Chapter 1
Informative top-k retrieval for advanced skill management

Simona Colucci and Tommaso Di Noia and Azzurra Ragone and Michele Ruta and Umberto Straccia and Eufemia Tinelli

Abstract The paper presents a knowledge-based framework for skills and talent management based on an advanced matchmaking between profiles of candidates and available job positions. Interestingly, informative content of top-k retrieval is enriched through semantic capabilities. The proposed approach allows to: (1) express a requested profile in terms of both hard constraints and soft ones; (2) provide a ranking function based also on qualitative attributes of a profile; (3) explain the resulting outcomes (given a job request, a motivation for the obtained score of each selected profile is provided). Top-k retrieval allows to select most promising candidates according to an ontology formalizing the domain knowledge. Such a knowledge is further exploited to provide a semantic-based explanation of missing or conflicting features in retrieved profiles. They also indicate additional profile characteristics emerging by the retrieval procedure for a further request refinement. A concrete case study followed by an exhaustive experimental campaign is reported to prove the approach effectiveness.

Simona Colucci
Politecnico di Bari, via E. Orabona 4, I-70125 Bari,Italy, e-mail: s.colucci@poliba.it
Data Over Ontological Models - D.O.O.M. srl, via N. Paganini 7, I-75100 Matera, Italy, e-mail: simona.colucci@doom-srl.it

Tommaso Di Noia
Politecnico di Bari, via E. Orabona 4, I-70125 Bari, Italy, e-mail: t.dinoia@poliba.it

Azzurra Ragone
Politecnico di Bari, via E. Orabona 4, I-70125 Bari, Italy, e-mail: a.ragone@poliba.it

Michele Ruta
Politecnico di Bari, via E. Orabona 4, I-70125 Bari, Italy, e-mail: m.ruta@poliba.it

Umberto Straccia
ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy, e-mail: straccia@isti.cnr.it

Eufemia Tinelli (corresponding author)
Politecnico di Bari, via Orabona 4, I-70125 Bari, Italy, e-mail: e.tinelli@poliba.it

1.1 Introduction

Human resources retrieval can be a very complex process due to the difficulties for recruiters in properly expressing their requirements. Given the increasing competitiveness in job market, e-recruitment systems should provide more complex and articulated procedures in order to avoid the money loss caused by a wrong –or simply non optimal– assignment. Hence, there is the need to face a change in the old paradigms and techniques for managing talents and skills. Currently, recruiters describe vacant job positions exploiting traditional recruiting methods, such as advertisements and referral systems. More recently, on-line recruitment websites (see `http://www.monster.com/`, `http://www.careerbuilder.com` to cite a few) have been employed, which mainly build their retrieval techniques on a keyword-based search. In spite of the speed-up reached in profile collection, such systems require a considerable engagement for the recruiter often producing unsatisfactory results. The process of describing the vacancy has to be particularly accurate to produce adequate results, turning into a very time-consuming task. Furthermore, the recruiter is not fully aware of retrieval and ranking criteria (if any) and she receives as output a list of possible candidates not following any explicit set criteria. So the choice of a particular profile in the returned set is made according to a further manual selection of the returned profiles. Moreover, considering that the system interrogation is keyword-based, the recruiter can express only mandatory requirements. On the other hand, a job offer is usually characterized by some features which are strictly required and some other ones which are only preferred, possibly with a preference degree. A system flexible enough to reflect such a diversification would allow a relevant distinction in job vacancy descriptions. In addition, having a ranked list of candidates (and not trivially a set of candidates) represents a significant added value, especially if the recruiter may choose the ranking criterion. Finally, after selecting a profile, a guided comparison between the required job position and the returned candidate description which explains the resulting match, would be very useful for a further request refinement step. Semantic-based techniques and technologies allow for making more efficient and flexible the recruitment process. The approach presented here is based on an automatic matchmaking process between available candidate profiles and vacant job positionsjob positions according to mandatory requirements and preferences provided by the recruiter. Both, candidates and job vacancies, are described in a formal language suitable for data intensive applications allowing a good trade-off between expressiveness and computational complexity. The proposed system combines flexibility in query formulation with an explanation of solutions. The matchmaking between job positions and candidates profiles exploits top-k retrieval techniques [20] by using a matching engine which performs top-k queries over a DLR-Lite [4] Knowledge Base (KB). It returns a ranked list of candidates managing also non-exact matches. Furthermore, the informative content of resulting outcomes is enriched with additional information explaining results of comparison of each retrieved candidate with respect to the job offer. The remainder of the chapter is organized as follows: in the next Section we report on the formal background underlying the approach; in Section 1.3 the

proposed framework is outlined whereas in Section 1.4 its effectiveness is proved through an exhaustive experimentation. A comparison with available tools and approaches to human resources retrieval is presented in Section 1.5. Conclusions and future work close the chapter.

1.2 Preliminaries

A logic-based approach is combined with a relational model to fully characterize the application field. On the one hand, Description Logics (DLs) are exploited to define proper axioms related to the knowledge domain. On the other hand, data related to candidate profiles are stored within a database so that conjunctive queries can be used to describe requests and to rank answers according to a scoring function. In this way the *Top-k* scored tuples satisfying the query are provided (the interested reader may refer to *Top-k query answering* [15, 18, 19, 20, 21] for a more detailed description).

Reference formalism. The specific DL we adopt is based on an extension of DLR-Lite [4], namely *top-k DLR-Lite*, not including negation operator but including built-in predicates. Differently from usual DLs, DLR-Lite supports also n-ary relations ($n \geqslant 1$).

A *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ stores the domain knowledge. Basically, it consists of three components: *facts* $\mathcal{F}$, *Ontology* $\mathcal{O}$ and *Abstraction* $\mathcal{A}$, which are defined as in the following:

– *facts Component:* $\mathcal{F}$ is a finite set of expressions in the form $R(c_1, \ldots, c_n)$, where R is an n-ary relation and every c_i is a constant. For each R, we represent the facts $R(c_1, \ldots, c_n)$ in $\mathcal{F}$ by means of a relational n-ary table T_R, containing the records $\langle c_1, \ldots, c_n \rangle$.
– *ontology Component:* $\mathcal{O}$ is used to define relevant axioms of the application domain involving concept classes and relationships among them. Given an alphabet of n-ary relations (denoted as R), and an alphabet of unary relations, namely *atomic concepts* (denoted as A), the component $\mathcal{O}$ is a finite set of *axioms* having the form $Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr$ where $m \geqslant 1$, all Rl_i and Rr have the same *arity*. Rl_i is a *left-hand relation* and Rr is a *right-hand relation*. The syntax of the relations appearing on the left-hand and right-hand side of ontology axioms is specified below:

$$
\begin{array}{ll}
Rr & \longrightarrow A \mid \exists[i_1, \ldots, i_k]R \\
Rl & \longrightarrow A \mid \exists[i_1, \ldots, i_k]R \mid \exists[i_1, \ldots, i_k]R.(Cond_1 \sqcap \ldots \sqcap Cond_h) \\
Cond & \longrightarrow ([i] \leqslant v) \mid ([i] < v) \mid ([i] \geqslant v) \mid ([i] > v) \mid ([i] = v) \mid ([i] \neq v)
\end{array}
$$

where A is an atomic concept, R is an n-ary relation with $1 \leqslant i_1, i_2, \ldots, i_k \leqslant n$, $1 \leqslant i \leqslant n$, v is a reference value for the concrete domain interpretation and $h \geqslant 1$. Here, $\exists[i_1, \ldots, i_k]R$ is the projection of the relation R on the columns $i_1, \ldots, i_k$. Hence, $\exists[i_1, \ldots, i_k]R$ has arity k. On the other hand, $\exists[i_1, \ldots, i_k]R.(Cond_1 \sqcap \ldots \sqcap$

$Cond_l$) further restricts the projection $\exists[i_1, \ldots, i_k]R$ according to the conditions specified in $Cond_i$. For instance, $([i] \leqslant v)$ specifies that the values of the i-th column have to be less or equal than the value v. Besides other constructs, *top-k DLR-Lite* supports concrete domains which are defined as pairs $\langle \Delta_{\mathrm{d}}, \Phi_{\mathrm{d}} \rangle$, where Δ_{d} is an interpretation domain and Φ_{d} is the set of n-ary *domain predicates* d and an interpretation $d^{\mathrm{d}} \colon \Delta_{\mathrm{d}}^n \to \{0, 1\}$.

– *Abstraction Component:* a set of "abstraction statements" allowing to connect atomic concepts and relations to physical relational tables. Basically, this component is a wrapper for the underlying database and thus, it prevents that relational table names occur in the ontology. Let R_1 be a relation symbol and let R_2 be an m-ary table in the database. Let $c_1, \ldots, c_n$ be $n \leqslant m$ column names of relation R_2 of type t_i. Hence, a *simple abstraction statement* is in the form $R_1 \mapsto R_2(c_1[t_1], \ldots, c_n[t_n])$ stating that R_1 is an n-ary relation of the ontology component mapped into the projection on columns $c_1 \ldots, c_n$ of R_2. We assume that R_1 occurs in $\mathcal{O}$, while R_2 occurs in $\mathcal{F}$. The retrieval of facts from the KB requires a proper query language. In what follows both language syntax and semantics are detailed.

Query language. Queries are in a conjunctive form and enable a scoring function to rank answers. Each *query* is structured as reported hereafter:

$$q(\mathbf{x})[s] \leftarrow \exists \mathbf{y}\ R_1(\mathbf{z}_1), \ldots, R_l(\mathbf{z}_l), \mathsf{OrderBy}(s = f(p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)) \quad (1.1)$$

where (1) q is an n-ary relation (every R_i is an n_i-ary relation) whereas $\mathbf{x}$ are the related n variables (*distinguished variables*); (2) $\mathbf{y}$ are the so-called *non-distinguished variables*; (3) $\mathbf{z_i}, \mathbf{z_j}'$ are tuples of constant or variable values in $\mathbf{x}$ or $\mathbf{y}$. Any variable in $\mathbf{x}$ occurs in some $\mathbf{z_i}$. Any variable in $\mathbf{z_j}'$ occurs in some $\mathbf{z_i}$; (4) p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, 1]_m$. Such predicates have been named *expensive predicates* in [5] as the related score is not pre-computed off-line, but it is calculated at runtime during query execution. Notice that the n-ary fuzzy predicate p is required to be *safe*, that is, no m-ary fuzzy predicates p' such that $m < n$ and $p = p'$ must be present; (5) f is a *scoring* function $f \colon ([0, 1]_m)^h \to [0, 1]_m$, which combines the scores of the h fuzzy predicates $p_j(\mathbf{c}''_j)$ into an overall *score* to be assigned to the rule head $q(\mathbf{c})$. We assume that f is *monotone*, *i.e.*, for each $\mathbf{v}, \mathbf{v}' \in ([0, 1]_m)^h$ such that $\mathbf{v} \leqslant \mathbf{v}'$, $f(\mathbf{v}) \leqslant f(\mathbf{v}')$ holds, where $(v_1, \ldots, v_h) \leqslant (v'_1, \ldots, v'_h)$ iff $v_i \leqslant v'_i$ for all i.

Finally, a *disjunctive query* $\mathbf{q}$ is, as usual, a finite set of conjunctive queries where all the rules have the same head. We omit to specify $\exists \mathbf{y}$ when $\mathbf{y}$ can be clearly elicited from the context. $R_i(\mathbf{z_i})$ may also be a concrete unary predicate in the form $(z \leqslant v), (z < v), (z \geqslant v), (z > v), (z = v), (z \neq v)$, where z is a variable, and v is a concrete domain proper value. We call $q(\mathbf{x})[s]$ its *head*, $\exists \mathbf{y}.R_1(\mathbf{z}_1), \ldots, R_l(\mathbf{z}_l)$ its *body* and $\mathsf{OrderBy}(\ldots)$ the *scoring atom*. The informal meaning of such a query is: if $\mathbf{z}_i$ is an instance of R_i, then $\mathbf{x}$ is an instance of q with a degree less or equal to $f(p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h))$. Also the scoring atom can be omitted. In this case we assume $s = 1$.

Top-k retrieval. Given a KB $\mathcal{K}$, and a disjunctive query $\mathbf{q}$, we denote with $ans_k(\mathcal{K}, \mathbf{q})$ the k tuples $\langle \mathbf{c}, s \rangle$ instantiating the query relation q with a maximal score (if such k tuples exist), and ranked in decreasing order w.r.t. the score s.

From a query answering standpoint, top-k retrieval extends the DL-Lite/DLR-Lite reasoning approach [4] to the fuzzy case. The reasoning algorithm is an extension of the one described in [4, 18, 20]. Given a query $q(\mathbf{x})[s] \leftarrow \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$, the algorithm proceeds as reported hereafter:

1. by considering $\mathcal{O}$, the original query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathbf{q}, \mathcal{O})$. Basically, the reformulation procedure closely resembles to a top-down resolution procedure for logic programming, where each axiom is seen as a logic programming rule;
2. we remove redundant ones from $r(\mathbf{q}, \mathcal{O})$;
3. the reformulated queries $q' \in r(\mathbf{q}, \mathcal{O})$ are translated into ranked SQL queries and evaluated. The query evaluation of each ranked SQL query returns the top-k answer set for that query. Specifically, the RankSQL [14] system is exploited for this purpose;
4. all the $n = |r(\mathbf{q}, \mathcal{O})|$ top-k answer sets have to be merged into the final top-k answer set $ans_k(\mathcal{K}, \mathbf{q})$. As $k \cdot n$ may be large, we apply a *Disjunctive Threshold Algorithm* (DTA, see for example [20]) to collect all the answer sets.

A detailed description of the top-k retrieval algorithm is beyond the scope of this work. An implementation of it is part of the SoftFacts system[1].

1.3 Human Resources Retrieval

The top-k retrieval approach described above can be targeted to several discovery problems. In this paper we describe its application to a Human Resources Management. The theoretical framework has been implemented into `I.M.P.A.K.T.` [22], a system for skills management developed by *Data Over Ontological Models s.r.l.*[2] as a commercial solution implementing the skill matching framework designed in [6]. In what follows the general features of the system as well as a description of the components architecture will be provided. The retrieval process starts by *querying* a *knowledge base* where candidate profiles (*f*acts expressed trough a domain-independent framework) are stored in a relational database. The reference vocabulary is provided by an *ontology*, representing main concepts and relations among them in the knowledge domain (see Appendix 1.5.2). In what follows, each system component will be explained in depth. The information in the ontology (Section 1.3.1) is exploited to compose queries (Section 1.3.3) addressed to the database component (Section 1.3.2). The final goal is to obtain a top-k retrieval answer set

[1] `http://gaia.isti.cnr.it/~straccia/software/SoftFacts/SoftFacts.html`

[2] `http://www.doom-srl.it/`

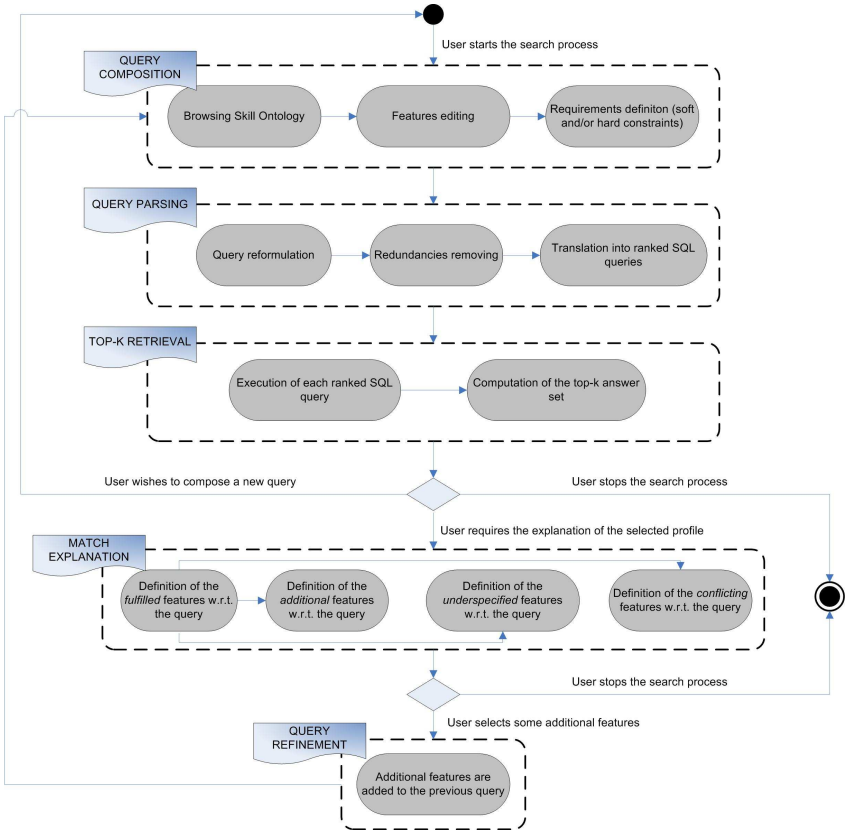

Fig. 1.1 The Informative top-k retrieval Process

(Section 1.3.4) enriched by information useful for outcomes explanation. The whole process is depicted in Figure 1.1 and summarized hereafter. - **Storage phase**

- *Knowledge modeling* - A candidate's CV includes personal and employ information. They are modeled according to an ontology $\mathcal{O}$ and, after a pre-processing phase, they are stored in a relational DBMS;
- *Knowledge pre-processing* - The semantic annotation of a CV is parsed to group in the same collection features related to the same area *e.g.*, *Job Title* or *Education Level*. We denote with p (for profile) the resulting description of a candidate. For each feature in a collection, the system will build a normalized version taking into account ontology $\mathcal{O}$ axioms so producing a normalized version of the profile description denoted as p_{norm};
- *Knowledge storage* - The system stores both p and p_{norm} in a relational database. Notice that p and p_{norm} will be maintained in different tables and for each feature collection there will be a specific table. The need to have two differently

parsed description of the same candidate –and consequently a different relational representation– will be clarified in the following *query phase*.

- Query phase

- *Query composition* - The recruiter expresses information useful to describe the profile to search for according to the ontology $\mathcal{O}$. Hence, the user query can be considered as a set of features grouped by area of interest. Each feature can be settled as either a strict requirement or a preference (hard or soft constraints, respectively).
- *Top-k retrieval* - The query composed by the user is reformulated according to the rules in Section 1.2 and the top-k retrieval process starts considering only the *non-normalized* profiles p stored in the relational database;
- *Match explanation* - Once top-k resources have been retrieved, their formal representation w.r.t. the ontology $\mathcal{O}$ may be exploited for explaining results. We name *match explanation* a semantic-based comparison between each feature of retrieved candidate and the corresponding request in the user query. By exploiting the *normalized* profiles p_{norm} stored in the relational database, the system provides the comparison outcomes classifying each candidate feature in the following classes: 1) fulfilled; 2) additional; 3) underspecified and 4) conflicting.

1.3.1 Ontology Component

The skill ontology underlying `I.M.P.A.K.T.` mainly focuses on ICT domain and contains 3341 `rdfs:class` classes representing both candidate's knowledge and her complementary skills. The former refer to the candidate background knowledge about specific technologies and tools, the latter represent personal and social characteristics. An upper level sketch of the ontology is shown in Fig. 1.2. The whole ontology has been developed by two domain experts working full time for 10 months following Methontology [11] specifications. The root class of the `I.M.P.A.K.T.` ontology is `Profile`. Every CV description is an instance of it. `Profile` class also is the domain of `rdf:Property` roles needed to represent both personal information (*i.e.*, first/second name, address, telephone[3]) and "ontological" characteristics modeling skills, knowledge and competences. Main ontology roles are reported hereafter:
-`Knowledge`. Its subclasses are, for instance, `Functional Programming`, `XML` to cite a few. Along with the skill representation, it is allowed to specify the related experience and competence expressed in years.
-`hasJobTitle`. It is exploited to model job positions as `Teacher Assistant` or `Database Administrator`. All the job positions represented in the ontology are subclasses of `Job Title`. Also in this case it is possible to specify the job experience (in years).

[3] Many properties are obtained with a mapping of `vcard` [12] and `foaf` [8] specifications.

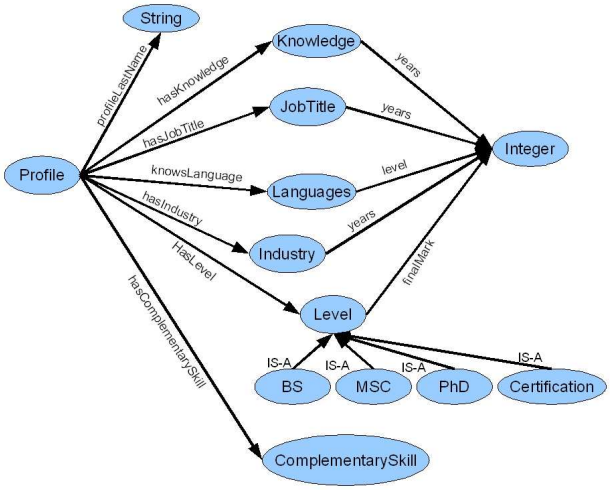

Fig. 1.2 A graphical sketch of the ontology structure

-`hasLanguage`. The knowledge of a specific `Language` may be further specified and classified in `verbal knowledge`, `reading knowledge` and `writing knowledge` followed by the reference level `Basic`, `Good` or `Excellent`.
-`hasIndustry`. It is used, along with its reference range `Industry`, to model institute areas, research laboratories, company departments where a candidate works or worked. Years of work experience in a specific field may be specified.
-`hasLevel`. It is adopted to represent everything related to candidate education and training. From basic education to `Master Degree`, the whole candidate's qualifications can be specified including specific `Certifications` she gained.
-`hasComplementarySkills`. It models complementary attitudes (namely soft skills) such as `Cooperation`, `Stress Tolerance`, `Leadership` which complete CV's basic information about knowledge, technical skills and competence and which often are very useful to correctly evaluate a candidate profile.

In its current implementation, the ontology does not report disjunction axioms. Reasonably, in the recruitment field it is quite rare to assert that *if you know "A" then you do not know "B"*. The above described properties represent what will be defined *entry points* in the domain ontology while their range represents what will be defined *main categories*. To help the reader in better understanding the case study reported later on, an ontology axioms excerpt is reported in Appendix 1.

1.3.2 Database Component

In order to perform the top-k retrieval, candidates must be stored as *facts* in a relational database. If additional ontological information underlying each profile are added in the database, a match explanation can be enabled to make the recruiter aware of the retrieval results and to perform a possible query refinement. The Entity-Relationship (ER) model designed to this aim is shown in Fig. 1.3. The slightly redundant relational structure is needed to separately optimize the above processes: top-k retrieval of candidates (Fig. 1.3 (a)) and match explanation (Fig. 1.3 (b)).

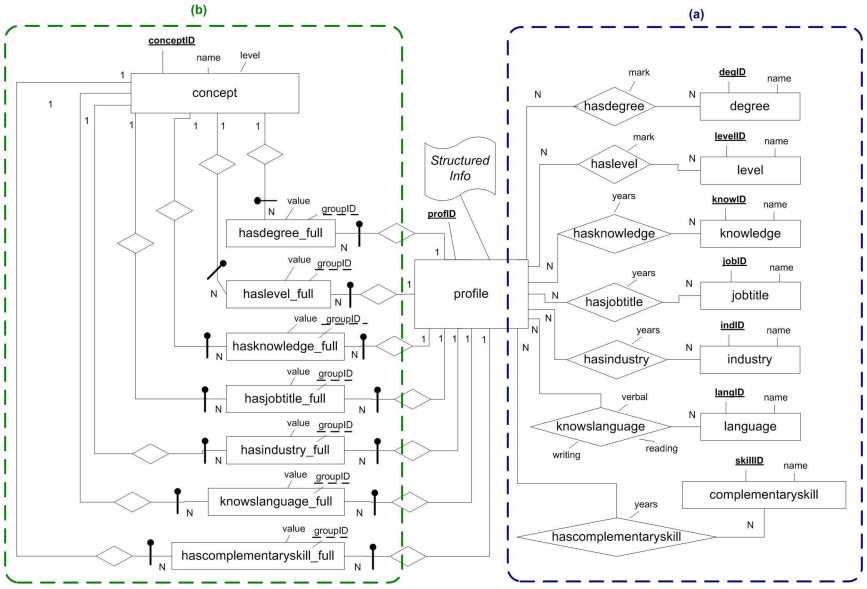

Fig. 1.3 Entity-Relationship model underlying the case study

The main entity is `profile` which bridges (a) and (b) model components. `profile` contains the so called *structured info* such as personal information (*i.e.*, last and first name, birth date –see `profile` table map-role in Fig. 1.4 for more details) and employ information (*i.e.*, preferred working hours, car availability). With respect to Fig. 1.3 (a), notice that the model reproduces the structure of the ontology component (see Fig. 1.2): for each main category in the ontology, one entity in a *many-to-many* relation with `profile` is modeled. The ontology is mapped into the relational tables in (a) using the abstraction component $\mathcal{A}$ of the underlying DLR-Lite KB (see Section 1.2) represented in Fig. 1.4. Attributes as `years`, `mark`, `reading level`, `writing level` and `verbal level` are modeled in specific tables (main categories) in order to maintain the corresponding data property values.

```
(MAP-ROLE Profile profile (profID, FirstName, LastName, Genre,
 BirthDate, CityOfBirth, Address, City, ZipCode, Country, IdentityCode,
 PhoneNumber, Email, WebPage, Nationality, ResidentIn,
 SuddenJobAvailability, JobLocation, FlexibleWorkHours,
 TravelingAvailability, CertificationInstitute, Salary, CarAvailability))
(MAP-ROLE degreeName degree (degID, name))
(MAP-ROLE levelName level (levelID, name))
(MAP-ROLE industryName industry (indID,name))
(MAP-ROLE jobName jobtitle (jobID, Name))
(MAP-ROLE knowledgeName knowledge (knowID, name))
(MAP-ROLE languageName language (langID, name))
(MAP-ROLE skillName complementaryskill (skillID, name))
(MAP-ROLE hasDegree hasdegree (profID, classID, mark))
(MAP-ROLE hasLevel haslevel (profID, classID, mark))
(MAP-ROLE hasIndustry hasindustry (profID, classID, years))
(MAP-ROLE hasJobTitle hasjobtitle (profID, classID, years))
(MAP-ROLE hasKnowledge hasknowledge (profID, classID, years))
(MAP-ROLE knowsLanguage knowslanguage (profID, classID, reading, verbal,
 writing))
(MAP-ROLE hasComplementarySkill hascomplementaryskill (profID, classID,
 years))
```

Fig. 1.4 Relational tables deriving from ontology mapping

The model in part (a) of Fig. 1.3 allows to perform a classical top-k retrieval problem over a knowledge base, whereas part (b) of the model has been properly designed to enable an ontology-based classification of profiles for making explicit differences between each retrieved candidate and submitted query. To this aim, a normalization procedure on each profile must be performed in order to put in evidence all the "information atoms" a single feature is made of. The normalized profile p_{norm} is obtained processing each feature by recursively applying the following rule: if a p_{norm} feature contains a concept C and in the ontology $\mathcal{O}$ there exists the axiom $C \sqsubseteq D$, then D is added to the feature.

Normalized profiles are stored in the table modeled by the (b) component of Fig. 1.3. Also in this case, profile features belonging to the same entry point are stored in the same table. Here, the use of a *many-to-many* relationship is not enough because, after normalization, the same "information atom" can occur many times in a profile as introduced by different features. Hence, a design issue is to properly allow the identification of features and rightly store the "information atoms" they are composed by. Tables named `<entrypoint>_full` (*i.e.*, `hasknowledge_full`, `hasjobtitle_full`, `hasdegree_full`) contain normalized features corresponding to the related main category whereas an attribute is exploited (namely `groupID`) to label information referred to a specified feature. `groupID` can be considered as a partial key of `<entrypoint>_full` table since the atoms of a normalized feature are distributed in different tuples of the table itself but they have the same `groupID`. Finally, `value` is adopted for datatype properties quantification. `concept` contains both concepts and data properties in the ontology; they are exploited to describe profile features. Among attributes, a specific relevance is assumed by `level`. It indicates the *depth level* of the concept name (represented by one tuple) in the ontology (taxonomy). This attribute is needed for match explanation to more efficiently detect the most specific concept in a concept set (*i.e.*, a set of features sharing the same `groupID`). Note that `level` is meaningful only when `conceptID` corresponds to a concept name.

profile			
profID	*FirstName*	*LastName*	...
1	Jack	Sparrow	...

hasdegree		
profID	*classID*	*mark*
1	29	110
...	...	...

degree	
degID	*name*
29	Computer_Science_Engineering
...	...

Fig. 1.5 `profile`, `hasdegree` and `degree` tables for Jack's profile

hasdegree_full			
profID	*groupID*	*conceptID*	*value*
1	1	30	null
1	1	56	null
1	1	16	null
1	1	4	110
...	...	...	...

concept		
concepID	*name*	*level*
4	mark	null
16	Degree	1
30	Computer_Science_Engineering	3
56	Engineering_Degree	2
...	...	...

Fig. 1.6 `hasdegree_full` and `concept` tables for Jack's normalized profile

In order to better clarify both usefulness and effectiveness of the previous E-R modeling w.r.t. skill and talent management domain, a toy profile description is proposed hereafter. Let us suppose *Jack* is searching for a job; his profile can be described as: *"Jack speaks English with a scholar level whereas he's doing better with written English. He has a degree in Computer Science Engineering, with mark equal to 110 (out of 110), an excellent experience in Java programming (5 years) and he is one year experienced in Web Design, ...". Jack*'s profile can be represented according to both *entry points* and *main categories* in Fig. 1.2:
- ***hasDegree*** - Computer Science Engineering (final mark = 110)
- ***hasKnowledge*** - Java (years of experience = 5); Web design (years of experience = 1)
- ***knowsLanguage*** - English (verbalLevel=1, writingLevel=2)

The previous profile will be normalized and split in components. So it is stored in proper tables. With reference to the model in Fig. 1.3, storage details are reported hereafter:
- *part (a)* - a tuple is added to `hasdegree`, `hasknowledge`, `knowslanguage` tables, respectively. It contains a concept name and one or more data property values. Tuple stored in the `hasdegree` table is shown in Fig. 1.5;
- *part (b)* - some tuples (the exact value depends on the depth of concept in the ontology) are added to tables `hasdegree_full`, `hasknowledge_full` and `knowslanguage_full`, respectively. Furthermore, a tuple is added for each information atom and data property value. Tuples stored in `hasdegree_full` table are reported in Fig. 1.6;

For the sake of clarity, in Fig. 1.3 we do not represent tables needed to store intermediate match results for computing the final explanation. After top-k retrieval, given a profile (labeled by a `profID`), denoting as i the i-th feature of the *reformulated* top-k query, i belongs to a specific entry point. Hence, the system stores all the *normalized* profile tuples containing at least the concept name in i in a proper auxiliary table just named `<entrypoint>_full_i`. For example, if the recruiter is looking for "a candidate having an Engineering Degree with a final mark equal

or higher than 103 (out of 110)" and she selects a profile from the top-k answer set for the explanation, then the system will retrieve all the tuples satisfying the query feature for the specific `profID` from the `hasdegree_full` table. Those tuples will be materialized in the `hasdegree_full_1` table created at runtime.

Given a `profID`, considering each feature i in the *reformulated* top-k query and the corresponding `<entrypoint>_full` and `<entrypoint>_full_i` tables, the system will compute the following classes of features:

1. ***fulfilled***: all the features in `<entrypoint>_full_i` table are de-normalized according to `level` value of concepts sharing a `groupID`. The resulting tuples represent the so called *fulfilled features*. In this class, duplicate features can exist so the class has to be preprocessed before showing it to the user (see `Object Oriented Programming` feature in Fig. 1.10 for a concrete example);
2. ***additional***: given a `groupID` collection, if there exists a concept in the related `<entrypoint>_full` table with a `level` higher than the one in the query and/or if there is a data property not specified in the query itself, these "information atoms" are classified as *additional features*. Features not required by the user but considered as significant for the skill domain also belong to this set of features. Particularly, all the features stored in `<hasknowedge>_full` table not required in the whole user query (*i.e.*, not only in a query feature) are also presented as additional features. However, the system distinguishes between these two different kind of additional features (see Fig. 1.9);
3. ***conflicting***: given a `groupID` collection, if there exists a data property in the `<entrypoint>_full_i` table lower than the one required by a query feature, that "information atoms" are classified as *conflicting features*;
4. ***underspecified***: if all the concept names and/or data properties required by the query are lacking in the corresponding `<entrypoint>_full_i` table, the missing "information atoms" are classified as *underspecified features*.

Obviously, in case of query features settled as "strict requirements" only *fulfilled features* have to be considered whereas *additional* ones are optionals.

1.3.3 Query process (by Example)

Let us consider a recruiter looking for "*a candidate having an Engineering Degree (possibly with a final mark equal or higher than 103 (out of 110)). A Ph.D is preferred and a good ability to write in English is required. She should be at least six years experienced in Java and she should have complex problem solving capabilities*". Such a job request, can be summarized as:

1. Strictly required constraints:
   a. Engineering degree;
2. Preferences:
   a. Engineering degree final mark $\geqslant 103$;
   b. Doctoral Degree;

c. Java programming and experience $\geqslant$ 6 years;
d. Complex problem solving capabilities;
e. Good written English.

In the following, we show how the approach we propose can provide an answer to the above request w.r.t. to the data-set in Appendix 2, made up by ten candidates whose profile has been chosen to evidence peculiarities of our approach. Particularly:

- 8 out of 10 selected profiles include features covering all introduced categories in order to show how the proposed modeling allows to convey most information needed in real-world recruitment;
- two candidates (*Carla Buono* and *Marcello Cannone*) do not fulfill strict requirements specified in the query and then they will not be part of the final result set;
- a small subset of candidate sample is made up by people (*Mario Rossi*, *Daniela Bianchi* and *Elena Pomarico*) with similar profiles: their CVs only differ by experience years associated to either job titles, enterprise working or exploitation of a given competence. Such a choice allows us to make clear how these differences, even slight, cause profiles to be differently ranked by the scoring mechanism and evaluated by the match explanation facilities;
- three candidates (*Lucio Battista*, *Mariangela Porro* and *Nicola Marco*) satisfy only a few characteristics other than strict requirements. The scoring mechanism will rank them lower than profiles better filling query preferences (*Domenico De Palo* and *Carmelo Piccolo*);
- two profiles (*Mario Rossi* and *Carmelo Piccolo*) include features slightly conflicting with query preferences. In the match explanation phase they will be automatically highlighted;
- many selected profiles have additional features w.r.t. the query. They will be presented in the match explanation.

`I.M.P.A.K.T.` provides a graphical interface to compose the recruiter's requests: Fig. 1.7 shows such a GUI reporting the example query description. In the menu (a), all the entry points are listed; panel (b) allows to search for ontology concepts according to their meaning; section (c) enables the user to explore both taxonomy and properties of a selected concept. Entry points in menu (a) represent main classes and relationships in Fig. 1.2. Once an item is selected in panel (c), the corresponding menu is dynamically filled and added to panel (d) enumerating the features requested in the query. For each of them, the GUI allows: **(1)** to define if the feature is a strict one –panel (f)– or if it is a preference –panel (c); **(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 either each feature atom or existing data property values. In what follows, we report the formalization of the job request according to the query language introduced in Section 1.2, aimed at top-k retrieval:

q(id, lastName)
←
profileLastName(id, lastName),
hasDegree(id, degreeId, mark), degreeName(degreeId, degreeName), Engineering_Degree(degreeId),
hasLevel(id, levelId, levelmark), levelName(levelId, levelName),
knowsLanguage(id, lanID, Reading, Verbal, Writing), languageName(langID, langName),
hasKnowledge(id, classID, years, type, level), knowledgeName(classID, hasKnowledge),
hasComplementarySkill(id, classID2)skillName(classID2, capabilities)

OrderBy(s = rs(mark; 102, 110) · 0.166 + pref1(levelName; Doctoral_Degree) · 0.166+
pref1(langName; English) · pref4(Writing; NotSpecified/1.0, Basic/3.0, Good/6.0, Excellent/9.0) · 0.166+
rs(years; 5, 10) · pref1(hasKnowledge; Java) · 0.166+
pref1(capabilities; Complex_Problem_Solving) · 0.166)

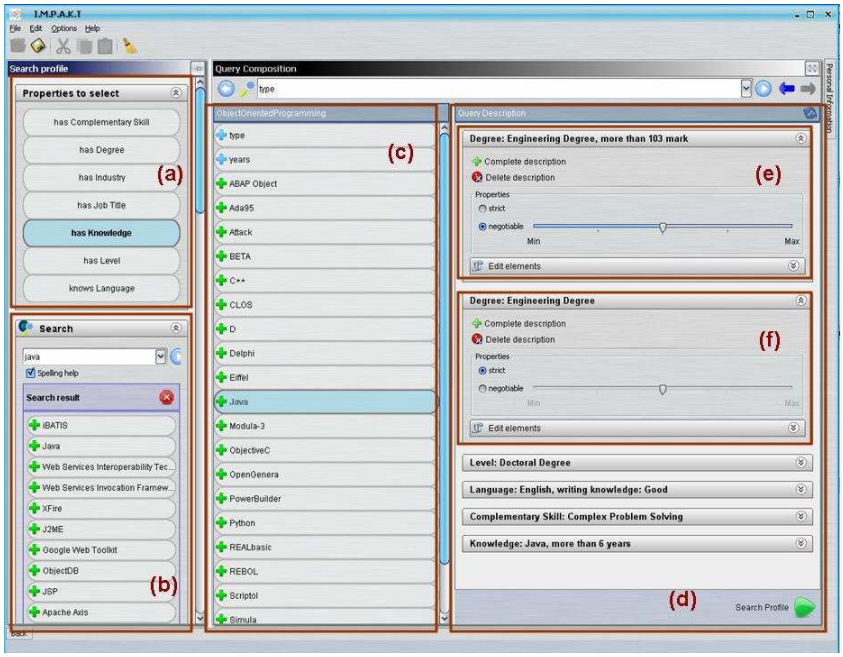

Fig. 1.7 Query composition GUI

1.3.4 Match explanation

Fig. 1.8 (a) shows how the I.M.P.A.K.T. GUI renders match explanation result. Candidates are sorted by their score. Noteworthy, only eight candidates are retrieved, given that neither *Carla Buono* nor *Marcello Cannone* have an Engineering Degree, which is strictly required by the query. Detailed information about the

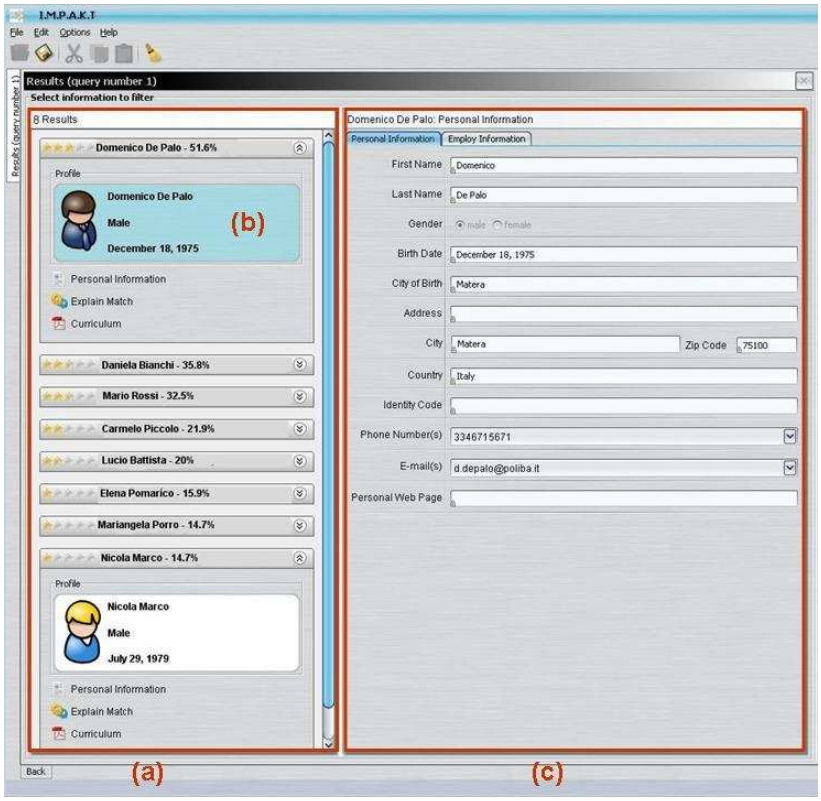

Fig. 1.8 Informative top-k retrieval Answer Set

two candidates can be shown expanding related panels as in Fig. 1.8 (b). Three options are made available for each retrieved candidate: i) *personal information*, ii) *textual CV*, iii) *explain match*.

By selecting view i), the panel displayed in Fig. 1.8 (c) is shown on the right side of the GUI. It gathers both personal and employ information related to the selected candidate. View ii) allows the rendering of textual CV of the selected candidate. In view iii) query and selected profiles are graphically compared. In Fig. 1.9 the best (*Domenico De Palo*) retrieved profiles in the answer set are examined, in order to explain the retrieval results. On the contrary, Fig. 1.10 shows a comparison between *Mario Rossi*'s profile and the example query. Here conflicting features are identified and shown to the end user.

We explain how query/result comparison behaves with the aid of Fig. 1.9. Panels (a) and (b) sum up strict requirements and preferences in the original query, respectively. Noteworthy, features in the query are numbered by an ID, which makes easier to identify them while comparing the retrieved profiles. The remaining of the GUI shows the following information:

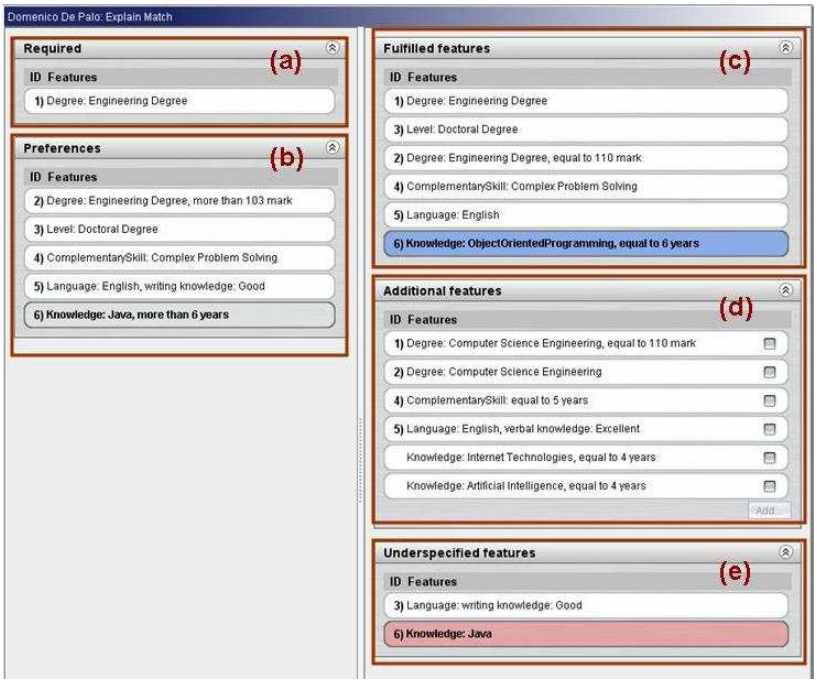

Fig. 1.9 Best candidate in the answer set

- Panel *(c)* shows *fulfilled features*, *i.e.*, features required with the query and either perfectly matched by the candidate or slightly conflicting in her profile;
- Panel *(d)* provides *additional features*, *i.e.*, technical skills in the candidate CV not required at all by the query or more specific than the required ones;
- Panel *(e)* gathers *underspecified features*, *i.e.*, parts of the query not explicitly specified in the retrieved CV.

The presentation of comparison results makes use of feature IDs: the ID of the related feature in the query is assigned to each fulfilled and underspecified characteristic in the profile; the same presentation is provided in case of additional characteristics. For example, by looking at Fig. 1.9 (c), we can assert that *Domenico De Palo* almost completely fulfills query features from 1 to 6, even though preference 3 (Java Programming) and preference 5 (good writing English knowledge) result underspecified (see panel (e)). In fact, *Domenico De Palo* is 6 years experienced in object oriented programming (a direct ancestor of Java programming in the reference ontology), and he knows English, even though nothing is specified within his CV about competence in writing English documents. It is noteworthy that by clicking on a query feature in (a) or (b), the GUI automatically highlights all characteristics labeled by the same ID in panels (c), (d), and (e). Finally let us have a look to panel (d): *Domenico De Palo* has several additional features w.r.t. the query, either related to elicited preferences and requirements (numbered features in (d)) or

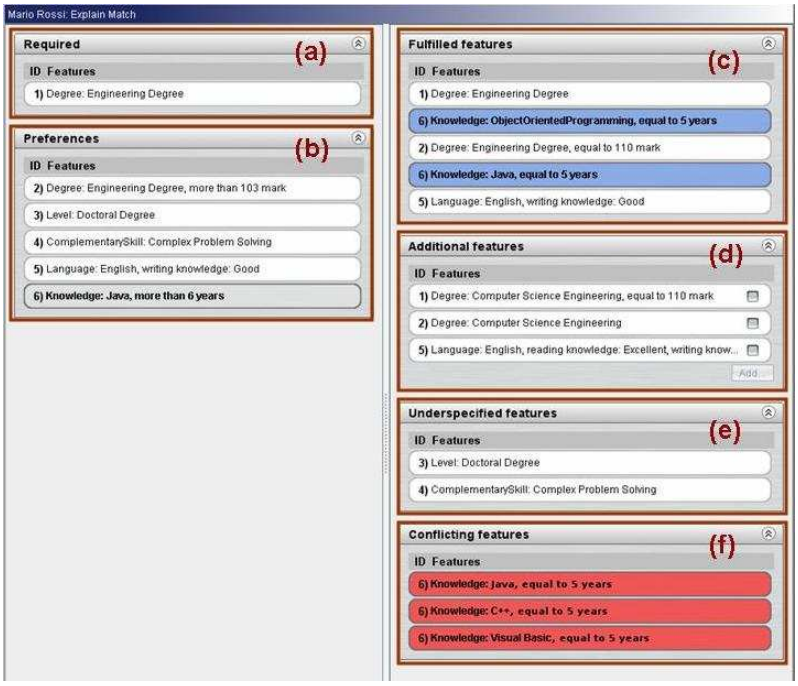

Fig. 1.10 Candidate in the answer set with conflicting features

to other profile competences (not numbered features in (d)). By looking at feature 1 in (d), for example, it can be noticed that the candidate is not only an Engineer, as required by the recruiter, but he is a Computer Science Engineer with full marks. Moreover he has a 4 years knowledge of *Internet Technologies* and *Artificial Intelligence*, and this could be an added value for the recruiter, even if it was not explicitly asked for. Fig. 1.10 represents an explanation of the match between *Mario Rossi*'s CV and the example query. Let us consider feature 6 in the query: "*at least six years of Java experience*". By looking at Fig. 1.10 (c), it can be noticed that *Mario Rossi* is 5 years experienced in Java and Object Oriented Programming. Such a conflict is highlighted in the GUI as shown in Fig. 1.10 (f) so making the recruiter aware of profile features slightly conflicting with the query. Notice that the same preference (*i.e.*, the one identified by ID 6) generates two information atoms in panel (c). Such a duplication in fulfilled features does not introduce redundancy. Instead, it is exploited to show in panel (f) CV conflicting features.

1.4 Experiments

In this section, we report on top-k retrieval performance in the proposed setting. Experimental evaluation has been conducted exploiting the *SoftFacts* system over the ontology component detailed in Section 1.3.1. It is based on $d = 100000$ automatically generated CVs. Several queries have been built according to the following criteria: i) scoring (resp. not scoring) atoms, ii) ranking (resp. not ranking) aggregates to be submitted to the system, iii) varying the size of CVs and iv) exploiting different values for k in the top-k retrieval ($k \in \{1, 10\}$). For each setting, it has been evaluated:

1. the number of queries generated after the reformulation process ($|r(\mathbf{q}, \mathcal{O})|$);
2. the number of reformulated queries after redundancy elimination (q_{DB});
3. the time needed for the reformulation process (t_{ref});
4. the time needed for the query redundancy elimination process (t_{red});
5. the query answering time of the database component ($t_{DB_{all}}$, t_{DB_1}, $t_{DB_{10}}$).

Note that measures in 1-4 neither depend on the number d of CVs nor on the k value. The submitted queries are reported hereafter (for illustrative purposes, for some of them we also provide the encoding in the used formalism):

1. Retrieve CV referred to candidates with some competences about Engineering Technology

```
q(id, lastName, hasKnowledge, Years)
← profileLastName(id, lastName),
  hasKnowledge(id, classID, Years, Type, Level),
  knowledgeName(classID, hasKnowledge),
  Engineering_and_Technology(classID)
```

2. Retrieve CV referred to candidates with an Engineering degree
3. Retrieve CV referred to candidates with a competence in Artificial Intelligence, having a degree with final mark greater than 100/110
4. Retrieve CV referred to candidates with a competence in Artificial Intelligence, having an Engineering degree with final mark greater than 100/110
5. Retrieve CV referred to candidates experienced in Information Systems (at least 15 years), having a degree with final mark greater than 100
6. Retrieve top-k CV referred to candidates with a competence in Artificial Intelligence, having a degree with final mark scored according to a right-shoulder fuzzy function $rs(mark; 100, 110)$

```
q(id, lastName, degreeName, mark, hasKnowledge, years)
← profileLastName(id, lastName), hasDegree(id, degreeId, mark), degreeName(degreeId, degreeName),
  hasKnowledge(id, classID, years, type, level), knowledgeName(classID, hasKnowledge),
  Artificial_Intelligence(classID), OrderBy(s = rs(mark; 100, 110))
```

7. Retrieve CV referred to candidates having an Engineering degree with final mark scored according to $rs(mark; 100, 110)$

8. Retrieve top-k CV referred to candidates with a competence in Artificial Intelligence, having an Engineering degree with final mark scored according to $rs(mark; 100, 110)$
9. Retrieve CV referred to candidates with a competence in Information Systems, having an Engineering degree with final mark and years of experience both scored according to $rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6$;
10. Retrieve CV referred to candidates with a good competence in Artificial Intelligence. Final mark, years and experience level are scored according to the function $rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot pref(level; Good/0.6, Excellent/1.0) \cdot 0.6$;

```
q(id, lastName, degreeName, mark, hasKnowledge, years, kType)
← profileLastName(id, lastName), hasDegree(id, degreeId, mark), degreeName(degreeId, degreeName),
  hasKnowledge(id, classID, years, type, level), knowledgeLevelName(level, kType), Good(level),
  knowledgeName(classID, hasKnowledge), Artificial_Intelligence(classID),
  OrderBy(s = rs(mark; 100, 110) · 0.4 + rs(years; 15, 25) · pref(level; Good/0.6, Excellent/1.0) · 0.6)
```

11. Retrieve CV referred to candidates with a competence in Artificial Intelligence, grouped by $MAX[rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6]$

```
q(id, lastName)
← profileLastName(id, lastName), hasDegree(id, degreeId, mark),
  hasKnowledge(id, classID, years, type, level), Artificial_Intelligence(classID),
  GroupBy(id, lastname), OrderBy(s = MAX[rs(mark; 100, 110) · 0.4 + rs(years; 15, 25) · 0.6])
```

12. Retrieve CV referred to candidates with a competence in Artificial Intelligence, having an Engineering degree, grouped by $AVG(rs(mark; 100, 110) \cdot 0.4 + rs(years; 15, 25) \cdot 0.6)$

Queries 1-5 are *crisp* since they do not involve any fuzzy predicate. As each answer scores 1.0, we would like to verify whether there is a difference between the retrieval time of all records and the one of just the best k. The remaining queries are top-k ones. In query 9, an example of score combination is shown, with a preference on the experience years. In query 10, we use the following preference scoring function: $pref(level; Good/0.6, Excellent/1.0)$, which returns neither 0.6 if *level* is *Good* nor 1.0 if *level* is *Excellent*. Queries 11 and 12 use ranking aggregates.

Tests have been performed on a MacPro PC equipped with a 2 x 3 GHz Dual-Core processor and 9 GB RAM with Mac OS X 10.5.5 onboard. Results are shown in Table 1.1 (time is expressed in seconds). Some consideration can be made about experimental outcomes:

- the response time is always acceptable if one takes into account the non negligible ontology size, the relevant number of CVs and the lacking of any indexation for relational tables;
- if the answer set is large (as in the case of query 1), there is a significant drop in response time for the top-k case;
- for each query, the response time increases while the retrieved records increase;

- the set size of reformulated queries $|r(\mathbf{q}, \mathcal{O})|$ may be non negligible, hence the redundancy elimination could remove almost one-third of the original set. Nevertheless the time needed for reduction phase is negligible;
- the database answering time increases with the number of top-k results to be retrieved.

Note that for query 9, most of time is spent for query reformulation phase.

Size 100000											
Query	All	top-1	top-10	$\|ans(\mathcal{K}, q)\|$	$\|r(\mathbf{q}, \mathcal{O})\|$	q_{DB}	t_{ref}	t_{red}	$t_{DB_{all}}$	t_{DB_1}	$t_{DB_{10}}$
1	7.507	2.489	2.581	3985	1599	1066	1.723	0.010	5.738	0.203	0.258
2	0.193	0.036	0.037	445	69	46	0.016	0.001	0.137	0.017	0.039
3	0.164	0.030	0.066	19	18	12	0.006	0.001	0.126	0.015	0.057
4	3.194	2.143	3.155	8	1242	552	2.072	0.015	1.075	0.106	0.875
5	0.348	0.067	0.186	19	75	50	0.027	0.001	0.300	0.021	0.141
6	0.114	0.052	0.102	93	18	12	0.005	0.001	0.088	0.036	0.082
7	0.207	0.053	0.146	445	69	46	0.013	0.001	0.166	0.014	0.118
8	3.090	2.353	3.080	21	1242	552	1.242	0.013	1.306	0.512	1.125
9	22.764	22.702	22.754	91	5175	2300	17.850	0.058	4.819	4.604	4.766
10	0.759	0.378	0.369	40	108	48	0.229	0.003	0.498	0.159	0.145
11	0.105	0.100	0.101	37	18	12	0.004	0.001	0.075	0.072	0.074
12	2.2	2.038	2.128	15	828	552	0.794	0.005	1.370	1.296	1.128
Average	3.834	3.0303	3.248	516.6	961.5	468.4	2.318	0.01	1.4053	0.601	0.761
Median	.0554	0.223	0.278	65.5	91.5	49	0.128	0.002	0.399	0.133	0.143

Table 1.1 Retrieval statistics

1.5 Related Work

In this Section we gather approaches to Human Resources retrieval related to our proposal, both in terms of research investigations and prototypes tools.

1.5.1 Research Approaches

Several approaches have been presented, where databases allow users and applications to access both ontologies and other structured data in a seamless way. Das et al. [7] developed a system that stores OWL-Lite and OWL-DL ontologies in Oracle RDBMSs, and that provides a set of SQL operators for ontology-based matching. *Jena 2 Ontology Stores* [23], *Sesame* [2] and *Oracle RDF Store* use a three columns relational table $\langle Subject, Property, Object \rangle$ to memorize RDF triples. In spite of the same storing model, these systems present different inference capabilities among them. Other ontology storage systems –such as *DLDB* [17] and *Sesame on Post-*

greSQL [2]– adopt binary tables. They create a table for each class in an ontology. A possible optimization is to cache the classification hierarchy in the database and to provide tables maintaining all the subsumption relationships between primitive concepts. This happens for example in *Instance Store* (***iS***) [1], a system for reasoning over OWL KBs specifically adopted in biomedical-informatics domains. ***iS*** is also able –by means of a hybrid reasoner/database approach– to reply to instance retrieval queries w.r.t. an ontology, given a set of axioms asserting class-instance relationships. A comparison between ***iS*** and the approach we present here show the former reduces instance retrieval to pure TBox reasoning and it is able to return only exact matches whilst we use an enriched relational schema storing only the ABox (*i.e.*, facts) in order to provide a logic-based ranked list of results and the not classified ontology. Other systems using RDBMS to deal with large amounts of data are *QuOnto*[4] and *Owlgres*[5]. They are DL-Lite reasoners providing consistency checking and conjunctive query services. Neither QuOnto nor OWLgres returns a ranked list of results. SHER ([9],[10]) is a highly-scalable OWL reasoner performing both membership and conjunctive query answering over large relational datasets using ontologies modeled in a subset of OWL-DL without nominals. It relies on an indexing technique summarizing database instances into a compact representation used for reasoning. It works by selectively uncompressing portions of the summarized representation relevant for the query, in a process called refinement. SHER uses Pellet to reason over the summarized data it enables motivations for data inconsistency. SHER allows for getting fast incomplete answers to queries, but it does not provide a ranked list of results.

Top-k queries [3] 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 [3] and new physical rank-aware operators [13] where rank relations are processed incrementally. RankSQL [14] 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 [16] 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 $[0, 1]$.

1.5.2 Commercial Tools

Currently, several commercial tools for talent management and e-recruitment are available. Most of them are enterprise suites supporting human resource management, including solutions that, even though improving the recruitment process by

[4] `http://www.dis.uniroma1.it/~quonto/`

[5] `http://pellet.owldl.com/owlgres/`

means of innovative media and tools, do not have a significant novelty charge. Available solutions in fact exploit databases to store candidate's personal and employment information, and do not ground on a logic-based structure. To the best of our knowledge, one of the few logic-based solutions for recruitment and referral process is STAIRS, a system in use at US Navy Department allowing to retrieve referral lists of best qualified candidates w.r.t. a specific mansion, according to the number of required skills they match. The commercial software supporting STAIRS is RESUMIX[6] an automated staffing tool making use of artificial intelligence techniques and adopted only as an internal tool. The system allows also to distinguish skills in *required* and *desired* ones in the query formulation: all required skills must be matched by the retrieved candidate, differently from *desired* ones. Among semantic-based tools for recruitment, noteworthy are products offered by Sovren[7], which provide solutions for parsing both CV and job requests starting from several text formats to HR-XML schema. A semantic-based matchmaking engine automatically returns a ranked list of best candidates given a job vacation, posted in any textual format. The matching process is by the way completely hidden to the recruiter, which does not receive any explanation about retrieved results. Sovren tools also manage the distinction between required and preferred skills in job offers posting.

References

1. Bechhofer, S., Horrocks, I., Turi, D.: The OWL Instance Store: System Description. In: The 20th International Conference on Automated Deduction (CADE '05), pp. 177–181 (2005)
2. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: The First International Semantic Web Conference (ISWC '02), pp. 54–68. Springer-Verlag (2002)
3. C. Li, K. C.-C. Chang, Ihab F. Ilyas and S. Song: RankSQL: Query Algebra and Optimization for Relational Top-k Queries. pp. 131–142. ACM (2005)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-06), pp. 260–270 (2006)
5. Chang, K.C.C., won Hwang, S.: Minimal probing: Supporting expensive predicates for top-k queries. In: SIGMOD Conference (2002). URL `citeseer.ist.psu.edu/ands02minimal.html`
6. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F.M., Ragone, A.: Semantic-based skill management for automated task assignment and courseware composition. Journal of Universal Computer Science **13**(9), 1184–1212 (2007)
7. Das, S., Chong, E.I., Eadon, G., Srinivasan, J.: Supporting ontology-based semantic matching in RDBMS. In: The 30th International Conference on Very Large Data Bases (VLDB'04), pp. 1054–1065. VLDB Endowment (2004)
8. Dodds, L.: An introduction to foaf (2004). `http://www.xml.com/pub/a/2004/02/04/foaf.html`

[6] `http://www.cpol.army.mil`

[7] `http://www.sovren.com/default.aspx`

9. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007) (2007)
10. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The summary abox: Cutting ontologies down to size. In: Proceedings of 5th International Semantic Web Conference(ISWC 2006), pp. 343–356 (2006)
11. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing). Springer (2004)
12. Iannella, R.: Representing vcard objects in rdf/xml (2001). `http://www.w3.org/TR/vcard-rdf`
13. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. Very Large Database Journal **13**(3), 207–221 (2004)
14. Li, C., Soliman, M.A., Chang, K.C.C., Ilyas, I.F.: RankSQL: supporting ranking queries in relational database management systems. pp. 1342–1345. VLDB Endowment (2005)
15. Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the semantic web. In: Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM-07), no. 4772 in Lecture Notes in Computer Science, pp. 16–30. Springer Verlag (2007)
16. P. Bosc and O. Pivert: SQLf: a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems **3**(1), 1–17 (1995)
17. Pan, Z., Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: The First International Workshop on Practical and Scalable Semantic Systems (PSSS1), vol. 89, pp. 109–113. CEUR-WS.org (2003)
18. Straccia, U.: Answering vague queries in fuzzy DL-Lite. In: Proceedings of the 11th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-06), pp. 2238–2245. E.D.K., Paris (2006)
19. Straccia, U.: Towards top-k query answering in deductive databases. In: Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC-06), pp. 4873–4879. IEEE (2006)
20. Straccia, U.: Towards top-k query answering in description logics: the case of DL-Lite. In: Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA-06), no. 4160 in Lecture Notes in Computer Science, pp. 439–451. Springer Verlag, Liverpool, UK (2006)
21. Straccia, U.: Towards vague query answering in logic programming for logic-based information retrieval. In: World Congress of the International Fuzzy Systems Association (IFSA-07), no. 4529 in Lecture Notes in Computer Science, pp. 125–134. Springer Verlag, Cancun, Mexico (2007)
22. Tinelli, E., Cascone, A., Ruta, M., Di Noia, T., Di Sciascio, E., Donini, F.M.: I.M.P.A.K.T.: an innovative, semantic-based skill management system exploiting standard SQL. In: 11th International Conference on Enterprise Information Systems (ICEIS'09), vol. AIDSS, pp. 224–229
23. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: The first International Workshop on Semantic Web and Databases (SWDB'03), pp. 131–150 (2003)

Appendix 1: Ontology axioms excerpts

Excerpt of Degree and Level classes hierarchy

```
(IMPLIES Engineering_Degree Degree)
(IMPLIES Computer_Science_Engineering Engineering_Degree)
(IMPLIES Managerial_Engineering Engineering_Degree)
(IMPLIES Electronics_Engineering Engineering_Degree)
(IMPLIES Mechanical_Engineering Engineering_Degree)
(IMPLIES Statistics Mathematical_Sciences)
(IMPLIES Mathematical_Sciences Degree)
(IMPLIES Management_degree Social_Studies)
(IMPLIES Social_Studies Degree)

(IMPLIES Doctoral_Degree Level)
(IMPLIES Master_Degree Level)
(IMPLIES Master_after_master Level)
(IMPLIES Secondary_School)
(IMPLIES Bachelor Level)
(IMPLIES CCDP Design_Certification)
(IMPLIES Design_Certification Cisco)
(IMPLIES Cisco Certification)
(IMPLIES Certification Level)
```

Excerpt of Job Title and Industry classes hierarchy

```
(IMPLIES Database_Administrator Computer_Specialist)
(IMPLIES Network_and_Computer_Systems_Administrator Computer_Specialist)
(IMPLIES Computer_Specialist Computer_and_Mathematical_Occupation)
(IMPLIES Computer_and_Mathematical_Occupation Job_Title)
(IMPLIES Training_and_Development_Manager Human_Resources_Manager)
(IMPLIES Human_Resources_Manager Manager)
(IMPLIES Project_Manager Manager)
(IMPLIES Manager Job_Title)
(IMPLIES Cost_Estimators Business_Operation_Specialist)
(IMPLIES Business_Operation_Specialist
 Business_and_Financial_Operations_Occupation)
(IMPLIES Business_and_Financial_Operations_Occupation Job_Title)
(IMPLIES Budget_Analysts Financial_Specialist)
(IMPLIES Financial_Specialist Business_and_Financial_Operations_Occupation)
(IMPLIES Patternmaker_Metal_and_Plastic
 Model_Makers_and_Patternmakers_Metal_and_Plastic)
(IMPLIES Model_Makers_and_Patternmakers_Metal_and_Plastic
 Metal_Workers_and_Plastic_Workers)
(IMPLIES Metal_Workers_and_Plastic_Workers Production_Occupation)
(IMPLIES Production_Occupation Job_Title)
(IMPLIES ProcessPlanner Job_Title)
(IMPLIES Teachers Job_Title)

(IMPLIES Banking_and_Consumer_Lending Banking)
(IMPLIES Banking Industry)
(IMPLIES IT_and_Telematics_Applications Industry)
(IMPLIES Business_Strategic_Management Business)
(IMPLIES Business Socio-economic_development_models_economic_aspects)
(IMPLIES Socio-economic_development_models_economic_aspects
 Social_and_Economics)
(IMPLIES Social_and_Economics Industry)
(IMPLIES Finance_Banking Financial_Services)
(IMPLIES Financial_Services Socio-economic_development_models_economic_aspects)
(IMPLIES Sales Industry)
(IMPLIES Engineering_Services Industry)
(IMPLIES Clothing_and_Textile_Manufacturing Industrial_Manufacture)
(IMPLIES Industrial_Manufacture
 Industrial_Manufacturing_Material_and_Transport)
(IMPLIES Industrial_Manufacturing_Material_and_Transport Industry)
```

Excerpt of Knowledge classes hierarchy

```
(IMPLIES Engineering_and_Technology Knowledge)
(IMPLIES Computer_Science_Skill Engineering_and_Technology)
(IMPLIES Artificial_Intelligence Computer_Science_Skill)
(IMPLIES Information_Systems Computer_Science_Skill)
(IMPLIES Java Object_Oriented_Programming)
(IMPLIES Cplusplus Object_Oriented_Programming)
(IMPLIES Visual_Basic Object_Oriented_Programming)
(IMPLIES Object_Oriented_Programming Programming_Languages)
(IMPLIES Programming_Languages Software_Development)
(IMPLIES Software_Development Computer_Science_Skill)
(IMPLIES Fuzzy Artificial_Intelligence)
(IMPLIES Data_Mining Artificial_Intelligence)
(IMPLIES Machine_Learning Artificial_Intelligence)
(IMPLIES Knowledge_Representation Artificial_Intelligence)
(IMPLIES Natural_Language Artificial_Intelligence)
(IMPLIES DBMS Information_Systems)
(IMPLIES Information_Systems Computer_Science_SKill)
(IMPLIES Internet_Technologies Computer_Science_SKill)
(IMPLIES Web_Technologies Development_Technologies)
(IMPLIES Workflow_Management Engineering_and_Technology)
(IMPLIES VBScript Script_Languages)
(IMPLIES Script_Languages Programming_Languages)
(IMPLIES Development_Technologies Software_Developmnent)
(IMPLIES Sales_and_Marketing Business_and_Management)
(IMPLIES Business_and_Management Knowledge)
(IMPLIES Administration_and_Management Business_and_Management)
(IMPLIES Economics_and_Accounting Business_and_Management)
(IMPLIES ProcessPerformanceMonitoring Managerial_Skill)
(IMPLIES Managerial_Skill Business_and_Management)
(IMPLIES Mathematics Mathematics_and_Science)
(IMPLIES Mathematics_and_Science Knowledge)
```

Excerpt of Complementary Skills and Language knowledge hierarchy

```
(IMPLIES Complex_Problem_Solving_Skills Cross_Functional_Skills)
(IMPLIES Cross_Functional_Skills Worker_Requirements)
(IMPLIES Worker_Requirements Skill)
(IMPLIES Cooperation Interpersonal_Orientation)
(IMPLIES Interpersonal_Orientation Work_Styles)
(IMPLIES Work_Styles Worker_Characteristics)
(IMPLIES Worker_Characteristics Skill)
(IMPLIES Leadership Social_Influence)
(IMPLIES Social_Influence Work_Styles)
(IMPLIES Learning_Strategies Process)
(IMPLIES Process Basic_Skills)
(IMPLIES Basic_Skills Worker_Requirements)
(IMPLIES Critical_Thinking Process)
(IMPLIES Monitoring Process)
(IMPLIES Visualization Spatial_Abilities)
(IMPLIES Spatial_Abilities Cognitive_Abilities)
(IMPLIES Cognitive_Abilities Abilities)
(IMPLIES Abilities Worker_Characteristics)
(IMPLIES Spatial_Orientation Spatial_Abilities)
(IMPLIES Verbal_Abilities Cognitive_Abilities)
(IMPLIES Systems_Skills Cross_Functional_Skills)
(IMPLIES Visual_Color_Discrimination Visual_Abilities)
(IMPLIES Visual_Abilities Sensory_Abilities)
(IMPLIES Sensory_Abilities Psychomotor_Abilities)
(IMPLIES Psychomotor_Abilities Worker_Characteristics)

(IMPLIES English Language)
(IMPLIES French Language)
```

```
(IMPLIES German Language)
```

Appendix 2: Example Candidate profiles set

1 - Mario Rossi

- *Degree*: Computer Science Engineering, mark 110
- *Level*: Secondary School (mark 60), Master Degree
- *JobTitle*: Database Administrator(4 years), Project Manager (2 years)
- *Industry*: Banking (4 years), IT and Telematics Applications (2 years)
- *Knowledge*: Cplusplus (5 years), Java (5 years), Visual Basic(5 years)
- *ComplementarySkill*: Cooperation (5 years), LeaderShip (5 years)
- *Language*: English (excellent writing, verbal and reading), French (good writing)

2 - Daniela Bianchi

- *Degree*: Computer Science Engineering (mark 110)
- *Level*: Secondary School (mark 60), Bachelor
- *JobTitle*: Database administrator (4 years), Project Manager (2 years)
- *Industry*: Banking (4 years), IT and Telematics Applications (2 years)
- *Knowledge*: Cplusplus (2 years), Java (6 years), Visual Basic (1 years)
- *ComplementarySkill*: Cooperation (5 years), LeaderShip(5 years)
- *Language*: English (excellent verbal, writing and reading), French (good writing)

3 - Lucio Battista

- *Degree*: Managerial Engineering (mark 104)
- *Level*: Secondary School (mark 60), Master Degree, CCDP
- *JobTitle*: Database Administrator (4 years), Project Manager (2 years)
- *Industry*: Banking (4 years), IT and Telematics Applications (2 years)
- *Knowledge*: DBMS (2 years)
- *ComplementarySkill*: Cooperation (5 years), LeaderShip (5 years)
- *Language*: English (excellent verbal, writing and reading), French (good writing)

4 - Mariangela Porro

- *Degree*: Managerial Engineering (mark 104)
- *Level*: Secondary School (mark 60), Master Degree, Master after master
- *JobTitle*: Database Administrator (4 years), Network computer systems Administrator (4 years)
- *Industry*: Banking (4 years), IT and Telematics Applications (2 years)
- *Knowledge*: DBMS (2 years), Web Technologies (2 years)
- *ComplementarySkill*: Learning Strategy (8 years)
- *Language*: English (good verbal, writing and reading)

5 - Nicola Marco

- *Degree*: Electronics Engineering (mark 104)
- *Level*: Bachelor, Master after Master
- *JobTitle*: Database Administrator (2 years), Network computer systems Administrator (2 years)
- *Industry*: Banking (4 years), IT and Telematics Applications (2 years)
- *Knowledge*: DBMS (5 years), Web Technologies (5 years)
- *ComplementarySkill*: Learning Strategy (8 years)
- *Language*: English (good writing, verbal and reading)

6 - Carla Buono

- *Degree*: Statistics (mark 106)
- *Level*: Master Degree, Master after Master
- *JobTitle*: Cost Estimator (4 years), Budget Analysts (10 years)
- *Industry*: Banking (4 years), Business Strategic Management (2 years), Finance Banking (1 years)
- *Knowledge*: Sales and Marketing (2 years), Administration and Management (4 years), Mathematics (10 years)
- *ComplementarySkill*: Critical thinking (8 years), monitoring (8 years)
- *Language*: English (excellent writing, verbal and reading knowledge), French (good writing knowledge)

7 - Marcello Cannone

- *Degree*: Management Degree (mark 106)
- *JobTitle*: Training and Development Manager (2 years)
- *Industry*: Sales, Banking and Consumer Lending
- *Knowledge*: Economics and Accounting (4 years), WorkflowManagement
- *ComplementarySkill*: Visualization, Spatial orientation, Verbal abilities
- *Language*: English, German (excellent writing and reading knowledge, basic verbal knowledge)

8 - Carmelo Piccolo

- *Degree*: Mechanical Engineering (mark 79)
- *JobTitle*: Patternmaker Metal and Plastic, Process Planner (6 years)
- *Industry*: Engineering Services (14 years), Clothing and Textile Manufacturing (11 years)
- *Knowledge*: VBScript, Process Performance Monitoring
- *ComplementarySkill*: Systems Skills, Complex problem solving (10 years), Visual Color Discrimination (14 years)
- *Language*: English (basic writing knowledge), French (excellent reading knowledge)

9 - Elena Pomarico

- *Degree*: Computer Science Engineering
- *Level*: Secondary School, Bachelor
- *JobTitle*: Database Administrator, Project Manager
- *Industry*: Banking, IT and Telematics Applications
- *Knowledge*: CplusPlus, Java, Visual Basic
- *ComplementarySkill*: Cooperation, Leadership
- *Language*: English (excellent writing, reading and verbal knowledge), French (good writing knowledge)

10 - Domenico De Palo

- *Degree*: Computer Science Engineering (mark 110)
- *Level*: Doctoral Degree
- *JobTitle*: Project Manager (4 years), Teachers (4 years), Database Administrator (4 years)
- *Knowledge*: OOprogramming (6 years), Artificial intelligence (4 years), Internet technologies (4 years)
- *ComplementarySkill*: Cooperation (6 years), Complex problem solving (5 years)
- *Language*: English (excellent verbal knowledge)

Index