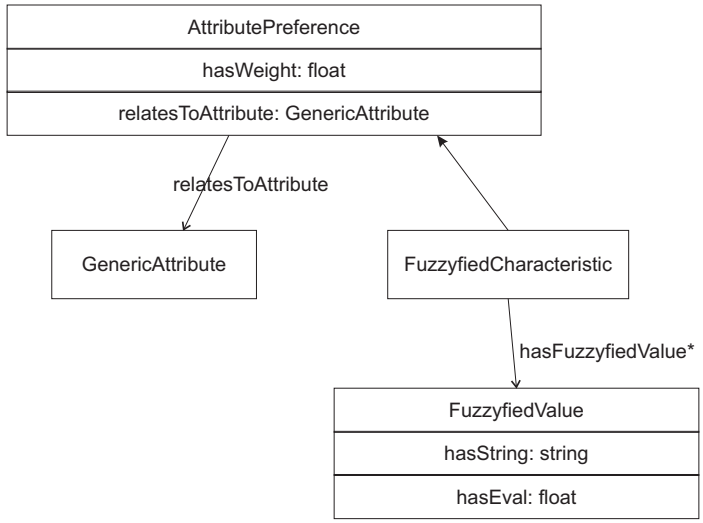


Fig. 4. Fuzzified attributes

set for domain specific attribute *Salary*, $\geq$ is an instance of class Relation and 0.5 is a datatype value. Ontology schema for classification rules is shown in Figure 5.

2.4 Domain Ontology

Domain ontology models the application domain. In the NAZOU project, it includes all information about concrete job offers collected from the web. The extraction of the job offers from the web can be performed in different ways through the different methods. In this paper the tool OMIN (see Section 3.1) represents methods of data extraction. Due to the experimental status of this tool we complement the extracted data with job offers that were entered manually via specially crafted web interface.

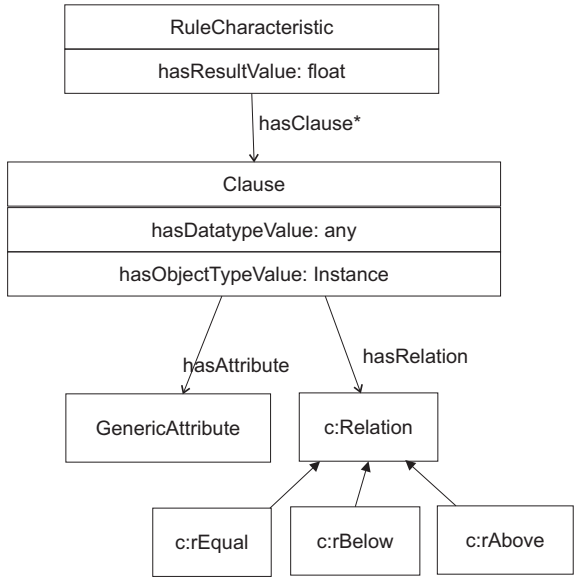


Fig. 5. Rule characteristic

The domain ontology consists of concepts, relations between concepts and instances of concepts. There are two types of concepts – domain independent and concepts that depend on domain. Therefore domain ontology of job offers includes several independent ontologies:

- *offer* – defines job offer in general,
- *offer-job* – defines concrete job offer, the organization which offers the job positions and additional concepts (e.g. salary and benefits),
- *region* – domain independent ontology, which defines geographical (prefix r) and administration regions and their instances,
- *classification* – classifies branch of industry, profession or education level (prefix *c:*),
- *offer-job-inst* – in contrast to previous ontologies which define only schema (metadata), this ontology contains instances (data). Simply said, *offer-job-inst* imports other four ontologies and extends them with instances. This approach ensures that the ontology schema is separated from data and it does not change when data is added or deleted.

Domain independent ontologies can be used in other application domains.

In the ontology *offer* general job offer is represented by class *offer* with *hasSource*, *hasCreator*, *hasValidityInterval*, *name* and *summary*. Otherwise, the most important class is in the ontology *offer-job* and it is called *JobOffer*, which represents the real job offer. In Figure 6, objects properties of the *JobOffer* are

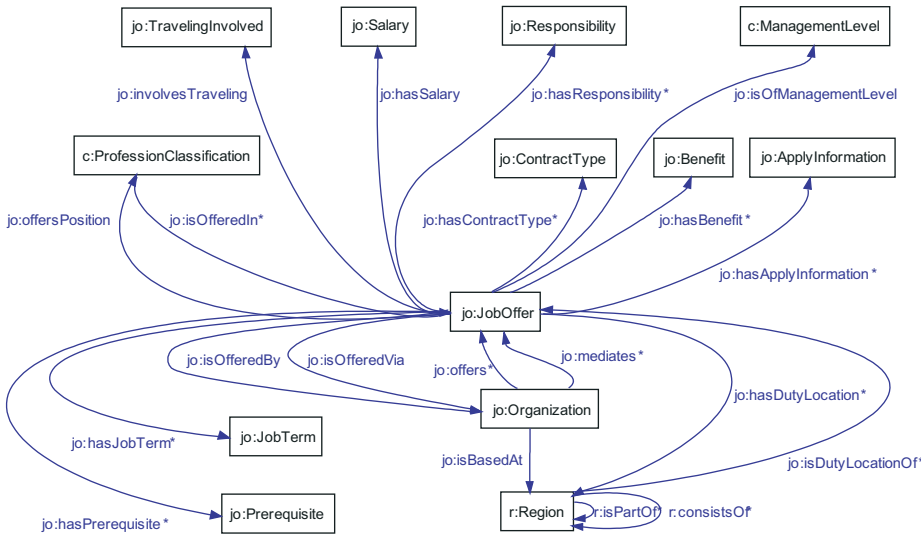


Fig. 6. Object properties of the class *JobOffer* designed in the NAZOU project [24]

shown, which are connected with the classes *ApplyInformation* (information about job application), *Benefit* (e.g. company car, insurance), *ContractType* (e.g. full-time, part-time), *c:ManagmentLevel* (company level of the position in hierarchy of management), *Organization* (organization that offers or mediates the job), *Prerequisite* (preconditions for job candidate), *ProfessionClassification* (profession classification according to OSN),*r:Region* (allocation of work position), *Salary* (highest/lowest salary, reward, currency, time interval of salary), *TravelingInvolved* (encodes the information if traveling is included in job content).

Except for object properties (attributes), there are also datatype properties:

- *hoursPerWeek* – the float number about work hours per week,
- *name* – any string to encode the name of the offer,
- *startDate* – date as a start date of the beginning the work,
- *startDateASAP* – property to encode if it is necessary to start work right now,
- *subordinateCount* – number of subordinates and
- *text* – text source of the offer (for test purpose).

Usually, it does not make a sense to develop the system for one use (for one domain). Therefore we have tried to design our methods in such a way that the system mentioned above (with small adjustments) can search for mutual funds, flats or car for sale or other entities. Therefore our tools are attribute-oriented. That means that every single tool which needs to handled with the data is not directly working with an ontology (domain or user preference), but it is working

with attributes and their values. Instances of the both ontologies are mapped into attributes.

For domain ontology, we identified the following attributes. Some attributes have finite number of values (degrees) identified from careful analysis of the domain of job offers and real data.

- *EducationLevel* – fuzzy attribute with 10 values denoting from elementary educational level to PhD level,
- *MinSalary* – minimal salary normalized to Slovak crowns per week,
- *Place* – duty location, which belongs to crisp attributes,
- *ExperienceLevel* – fuzzy attribute with 5 values (from *basic* to *expert*),
- *ManagementLevel* – fuzzy attribute, which has 10 levels (from *worker* to *president of the company*),
- *TravelingInvolved* – fuzzy attribute to express amount of traveling with 3 levels (from *none* to *often*),
- *StartDate* – start date of the job position,
- *Profession* – offered position (crisp attribute),
- *HoursPerWeek* – number of hours per week,
- *Benefits* – benefits besides salary (crisp attribute),
- *ContractType* – fuzzy attribute to denote type of contract with possible values, e.g. *Permanent*, *Temporary* or *SelfEmployed*,
- *Industry* – crisp attribute denoting industry sector of the job offer.

3 METHODS

In this section, we introduce the methods used in our system, namely web information extraction, preference acquisition, preference learning and top-k search.

3.1 Web Information Extraction

The information extraction module is the foremost part that retrieves data from the web. We consider the HTML pages, which individually contain the data for a single domain object (we shall call them *detail pages*). Each retrieved HTML is processed by the OMIN tool [22] in order to retrieve document the domain object attribute values. The actual process is based on the two approaches: on the algorithm of comparison of two HTML sources and on the simple extraction ontology based on the regular expressions, which refines found attribute values.

Many detail pages are dynamically generated on the server side by some kind of a template engine. A typical template represents a skeleton of HTML page with variables, which are substituted with dynamic data, thus yielding a result HTML

page which is then sent to client. Therefore, we can expect considerable similarity between two pages generated from a single template.

We compare HTML sources of two web pages by using the well-known *diff*[2] algorithm and focus on the *difference spots*, i.e., places in which do the two sources differ. They can correspond to the variables from the original template (which is unknown) and most often, these variables represent the domain object attributes.

By applying various heuristics and picking a large number of page pairs for comparison, we can deduce the object attribute values. A simple extraction ontology with regular expression patterns can further reduce the number of invalid or misappropriate entries.

3.2 User Preference Acquisition

We use an explicit acquisition method with a GUI that enables users to define both local and global preferences. This GUI also lets user skip explicit specification of his preferences if it is too complicated for him or her. User profile can be also modified later.

The graphical interface for global preferences contains one slider for every attribute (see Figure 7). User can drag this slider from the value "Unimportant" to "Important" and thus specify a weight of particular attribute. This weight is used in weighted average, i.e., initial aggregation function. This weighted average is later replaced with classification rules learned from evaluated results.

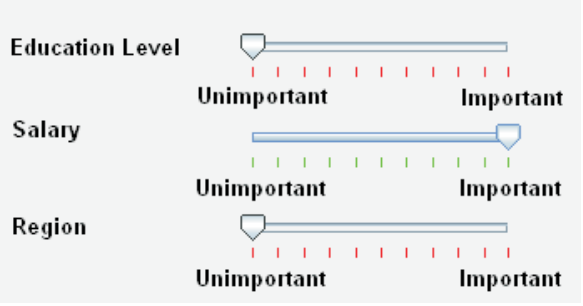


Fig. 7. GUI for explicit specification of global user preference

Local preference specification is different for fuzzy and crisp attributes. In the case of fuzzy attributes, the interface shows possible values of this attribute in the first column. Every value has a slider in the second column. The slider enables the user to specify a degree of preference for corresponding attribute value (see Figure 8).

Crisp attributes have a finite range of possible values, usually strings. These values have no apparent ordering. The interface shows a combo box with possible string values. If the user selects some value, it appears on the right side of the GUI

[2] `http://www.gnu.org/software/diffutils/`

Fig. 8. GUI for local user preference to fuzzy attributes

together with a slider and a "remove" button. This button removes the corresponding slider from the GUI and returns the value back to the combo box. This may be useful if the user decides that he is not interested in the value anymore. The slider has the same functionality as with fuzzy attributes: it specifies a degree of preference for corresponding value. Thus we specify a fuzzification of crisp attribute (see Section 2.3). Figure 9 shows the interface for the Region attribute. Three values (Europe_Central, Europe_Western and Europe_Northern) have been previously selected and their preference degrees have been set with sliders. All user preferences specified with this interface are stored to ontology and used for searching the best objects (see Section 3.5).

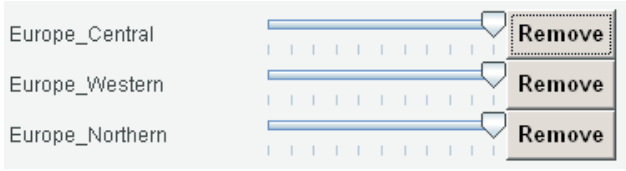


Fig. 9. GUI for local user preference to crisp attributes

Missing parts of user profile are filled by modified collaborative filtering method. Collaborative filtering systems [7] are able to find a group of users similar to the current user and to predict his/her preferences as average preferences within the group. We find a group of similar users according to basic user's personal information from the registration questionnaire. This information is stored in the domain independent part of our user ontology (see Section 2.3). We store only a few user characteristics with finite number of values (e.g., education level, gender, age). Different combinations of these values are called user types. There is also a finite number of attributes. We can compute a hash function for every user type and every attribute. Thus we get two-dimensional hash table.

Table 1 shows a simplified example of such hash table. The first column contains hashed user types, the first line contains attribute names (in real implementation, these attribute names are also hashed). For spatial reasons we restrict this table to five user types and two attributes, EducationLevel and Salary. This table contains four numbers for every user type and attribute: number of higher-best, lower-best,

	EducationLevel				Salary			
	HB	LB	MID	MAR	HB	LB	MID	MAR
−34 115	1	7	0	0	5	0	3	0
44 125	4	10	2	0	12	1	3	0
23 112	2	2	12	1	10	3	4	0
−11 213	1	4	6	0	2	1	7	1
−24 212	0	6	2	0	1	5	0	2
total	8	29	22	1	30	10	17	3

Table 1. Two-dimensional hash table of user types

middle-best and marginal-best fuzzy sets for each user type and attribute taken from previous users.

If a new user would skip explicit specification of his/her preferences, we can take his/her personal information from the questionnaire, compute a hash function and look for the result in our hash table. E.g., if the hash function returns a result 23 112, we search for the corresponding row in Table 1 and choose the prevailing type of fuzzy set for every attribute. The new user would receive a universal middle-best fuzzy set for the attribute EducationLevel and a universal higher-best fuzzy set for the attribute Salary.

Note that the result of the hash function does not have to be found in the hash table. In our example, Table 1 contains only 5 user types and it does not contain e.g., type 11 110. This would mean that no user of the same type (11 110) ever entered our system before. In this case, the user would receive fuzzy sets prevailing for all users, not only for users of the same user type. The new user 11 110 would have lower-best fuzzy set for EducationLevel and higher-best fuzzy set for Salary. Thus we can acquire approximate preferences even for those users who do not provide them explicitly.

Other possibilities of acquiring user preferences are inductive and statistical learning methods described in the following sections.

3.3 Detecting Local Preferences

To learn local preferences, we have several possibilities. The first one is, we present to user a representative sample of job offers (see Table 2) with several attributes, e.g., salary, education, responsibility, location, traveling. Real job offers seldom specify the amount of traveling – this is a typical example of attribute with many missing values. The user classifies jobs into categories *unsuitable(U)*, *rather unsuitable(RU)*, *neutral(N)*, *rather suitable(RS)* and *suitable(S)* according to the relevance of the job offer to him/her.

We distinguish four basic types of local preferences, according to user's most preferable attribute values. We call these basic types *higher-best*, *lower-best*, *middle-best* and *marginal-best.*

Job offer	**Salary**	**Place**	**Travel**	**Eval**
Programmer Analyst (Chicago)	25 812 Sk/W	US_IL	Often	U
VP of Application Development	91 778 Sk/W	US_MD		S
Java Developer (Lake Oswego)	42 000 Sk/W	US		N
Bank Quantitative Developer	Sk/W	London		RU
HMS Associates Of Tri-State Inc.	66 000 Sk/W	US		RS
Secretarial Support	Sk/W	FR		U
Systems Programmer	48 757 Sk/W	US		N
VC++ Makefile Generation	36 000 Sk/W	US_AZ		N
PeopleSoft Programmer	64 572 Sk/W	US_NC		RS

Table 2. Representative sample job offers evaluated by the user

The local preferences can be detected by statistical methods, e.g., regression or QUIN [6]. Using the linear regression we can detect just two basic types (*higher-best* and *lower-best*). In contrast to QUIN, the regression is resistant against statistically irrelevant values. Thus for the purposes of detecting the aforementioned four basic types, the polynomial regression is the most appropriate approach. In the case of Table 2, we checked that the higher the salary (*higher-best* type of ordering) and closer to US (*lower-best* type of ordering) are appropriate for the user. Albeit this method is not fully integrated in the experimental implementation, it is useful for the motivation purposes.

In our system, we offer additional possibility, namely, to learn local preferences explicitly from the user. The user has the possibility to specify his/her preferences by explicit choice of fuzzy functions. We have used this method in our experiments (see Section 5).

3.4 Learning Global Preferences

We learn user's global preferences by the method of ordinal classification with monotonicity constraints [18] based on multiple use of the Inductive Logic Programming (ILP) system ALEPH [28].

The user's global preferences are computed by using his/her local preferences which can be well represented in ILP. For illustration, consider that we have discretized the values of salary to three classes: *small*, *medium* and *large*. The different orderings of these classes for different users can be:

$$\begin{aligned}
\text{small} &\leq \text{medium} \leq \text{large}\\
\text{small} &\leq \text{large} \leq \text{medium}\\
\text{large} &\leq \text{medium} \leq \text{small}\\
\text{large} &\leq \text{small} \leq \text{medium}\\
\text{medium} &\leq \text{small} \leq \text{large}\\
\text{medium} &\leq \text{large} \leq \text{small}
\end{aligned}$$

These orderings indicate which values of an attribute are preferable for a given user (e.g. the medium values of the attribute salary are preferred to the large values, which are preferred to the small values – as depicted in the last case).

An usual aggregation function can be easily simulated by monotone classification rules in the sense of many valued logic:

job evaluation	=	suitable IF salary large AND Place close to US
		AND Traveling involved
job evaluation	=	rather suitable IF salary large AND ...
		OR salary medium AND Place close to US

We can see that the ordered meaning of the classification is preserved in our results (this is proved in [17] as the *igap-consistency* of our approach): jobs classified in the grade "suitable" by the user also fulfills the requirements for "rather suitable" and "neutral" jobs, and "rather suitable" job fulfills the requirements for "neutral" ones, e.g., the job with large salary and in US with traveling involved is "at least as appropriate as" the job with medium salary and in Europe according to the local preferences (higher salary and closer to US is the better).

From our results we can also obtain additional information about crisp attributes (e.g. traveling involved, area of expertise). The meaning of any classification rule is as follows: If the attributes of 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 suitable or rather suitable. During the simulation of computation of aggregation function 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. Since we test the sorted rules, we always rank the object with the highest possible value.

It is a well known fact in data mining that the computation of an ILP system is complex, since it deals with joins of relations (and several substitutions). In our case, an RDF-triple "(Object-Attribute-Value)" is represented as a predicate attribute(object,value). Thus, all predicates are joined via an object identifier and substitutions are considered just for attribute values. All these settings can be determined in the ALEPH background knowledge. Moreover, the computation is made on a small dataset (user does not rate more than a few objects), so the learning process is relatively fast.

Similarly to our approach, in [9] user preferences are learned using an Inductive Logic Programming system TILDE [5]. In this work, monotonicity is not considered. However, due to its expressiveness, input and output of ILP systems are readable, in contrast to sub-symbolic systems like e.g. neural networks.

3.5 Relevant Object Search

Now we have orderings of properties from learning of local preferences and rules from learning of global preferences. Our last task is to find top k objects which are

suitable for particular user. We use the extension of middleware search of Ronald Fagin [10]. The main idea is to browse only the necessary data until the system is sure that it has the top-k objects already. Thus we do not need to calculate with whole domain data. Limiting the retrieval to the k best objects is often sufficient for the user and saves the time. The efficiency grows with the number of objects stored in domain ontology and also with the number of properties we consider.

The model in [10] works with data stored in possibly distributed lists that are ordered from the best to the worst in particular property. Fagin considered two kinds of accesses to 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 particular object. We want to minimize the number of accesses to lists and of course the time of searching. The random access has one large drawback. Each random access requires searching of the value of specific object. In the case of sorted access the results are prepared immediately (values can be preordered according to various criteria). Moreover, data can be sent in blocks. These important differences are the reason why we prefer the sorted access algorithms to the random access in our implementation.

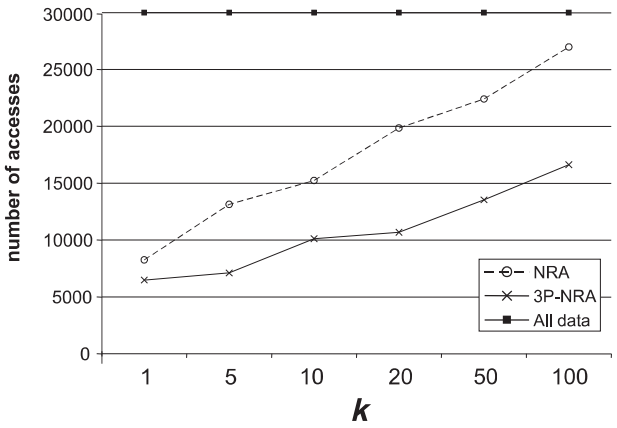


Fig. 10. Number of I/Os to disk needed to retrieval of top-k objects

In our application we use 3P-NRA (3 Phased No Random Access) algorithm [13], which is an improvement of NRA [11] algorithm with the support of various indices.

In Figure 10, we can see the effectiveness of top-k search from the tests with 2 exponential and 2 logarithmic distributions of properties with 10 000 objects and 6 types of aggregation functions. For all experiments, we have considered 3 properties. Various combinations of properties and aggregation functions were used for composition of 25 inputs for algorithms. The final results show the averages of particular results. The techniques of top-k search significantly reduces the number

of accesses and correspondingly decreases the search time, especially in the case of a large set of objects.

Indices used by 3P-NRA algorithm [15] allow the simulation of sorted access, used by 3P-NRA, for different local preferences over various attribute domain types: ordinal, nominal, metric and hierarchical. Without indices, i.e., with data stored in lists we have the problem that fuzzy sets and fuzzifications of different users cause different ordering of attribute domains. Reordering of the lists according to local preferences is more time consuming than processing of all objects (that we want to avoid).

For example, when simulating sorted access over the ordinal attribute defined by local preference of a user we use the B+tree index traversal. First, we identify the local maxima of the fuzzy function, find the nearest objects to local maxima and traverse the leaves according to descending fuzzy values of domain values (see [15]). For example, if the fuzzy function has the "middle-best" type we have to create two pointers, both starting at single local maximum. The first pointer traverses the tree in descending direction and the second in ascending direction. Having simple functions (partially linear) we can very easily find points of extremes to identify the pointers and directions.

3.6 Fulltext Search

Fulltext search is a usual part of many middleware systems which provide the ability to retrieve the relevant objects. Its main advantage is the ease of use allowing the use of natural language queries (in complex use-cases) and the high relevance, provided that the user supplies the useful terms. Such method is available in our system, too, in the form of the JDBSearch tool [21].

This tool, based on the vector model, provides both basic functionalities of fulltext search engine: the indexation and the querying processed. The actual indexing consists of building a special data structure – an *inverted list*, which can be described as a list, where each element contains a term and a list of documents with occurrences of that term.

Indexed documents can be retrieved from the arbitrary datasource; however, we can assume that we process plaintext files from filesystem. Each document is at first tokenized into single *terms*. Next, each term is processed by various tools, i.e. lemmatizer or stemmer. Finally, each term is put into the inverted list and the document weights are calculated according the inversion document frequency formulae; we will calculate the weight using the inversion document frequency formula

$$w_{ij} \quad = \quad \mathrm{tf}_{ij} \cdot \log\left(\frac{n}{\mathrm{df}_j}\right).$$

In this equation, df_j represents the number of documents which contain the term t_j, and tf_{ij} denotes the number of occurrences of term t_j in the document D_i. The w_{ij} then can be seen as a weight of the term t_j in the document D_i.

This inverted list can be stored in a relational database (for example by mapping terms and documents on separate tables while maintaining the term occurrence and document weights in the separate table). The actual storage process is, however, the matter of implementation.

The querying part is somewhat similar. Query terms are also tokenized and optionally lemmatized or stemmed. We support two modes of operation: the first with terms separated by AND connective and the second with terms separated by OR connective. After the query preprocessing part, the whole query is mapped on the SQL statement, which is executed against a relational database, which calculates the result document relevances.

This fulltext query method can be integrated in many ways. The obvious association is with the Morphonary tool. This linguistic method fulfills the role of lemmatizer in the term processing. Another integration is available by connecting this tool with the relevant object search tool Top-k. The results returned from the JDBSearch tool can be plugged into the top-k search as one of the distributed lists. This way, we can achieve preferential fulltext search, which can possibly reorder the results retrieved from JDBSearch according to user's global preferences.

4 IMPLEMENTATION

Our models and methods were implemented, integrated and experimentally tested in the NAZOU project, atop the application domain of job offers. Our implementation is based on the multiple tools corresponding to the aforementioned methods and algorithms. Each tool can be used as a stand-alone component. However, this is not our goal, since the fundamental part of our middleware deals with the integration of these components. We have chosen Spring Framework [27] as the architectural basis for each component. It glues together and manages the associations between classes, thus maintaining low coupling and high cohesion. Moreover, the utility classes provided by this framework (e.g. Spring's JDBC abstraction) unifies the access to the relational and RDF databases by various tools. Our implementation is web accessible at `http://x64.ics.upjs.sk:8080/nazou/`. Tools are running also at the site of whole project application.

4.1 Attribute Labels Detection in OMIN

Attribute labels detection can be achieved in three different manners. Currently we have implemented the first one [22], which is based on the observation that the attribute labels are explicitly denoted by some kind of separator or delimiter (usually a colon). An example of attribute label is *Salary:*, where ":" denotes a separator. For each element containing a difference spot we can find a nearest left text node and detect whether it contains the separator. If so, we can match the given value with the appropriate attribute. If the element does not contain a separator, we can still consult the ontology, particularly the name annotation property, if it is equal

with enumerated labels. Moreover, we plan to implement the approach in which we consult the retrieved labels with either WordNet or a suitable word hierarchy (e.g., Google Topic Maps). Besides the implemented methods, we have examined other approaches to attribute labels detection. Two are based on the visual emphasis of HTML elements. We can expect that the attribute labels which are in the vicinity of attribute values are often presented in some kind of accentuated form (typically in bold text), what is typically achieved via CSS rules. We can possibly calculate the relative visual weight of each element containing text nodes.

4.2 The Top-k Search

The architecture of our top-k search component allows us to combine various algorithms, heuristics and types of sources in order to achieve the best searching strategy for particular data sources and data types. The architecture takes into consideration an analytical aggregation function as well as classification rules.

The overall design comprises four classes: an `Algorithm` represents the procedure used to evaluate the object order. We provide multiple implementations, which use random or sorted accesses to the lists (or their combinations) – generalized Threshold algorithm and NRA algorithm from [11] and 3P-NRA algorithm [15]. A `DataSource` allows to abstract from the physical location of the input data and is subclassed for various sources – 4 main types of indices: B+tree (for ordinal attributes), Dis-index (nominal attributes), M-tree (metric attributes) and Hierar-tree (hierarchical attributes), then various alternative sources like database, files, SOAP client, ontology and memory list.

A `Heuristic` is used by an `Algorithm` to optimize the time and space requirements. Their implementations are based on several theoretical researches ([10, 12, 13]. Currently, there are seven available options to choose from.

The last important class, `Evaluator`, is used in the process of calculation of the overall value of an object. This is used in the determination of the final order of the objects. One subclass is based on the concept of the aggregation functions and the other uses the rules of the monotone graded classification obtained from the user evaluation.

Each fundamental component is based on the Strategy design pattern. This allows great flexibility in the methods used in the evaluation of the best objects by Ordinal Classification with Monotonicity Constraints.

4.3 IGAP

The architecture of IGAP is based on classes which prepare data in the Prolog-like notation, run the ILP system ALEPH, and parse its results into an object representation.

1. Gather objects and attributes and map them on the instances of `Domain Object` (these are just mappings from attribute names to attribute values).

2. For each degree (except the lowest) iterate and prepare data for
   (a) `BackgroundKnowledgeGenerator` transforms the object into Prolog facts and adds monotonicity constraints in the form of Prolog rules.
   (b) `Classifier` distributes the objects into positive and negative examples and writes them into Prolog facts.
   (c) `IgapRunner` executes the ILP system ALEPH and catches its output.
   (d) `ResultsInterpreter` retrieves the Prolog-rules yielded by the ALEPH and transforms them back into the object

The architecture is designed so that the ALEPH system could be possibly replaced by a customized ILP system. Moreover, we could plug-in additional modules, for example for the discrete attributes, or for the various regression algorithms on the attributes.

4.4 UPreA

Tool UPreA (User Preference Acquirer) [29], [30] is responsible for all actions concerning user preferences: explicit specification, storing to ontology, retrieving and updating, predicting preferences for users who do not specify them explicitly. Thus UPreA provides a software facade for other tools that work with preferences, namely IGAP and Top-k.

The most important classes are:

- `UserProfileDao` is responsible for serializing Java objects to and from RDF format and provides access to Sesame database with ontology data.
- `UPreA` contains two-dimensional hash table with user types. This class implements simplified collaborative filtering to fill missing preferences.
- `FuzzyApplet` provides a GUI for explicit preference specification. It handles both global preferences (represented as attribute weights) and local preferences (represented as fuzzy set membership functions). Screenshots of this GUI can be seen in Section 3.2.
- `UpreaFacade` provides high-level access to preference searching. It has a method `getRatedInstances` which takes user URI identifier and (optional) list of rated results as arguments. It calls other methods of Top-k, IGAP and itself as necessary. If the user previously rated some results, `UpreaFacade` calls IGAP learning method. Then it updates user preferences in the ontology and calls Top-k that retrieves new results according to actual preferences. This method enables cyclic searching for best objects and refining user preferences.

4.5 JDBSearch

JDBSearch consists of two main components. `DocumentIndexStorage` represents the class that writes the inverted list into a relational database. Its main task

is to perform all tasks of the aforementioned indexing process. There are various available data sources which can retrieve the documents and their metadata, e.g., filesystem, ontology, web, etc. The parsing and tokenizing parts comes in the form of two implementations: the `PorterStemmer` performs the stemming of the document terms. `SlovakLemmatizer` is an implementation of lemmatizer customized for inflective Slovak language.

The actual storage process is targeted for the MySQL database (although it is possible to use another database). It allows to index very large collection of data by splitting the indexation into the batches which are directly loaded into the database. These batches can be even indexed independently on the multiple machines, thus achieving parallel indexation.

The second component deals with the actual querying process and is represented by the `FulltextQuery` class. Each query is parsed and tokenized (optionally transformed via stemming or lemmatizing processes) and mapped on a SQL query, which is executed against the database.

4.6 Tvaroslovník/Morphonary

The process of searching in text documents should consider an important observation that one word (usually reflecting one notion) can have different forms. Linguistic processes are used in the processes of indexation and querying. The original implementation of our full-text searcher JDBSearch working with English documents uses Porter's stemming algorithm ([25]) in both cases. By replacing Porter's stemming by the lemmatization provided by our tool Morphonary/Tvaroslovník we obtain a version of tool able to work correctly with Slovak language. More information about Morphonary/Tvaroslovník can be found in [20] (in this volume).

4.7 Tools Interaction Summary

The aforementioned tools can be separated into two groups: the tools that work offline and those which work online.

Offline tools acquire information from the web (Omin) and store it into domain ontology. They are not active during the user's interaction, but work in a "batch" mode (gather data in a single run which can be scheduled on particular time). E.g. Omin is executed whenever new jobs are retrieved from the web. The processed data are not only stored into the ontology, but also indexed. Other tools do not access the ontology directly, but only via index, which rapidly speeds-up the query phase. The reindexation happens on scheduled times or when there is a rapid change in the domain ontology (a job offer was added, changed or deleted). Indices are maintained by the Top-k tool and primarily used by the IGAP and UPreA tools.

Online tools (UPreA, IGAP, Top-k, JDBSearch) are in operation as shown in Figure 11. The user has to provide the data about him/herself through the questionnaire, adds/removes his/her preferences and their relevances (thus defining fuzzy sets) and the tool UPreA stores the information into user ontology. This process

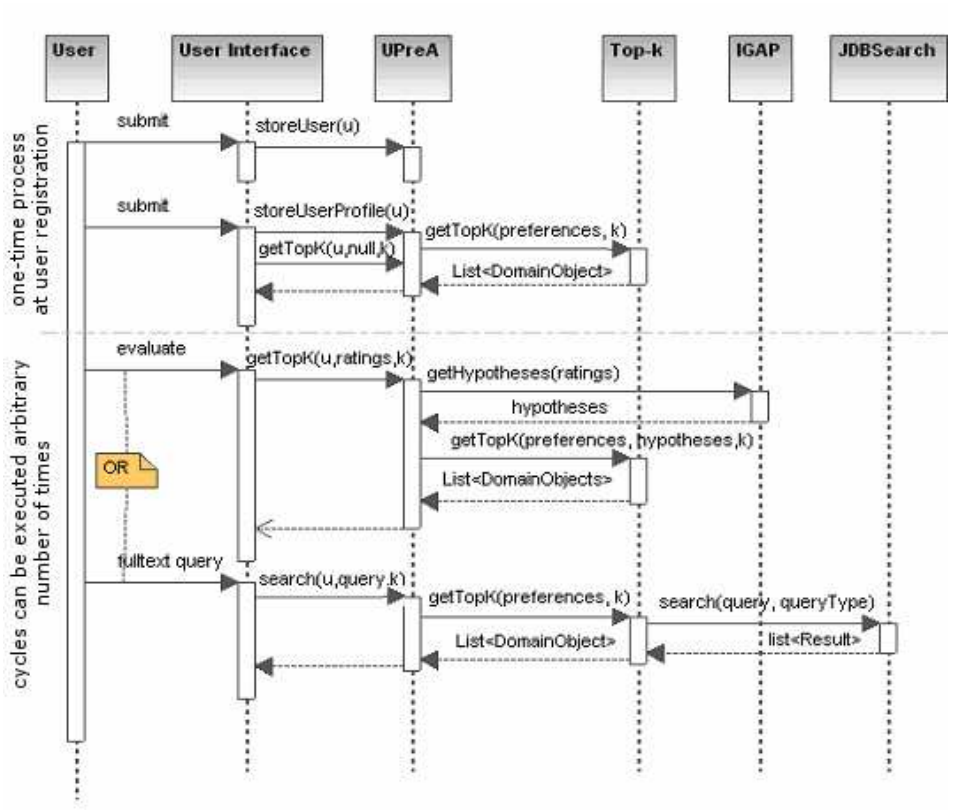


Fig. 11. Sequence diagram of the tools chain

runs just one time and it is shown on the mentioned figure and it is labeled "one-time process at user registration". Then it processes user information and sends it to the Top-k which finds top-10 job offers for the selected preferences. The user has two choices in this step:

- Evaluate the represented offers and submit them to UPreA which calls the IGAP to induce the hypotheses (rules). These are used in the Top-k search process, which produces another 10 job offers.
- Alternatively, the user can use keyword search which means that user preferences are extended with fulltext attribute. With assistance of fulltext attribute it is possible to reduce job offers ordered by user preferences to satisfied fulltext query.

Furthermore, the user has an option to use one of the following activities repeatedly in arbitrary order (in Figure 11 the part labeled "cycles can be executed arbitrary number of times"):

- Object evaluation – new preferences (hypothesis) are inferred by IGAP.
- Searching for the k best offers with regard to user preferences and fulltext query.

5 EVALUATION AND EXPERIMENTS

We use a sample implementation of preference based search system for experiments. The complete user dependent searching process is illustrated in Figure 12. We used sample domain ontology in OWL format with 500 instances of job offers.

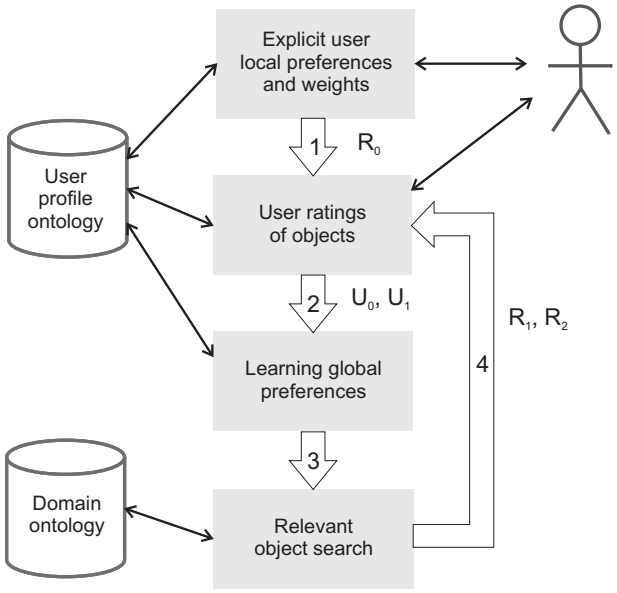


Fig. 12. Flow diagram of user profile dependent part

In our system and experiments, different users can explicitly specify local preferences. We consider job offer attributes like required education and experience level of the applicant, offered salary, manager position and the amount of traveling involved in the job (EducationLevel, ExperienceLevel, MinSalary, ManagementLevel, TravelingInvolved). The testing implementation first asks the user to specify his/her explicit preferences to these attributes. The user can choose if s/he prefers higher, lower, middle or marginal values of these attributes and s/he can even define how much s/he prefers every value. Thus we can recognize *higher-best*, *lower-best*, *middle-best* and *marginal-best* preference types. Additionally, a weight assigned to each attribute can be specified. This is used for combination function being weighted average. These parameters are used to retrieve (arrow 1) top-k objects to user. We denote this list of results as R_0.

User preferences can be changed or stated more precisely during several search cycles. We follow the idea that everybody can easily say how suitable is a concrete object. Thus we will require the user to evaluate the results in scale from the worst to the best.

User evaluation uses 5 possible values: worst, bad, neutral, good, best (see Figure 13). The k-tuple of initially retrieved objects R_0 is transformed to new set U_0 (a fuzzy set with different order induced by user ratings). This is sent (arrow 2) to IGAP tool learning global preferences. These are used (sent via arrow 3) to relevant object search to retrieve top-k objects from the whole set of objects; let us denote this by R_1. After new objects are given to user, s/he can evaluate them (creating U_1) and start the whole process again. We created a database log for user preferences, result sets and user ratings.

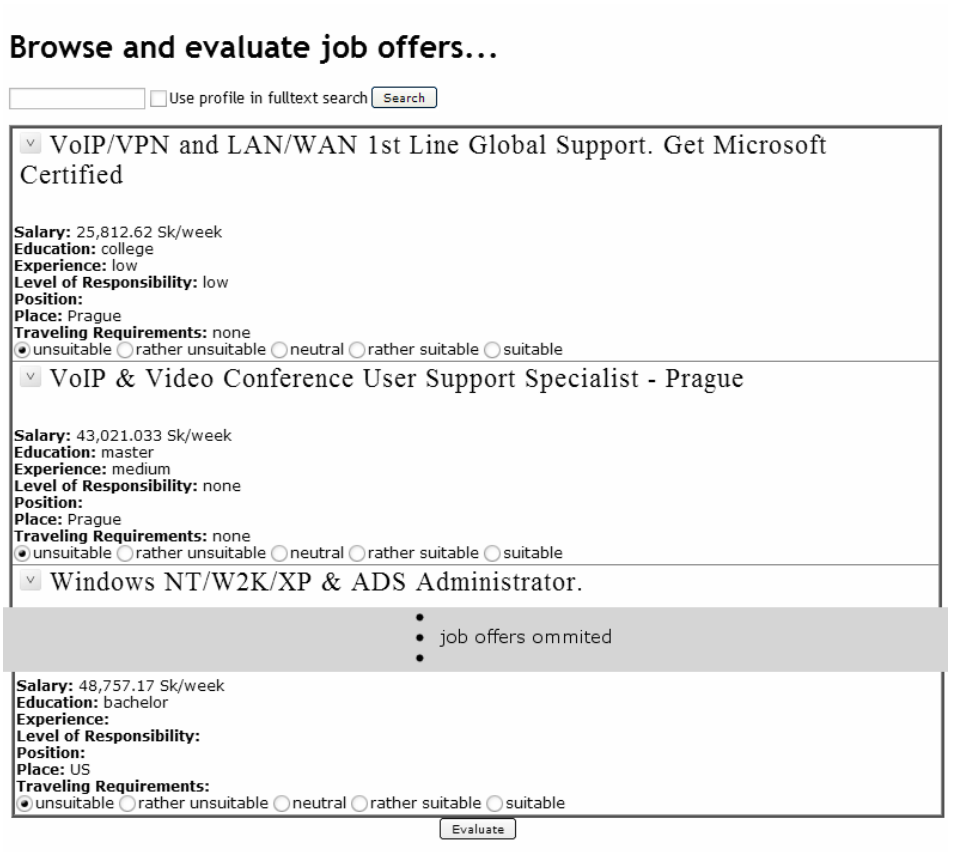


Fig. 13. Screenshot of user interface

We performed two experiments with different groups of users. There were 62 users in the first group and 136 users in the second group. The basic statistics is shown in Table 3.

Users	Text Searches	Preference Searches	Ratings
62	0	333	3 547
136	77	482	4 402

Table 3. A comparison of the experiments

We have already excluded users who were "just browsing" or testing the system availability. In the following analysis, we consider only those users who performed at least one complete cycle, i.e., those who viewed and rated first 10 results and more precise global preferences were (possibly) found from their ratings. Some users passed only one cycle and stopped searching, while other users repeated and refined their search. Most users performed three searching cycles, so we analyze only the results of the first three cycles.

We find that global preferences were found from user ratings in 71 % for the first experiment, 68 % for the second experiment. This means that about one third of user ratings were inconsistent and the induction tool IGAP failed to find global preferences for such ratings. Successive cycles usually improve global preferences; the number of rules was higher or equal in 75 % of successive cycles, 68 % for the second experiment. Sometimes the number of rules oscillates from one cycle to another. This happens for 30 % of users. We suppose that these users do not know exactly what they are looking for or their preferences do not remain the same throughout the testing.

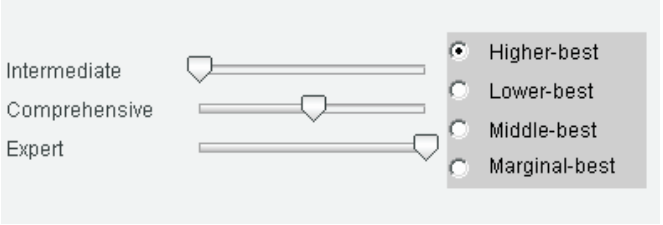


Fig. 14. Graphical user interface for explicit preference specification

Although user's ratings are used for finding global preferences, they also provide us an important feedback. They define some ordering of results. If this ordering is similar to the ordering returned by top-k searching algorithm, it indicates that our preference model resembles real preferences well. We compare two orderings (the ordering R_i given by top-k algorithm and the ordering U_i given by user ratings) with Kendall tau rank correlation coefficient [19]. This coefficient determines a degree of correspondence between two orderings of the same objects.

Kendall tau coefficient is defined as $\tau = \frac{2P}{\frac{1}{2}\cdot n\cdot(n-1)} - 1$ where P is the number of concordant pairs in both ratings. If two orderings are the same, they have tau coefficient 1. If one ordering is the reverse of the other, they have tau coefficient -1.

R_0 and U_0 have average tau coefficient 0.44 (the first experiment) and 0.52 (the second experiment) representing a strong correlation. If we analyze second cycles (R_1 and U_1), we have average tau coefficients 0.60 and 0.62; third cycles (R_2 and U_2) have coefficients 0.58 and 0.63. Note that some users stopped after the first or the second cycle, so the number of results is smaller for the third cycle. The correlation coefficients show that the inductive methods really improve global preferences, see Table 4. The numbers of ratings used for tau computations are shown in Table 5.

Experiment	1. cycle	2. cycle	3. cycle
1	0.44	0.60	0.58
2	0.52	0.62	0.63

Table 4. Correlations of result ordering and user evaluation

Experiment	1. cycle	2. cycle	3. cycle
1	85	33	23
2	96	79	60

Table 5. Numbers of rated result sets

We present a complete record of one typical user. Table 6 shows that this user rated the first 10 results and this ordering by ratings was different from the ordering given by top-k algorithm (tau correlation of R_0 and U_0 is only 0.1556). However, two rules were found from user's ratings and the global preferences were refined. The second result set R_1 has tau correlation 0.5111 compared to U_1. In the third cycle, the number of rules increased to 3 and tau correlation increased to 0.9111. The user was content with his results and he stopped searching.

We are also interested in the correlation of k-tuples R_i, R_{i+1}. If R_i and R_{i+1} have empty intersection, this correlation will be 0. In case of non-empty intersection, we can compute usual tau correlation coefficient of the common elements and multiply the result with $\frac{\text{number of common elements}}{\text{number of all elements}}$. The results are shown in Table 7.

From the mathematical point of view the Kendall-tau coefficient is (the $[-1, 1]$-normalization of) the number of inversions of a permutation. This approach does not consider relevance of place of a change but it is meaningful to think that the interchange of the 1st and the 2nd place is more significant than the interchange of the 8th and the 9th place.

Therefore we propose a modification of this approach which will reflect the place of potential changes. To be more formal, let $p = \langle p_1, p_2, \ldots, p_n\rangle$ be the permutation of positions $\langle 1, 2, \ldots, n\rangle$ and let $w = \langle w_1, w_2, \ldots, w_n\rangle$ be a vector of position weights (or *importances*) which are non-negative reals with property $w_1 \geq w_2 \geq \ldots \geq w_n$.

Offer ID	R_0	U_0	R_1	U_1	R_2	U_2	R_3
01096	2.837	1					
0f1b9	2.802	1					
01004	2.757	4					
01082	2.729	5					
01059	2.723	2					
01011	2.681	4					
9fe41	2.588	5	4	3			
8bc3a	2.588	4	4	4			
01090	2.54	5					
01095	2.54	2			3	2	
e4464			4	5	5	5	5
4c537			4	5	5	5	4
1d41d			4	1			
c56aa			4	5	5	4	4
01007			4	5	5	4	3
8f22d			4	5	5	3	3
01063			4	4	4	2	2
ef534			4	1			2
01069					3	2	
01068					3	1	
01100					3	1	
31a44							2
a8bbc							2
01036							2
τ	0.1556		0.5111		0.9111		
Rules		2		3		4	

Table 6. Testing records for one user and three cycles

Cycle	Result sets	Number of result sets	Correlation
1, 2	R_1, R_2	90	0.52
2, 3	R_2, R_2	80	0.65

Table 7. Correlation of R_i and R_{i+1}

Then define the function τ'_w by the following formula

$$\tau'_w(p) = \sum_{i=1}^{n} |w_{p_i} - w_i|.$$

For example if the permutation is $p = \langle 4,1,3,5,2\rangle$ and the weights are $w = \langle 10,7,4,2,1\rangle$, then

$$\tau'_w = |w_4 - w_1| + |w_1 - w_2| + |w_3 - w_3| + |w_5 - w_4| + |w_2 - w_5|$$

$$= |5-10| + |10-7| + |4-4| + |1-5| + |7-1| = 5+3+0+4+6 = 18.$$

We can see that the minimum of this function is 0 (it is gained for the identity permutation), and its maximum is $\sum_{i=1}^{n} |w_i - w_{(n+1)-i}|$ (for bottom-up permutation). It can be assumed that in our iteration process this number will decrease.

Of course, this function can be normalized to interval $[-1,1]$, namely,

$$\overline{\tau'_w}(p) = 2 \cdot \frac{\sum_{i=1}^{n} |w_{p_i} - w_i|}{\sum_{i=1}^{n} |w_i - w_{(n+1)-i}|} - 1.$$

In our example $\overline{\tau'_w}(p) = 2\frac{18}{28} - 1 = 0.28$, whereas $\tau(p) = -\frac{35}{45}$. We see that two sequences which are rather uncorrelated in tau are rather correlated in this new coefficient. We would like to experiment with this in future.

We can also analyze the local preference types and find which type is the most common for every attribute (see Table 8). It is easy to see that *higher-best* is generally the most common type and marginal-best is the least common. Lower-best type is the second most frequent for ManagementLevel and TravelingInvolved, while middle-best is the second most frequent for EducationLevel, MinSalary and ExperienceLevel. These results reflect some intuitive ideas, e.g., that most people are not satisfied with low salary, seek a job for some specific required level of education or experience and that many of them are not willing to travel.

Attribute	**Higher**	**Lower**	**Middle**	**Marginal**
EducationLevel	41	12	23	4
MinSalary	67	1	10	2
ExperienceLevel	50	10	15	5
ManagementLevel	43	24	8	5
TravelingInvolved	42	25	12	1

Table 8. A summary of local preference types

6 TRANSFER TO OTHER DOMAINS AND CONCLUSIONS

As we have mentioned before, the design and implementation of each method should allow fairly easy migration to other application domain. However, this process is not fully automatic and requires multiple steps. The most important part is the creation of the new ontology. Then the configuration file which contains the attribute definitions, their types and corresponding ontology mapping should be accommodated to the new domain. Finally, the user interface has to be adapted to new conditions.

Our first domain migration experiment is based on the application domain of mutual funds. The main goal is now slightly different. To find the suitable objects means to retrieve the best funds which the potential investor can invest in.

According to the migration steps, we created the prototype ontology, which can be extended with help of prepared subontologies from application domain of job-offers (e.g., *region* or *classification*).

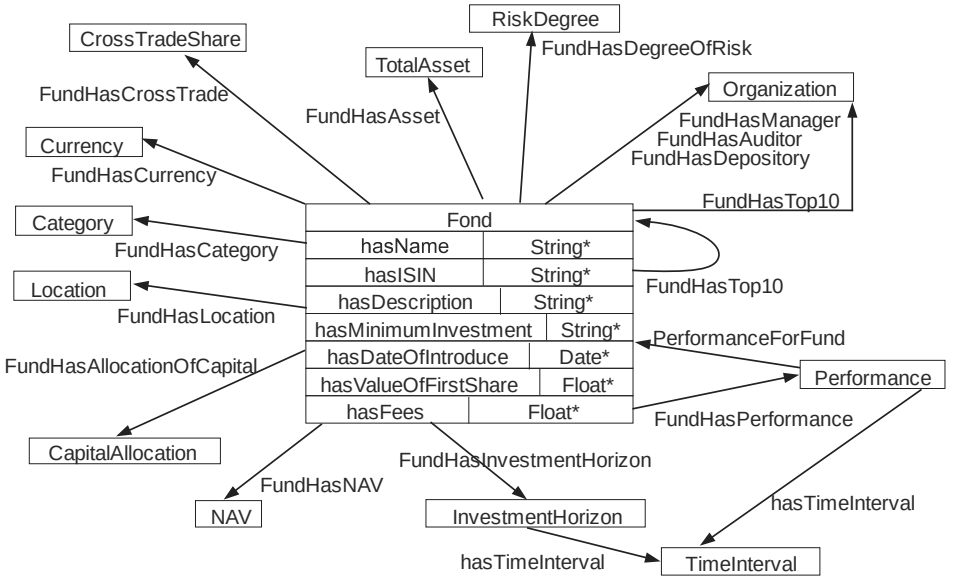


Fig. 15. Object properties of the *Fund* class

The class *Fund* is cardinal class of ontology and its instances present mutual funds (see Figure 15). *Fund* has many datatype properties, from which the most important are:

- general information – *hasName, hasISIN, hasMininumInvestment, hasDescription*,
- information about history of the funds – *hasDateOfIntroduce, hasValueOfFirst-Share*,
- fees – *hasManagementFee, hasPreliminaryFee, hasFinalFee, hasDepositoryFee*.

Object properties can be divide into temporal and non-temporal. The temporary properties are:

- *FundHasNAV* – net asset value to date (the range is the *NAV* class),
- *FundHasPerformance* – performance of the fund to date (calculated from NAV's; the range is the *Performance* class),

- *FundHasAsset* – whole asset of the fund to date (the range is the *TotalAsset* class),
- *FundHasCrossTrade* – total sales and purchases of fund shares to date (the range is the *CrossTradeShare* class) and
- *FundHasInvestmentHorizon* – recommended duration (time interval) of investment (the range is the *InvestmentHorizon* class).

Except for temporal object properties there are the following properties:

- *FundHasCurrency* – information about reference currency of the fund (the range is the *Currency* class),
- *FundHasCategory* – category of the mutual fund, e.g., bond fund or equity fund (the range is the *Category* class),
- *FundHas*{*Auditor/Depository/Manager*} – information about auditor, depository and manager of the fund (the range is the *Organization* class),
- *FundHasLocation* – describes geographical location of investments of the fund (the range is the *Location* class), for now we consider just two locations – Slovakia and abroad.
- *FundHasAllocationOfCapital* – is information about composition of shares in percentage (the range is the *CapitalAllocation* class),
- *FundHasDegreeOfRisk* – is instrument to express normalized degree of the risk (the range is the *RiskDegree*),
- *FundHasTop10* – is list of the 10 biggest organizations or another fund, where the fund has invested.

There are many cases of similarity between the default domain and this new domain. However, there is one issue not occurring in the job offers ontology – the presence of temporal data (such data are not present in our job offers model). This issue is present in properties such as *PerformancePerMonth* or *PerformancePerYear*. This is not unnatural, since *PerformancePerMonth* calculated today is obviously different from *PerformancePerMonth* calculated yesterday.

Therefore we decided to find out just performance for the whole month (or half-year and year). If we want to calculate the performance on March 31st 2008, the *PerformancePerMonth* is calculated from February 1st to 29th 2008. When the calculation day is April 1st 2008, then *PerformancePerMonth* is calculated from March 1st to 31st 2008. For the same reasons we decided to consider only the performance per month, half of the year, year and annum.

The next migration step deals with the configuration of domain object attributes. At first, we have identified the following mutual funds attributes:

- *PerformancePer*{*Month/Half-Year/Year/Annum*},
- {*Preliminary/Final/Management*}*Fee*
- *Manager, Depository, Auditor*

- *Currency*
- *Category*
- *Allocation*
- *InvestmentHorizon*
- *MinimumAmountOfInvestment*
- *RiskDegree*.

Moreover, we have to migrate the user preference ontology. In this case, it is the same as the ontology of job-offers. Also the semantics of user preference model is the same as in the default application domain.

6.1 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 and are processed to vector index and to a domain ontology. The system implements personalized search for users with different preferences. Our theoretical model integrates all parts of the system from collected data in classical RDF form to user query answering. We do not have a model for web resource downloading and ontology annotation part. For this we present only methods and a tool.

The model of user dependent search is based on modeling preferences by fuzzy sets and fuzzy logic. We present the semantics of fuzzy description logic f-$\mathcal{EL}$. User dependent search integrates both inductive and deductive approach. By induction we can learn local and global preferences (although currently we have implemented only global preferences). Results of induction as well as hand-filled orderings can be easily modeled in user ontology and fuzzy RDF. Deductive part of the system uses the preferences to find suitable objects for user, who can express his/her preferences precisely by fuzzy functions and aggregation function. The easiest way is to evaluate several objects in a scale and to let the system learn preferences. Repeating this process (evaluating a set of objects and finding new objects) leads to refining the user profile. The user who is satisfied with his profile and with the presented search results does not have to evaluate objects again. When the domain ontology is updated, user can just get the best objects according to his/her profile. Effective identification of suitable user type for a new user is the aim of our future research. It can be done by collecting more data from real users. We are collecting some personal information like age, gender, job position, etc. and the explicit specification of preferences from each user and then we will analyze the data in search for dependency.

The presented model was experimentally implemented and integration was tested using Spring Framework [27]. Both searching methods (keyword text search and preference search) consist of tools which can be further improved separately. After the phase of gathering data about users, it will be possible to compare the efficiency, speed and complexity of these methods and decide about their practical usage. In

future we plan to experiment with infrastructure of [34] and [4], which could possibly replace [27].

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* [24] to find relevant job offers for the user according to his/her preferences.

Similar strategy of communication with users (evaluation of sample objects) was used in [23], where the learning part of the system was covered by a neural network. This approach does not permit to model user preferences. We did not find any other similar approach to the whole process.

Acknowledgements

Partially supported by Czech projects 1ET100300419 and Slovak projects VEGA 1/3129/06 and NAZOU (Košice branch of the project under the leadership of P. Vojtáš).

REFERENCES

[1] Baader, F.—Kuesters, R.—Wolter, F.: Extensions to Description Logics. In Description Logic Handbook, Cambridge University Press, 2003, pp. 219–261.

[2] Balke, W.—Gűntzer, U.—Zheng, J.:Efficient Distributed Skylining for Web Information Systems. EDBT 2004, LNCS 2992, Heraklion, Crete, Greece, Springer 2004.

[3] Bartalos, P.—Barla, M.—Frivolt, G.—Tvarožek, M.—Andrejko, A.—Bieliková, M.—Návrat, P.: Building an Ontological Base for Experimental Evaluation of Semantic Web Applications. In Proc. of SOFSEM 2007, Lecture Notes in Computer Science, Vol. 4362, Springer-Verlag 2007, pp. 682–692.

[4] Bednárek, D.—Obdržálek, D.—Yaghob, J.—Zavoral, F.: Data Integration Using DataPile Structure. In Advances in Databases and Information Systems, Springer-Verlag 2005, pp. 178–188.

[5] Blockeel, H.—de Raedt, L.—Jacobs, N.—Demoen, B.: Scaling up Inductive Logic Programming by Learning from Interpretations. In Data Mining and Knowledge Discovery, Vol. 3, 1999, No. 1, pp. 59–93.

[6] Bratko, D.—Šuc, D.: Learning Qualitative Models. AI Magazine, Vol. 24, 2003, pp. 107–119.

[7] Candillier, L.—Meyer, F.—Boullé, M.: Comparing State-of-the-Art Collaborative Filtering Systems. Machine Learning and Data Mining in Pattern Recognition, Vol. 4571, 2007, Springer-Verlag 2007, pp. 548–562.

[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 ISIM '06, Czech Republic, 2006, pp. 185–192.

[9] Dastani, M.—Jacobs, N.—Jonker, C. M.—Treur, J.: Modelling User Preferences and Mediating Agents in Electronic Commerce. In Agent Mediated Electronic Commerce, The European AgentLink Perspective, Springer-Verlag 2001, pp. 163–193.

[10] Fagin, R.: Combining Fuzzy Information from Multiple Systems. J. Comput. System Science, Vol. 58, 1999, pp. 83–99.

[11] Fagin, R.—Lotem, A.—Naor, M.: Optimal Aggregation Algorithms for Middleware. In Proc. 20th ACM Symposium on Principles of Database Systems, 2001, pp. 102–113.

[12] Gűntzer, U.—Balke, W.—Kiessling, W.: Towards Efficient Multi-feature Queries in Heterogeneous Enviroments. ITCC, Las Vegas, Nevada, IEEE Computer Society, 2001, pp. 622–628.

[13] Gurský, P.: Towards Better Semantics in the Multifeature Querying. In Proceedings of Dateso 2006, CEUR Workshop Proceedings, available from http://CEUR-WS.org/Vol-176/paper9.pdf.

[14] Gurský, P.—Horváth, T.—Novotný, R.—Vaneková, V.—Vojtáš, P.: UPRE: User Preference Based Search System. In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, WI '06, Hong Kong, 2006, pp. 841–844.

[15] 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.

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

[17] Horváth, T.—Vojtáš, P.: Induction of Fuzzy and Annotated Logic Programs. In ILP 2006, LNAI 4455, Springer-Verlag, 2007, pp. 260–274.

[18] Horváth, T.—Vojtáš, P.: Ordinal Classification with Monotonicity Constraints. In ICDM 2006, Leipzig, Germany, LNAI 4065, Springer-Verlag, 2006, pp. 217–225.

[19] Kendall, M. G.: Rank Correlation Methods. Hafner Publishing Co., New York, 1955.

[20] Krajči, S.—Laclavík, M.—Novotný, R.—Turlíková. L.: The Tool Morphonary/Tvaroslovník: Using of Word Lemmatization in Processing of Documents in Slovak. Znalosti 2009, pp. 119–130, Vydavateľstvo STU 2009.

[21] Lencses, R.: Indexing for Information Retrieval System supported with Relational Database. In Sofsem 2005 Communications, SIS Bratislava, 2004, pp. 81–90.

[22] Maruščák, D.—Novotný, R.—Vojtáš, P.: Unsupervised Structured Web Data and Attribute Value Exctraction. Znalosti 2009, pp. 143–154, Vydavateľstvo STU 2009.

[23] Naito, E.—Ozawa, J.—Hayashi, I.—Wakami, N.: A Proposal of a Fuzzy Connective with Learning Function and Query Networks for Fuzzy Retrieval Systems. In Fuzziness in database management systems, P. Bosc and J. Kacprzyk (Eds.), Physica Verlag 1995, pp. 345–364.

[24] NAZOU. Tools for Acquisition, Organization and Maintenance of Knowledge in an Environment of Heterogeneous Information Resources. Available from http://nazou.fiit.stuba.sk.

[25] Porter, M. F.: An Algorithm for Suffix-Stripping. Program, Vol. 14, 1980, No. 3, pp. 130–137.

[26] Smutná-Hliněná, D.—Vojtáš, P.: Graded Many-valued Resolution with Aggregation. Fuzzy Sets and Systems, Vol. 143, 2004, No. 1, pp. 157–168.

[27] Spring Framework. System for Assembling Components via Configuration Files. Available from `http://www.springframework.org`.

[28] Srinavasan, A.: The Aleph Manual. Technical Report, Comp. Lab., Oxford University.

[29] Vaneková, V.: Methods of Acxquiring User Preferences. Znalosti 2008 – Zborníík príspevkov, Vydavateľstvo STU, 2008, pp. 387–390 (in Slovak).

[30] Vaneková, V.: Representation and Processing of User Preferences in RDF. Proceedings of MIS 2007 (in Slovak).

[31] Vojtáš, P.: A Fuzzy EL Description Logic with Crisp Roles and Fuzzy Aggregation for Web Consulting. In Proc. IPMU '2006, B. Bouchon-Meunier et al. (Eds.), EDK 2006, pp. 1834–1841.

[32] Vojtáš, P.: Fuzzy Logic Aggregation for Semantic Web Search for the Best Top-k Answers. In Fuzzy Logic and the Semantic Web, E. Sanchez (Ed.), Elsevier, 2006, pp. 341–360.

[33] Vojtáš, P.: Fuzzy Logic Programming. Fuzzy Sets and Systems, Vol. 124, 2001, No. 3, pp. 361–370.

[34] Yaghob, J.—Zavoral, F.: Semantic Web Infrastructure using DataPile. In WI-IATW '06, Los Alamitos, California, 2006, pp. 630–633.

Peter Gurský has finished his Ph. D. study at Pavol Jozef Šafárik University in Košice, Slovakia in 2008. Nowadays he works as a research assistant at the same university. He works in the field of top-k search algorithms, data indexing and semantic web. He participated in the NAZOU project as a developer of the Top-k aggregator tool.

Tomáš Horváth has finished his Ph. D. study at Pavol Jozef Šafárik University in Košice, Slovakia in 2008. Nowadays he works as a research assistant at the same university. His research interest includes fuzzy multi-relational data mining, preference learning and recommender systems. He participated in the NAZOU project as a developer of the IGAP tool.

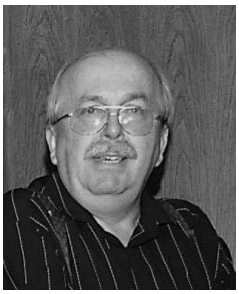
Jozef Jirásek graduated from Charles University, Faculty of Mathematics and Physics, majoring in system theory and mathematical structures, in 1984. Since 1991 engaged as Assistant Professor at the Institute of Computer Science, Pavol Jozef Šafárik University in Košice. His teaching concerns computer networks, computer graphics, cryptography and network security. His work in structure of discrete graphs resulted in a Ph. D. obtained in 2001. His recent research interests focus on the theory of formal languages and formal analysis of security protocols.

Stanislav Krajči is an Associated Professor of computer science at the Institute of Computer Science of Pavol Jozef Šafárik University in Košice, Slovakia. He graduated in 1993 in theoretical cybernetics from Pavol Šafárik University in Košice. His recent research interest includes applied algebra, logic, and natural language processing. He participated in the NAZOU project as a developer of the Morphonary (Tvaroslovník) tool.

Róbert Novotný is a Ph. D. student at Pavol Jozef Šafárik University in Košice. The main area of his research is information extraction from the web. He participated in the NAZOU project and contributed to the Omin extraction tool, cooperated on the Morphonary tool and served as software integrator and architecture maintainer.

Jana Pribolová is recently finishing her Ph. D. study at Pavol Jozef Šafárik University in Košice. She graduated in 2005 in computer science at the Institute of Computer Science of Pavol Jozef Šafárik University. Her research focuses on ontologies, efficiency of querying under ontologies, description logic and sematic web. She participated in the NAZOU project and supplied extensibility of the UPreA – Top-k – IGAP chain to other application domains.

Veronika Vaneková is a Ph. D. student at Pavol Jozef Šafárik University in Košice. She graduated in 2006 at the Institute of Computer Science of Pavol Jozef Šafárik University. Her research focuses on user preference, ontologies, description logics and semantic web. She also participated in the NAZOU project and developed the UPreA tool to acquire and maintain user preferences.



Peter Vojtáš is Professor of computer science at the Department of Software Engineering, School of Computer Science at Charles University in Prague, Czech Republic. He graduated in 1974 in theoretical cybernetics at Charles University. After research in applications of mathematical logic in Slovak Academy of Science in the 90's he moved to computer science research at Pavol Jozef Šafárik University in Košice, especially logic programming. His recent research interest is in flexible querying; uncertainty and vagueness (imperfection) of information; logic programming – deduction, induction, abduction; semantic web, user preferences. He is the leader of the Košice group in the NAZOU project.