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] and Francesco M. Donini[4]
[1]*Politecnico of Bari, via Re David 200, I - 70125 Bari, Italy*
[2]*University of Bari, Orabona 4, I - 70125 Bari, Italy*
[3]*DOOM s.r.l, Paganini, 7 - 75100 Matera, Italy*
[4]*University of Tuscia, S. Carlo, 32 - 01100 Viterbo, Italy*

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.

1 INTRODUCTION

We present 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 for skill management. It uses an inference engine which performs non-standard reasoning services (Di Noia et al., 2004; Colucci et al., 2005) over a Knowledge Base (KB) by means of a flexible query language based on standard SQL. Noteworthy is the possibility for the recruiter to explicit mandatory requirements as well as preferences during the matching 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, providing a useful result explanation. 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[1] and, in order to ensure scalability and responsiveness of the system, the deductive closure of the ontology has been mapped within an appropriate relational schema.

2 LANGUAGE AND SERVICES

Our 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), the reference formalism we adopt here. W.r.t. 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 point* $\overline{R}$ as a role whose domain is $\top$ and whose range is a *main category*. Furthermore, for each main class the 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. The reference domain ontology is modeled as an $\mathcal{AL}(\mathrm{D})$ one and the following axioms are allowed:

[1]http://www.w3.org/TR/owl-guide/

Tinelli E., Cascone A., Ruta M., Di Noia T., Di Sciascio E. and Donini F. (2009).
I.M.P.A.K.T.: An Innovative Semantic-based Skill Management System Exploiting Standard SQL.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Artificial Intelligence and Decision Support Systems*, pages 224-229
DOI: 10.5220/0002008802240229

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

where $\overline{R}$ and $\overline{S}$ are *entry points* whereas C is an $\mathcal{AL}(\mathrm{D})$ concept defined as[2]:

$$
C,D \rightarrow \left| \begin{array}{c} CN \\ \exists R \sqcap \forall R.CN \\ \leq_n a \\ \geq_n a \\ =_n a \\ C \sqcap D \end{array} \right.
$$

All the requests submitted to the system as well as the description of *curricula* can be represented as DL formulas to be 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:

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

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 above. Similarly to the approach adopted in Instance Store (***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). 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 1 where the `profile` table maintains the so called *structured info*, exploited to take into account non ontological information referring to a specific curriculum vitae (CV) description.

In Figure 1(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

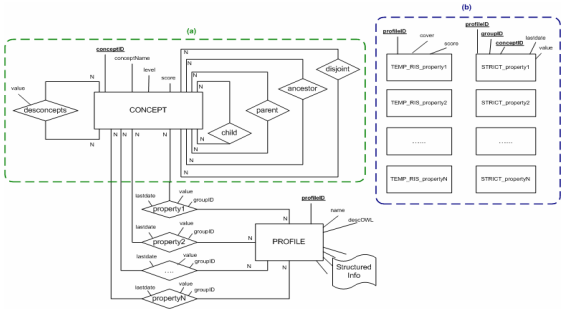

Figure 1: KB schema.

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[3]. 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. In Figure 1(b) the auxiliary tables are modeled (not fully represented here due to the lack of space) needed to store intermediate match results with their relative score. 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. To assign a score to each individual data property a, specifications in the form $\geq_n a$ are managed by the function in Figure 2(a) whereas properties in the form $=_n a$ will be managed by the one in Figure 2(b) respectively[4]. 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* are the maximum and the minimum value stored in `value` attribute for the data property a in the related `propertyR` table.

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.,

[2] 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*.

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

[4] 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 2(a).

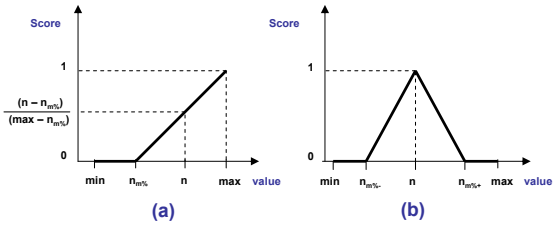

Figure 2: Score functions.

2004), whereas a Soft Match is a revised version of a *Potential* one (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 as the resources themselves are seen as carrying out conflicting features w.r.t. the user request. The match process is hereafter 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: a) if no elements of $\{ \leq_n a, \geq_n a, =_n a \}$ occur in C[5] then a single query Q is built. W.r.t. the specific *entry point* $\overline{R}$, the match process will retrieve the profile features containing at least one among syntactic element occurring in C; b) 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 2. 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 2 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[6]. The same above considerations outlined for Q_n can be applied to the syntactic structure of $Q_{n_{m\%}}$. The above queries, to some extent, grant the "Open-World Assumption" upon a database which is notoriously based on the well-known "Close-World 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 weighted with a specified $\overline{R}$. Hence, for example, the profile feature $\exists hasKnowledge.(Java \sqcap =_5 years \sqcap =_{2008-12-10} lastdate \sqcap \forall skillType.programming)$ will be stored in `hasKnowledge` table filling 4 tuples. 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, using 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)**

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

[6] It is similar to the λ-cut operator of SQLf language (P. Bosc and O. Pivert, 1995).

$\texttt{cover} = 1$ in case the tuples have been retrieved by Q_n, Q_{NULL} or Q queries; **(b)** $\texttt{cover} = 0.5$ 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. 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 the cover value c_i will be computed; **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 according to a linear combination of scores obtained at the previous step. The following formula is exploited:

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

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*** (fully satisfying the corresponding request features); ***Conflicting*** (containing a data property value slightly conflicting with the corresponding request feature) [7]; ***Additional*** (either more specific than required ones or belonging to the first relevance level but not exposed in the user request); ***Underspecified*** (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 computation one. To this aim the score explanation is used. In fact, by analyzing fulfilled and conflicting features, a recruiter can decide to negotiate either features themselves or data property values, and 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. $p_i \prec_Q p_j$ 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* one, we can divide the previous n profiles into the ones which fulfill fp, the ones which do not and the ones for which data property a is unknown. If p_i, p_j both belong to the the same class, 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. 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}.

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

3 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 as main foreign APIs. Furthermore, it embeds Pellet (pellet.owldl.org) 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. I.M.P.A.K.T. is built upon the open source database system PostgreSQL 8.3 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. 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. Thanks to *lastdate* data property, we can say for example 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", "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 have been determined in several tests involving different specialist users engaged in a proactive tuning process of the software.

4 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 recruiter. 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 3 whereas deriving ranked results are reported in Figure 4. The GUI for browsing the ontology and to compose the query is also shown in Figure 3. Observe that the interface for defining/updating the candidate profile is exactly the same.

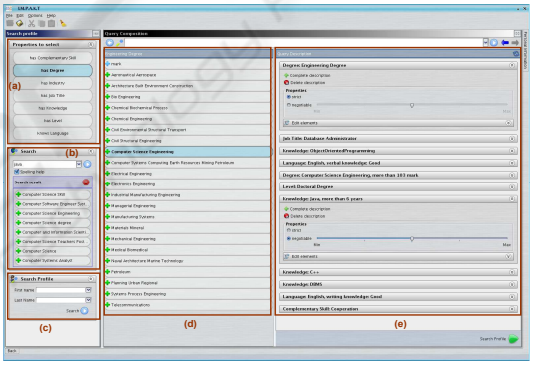

Figure 3: Query composition GUI.

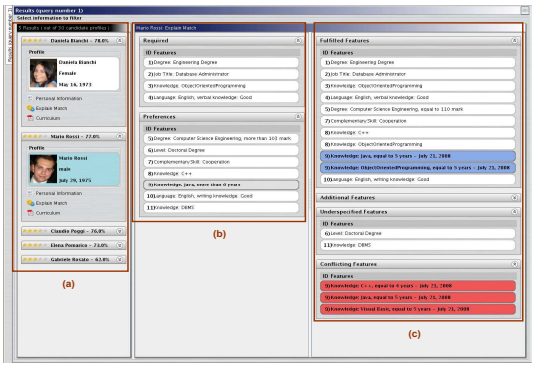

Figure 4: Results and score explanation GUI.

W.r.t. Figure 3, (a), (b) and (d) panels allow the recruiter 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 3 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 the name field of the known candidate whose profile is considered as starting request (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 4 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 recruiter 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 4– 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 $\exists hasKnowledge.(OOprogramming \sqcap =_{2008-07-21} lastdate \sqcap =_5 years)$ 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 recruiter's 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 3) and they can be further manipulated.

5 CONCLUSIONS

We presented 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 system allows to describe features of a required job position 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 weighting the relevance of each of them and at testing other strategies for score calculation in the match process. We are grateful to Umberto Straccia for discussions and suggestions and Angelo Giove for useful implementations.

REFERENCES

Bechhofer, S., Horrocks, I., and Turi, D. (2005). The OWL Instance Store: System Description. In *proc. of CADE '05*, pages 177–181.

Chomicki, J. (2002). Querying with Intrinsic Preferences. In *proc. of EDBT 2002*, pages 34–51.

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

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

Kießling, W. (2002). Foundations of preferences in database systems. In *proc. of VLDB'02*, pages 311–322.

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