I.M.P.A.K.T.: an innovative, semantic-based skill management system exploiting standard SQL

Eufemia Tinelli[1,2], Antonio Cascone[3],Michele Ruta[1],Tommaso Di Noia[1],Eugenio Di Sciascio[1], Francesco M. Donini[4]
Politecnico of Bari, via Re David 200, I - 70125 Bari
University of Bari, Orabona 4, I - 70125 Bari
DOOM s.r.l, Paganini, 7 - 75100 Matera
University of Tuscia, S. Carlo, 32 - 01100 Viterbo
e.tinelli@poliba.it, antonio.cascone@doom-srl.it, m.ruta,t.dinoia,disciascio@poliba.it, donini@unitus.it

Keywords: Skill Management, Logic-based Ranking, Match Explanation, Soft constraint

Abstract: The paper presents I.M.P.A.K.T. (Information Management and Processing with the Aid of Knowledge-based Technologies), a semantic-enabled platform for skills and talent management. In spite of the full exploitation of recent advances in semantic technologies, the proposed system only relies on standard SQL queries. Distinguishing features include: the possibility to express both strict requirements and preferences in the requested profile, a logic-based ranking of retrieved candidates and the explanation of rank results. System features are discussed in comparison with similar approaches, *e.g.*, SQLf, and both quantitative and qualitative experimental results are proposed.

1 INTRODUCTION AND MOTIVATION

One of the most critical aspect recruiters have to face with, is the allocation of people to cover specific tasks based on both the *knowledge* the workers should have (in terms of owned skills) and the *knowledge* recruiters themselves should have related to the specific recruiting domain. Nevertheless, two fundamental issues affect the skill management. First of all, skills are often related with each other and then the knowledge of a particular technology or tool may imply to know something else. Furthermore, in a more practical way, due to their daily work, recruiters should be experts in whatever domain or at least in the overall knowledge domain related to the recruiting company.

The above issues call for the adoption of e-recruiting systems allowing to electronically manage the whole recruitment process (or part of it) reducing costs. The efficiency of such tools is determined by the efficacy of their underlying frameworks able to perform the match among employment requirements and job positions. The state of the art of such systems[1] basically relies on keyword-based indexing and use RDBMSs to store the indexed information. Exploiting standard relational database techniques to model an e-recruiting framework, there is the need to completely align the attributes of the offered and requested human resources, in order to perform a match. If requests and offers are described by means of string *keywords*, the only possible match would be identity, resulting in an all-or-nothing outcome. From this point of view, it is noteworthy that non-logical approaches to resource retrieval and matchmaking have serious limitations. On the other hand, pure knowledge-based approaches both often require heavy computational capabilities (with unacceptable response times) and use only pure deductive inference to entail implicit knowledge starting from the explicitly stated one.

A small example will clarify features and main differences of the above mentioned approaches to skill management. Let us suppose a recruiter is looking for an *expert in AJAX programming at least two years experienced*. Three possible candidates are available –**Jack**, **John** and **Al**– whose profiles are reported in what follows:

Jack. *He has a basic oral knowledge of the English language whereas he's doing better with written English. Furthermore, he has an excellent experience in Java programming (5 years until Decem-*

[1] `http://www.attract-hr.com/cm/about`, `http://www.oracle.com/applications/human_resources/irecruit.html`

ber 10, 2008), ...

John. *He is one year experienced in Web Design up today with a good knowledge of XHTML, CSS, DHTML, PHP, ...*

Al. *He is skilled in server-side Web programming, ...*

The candidates will be ranked based on their profile descriptions and on the original request as reported hereafter:
(1) John; **(2)** Al; **(3)** Jack.

Arrangement motivation is that the *AJAX knowledge* implies the *knowledge of XHTML, JavaScript, CSS* (among athers)[2] then, reasonably, skills owned by John are very close to the requested ones. Furthermore, since AJAX programming refers to Web technologies Al's skills seem to be more suitable than Jack's ones. Note that, simply adopting a keyword-based search, it is very difficult to rank the managed profiles; on the contrary a semantic-based approach helps in building a list of results arranged in order of relevance thanks to the exploitation of a domain ontology.

Now, a new question arises for a recruiter: how to explain the ranking results? If she is a domain expert, she can easily write down a report exposing the outcome motivation. But, what happens if she is not a specialist (and this is a very frequent situation in recruitment agencies)? Could the e-recruitement system help her in suggesting the reason why John is better than Al who is in turn better than Jack? Moreover, often recruiters posting job offers could have the need to specify a subset of the whole requirements as mandatory and the remaining part as preferred (likely with a preference degree). Could the system take into account in the matchmaking procedure differences among strict constraints and preferences in a required profile?

Current e-recruitment tools do not cope with this issues. Information about the employment, personal data as well as certifications and competence of candidates are generally modeled exploiting relational databases with customized and structured templates. Nevertheless, in spite of a Data Base Management System is surely suitable for efficient storage and retrieval, SQL does not allow the necessary flexibility to support a complex discovery process as the recruitment one. The `order by` statement and the `min` and `max` aggregation operators are generally used to "trivially" retrieve the *best* tuples *i.e.*, the best candidate for a specific task. Furthermore, currently, a job-seeker describes vacant job positions using traditional methods, such as advertisements and referral systems or on-line recruitment portals[3]. Nevertheless, the recruitment procedure has became more complex and articulate due both to the increase of competitiveness in work environments and to the high number of specific competences (as for example in the ICT field).

Classical DB-based techniques show their limits in managing complex domains. Furthermore, in on-line recruitment portals the search processes can be very time-consuming but often unsatisfactory because underlying frameworks basically rely only on keyword-based approaches where recruiters can express only mandatory requirements (there is no possibilities to select positions according to some negotiable constraints). Finally, such systems usually do not return arranged outcomes (a priori excluding results summarily deemed as not relevant) and above all they do not provide any matchmaking explanation.

In this paper a novel approach to skill management is presented resulting in `I.M.P.A.K.T.` (**I**nformation **M**anagement and **P**rocessing with the **A**id of **K**nowledge-based **T**echnologies), an innovative application based on a hybrid approach. It takes use of an inference engine which performs non-standard reasoning services presented in (Di Noia et al., 2004; Colucci et al., 2005a) over a Knowledge Base (KB) by means of a flexible query language exploiting standard SQL. Noteworthy is the possibility for the recruiter to explicit mandatory requirements as well as preferences during the matchmaking process. The former will be considered as *strict constraints* and the latter as *soft constraints* in the well-known sense of strict partial orders (Kießling, 2002). Moreover, the proposed tool is able to cope also with non-exact matches always providing a result explanation.

Undoubtedly, logic-based approaches increase the efficiency and the flexibility of recruitment. An automatic matchmaking process between candidate profiles and job positions –expressed according to mandatory requirements and preferences provided by the recruiter– allows to discover the most qualified candidates w.r.t. a requested job position in a straightforward way. Hence, the retrieval process is not bound to a simple but quite inefficient string matching. `I.M.P.A.K.T.` exploits a specific *Skills Ontology* modeling experiences, certifications and abilities along with personal and employment information of candidates. It has been designed and implemented using (a subset of) `OWL DL` and, in order to ensure scalability and responsiveness of the system The deductive closure of the ontology has been mapped within an appropriate relational database. Using `I.M.P.A.K.T.`, both job-seekers and candidates refer to the same knowledge domain model.

[2] Note that this is not a trivial IS-A relation.

[3] `http://www.monster.com/, http://www.careerbuilder.com`

Thanks to a friendly GUI, browsing the *Skills Ontology* the job-seeker defines a vacant job position as conjunction of features. Each of them can be treated as a mandatory requirement (strict constraint) or as a preference (soft constraint). Hence, the system translates a user request into a set of SQL queries for retrieving the best candidates to cover a given position.

For each expressed preference, a proper weight function is exploited to compute a score allowing to rank results. If needed, a match explanation (Colucci et al., 2005b) is computed only using SQL queries for each returned candidate. In fact, for each retrieved profile, the system is able to provide information about additional, fulfilled or underspecified features w.r.t. the request. In the same way, it will indicate characteristics conflicting with the request itself.

The remaining of the paper is structured as follows: Section 2 reports on relevant related work, whereas the following Section 3 outlines language and algorithms we adopt in I.M.P.A.K.T.. In Section 4 we describe the system architecture while in Section 5 an illustrative example is used to clarify the approach and the rationale behind it. A preliminary system performance evaluation is presented in Section 6. Finally conclusion closes the paper.

2 STATE OF THE ART

Several frameworks and systems have been conceived and developed in the e-recruitment field. Here we will focus on logic-based approaches.

In order to improve the recruitment and referral process, the US Navy Department make use of STAIRS[4], a system (not exploited for recruitment of non-military people but adopted as an internal tool) allowing to develop referral lists of best qualified candidates according to the number of skills they match w.r.t. a specific mansion. The commercial software supporting STAIRS is RESUMIX[5]. It is an automated staffing tool making use of artificial intelligence techniques. As for I.M.P.A.K.T., it allows to distinguish between *required* and *desired* skills in the query formulation. All the required skills must be matched by the retrieved candidate. To the best of our knowledge, the above two systems are currently the only ones exploiting logic-based formalisms. Many other different solutions for talent management and e-recruitment exist. They mostly aim at improve the recruitment process by exploiting innovative media and tools, but their concrete novelty charge is more limited.

The issue of managing preferences is not new in information retrieval systems. From this viewpoint, two competing approaches have emerged so far, thanks to Chomicki (Chomicki, 2002), even if they do not have been specifically applied to the skill management research field. The first one –defined as *quantitative*– models preferences exploiting utility functions (C. Li, K. C.-C. Chang, Ihab F. Ilyas and S. Song, 2005; P. Bosc and O. Pivert, 1995), whereas the second one –Chomicki named *qualitative*– uses logical formulas (Kießling, 2002; Chomicki, 2002; Hafenrichter and Kießling, 2005). In particular, in the latter approach the Chomicki *skyline* operator is exploited. Furthermore, some relevant theoretical aspects and possible optimizations of Sort Filter Skyline (SFS) algorithm for computing skyline queries have been introduced (Chomicki et al., 2005). Preferences are modeled as *strict partial orders* and interpreted as *soft selection constraints* under the query model defined as *Best Matches Only (BMO)* by Kießling and called *Winnow* by Chomicki. Several implementations of such BMO/Winnow query languages, supporting different preference constructors, have been conceived besides the above mentioned skylines. The use of BMO/Winnow query languages has been also investigated in practical database applications (Kießling et al., 2004; Döring et al., 2008). Various approaches using numerical ranking in combination with either the top-k model (Li et al., 2005; Hristidis et al., 2001; Yu et al., 2005), the Preference SQL (Kießling and Köstler, 2002) or the Preference XPath (Kießling, 2002) have been also devised. Top-k queries ensure an efficient ranking support in RDBMSs letting the system to provide only a subset of query results, according to a user-specified ordering function (which generally aggregates multiple ranking criteria). The algebra is implemented by means of both an efficient query execution model (C. Li, K. C.-C. Chang, Ihab F. Ilyas and S. Song, 2005) and new physical rank-aware operators (Ilyas et al., 2004) where rank relations are processed incrementally. RankSQL (Li et al., 2005) is the first RDBMS that fully integrates a ranking support as a first-class functionality. In other systems, basically the user adopts terms like *ideal*, *good* for expressing her preferences and *high*, *medium* for setting the relevance she assigns to a ranking criterion. SQLf (P. Bosc and O. Pivert, 1995) is another SQL extension to cope with user preferences. It allows to formulate queries on atomic conditions defined by fuzzy sets. Each attribute of a tuple is associated to a satisfaction degree μ in $[0,1]$. Goncalves& Tineo (Goncalves and Tineo, 2006), estimate SQLf to be more expressive and less time-consuming than skyline queries.

[4] http://www.hrojax.navy.mil/forms/selectguide.doc

[5] http://www.cpol.army.mil

Differently from the above mentioned approaches, a relevant aspect of our work is the exploitation of classical relational database systems and languages *i.e.*, SQL, for storing the reference ontology and to perform reasoning tasks. Databases allow users and applications to access both ontologies and other structured data in a seamless way. Das et al. (Das et al., 2004) present a prototype implementation storing OWL-Lite and OWL-DL ontologies in Oracle RDBMSs, which provides a set of SQL operators for ontology-based semantic matching. *Jena 2 Ontology Stores* (Wilkinson et al., 2003), *Sesame* (Broekstra et al., 2002) and *Oracle RDF Store* use a three columns relational table $\langle Subject, Property, Object\rangle$ to memorize RDF triples; in spite of a similar internal structure, those systems present different inference capabilities among them. Other ontology stores –such as *DLDB* (Pan and Heflin, 2003) and *Sesame on PostgreSQL* (Broekstra et al., 2002)– adopt binary tables. They create a table for each ontology class caching the classification hierarchy in the database and providing tables which maintain all the subsumption relationships between primitive concepts. This happens for example in *Instance Store* (***iS***) (Bechhofer et al., 2005), a system for reasoning over OWL KBs specifically adopted in bio and medical-informatics domains. Given a selected ontology, ***iS*** replies to instance retrieval queries using a hybrid reasoner/database approach working on a set of axioms asserting class-instance relationships. A comparison between ***iS*** and the framework we present here points out the former reduces instance retrieval to pure TBox reasoning also coping only on exact matches (*i.e.*, instance retrieval). On the contrary we use an enriched relational schema able to support either potential or partial matches and to provide logic-based result explanations.

3 I.M.P.A.K.T. LANGUAGE AND SERVICES

Basically, `I.M.P.A.K.T.` framework aims to efficiently store and retrieve KB individuals taking into account their *strict* and *soft* constraints and only exploiting SQL queries over a relational database. In what follows we report details and algorithms of the proposed approach assuming the reader be familiar with basics of Description Logics (DLs)(Baader et al., 2002), the reference formalism we adopt here.

With reference to the domain ontology, we define:

- *main categories* and *entry points*. Given a concept name CN, if $CN \sqsubseteq \top$, then it is defined as a *main category* for the reference domain. For what concerns role names, we define an *entry points* $\overline{R}$ as a role whose domain is $\top$ and whose range is a *main category*. Furthermore, for each main class a relevance for the domain is expressed as an integer value L.
- *relevance classes*. For each concept name CN, a set of *relevance classes* either more generic than CN or in some relation with CN can be defined. For example, in the ICT Skill Management domain, the concept $J2EE$ could have as relevance classes $Object\_Oriented\_Programming$ and *Java* among others.

$$\begin{array}{rcl} C, D & \rightarrow & CN \\ & & \exists R \sqcap \forall R.CN \\ & & \leq_n a \\ & & \geq_n a \\ & & =_n a \\ & & C \sqcap D \end{array}$$

Figure 1: Syntax rules for $\mathcal{AL}(\mathrm{D})$ concepts used in `I.M.P.A.K.T.`

The reference domain ontology is modeled as an $\mathcal{AL}(\mathrm{D})$ one and the following axioms are allowed:

$$\begin{array}{rcl} CN_0 & \sqsubseteq & CN_1 \sqcap \ldots CN_m \\ CN_0 & \equiv & CN_1 \sqcap \ldots CN_m \\ CN_1 & \sqsubseteq & \neg CN_2 \\ \exists \overline{R}.(CN_1 \sqcap \ldots \sqcap CN_k) & \sqsubseteq & \forall \overline{S}.C \end{array}$$

where $\overline{R}$ and $\overline{S}$ are *entry points* whereas C is an $\mathcal{AL}(\mathrm{D})$ concept defined as in Figure 1[6].

All the requests submitted to the system as well as the description of CVs can be represented as DL formulas possibly mapped in standard SQL queries. In such queries, `WHERE` clause is used for select relevant tuples and `GROUP BY/ORDER BY` operators to compute the final score. Notice that we do not use a specific preference language as in (Kießling, 2002; Chomicki, 2002; P. Bosc and O. Pivert, 1995) but we only exploit a set of *ad-hoc* SQL queries built supposing the following DL template for expressing user requirements (soft and strict constraints) and a candidate profile. (see Section 3.2 for further details):

$$\exists \overline{R}_1.C_1 \sqcap \ldots \sqcap \exists \overline{R}_n.C_n \qquad (1)$$

where $\overline{R}_1, \ldots, \overline{R}_n$ are *entry points* and $C_1, \ldots, C_n$ are $\mathcal{AL}(\mathrm{D})$ concepts defined w.r.t. the syntax reported in Figure 1.

[6] Observe that in the current *Skills Ontology* we do not use disjunction axioms. In fact, in the recruitment domain it is quite rare to assert that *if you know A then you do not know B*.

Similarly to the approach adopted in ***iS***, we use role-free ABoxes, *i.e.*, we reduce reasoning on the ABox to reasoning on the TBox (Bechhofer et al., 2005). Furthermore, individuals in the knowledge base are normalized w.r.t. a Concept-Centered Normal Form (CCNF) (Baader et al., 2002).

3.1 KB relational schema

In order to store both the classified TBox and the normalized ABox we have modeled a proper relational schema. It is also optimized for individual instances retrieval and ranking (in case of strict and soft matches) and for providing match explanations. The E-R model of the reference database is sketched in Figure 2 where the `profile` table maintains the so called *structured info*, exploited to take into account non ontological information referring to a specific CV description.

In Figure 2(a) tables referring to the TBox are reported. The `concept` table stores primitive and defined concepts along with data and object properties. Actually, also descriptions in the form $\forall R.\forall S \ldots \forall T.C$, being C a primitive concept name, are stored in the `concept` table itself. For each defined concept C in `concept`, the `desconcepts` table will store the atomic elements belonging to the C CCNF. `parent` and `child` tables will respectively store parents and children of a given concept and, finally, the `disjoint` table maintain disjunction sets[7]. Each `propertyR` table ($R = 1,\ldots,N$) refers to a specific *entry point* among the N ones defined in the domain ontology. Each of them will store features of normalized individuals referred to a specified ontology main category.

Simply recalling the introductory example, we can describe the *Jack*'s profile according to the template in (1) as feature conjunction:
$\exists hasKnowledge.(Java \sqcap \forall skillType.programming \sqcap =_5 years \sqcap =_{2008-12-10} lastdate) \sqcap \exists knowslanguage.(English \sqcap =_1 verbalLevel \sqcap =_2 writingLevel)$ where *hasKnowledge* and *knowslanguage* are entry points (see Section 3.2 for details). It will be split and stored in two tables named `hasKnowledge` and `knowslanguage`. Finally, according to the model, `value` and `lastdate` attributes respectively represent a numeric data property (*years* value in the above example) and the last update of that value (*lastdate* value in the previous example).

In Figure 2(b) are modeled the auxiliary tables (not fully represented here due to the lack of space) needed to store intermediate match results with their relative score.

[7] As stated before, in the Skills domain this table was empty.

Since the ontology contains classes and object properties (*i.e.*, qualitative information), and datatype properties (*i.e.*, quantitative information), in order to rank final results w.r.t. an initial request we have to manipulate in two different ways qualitative and quantitative data. In particular, to assign a score to each individual data property a, specifications in the form $\geq_n a$ are managed by the function in Figure 3(a) whereas properties in the form $=_n a$ will be managed by the one in Figure 3(b) respectively[8]. n is the value the user imposes for a given data property a whereas, in both functions, we indicate with $n_{m\%}$ the threshold value for accepting the individual features containing a. $n_{m\%}$ is a cut-coefficient calculated according to the following formula:

$$n_{m\%} = n \pm [(Max - min)/100] * m \qquad (2)$$

where *Max* and *min* respectively are the maximum and the minimum value stored in `value` attribute for the data property a in the related `propertyR` table.

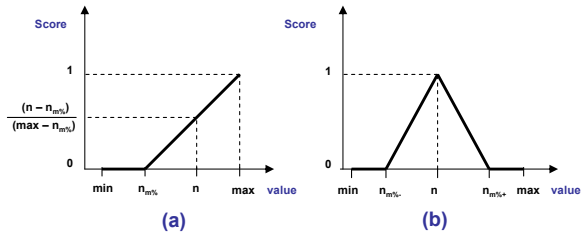

Figure 3: Score functions

3.2 Match classes and results explanation

In order to cope with soft and strict constraints `I.M.P.A.K.T.` performs a two step matchmaking process. It starts computing a Strict Match and, in case, it exploits obtained results as initial profile set for computing the following Soft Matches.

A Strict Match is similar to an *Exact* one (Di Noia et al., 2004), whereas a Soft Match is a revised version of a *Potential* one (see also (Di Noia et al., 2004)) which takes into account information related to datatype properties. Given a request containing a soft constraint on a datatype property in the form $\{ \leq_n a, \geq_n a, =_n a \}$, `I.M.P.A.K.T.` will also retrieve resources whose value for the property a is in the range $n \pm n_{m\%}$. This is not allowed by a Potential Match because the resources themselves are seen

[8] For query features containing concrete domains in the form $\leq_n a$, we will use a scoring function which is symmetric w.r.t. the one in Figure 3(a).

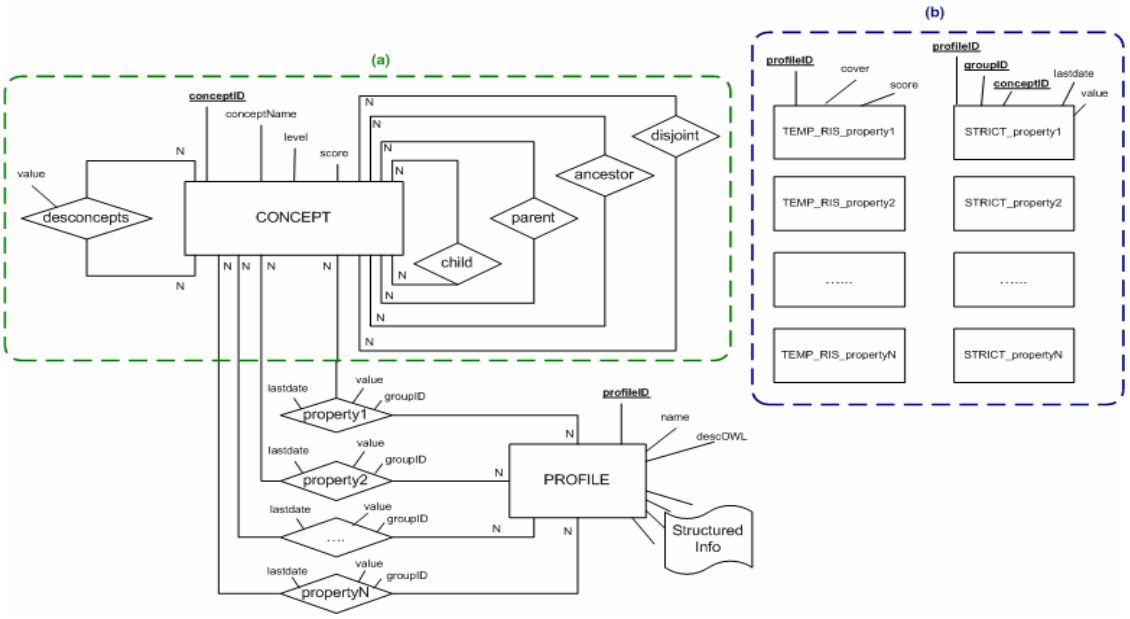

Figure 2: KB schema

as carrying out conflicting features w.r.t. the user request.

In what follows the matchmaking process performed by the proposed system will be detailed. First of all, it separates soft features fp from strict ones fs within the request and it normalizes both fp and fs in their corresponding $CCNF(fp) = \exists\overline{R}.C$ and $CCNF(fs) = \exists\overline{S}.C$ respectively. For each entry point $\overline{R}$ in soft features the corresponding set $\mathcal{FP}_{\overline{R}} = \{\exists\overline{R}.C\}$ is identified. Similarly, for strict features, the sets $\mathcal{FS}_{\overline{S}}$ are defined. If needed, soft and strict features can be grouped to build the two sets $\mathcal{FP} = \{\mathcal{FP}_{\overline{R}}\}$ and $\mathcal{FS} = \{\mathcal{FS}_{\overline{S}}\}$.

After this preliminary step, for each element $\exists\overline{R}.C \in \mathcal{FP}$ a single query Q or a set of queries Q^a are built according to following schema:

- if no of $\{ \leq_n a, \geq_n a, =_n a \}$ elements occur in C[9] then a single query Q is built. W.r.t. the specific *entry point* $\overline{R}$, it will retrieve the profile features containing at least one among syntactic element occurring in C;
- otherwise a set of queries $Q^a = \{Q_n, Q_{NULL}, Q_{n_{m\%}}\}$ is built. W.r.t. the specific *entry point* $\overline{R}$, Q^a will retrieve the profile features containing at least one among syntactic elements occurring in C also satisfying –either fully or partially– the data property a according to the threshold value $n_{m\%}$ and the scoring functions in Figure 3. The final result is the `UNION` of all the tuples retrieved by queries in Q^a.

In the latter case Q_n, Q_{NULL} and $Q_{n_{m\%}}$ represent respectively:
- Q_n retrieves only tuples containing, for the *entry point* $\overline{R}$, both at least one syntactic element occurring in C and the satisfied data property a. In this case, the structure of Q_n changes according to requested constraint ($\leq_n a$, $\geq_n a$ or $=_n a$) as well as the proper scoring function in Figure 3 has to be used in order to opportunely weight each feature;
- Q_{NULL} retrieves only tuples containing, for the *entry point* $\overline{R}$, both at least one syntactic element occurring in C and not containing the data property a; *i.e.*, it returns also tuples where a, corresponding to `value` attribute of `propertyR` table, is `NULL`;
- $Q_{n_{m\%}}$, retrieves only tuples containing, for the *entry point* $\overline{R}$, both at least one syntactic element occurring in C and a data property value for a within the interval $[n_{m\%}, \ldots, n]$. Hence, $n_{m\%}$ can be seen as threshold value for accepting profile features[10]. About the syntactic structure of $Q_{n_{m\%}}$, the same above considerations for Q_n can be repeated here.

The above queries, to some extent, grant the "Open-World Assumption" upon a database which is notoriously based on the well-known "Close-World

[9] Recall that at this stage C has been translated in its normal form w.r.t. the reference ontology

[10] It is similar to the $\lambda - cut$ operator of SQLf language (P. Bosc and O. Pivert, 1995).

Assumption". The queries Q and Q^a are used in the Soft Match step of the retrieval process.

At the beginning of the retrieval process, the Strict Match algorithm searches for profiles fully satisfying all the formulas in $\mathcal{FS}$. Furthermore, starting from tuples selected in this phase, the Soft Match algorithm, by means of Q and Q^a, will extract profile features either fully or partially satisfying a single formula in $\mathcal{FP}$.

Obviously, the same profile could satisfy more than one formula in $\mathcal{FP}$. Candidate profiles retrieved by means of a Strict Match have a 100% covering level of the user request, whereas a measure has to be provided for ranking profiles retrieved by means of a Soft Match. To this aim, each tuple of a `propertyR` table corresponding to one element of C is opportunely weighted for a specified $\overline{R}$. Hence, for example, the profile feature $\exists hasKnowledge.(Java \sqcap =_5 years \sqcap \forall skillType.programming \sqcap =_{2008-12-10}\ lastdate)$ will be stored in `hasKnowledge` table filling 4 tuples.

In particular, by means of Q^a, the system assigns a $\mu \in [0,1]$ value only to elements (tuples) in the form $=_n a$ according to the scoring function related to user requested constraint for a, by means of Q and Q^a, it will assign 1 to the other elements in C. Once retrieved, these "weighted tuples" are so stored in proper tables named `propertyR_i` ($i = 1,\ldots,M$ for a query where $|\mathcal{FP}_{\overline{R}}| = M$) created at runtime.

In other words, the `propertyR_i` table will store tuples (*i.e.*, features elements) satisfying the i-th soft requirement of the user request belonging to $\mathcal{FP}_{\overline{R}}$ and having `propertyR` as entry point. The `propertyR_i` schema enriches the `propertyR` schema by means of two attributes, namely **`score`** and **`cover`**. The former is the score related to each tuple (computed as described above), the latter marks each feature piece as fully satisfactory or not. The `cover` attribute can only assume the following values: **(a)** `cover` $= 1$ in case the tuples have been retrieved by Q_n, Q_{NULL} or Q queries; **(b)** `cover` $= 0.5$ in case the tuples have been retrieved by a $Q_{n_{m\%}}$ query and they represent a data property a. The overall *score* and *cover* values of a retrieved profile are calculated combining `score` and `cover` values of each tuple, as described in what follows.

The whole Soft Match process can be summarized in the following steps. Here, we introduce L_i as the relevance level the user assigns to the i-th soft feature of a request belonging to $\mathcal{FP}_{\overline{R}}$.

step I. For each $\exists \overline{R}.C \in \mathcal{FP}$ the "weighted tuples" of `propertyR_i` tables are determined and, for each retrieved feature, the score value s_i is computed by adding the `score` value of each tuple. In the same way, will be calculated the cover value c_i;

step II. for each profile and for each `propertyR_i` table, only features with the maximum s_i value are selected;

step III. the profile features belonging to the same level L_i are aggregated among them. For each retrieved profile, the system provides a global score s_{L_i} adding the scores s_i of features belonging to a given L_i;

step IV. the retrieved profiles are ranked acording to a linear combination of scores obtained at the previous step. The following formula is exploited:

$$score = s_{L_1} + \sum_{i=1}^{N-1} w_i * s_{L_{i+1}} \tag{3}$$

where w_i are heuristic coefficients belonging to the $(0,1)$ interval and N is the number of levels defined for the domain ontology (L_1 is the most relevant one).

PropertyR_i tables are also exploited for score explanation and to classify features of each retrieved profile. They can be divided into:

Fulfilled. That is fully satisfying the corresponding request features.

Conflicting. That is containing a data property value slightly conflicting with the corresponding request feature[11].

Additional. That is either more specific than required ones or belonging to the first relevance level but not exposed in the user request;

Underspecified. That is absent in the profile –and then unknown for the system– but required by the user.

Observe that features with a non-integer value for c_i are conflicting by definition.

The request refinement process follows the match calculation one. To this aim the score explanation is surely useful. In fact, by analyzing fulfilled and conflicting features, a job-seeker can decide to negotiate either features themselves or data property values, as well as she can also enrich the original request by adding new features taken from the additional ones.

About the refinement process, the following result ensues. Consider a request Q, and let us suppose Q allows to retrieve profiles $p_1, p_2, \ldots, p_n$ by means of the Soft Match –exactly in the reported order. In general, $p_i \prec_Q p_j$ denotes that profile p_i is ranked by Q better than p_j. Hence, in the previous case, p_1 is ranked better than p_2 and so on. Now, if Q_{fp} is obtained by adding to Q another feature fp as *negotiable* feature, we can divide the previous n profiles into the ones which fulfill fp, the ones which do not it and the ones for which data property a is unknown.

[11] The possibility to identify and extract components in a slight disagreement with the request represents a significant added value w.r.t. approaches based on Fuzzy Logic.

If p_i, p_j both belong to the the same class –*i.e.*, if either both fulfill fp or both do not fulfill it or both do not specify a– then $p_i \prec_Q p_j$ *iff* $p_i \prec_{Q_{fp}} p_j$. This can be proved by considering the rank calculation procedure, but we omit the details of the demonstration due to the lack of space. Thanks to the above property, the user can refine Q as Q_{fp} knowing that, when browsing results of Q, the relative order among profiles that agree on a is the same she would find among the ones deriving from Q_{fp}.

4 IMPLEMENTATION DETAILS

I.M.P.A.K.T. is a multi-user, client-server application implementing a scalable and modular architecture. It is developed in Java 1.6 (exploiting J2EE and JavaBeans technologies) and it uses JDBC and Jena[12] as main foreign APIs. Furthermore, it embeds Pellet[13] as reasoner engine to classify more "complex" ontologies. If the reference ontology does not present implicit axioms, it is possible to disable the reasoner services so improving performances.

The I.M.P.A.K.T. prototype is built upon the open source database system PostgreSQL 8.3[14] and uses: **(1)** auxiliary tables and views to store the intermediate results with the related scores and **(2)** stored procedures and b-tree indexes on proper attributes to reduce the retrieval time. Moreover, the compliance with the standard SQL makes I.M.P.A.K.T. available for a broad variety of platforms.

In the current implementation, all the features in the user request are considered as negotiable constraints by default. The exploited reference **Skills Ontology** basically models ICT domain. It owns seven entry points (*hasDegree*, *hasLevel*, *hasIndustry*, *hasJobTitle*, *hasKnowledge*, *knowsLanguage* and *hasComplementarySkill*), six data properties (*years* (meaning *years of experience*), *lastdate*, *mark*, *verbalLevel*, *writingLevel* and *readingLevel*), one object property (*skillType*) and nearly 3500 classes.

The skill reference template follows the above structure (Section 3) –reported in Table 1. Notice that the data property *lastdate* is mandatory only when the data property *years* is already defined in a profile feature. Moreover, data properties in the form $\{ \leq_n a, \geq_n a, =_n a \}$ are usable only in the retrieval phase whereas in the profile storing phase only $=_n a$ is allowed. Finally, only the *knowsLanguage* entry point –referred to the knowledge of foreign languages– follows an autonomous match query structure w.r.t. the others ones. In fact the three possible data properties for expressing oral, reading and writing language knowledge have to be tied to the language itself. In this case, each data property is an attribute whose domain is the *Language* main category and whose range is the set { 1,2,3 } where 3 represents an excellent knowledge and 1 a basic one.

Table 1: Skill Reference Template

Entry point	Main Category	Feature Description (DL syntax)
hasDegree	Degree	$\exists hasDegree.(Degree \sqcap (\geq,\leq,=)_n mark)$
hasLevel	Level	$\exists hasLevel.(Level \sqcap (\geq,\leq,=)_n mark)$
hasJobTitle	JobTitle	$\exists hasJobTitle.(jobTitle \sqcap (\geq,\leq,=)_n years \sqcap =_n lastdate)$
hasIndustry	Industry	$\exists hasIndustry.(Industry \sqcap (\geq,\leq,=)_n years \sqcap =_n lastdate)$
hasKnowledge	Knowledge	$\exists hasKnowledge.(Knowledge \sqcap \forall skillType.Type \sqcap (\geq,\leq,=)_n years \sqcap =_n lastdate)$
hasComplementarySkill	Ability	$\exists hasComplementarySkill.(Ability \sqcap (\geq,\leq,=)_n years \sqcap \exists =_n lastdate)$
knowsLanguage	Language	$\exists knowsLanguage.(Language \sqcap =_n readingLevel \sqcap =_n verbalLevel \sqcap =_n writingLevel)$

Thanks to *lastdate* data property, we can say that "John Doe" was 4 years experienced of Java but this happened 4 years ago and at the present time he knows DBMS by 2 years. In other words, our system can handle a temporal dimension of knowledge and experience considering time intervals as for example "now", "short time ago", "long time ago" to improve the score computation process. Actually I.M.P.A.K.T. uses a step function to weigh the effective value of the experience according to the formula $n_t = w_t * n$. A trivial example will clarify this feature. Assertions as "now" or "one year ago" have both $w_t = 1$, hence the value of the related experience is the same. On the contrary, a time interval represented as "two years ago" has $w_t = 0.85$, *i.e.*, the concrete value of experience is decreased w.r.t. the previous cases. When a temporal dimension is specified in the stored profile, I.M.P.A.K.T. retrieves the best candidates and calculates the related score according to the experience value n_t and not trivially taking into account n.

The adopted ontology has three relevance levels. The following rules ensue: the entry point *hasKnowledge* belongs to the L_1 level, the entry points *hasComplementarySkill*, *hasJobTitle*, *hasIndustry* belong to the L_2 level and the entry points *hasLevel*, *hasDegree*, *knowsLanguage* belong to the L_3 level. Obviously L_1 is the most important level and L_3 the least significant one. In the current implementation, the formula (2) fixes $m = 20$, *i.e.*, I.M.P.A.K.T. considers as possible a deviation of 20% w.r.t. n for *years* or *mark* features requested by the user. Moreover, in the formula (3): $N = 3$, $w_2 = 0.75$ and $w_3 = 0.45$. These values

[12] http://jena.sourceforge.net/

[13] http://pellet.owldl.org/

[14] http://www.postgresql.org

have been experienced in several tests involving different specialist users engaged in a proactive tuning process of the software.

5 I.M.P.A.K.T. GUI

"I'm looking for a candidate having an Engineering Degree (preferably in Computer Science with a final mark equal or higher than 103 (out of 110)). A doctoral Degree is welcome. S/He must have experience as DBA, s/he must know the Object-Oriented programming paradigm and techniques and it is strictly required s/he has a good oral knowledge of the English language (a good familiarity with the written English could be great). Furthermore s/he should be at least six years experienced in Java and s/he should have a general knowledge about C++ and DBMSs. Finally, the candidate should possibly have team working capabilities".

The previous one could be a typical request of a job-seeker. It will be submitted to the I.M.P.A.K.T. by means of the provided Graphical User Interface (GUI). The above requested features can be summarized as: **(1) strict ones:** 1.1) Engineering degree; 1.2) DBA experience; 1.3) OO programming; 1.4) good oral English; **(2) preferences:** 2.1) Computer science degree and $mark \geq 103$; 2.2) doctoral degree; 2.3) Java programming and $experience \geq 6 years$; 2.4) C++ programming; 2.5) DBMSs; 2.6) team working capabilities; 2.7) good written English.

They are shown in the (e) panel of Figure 4 whereas deriving ranked results are reported in Figure 5. The GUI for browsing the ontology and to compose the query is also shown in Figure 4. Observe that the interface for defining/updating the candidate profile is exactly the same.

With reference to Figure 4, (a), (b) and (d) panels allow the job-seeker to compose her semantic-based request. In fact, in the (a) menu all the entry points are listed, the (b) panel allows to search for ontology concepts according to their meaning, whereas the (d) part enables the user to explore both taxonomy and properties of a selected concept. The related panel is dynamically filled. The (e) panel in Figure 4 enumerates all the composing features. For each of them, the I.M.P.A.K.T. GUI allows: **(1)** to define the "kind" of feature (strict or negotiable); **(2)** to delete the whole feature; **(3)** to complete the description showing all the elements (concepts, object properties and data properties) that could be added to the selected feature; **(4)** to edit each feature piece as well as existing data properties. Finally, the (c) panel enables searches as for example *"I'm searching a candidate like John Doe"*. In this case, the job-seeker fills first and/or last name field of the known candidate and the system will consider her/him profile as starting request whose features are set as negotiable constraints by default. The user can view the query –automatically generated– and furthermore s/he can edit it before starting a new search.

In Figure 5 the results GUI is shown. Part (a) presents the ranked list of candidates returned by I.M.P.A.K.T. with the related score. For each of them, the job-seeker can ask for: **(1)** viewing the CV; **(2)** analyzing the employment and personal information and **(3)** executing the match explanation procedure. Match explanation outcomes are presented in the (c) panel, whereas in the (b) panel an overview of the request is shown (differentiating strict constraints from preferences). Observe that the system assigns a numeric identifier –namely $ID feature$– to each query feature. It will be used in the explanation phase to create an unambiguous relationship among the features in the panel (b) and the ones in the panel (c).

Let us exploit the second ranked result to explain the system behavior. It corresponds to "Mario Rossi" –as shown in Figure 5– which totals 77% w.r.t. the above formulated request. Why not a 100% score? Notice that, at the present time, "Mario" has the following programming competences:

1) $\exists hasKnowledge.(Java \sqcap =_5 years \sqcap =_{2008-07-21} lastdate)$;
2) $\exists hasKnowledge.(VisualBasic \sqcap =_5 years \sqcap =_{2008-07-21} lastdate)$;
3) $\exists hasKnowledge.(C++ \sqcap =_4 years \sqcap =_{2008-07-21} lastdate)$.

Hence, if one considers the requested feature $\exists hasKnowledge.(Java \sqcap \geq_6 years)(ID features = 9)$, the I.M.P.A.K.T. explanation returns the following:

a) $\exists hasKnowledge.(Java \sqcap =_5 years \sqcap =_{2008-07-21} lastdate)$
b) $\exists hasKnowledge.(OOprogramming \sqcap =_5 years \sqcap =_{2008-07-21} lastdate)$

as fulfilled features (they correspond to desired candidate characteristics), but they are also interpreted as conflicting ones because the experience years not fully satisfy the request.

In particular, the $\exists hasKnowledge.(OOprogramming \sqcap =_5 years \sqcap =_{2008-07-21} lastdate)$ is considered as a fulfilled feature thanks to the "Mario's" competence about *VB*. Finally, besides the conflicting features, "Mario Rossi" also has some underspecified ones and then, he cannot fully satisfy the job-seeker request. S/he can enrich her/his original query selecting some additional features among them displayed in the related panel. The checked ones are automatically added to the original query panel (part(e) in Figure 4) and they can be further enriched or modified as the other features.

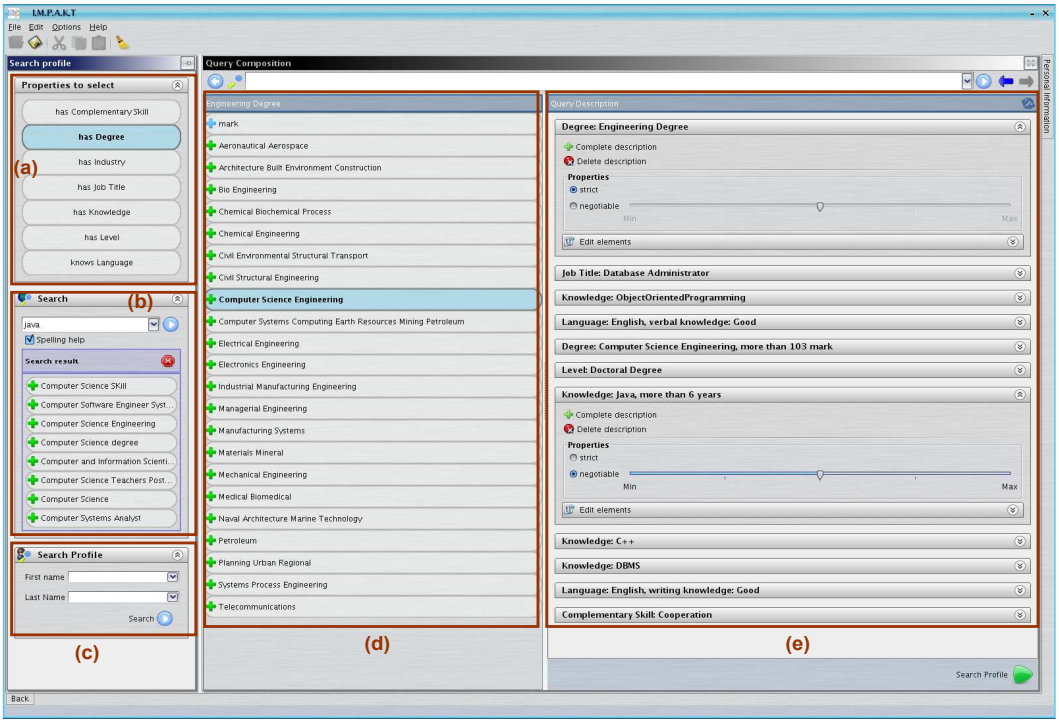

Figure 4: Query composition GUI

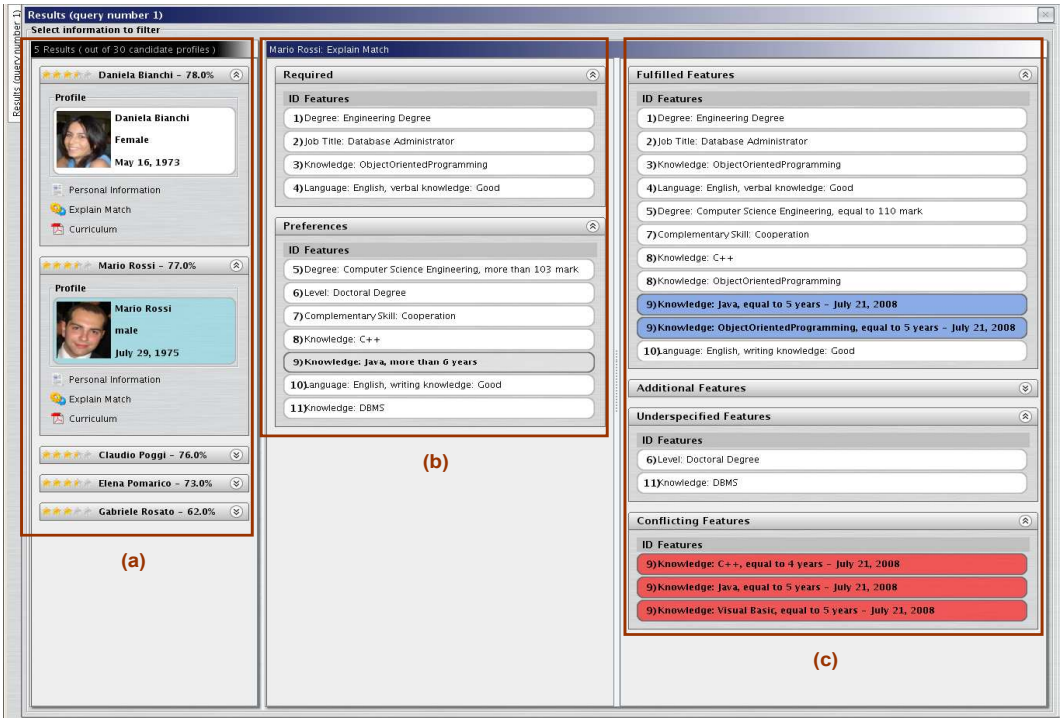

Figure 5: Results and score explanation GUI

6 PERFORMANCE EVALUATION

I.M.P.A.K.T. has been ran upon an Intel Dual Core server, equipped with a 2.6 GHz processor and a 3 GB RAM. The application has been tested by means of three kinds of query with a different expressiveness in order to obtain an omni-comprehensive evaluation of its capabilities. The first query –named $Q1$– is only composed by preferences rather generic w.r.t. the ontology taxonomy (see Table 2 for details), the $Q2$ query is composed of strict requirements and preferences (it is reported on the (e) panel of Figure 4) and finally $Q3$ is only composed by preferences more specific than $Q1$ features (see Table 3 for further specifications).

Table 2: Query Q1 - generic soft constraints

Category	Negotiable Feature (DL syntax)
Degree	$\exists hasDegree.(Managerial\_Engineering) \sqcap \exists hasDegree.(Computer\_scienceengineering)$
JobTitle	$\exists hasJobTitle.(Computer\_Software\_Engineer \sqcap \geq_3 years)$
Knowledge	$\exists hasKnowledge.(OOprogramming \sqcap \geq_4 years)$
Complementary Skill	$\exists hasComplementarySkill.(Cooperation)$
Language	$\exists knowsLanguage.(English \sqcap =_3 verbalLevel)$

Table 3: Query Q3 - specific soft constraints

Category	Negotiable Feature (DL syntax)
Degree	$\exists hasDegree.(Managerial\_Engineering \sqcap \geq_{104} mark)$
Level	$\exists hasLevel.(Doctorate)$
JobTitle	$\exists hasJobTitle.(Project\_Manager \sqcap \geq_3 years)$
Industry	$\exists hasIndustry.(IT\_Software\_Development \sqcap \geq_2 years)$
Knowledge	$\exists hasKnowledge.(Java \sqcap \exists years. \geq_3) \sqcap \exists hasKnowledge.(UML) \sqcap \exists hasKnowledge.(DBMS) \sqcap \exists hasKnowledge.(pl/SQL)$
Complementary Skill	$\exists hasComplementarySkill.(Cooperation) \sqcap \exists hasComplementarySkill.(Complex\_Problem\_Solving)$
Language	$\exists knowsLanguage.(English \sqcap \geq_2 verbalLevel \sqcap \geq_2 readingLevel \sqcap \geq_2 writingLevel)$

Moreover, we have considered datasets automatically generated with an increasing size from 300 to 2100 profiles, where each profile has nearly 10 features for each main category. Hence, property_R tables –actually used for implementing both Strict and Soft Match– store tuples ten times grater than the one in the profile table. For each dataset, the Figure 6 shows the retrieval time obtained by averging 50 subsequent executions.

First results have been calculated without considering delays caused by client-server communication and connection overhead, as they are quite not influent. Note that, for the $Q2$ query, the retrieved profiles are reported in Figure 5. They are independent datasets as the strict matching procedure always returns the same five results. Finally, it has to be observed that for queries only expressing preferences, the retrieval time quite linearly increases with the data size, *i.e.*, time nearly doubles as data size grows-up 6-7 times; whereas if the results number does not change (as in the $Q2$ case), the retrieval time increases more slowly.

Figure 6: I.M.P.A.K.T. performances

7 CONCLUSION AND FUTURE WORK

The paper presents I.M.P.A.K.T., a novel logic-based tool for efficiently managing technical competences and experiences of candidates in the e-recruitment field. The features of a required job position can be described as mandatory requirements and preferences. Exploiting only SQL queries, the system returns ranked profiles of candidates along with an explanation of the provided score. Preliminary performance evaluation conducted on several datasets shows a satisfiable behavior. Future work aims at enabling the user to optimize the selection of requested preferences by properly weighting the relevance of each of them and at testing other strategies for score calculation refinement in the match process.

REFERENCES

Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., and Patel-Schneider, P. (2002). *The Description Logic Handbook*.

Bechhofer, S., Horrocks, I., and Turi, D. (2005). The OWL Instance Store: System Description. In *The 20th International Conference on Automated Deduction(CADE '05)*, pages 177–181.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *The First International Semantic Web Conference (ISWC '02)*, pages 54–68.

C. Li, K. C.-C. Chang, Ihab F. Ilyas and S. Song (2005). RankSQL: Query Algebra and Optimization for Relational Top-k Queries. pages 131–142. ACM.

Chomicki, J. (2002). Querying with Intrinsic Preferences. In *Advances in Database Technology - EDBT 2002*, pages 34–51.

Chomicki, J., Godfrey, P., Gryz, J., and Liang, D. (2005). Skyline with Presorting: Theory and Optimizations. In *IIPWM'05*, pages 595–604.

Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F. M., and Mongiello, M. (2005a). Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. 4(4):345–361.

Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F. M., and Ragone, A. (2005b). Knowledge elicitation for query refinement in a semantic-enabled e-marketplace. In *ICEC 05 ACM Press*, pages 685–691. ACM.

Das, S., Chong, E. I., Eadon, G., and Srinivasan, J. (2004). Supporting ontology-based semantic matching in RDBMS. In *The 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 1054–1065. VLDB Endowment.

Di Noia, T., Di Sciascio, E., Donini, F. M., and Mongiello, M. (2004). A System for Principled Matchmaking in an Electronic Marketplace. 8(4):9–37.

Döring, S., Preisinger, T., and Endres, M. (2008). Advanced preference query processing for e-commerce. In *The 2008 ACM symposium on Applied computing (SAC '08)*, pages 1457–1462, New York, NY, USA.

Goncalves, M. and Tineo, L. (2006). SQLf vs. Skyline - Expressivity and Performance. In *IEEE International Conference on Fuzzy Systems*, pages 2062–2067.

Hafenrichter, B. and Kießling, W. (2005). Optimization of relational preference queries. In *ADC '05*, pages 175–184, Darlinghurst, Australia. Australian Computer Society, Inc.

Hristidis, V., Koudas, N., and Papakonstantinou, Y. (2001). PREFER: A system for the efficient execution of multi-parametric ranked queries. pages 259–270, New York, NY, USA. ACM.

Ilyas, I. F., Aref, W. G., and Elmagarmid, A. K. (2004). Supporting top-k join queries in relational databases. 13(3):207–221.

Kießling, W. (2002). Foundations of preferences in database systems. pages 311–322.

Kießling, W., Fischer, S., and Döring, S. (2004). COSIMA B2B - Sales automation for E-procurement. In *The IEEE International Conference on E-Commerce Technology (CEC '04)*, pages 59–68, Washington, DC, USA. IEEE Computer Society.

Kießling, W. and Köstler, G. (2002). Preference SQL - design, implementation, experiences. pages 990–1001.

Li, C., Soliman, M. A., Chang, K. C.-C., and Ilyas, I. F. (2005). RankSQL: supporting ranking queries in relational database management systems. pages 1342–1345. VLDB Endowment.

P. Bosc and O. Pivert (1995). SQLf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 3(1):1–17.

Pan, Z. and Heflin, J. (2003). DLDB: Extending Relational Databases to Support Semantic Web Queries. In *The First International Workshop on Practical and Scalable Semantic Systems (PSSS1)*, volume 89, pages 109–113. CEUR-WS.org.

Wilkinson, K., Sayers, C., Kuno, H. A., and Reynolds, D. (2003). Efficient RDF Storage and Retrieval in Jena2. In *The first International Workshop on Semantic Web and Databases (SWDB'03)*, pages 131–150.

Yu, H., Hwang, S.-W., and Chang, K. C.-C. (2005). RankFP: A Framework for Supporting Rank Formulation and Processing. pages 514–515, Washington, DC, USA.