Knowledge Compilation for Automated Team Composition Exploiting Standard SQL

Eufemia Tinelli
Politecnico di Bari
e.tinelli@poliba.it

Simona Colucci
Università della Tuscia
simona.colucci@unitus.it

Eugenio Di Sciascio
Politecnico di Bari
disciascio@poliba.it

Francesco M. Donini
Università della Tuscia
donini@unitus.it

ABSTRACT

Automatically finding suitable candidates in an organization to compose a team able to solve a given task is a typical problem in large companies. In this paper we present a Description Logics approach to *Team Composition* based on candidates technical knowledge and on tasks descriptions, modeled according to a skills ontology in $\mathcal{ALE}(\mathrm{D})$. The novelty of our approach is that our implemented service exploits standard-SQL querying expressiveness to emulate the proper reasoning procedures. The Team Composition service has been deployed as part of `I.M.P.A.K.T.`, a skill management system, and results show the effectiveness of the proposed approach.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalism and Methods**]: Representation Languages; H.3.3 [**Information Search and Retrieval**]: Query Formulation

General Terms

Algorithm, Management

Keywords

Description Logics, Semantic Team Composition, RDBMS, SQL

1. INTRODUCTION

We present a Semantic-based Automated Team Composition approach, deployed in the framework of a skill management system. While a number of semantic-based systems have been proposed in the recent past –see the following for a brief survey– their widespread use has been usually prevented by the huge computational resources needed. We face these issues combining the richness in informative content typical of semantic-based approaches with efficient data management and scalability characterizing the ones based on RDBMS. Our approach follows therefore a Knowledge Compilation scheme [5]. Knowledge Compilation has been employed to make computationally easier to reason over the information contained in a Knowledge Base (KB), by splitting the reasoning process in two phases:(i) a KB is pre-processed, thus parsing it in a proper data structure (*off-line reasoning*); (ii) the query is answered exploiting the structure coming from the first phase (*on-line reasoning*). The system adopts the conceptual approach presented in [11], by implementing DLs inference services specifically developed for knowledge management. In particular, we focus here on semantic-based *team composition*, within the framework of `I.M.P.A.K.T.`, an innovative system based on a hybrid approach combining semantics and database technologies [23], able to compute, beyond classical inference services such as satisfiability and subsumption, non standard inferences specifically designed for matching descriptions and logically rank them.

In order to fully model the features of the human resources management we deal with, at least the expressiveness of $\mathcal{ALE}(\mathrm{D})$ should be adopted. Nevertheless, existential qualification introduces a source of exponentiality in reasoning computational complexity both for standard and non-standard services, even for rather inexpressive DLs ([2], [3]). Therefore `I.M.P.A.K.T.` exploits the modeling potential of its underlying RDBMS framework to represent and manage all of the information needed to perform the addressed business processes, by adopting $\mathcal{FL}_0(\mathrm{D})$ for knowledge representation. Moreover, we adopt the classical KB definition $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$specifies the intensional knowledge (*i.e.*, our skills ontology), and the ABox $\mathcal{A}$ specifies the extensional one (*i.e.*, candidate profiles stored in our relational model).

Though several frameworks and systems have been conceived and developed for Skills management, we focus here on logic-based approaches only. The use of ontologies as knowledge repositories, widely recognized as useful to provide a common vocabulary and to model general Knowledge Management procedures [16], asks for an intense use of inference services ([18], [19]) to justify the computational cost of their performance. Although several semantic facilitators have been proposed in literature for several scenarios ([21, 22, 24, 13, 9]) often they do not fully leverage the ontological structure limiting their inferences to simple subsumption matching. [10] gathers several semantic-based approaches to retrieval, based on specifically devised non-standard services. Also in [4] an approach seemingly similar in overcoming exact matches is proposed. It extends the one for measuring similarities in ontologies by Ehrig et al. [15] for combining the advantages of similarity-based search with those of ontology-based systems. Nevertheless, such an approach does not allow to provide explanation of results in case of non-exact matches. Differently from the above mentioned approaches, a relevant aspect of our work is the exploitation of classical relational database systems and languages *i.e.*, SQL, for storing the reference ontology and to perform reasoning tasks. Recently, an approach aimed at classification in SQL databases of ontologies formalized in $\mathcal{ELH}$ has been presented [12]. Das et al. [7] developed a system that stores OWL-Lite KBs in Oracle RDBMSs providing a set of SQL operators for ontology-based matching. *QuOnto*[1] is other system using RDBMS to deal with large amounts of data modeled in DL-Lite. SHER [14] is a highly-scalabile OWL reasoner for large relational datasets modeled in a subset of OWL-DL without nominals. PelletDB[2] improves Pellet reasoner's OWL capabilities using Oracle Database 11g. All the cited systems, though often allow an expressiveness greater than the one enabled by the SQL engine proposed here, are only able to return either exact matches (*i.e.*, instance retrieval) or query answering. On the contrary, we use an enriched relational schema to provide non-standard inference services.

In fact the match between individual profiles we are interested in is, obviously, not an exact one, which is both quite simple and rare. Given a task description and individual profile descriptions, the matchmaking process has to return one or more best possible matches among the available ones. Such a set of potential candidates represents the beginning set of available profiles the *Team Composition* service works on. In particular, `I.M.P.A.K.T.` *TeamComposition* service supports the process of assigning more than one task to different groups of available individuals: a process we denote by *many to many* skill matching. Such service supports therefore a project manager by automatically providing a set of possible working teams for each task or project activity. A *many to one* skill matching, searching for ad-hoc team performing a single task has been presented in [8]. Here we propose a different composition strategy in order to take into account several criteria emerging in the new problem at hand: (i) format of the job request – split in soft and strict constraints; (ii) temporal availability of candidates; (iii) temporal relations among project activities – as described in Gantt diagrams.

The remaining of the paper is organized as follows: next section provides an overview of the proposed approach, both in terms of adopted knowledge modeling and of performed team composition strategy; in Section 3 we evaluate effectiveness of the proposed service carried out on PostgreSQL 8.4 DBMS; conclusions close the paper.

2. SYSTEM OVERVIEW

As hinted before, `I.M.P.A.K.T.` framework has to properly store candidate CV descriptions in order to efficiently perform reasoning services only via SQL standard queries over a relational database, fully mapping the KB. Before introducing the *Team Composition* process, we report on the services needed for its execution: KB compilation, candidates profiles storing and matchmaking.

2.1 Compiling $\mathcal{ALE}(\mathrm{D})$ into SQL

`I.M.P.A.K.T.` takes all the information needed to understand and manage the domain of human resources from a specifically developed modular ontology $\mathcal{T}$, modeled according to the $\mathcal{ALE}(\mathrm{D})$ formalism. Such an ontology includes nearly 5000 concepts modeling both the technical and the complementary skills –*i.e.*, personal and social abilities– a possible candidate may hold. All the classes useful to describe candidate CV belong to one of the ontology submodules M_i, with $i > 0$, where each module is modeled according to the formalism of $\mathcal{FL}_0(\mathrm{D})$ and may include the following items: i) a class hierarchy; ii) optional properties, defined over the classes specifying the module hierarchy; iii) optional concrete features p_i either in the natural numbers or in the calendar dates domain. Hereafter, the content modeled in each ontology module is shortly described: **`ComplementarySkill`** models the class hierarchy about complementary attitudes; **`Industry`** models the hierarchy of industries where a candidate may have worked; **`Knowledge`** models the hierarchy of possible candidate competence and technical tools usage ability; moreover, the module provides a `type` property to specify, for each competence, the related experience role (*e.g.*, developer, administrator, and so on); **`JobTitle`** models the hierarchy of possible job positions; **`Level`** models the hierarchy of candidate education and training levels: from basic education to `MasterDegree`, the whole candidate qualifications can be specified; **`Language`** models the hierarchy of possible languages known by the candidate and provides three concrete features to further classify language knowledge in `verbalLevel`, `readingLevel` and `writingLevel` and assign a reference level (from 1 – basic – to 3– excellent) to such language skills. Modules `Industry`, `ComplementarySkill`, `Knowledge` and `JobTitle` provide also a predicate `year` to specify the experience level (in years). Thanks to the knowledge modeling outlined so far, it is possible to model *Candidate Profiles* in the ABox. The CV classification approach we propose is based on a role-free ABox, which then includes only concept assertions of the form $P(a)$, stating that the candidate a (*i.e.*, her CV description) offers profile features P (see Definition 1).

Definition 1 (Profile)**.** *Given the skill ontology $\mathcal{T}$, a profile P is a $\mathcal{ALE}(\mathrm{D})$ concept defined as a conjunction of existential quantifications, $P = \sqcap(\exists R^0_j.C)$,with $j = 1 \ldots 6$ where R^0_j is a role we name **entry point** and C is a concept in $\mathcal{FL}_0(\mathrm{D})$ modeled in the ontology module M_j.*

As hinted before, the reasoning phase relies on the information gathered in the DBMS during the storage phase thanks to a proper E-R modeling. In particular, information storage involves both candidate profiles and TBox axioms, which could adopt the full expressiveness of $\mathcal{ALE}(\mathrm{D})$. Notice that all the reasoning services performed by `I.M.P.A.K.T.` process the atomic information composing the knowledge descriptions to be stored –rather then the concept as a whole– and then ask for the reduction in normal form of such descriptions. We recall that $\mathcal{FL}_0(\mathrm{D})$ concepts can be normalized according to the *Concept-Centered Normal Form* (CCNF), [1, Ch.2], through the recursive application of the formulas in Fig. 1, until no rule is applicable at every nesting level. A finite normal form is instead not available for $\mathcal{ALE}(\mathrm{D})$ concepts.

TBox reduction	Concept reduction
$A \rightarrow A \sqcap C$ if $A \sqsubseteq C \in \mathcal{T}$ $A \rightarrow C$ if $A = C \in \mathcal{T}$	$\forall R.(D \sqcap E) \rightarrow \forall R.D \sqcap \forall R.E$

Figure 1: Rules for $\mathcal{FL}_0(\mathrm{D})$ CCNF.

According to such an approach the Tbox $\mathcal{T}$ is mapped to the database according to the following design rules: 1) an entity CONCEPT is created to store the CCNF of all the $\mathcal{FL}_0(\mathrm{D})$ concepts included in the modules M_i with $i = 1, \ldots, 6$; 2) an entity $T_j(X)$, with $j = 1, \ldots, 6$ is created for each entry point R^0_j.

An excerpt of the consequent database logical model is shown in Fig. 2 to improve readability of the rest of the paper (we do not delve into details about the whole logical model, due to the lack of space).In order to populate the DB with the information in

[1]http://www.dis.uniroma1.it/ quonto/

[2]http://clarkparsia.com/pelletdb/

$CONCEPT(\underline{conceptID}, name, level)$

$PROFILE(\underline{profileID}, \cdots structured\_info \cdots)$

$HASKNOWLEDGE(\underline{profileID, groupID, conceptID}, value, lastdate)$

$AVAILABILITY(\underline{profileID}, start\_date, end\_date, activity\_cod)$

Figure 2: DataBase logic model excerpt

`Programming` $\sqsubseteq$ `KnowledgeType`

`ComputerScienceSkill` $\sqsubseteq$ `EngineeringAndTechnologies` $\sqsubseteq$ `Knowledge`

`Java` $\sqsubseteq$ `OOP` $\sqsubseteq$ `ProgrammingLanguage` $\sqsubseteq$ `SWDevelopment` $\sqsubseteq$ `ComputerScienceSkill`

`MySQL` $\sqsubseteq$ `RDBMS` $\sqsubseteq$ `OpenSourceDBMS` $\sqsubseteq$ `DBMS` $\sqsubseteq$ `InformationSystem` $\sqsubseteq$ `ComputerScienceSkill`

Figure 3: The ontology sketch used as reference in examples

the assertional component $\mathcal{A}$ of the KB, `I.M.P.A.K.T.` needs to store all the candidate profiles, modeled according to Definition 1 in the relational schema. In Fig. 2 also the entities needed to store all features of profile $P(a)$ are modeled: the `Profile` entity includes the so called *structured information*, exploited to take into account extra-ontological content, such as personal data (*e.g.*, last and first name, birth date) and employ information (*e.g.*, preferred working hours, car availability). Of course, each individual a is involved in one assertion only, while the same feature P could be offered by more than one candidate; the other six modeled entities derive from the second design rule and are one-one with the entry points.

In order to store a profile $P = \sqcap(\exists R^0_j.C)$ – with $j = 1 \dots 6$– for each conjunct $\exists R^0_j.C$, `I.M.P.A.K.T.` adds one tuple for each atom of the CCNF of C to the related entity $T_j(X)$ in the database (we recall that the logic model includes one table for each entry point –HASCOMPLEMENTARYSKILL, HASINDUSTRY and so on– even though in Fig. 2 we only show HASKNOWLEDGE). Each tuple X corresponds to a set of attributes $X = \{profileID, groupID, conceptID, value, lastdate\}$, where:

– **profileID** stores unique identifier assigned to a profile $P(a)$;

– **groupID** identifies each conjunct $\exists R^0_j.C$ and assigns a label to the CCNF of C: its atoms could be distributed in different tuples of the table itself but they share the same unique `groupID`;

– **conceptID** stores unique identifiers assigned to the atoms deriving from the CCNF of C (we recall that C is defined as an $\mathcal{FL}_0$(D) concept);

– **value** is not $NULL$ iff `conceptID` specifies any features;

– **lastdate** is not $NULL$ iff `conceptID` refers to the feature $years$ and it stores the last temporal update of work experience.

The set of tuples $T_j(X)$, $j = 1 \dots 6$, related to the same profile, identified by `profileID`, convey all of the information in the profile identified by `profileID`. Finally, AVAILABILITY table is defined to store candidates temporal constraints needed during the team composition process as reported in Section 2.2.

A toy profile description is proposed hereafter. Let us suppose *Robert* is a candidate; his profile can be described as: *"Robert speaks English with a basic 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 until December 10, 2010) and he is two years experienced in MySQL DBMS (until July, 2011), ...".* Robert profile can be represented according to Definition 1: - ***hasLevel*** - Computer Science Engineering Degree (final mark = 110); - ***hasKnowledge*** - Java (knowledge type = programming, years of experience = 5, last update = 2010-12-10); MySQL (years of experience = 2, last update = 2011-07-31); - ***knowsLanguage*** - English (verbalLevel = 1, writingLevel = 2).

We can describe the *Robert*'s profile as P features conjunction in $\mathcal{ALE}$(D): $\exists hasLevel.(ComputerScienceEngineeringDegree \sqcap =_{110} mark) \sqcap \exists hasKnowledge.(Java \sqcap \forall skillType.Programming \sqcap =_5 years \sqcap =_{2010-12-10} lastdate) \sqcap \exists hasKnowledge.(MySQL \sqcap =_2 years \sqcap =_{2011-07-31} lastdate) \sqcap \exists knowslanguage.(English \sqcap =_1 verbalLevel \sqcap =_2 writingLevel)$. P will be normalized and split in components. So it is stored in the three tables: HASKNOWLEDGE, HASLEVEL and KNOWSLANGUAGE.

With reference to the model in Fig. 2 and ontology sketch in Fig. 3, tuples stored only for the conjunct $\exists hasKnowledge.(Java \sqcap \forall skillType.Programming \sqcap =_5 years \sqcap =_{2010-12-10} lastdate)$ are reported in Fig. 4. We outline that obviously conjunct $\exists hasKnowledge.(MySQL \sqcap =_2 years \sqcap =_{2011-07-31} lastdate)$ of the Robert's profile will have a `groupID` value different from 1 in table HASKNOWLEDGE.

More formally, a *Job Request* $\mathcal{F}$ is defined as follows:

Definition 2 (Job Request)**.** *A* Job Request $\mathcal{F}$ *is a Profile (according to Definition 1), $\mathcal{F} = \sqcap(\exists R^0_j.C)$, defined as a pair of feature sets, $\mathcal{F} = \langle \mathcal{FS}, \mathcal{FP} \rangle$ such that:*
— $\mathcal{FS} = \{fs_i | i = 1, \dots, s\}$ is a set of strictly required features fs_i, of the form $\exists R^0_j.C_i$;
— $\mathcal{FP} = \{fp_k | k = 1, \dots, p\}$ is a set of preferred features fp_k, of the form $\exists R^0_j.CCNF(C_k)$.

According to such a distinction in preferences and strict constraints, `I.M.P.A.K.T.` performs a two-step matchmaking approach –we refer to it as $Matchmaking$ process in the rest of the paper– which starts with a process – namely *StrictMatch* – computing a set of profiles fully satisfying strict requirements and proceeds with a process – namely *SoftMatch*– trying to approximately match preferences with profiles belonging to the set returned by *StrictMatch*. In particular, `I.M.P.A.K.T.` maps Job Requests to standard SQL queries. Notice that we do not use a specific preference language as in [17, 6, 20] but we only exploit a set of *ad-hoc* SQL queries, built on-the-fly according to user requirements.

2.2 Team Composition

On the basis of the information storage process detailed so far, `I.M.P.A.K.T.` performs team composition by implementing Algorithm 1. It takes as input the set $\mathcal{P} = \{P(a_1), \dots, P(a_P)\}$ of available candidate profiles and the set $\mathcal{PA} = \{PA_1, \dots, PA_A\}$ of project activities to be assigned. Each project activity description PA_i is composed by a description K_i of the knowledge required for the task, a set of temporal constraints D_i, and a number of required team members m_i; formally $PA_i = \langle K_i, D_i, m_i \rangle$. We notice that both K_i and P are described according to Definition 1 and that K_i conjuncts of the form $\exists R^0_j.C$ employ only the entry point $R^0_j = hasKnowledge$ (*i.e.*, currently *TeamComposition* involves only technical skills in the search process).

Algorithm 1 Teams Composition

```
Require: $\mathcal{PA} = \{PA_i\}, \mathcal{P} = \{P(a_1), \dots, P(a_P)\}$
Ensure: $Results = \langle PA_i, \{P(a_1), \dots, P(a_m)\} \rangle$
1: $Results := \emptyset$
2: $Assignments := \emptyset$
3: for each $PA_i = \langle K_i, D_i, m_i \rangle$ s. t. $K_i = \exists hasKnowledge.C_1 \sqcap \dots \sqcap \exists hasKnowledge.C_J$ do
4:     $PM_i := pmatrix(\mathcal{P}, K_i)$
5:     $Assignments := Assignments \cup \langle PA_i, TaskTeams(PM_i, J, m_i, 0) \rangle$
6: end for
7: $Results := CSPsolver(Assignments, \{D_i\})$
```

Algorithm 1 returns the set $Results$ of possible solutions as pairs $\langle PA_i, \{P(a_1), \dots, P(a_m)\} \rangle$ of team member profiles $P(a_g)$

concept		
conceptID	*name*	*level*
1	mark	null
2	year	null
3	skillType	null
4	lastdate	null
5	skillType.Programming	2
6	skillType.KnowledgeType	1
7	Knowledge	1
8	EngineeringAndTechnologies	2
9	ComputerScienceSkill	3
10	SWDevelopment	4
11	ProgrammingLanguage	5
12	OOP	6
13	Java	7
...	...	...

hasKnowledge				
profileID	*groupID*	*conceptID*	*value*	*lastdate*
100	1	13	null	null
100	1	12	null	null
100	1	11	null	null
100	1	10	null	null
100	1	9	null	null
100	1	8	null	null
100	1	7	null	null
100	1	6	null	null
100	1	5	null	null
100	1	2	5	2010-12-10
...	...	...	...	...

profile			
profileID	*first_name*	*second_name*	...
100	Robert	Wane	...
...	...	...	...

Figure 4: Tables filled to store one feature of Robert profile ($profileID = 100$)

assigned to each PA_i in $\mathcal{PA}$. Coherently with an Open World Assumption, it allows not only to find a team that, based on provided skills descriptions, fully covers K_i, but also teams only approximating such a full cover. Moreover, the algorithm considers equivalent all possible solutions – several combinations of candidates allocation may exist– and leaves the selection of the most proper set of assignments up to the Project Manager, given the high level of subjectivity of the task.

Given a set of project activities $PA_i = \langle K_i, D_i, m_i \rangle$, *Team-Composition* process is basically performed according to the following steps:
1. for each PA_i the candidates availability is checked by considering tuples in the AVAILABILITY table – specifically defined in the DB to store candidates temporal constraints. Only candidate profiles satisfying the constraints in D_i are returned;
2. the *Matchmaking* process is performed by taking as input the set of profiles returned by the previous step and a job request $\mathcal{F}$ including only *hasKnowledge* entry point, accordingly with the above introduced considerations on K_i;
3. for each PA_i the set of all candidate teams ($Assignment = \{Assignments(PA_i)|i = \ldots, A\}$) is computed by taking into account both the need for covering as much as possible required skills and the required number of team members m_i; each solution $Assignments(PA_i)$ is a set of possible teams covering PA_i, computed without taking into account the need for executing $PA_j \in \mathcal{PA}$ with $i \neq j$
4. a Constraint Satisfaction Problem (CSP)[25] solver computes the set of all possible solution teams solving the whole set of project activities $\mathcal{PA}$; such solutions are obtained by considering the elements in the set $Assignment$ returned at *step 3* as variables and temporal information in D_i as constraints, for each activity PA_i. The final goal is obviously that of returning an assignment set such that concurrent activities never involve the same profile.

Before delving into details of Algorithm 1, we need to provide the following definition.

Definition 3 (Profiles Matrix). *Let $PA_i = \langle K_i, D_i, m_i \rangle$ be a required task description and let $Profiles$ be the set of profiles $P(a_i)$ satisfying temporal constraints D_i.*
Let now $\mathcal{P}_M = P_1, \ldots, P_p = Matchmaking(K_i, Profiles)$ be a set of available profiles matching at least one features in K_i, with each profile modeled according to Definition 1.
Given the $Matchmaking$ process, we denote by ***Profiles Matrix*** *$PM = (pm_{gj})$, with $g = 1 \ldots p$ and $j = 1 \ldots J$ with J equal to the number of conjuncts in K_i ($K_i = \sqcap \exists hasKnoweldge.C_1 \sqcap \ldots \sqcap \exists hasKnoweldge.C_J$), the matrix defined as follows:*
— $pm_{gj} = 1$ if $P_g \in Matchmaking(\exists hasKnoweldge.C_j, Profiles)$, i.e., *if the profile P_g is returned in the results set of $Matchmaking$ process called over the j-th features of K_i;*
— $pm_{gj} = 0$ otherwise.

By looking at Algorithm 1, the first two steps introduced before are performed by a function – $pmatrix(\mathcal{P}, K_i)$– returning a PM *Profile Matrix* (see Definition 3) defined over the whole set of stored profiles $\mathcal{P}$, for each project activity PA_i.

The recursive algorithm $TaskTeam$ takes as input a Profile Matrix PM, the number J of required features in K_i, the required number of members m_i and the number of members m already added at the stage of the specific recursive call (Algorithm 1 calls in fact $TaskTeam$ with $m = 0$). $TaskTeam$ returns a candidate teams set, $Assignments$, as set of possible candidate teams covering PA_i. We do not delve into algorithm details but we just hint that the composition strategy relies on the general assumption that the minimum set of profiles is assigned to each task, while still trying to maximize coverage of required skills.

Final $Results$ set is computed by invoking a CSP engine, namely $CSPsolver$ in row 7, which takes as input the set $Assignements$ built upon the results returned by *TaskTeam* and the set of temporal constraints and returns a set of possible solutions. Each solution is composed by one team for each activity and is such that concurrent activities never involve the same profile.

The engine works by building a graph G whose nodes (variables) are project activities and edges (constraints) represent temporal overlap between them. On the basis of G, the $CSPsolver$ builds the search tree ST for computing the problem solutions. In particular, a G node is an activity whereas a ST node is an assignment for such an activity. To this aim, $CSPsolver$ exploits the three different strategies described in the following:

Degree Heuristic - it is useful for selecting the first node from G in order to build the related ST. The heuristic consists in detecting G nodes having the greater degree, so that the activity PA_i involving the biggest number of constraints is chosen. Then one ST for each assignment of the selected PA_i is built;

Forward Checking - for each node n_i in ST, the strategy considers all of the activities connected to n_i in G and discards candidate teams including profiles already selected by n_i. If all of the involved activities run out of candidate teams, then the ST branch coming from n_i does not admit solutions. In particular, for each ST node, the engine traces out the discarded assignments in order to provide a possible explanation in case of no solutions;

Minimum Remaining Values (MRV) - after each execution of *Forward Checking* , candidate teams list is uploaded and then MRV technique is exploited in order to define the next activity to consider. The preferred candidate activity is the one having less remaining candidate teams, in order to reach a possible failure as fast as possible.

For each node added to ST, the engine executes Forward Checking and MRV for computing all the solutions trees ST.

3. SYSTEM BEHAVIOR

I.M.P.A.K.T. is a client-server application developed in Java and it is built upon the open source database system PostgreSQL 8.4[3]. Moreover, compliance with standard SQL makes our application available for a broad variety of platforms.

We notice that the Matchmaking service underlies the Team Composition one. For this reason, as preliminary investigation we ground the whole system performance evaluation on Skill Matching, and we show system behavior for Team Composition service. We here show I.M.P.A.K.T. working mode through an example query. I.M.P.A.K.T. provides a GUI specifically built to compose the inputs to *Team Composition* algorithm. In particular panels (a)-(b) of Fig. 5 show a *project manager* request for a work team to employ in a new project composed by 3 activities:

A1 – Architecture Design: *Start Date*: 2011-09-01; *End date*: 2011-11-30; *Team*: from 2 to 3 candidates; *Skill*: Modeling tool (preference), Software Development (preference)

A2 – Data Layer: *Start Date*: 2011-10-15; *End date*: 2011-12-20; *Team*: 2 candidates; *Skill*: Object Oriented Programming (preference), DBMS (preference)

A3 – Layer Implementation: *Start Date*: 2011-12-15; *End date*: 2012-04-30, *Team*: from 2 to 3 candidates, *Skill*: J2EE (strict), Hibernate (preference)

I.M.P.A.K.T. has been ran upon an Intel Dual Core server, equipped with a 2.6 GHz processor and a 3 GB RAM. The application has been tested with several queries having different expressiveness applied to a number of data sets, to obtain a comprehensive evaluation of the approach. Our tests measure the retrieval time calculated as average time over ten repetitions and they include also the request normalization process time. In order to perform our experiments, we implemented a synthetic KB instances generator. In this way, we have available data sets having different size, ranging from 500 to 3000 profiles, where each profile has at least 10 features according to a given format (*i.e.*, number of features for each relevance level, at least two specified technical skill, number of numeric restrictions, etc.). We outline that actual semantic profiles can be imported by the editing I.M.P.A.K.T. GUI specifically built for inserting both structured information and ontologically one by means of ontology browsing.

Tests have been performed composing several kinds of queries including: (i) only strict requirements represented by either generic concepts or features with an higher specificity; (ii) only preferences represented also in this case by either generic concepts (see skills of activities $A1$ in the previous example) or features with an higher specificity (*e.g.*, $Java$, $Oracle11g$, $J2EE$); (iii) a combination of them (see skills of activities $A3$). Results show that queries of type (i), calling only the *Strict Match* procedure, have execution times by far lower than *Soft Match* ones, computed on the same requirements set as preferences. It is an expected result because the *Strict Match* both by construction performs a number of conditions/queries smaller than *Soft Match* and manages a number of intermediate results generally limited according to the strategy proposed in [23]. For this reason, retrieval times of skill request K_3 in Table 1 are lower than other request. Moreover, considering only queries (ii) our tests show that retrieval times moderately increase with the complexity of queries addressed to the system (see skill

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

Table 1: I.M.P.A.K.T. Skill Matching times (in ms)

No. profiles	500	1000	2000	3000
$K_1 = \{ModelingTool, SWDevelopment\}$	109	208	316	450
$K_2 = \{OOP, DBMS\}$	132	228	330	465
$K_3 = \{J2EE, Hibernate\}$	89	116	150	201

requests K_1 and K_2 in Table 1 where K_2 containing features more specific than K_1 ones).

Observe that, by using I.M.P.A.K.T. GUI in Fig. 5, the project manager can describe each activity by means of the technical knowledge needed to solve it. It is also possible to set such required competence as strict requirements or preferences. For each project activity, panel (d) in Fig. 5 enables ontology browsing only for defining required knowledge. In particular, (c), (d) and (e) panels provide a sort of keyword-based search on ontology entities, ontology classes and properties browsing and features editing respectively. Panel (b) is introduced in order to edit all the activity features: name, start date, end date, number of team member. Moreover, panel (a) shows an overview about all the selected skills for each activity. It is noteworthy that by adding a required knowledge in panel (e), I.M.P.A.K.T. automatically performs the $Matchmaking$ process and then it highlights by means of a red icon those skills not covered by any profiles. In our example, panel (a) shows that the knowledge of *Modeling Tool* cannot be satisfied.

With reference to an example dataset, team solutions are presented in panel (b) of Fig. 6. In particular, the figure shows team members assigned by *Solution 1*. We point out that, for example, *Daniela Bianchi* is assigned to activities $A1$ and $A3$ because they are not overlapped, whereas *Mario Rossi* and *Lucio Battista* are the selected team members for activity $A2$ for the experience in object oriented programming (*i.e.*, Java, C++ and Visual Basic) and DBMS respectively as showed in their "‘Compentences"’ list in panel (c) of Fig. 6.

For each activity, the previous panel presents candidate lists to solve the task, and I.M.P.A.K.T. GUI also supports team completion process in case of selection of teams smaller than required (*i.e.*, teams indicated with a "warning" icon in the same panel). The team completion process is performed using again the CSP solver having as input the same temporal constraints but different variables. An incomplete assignment means that no other candidate covering both the required skills and the selected temporal view exists. Hence, in order to select candidates among the available ones, several strategies can be tested. I.M.P.A.K.T. currently prefers candidates with the highest rank value returned by the Matchmaking process and, among equally ranked ones, it chooses candidates with the highest number of additional skills w.r.t. the required ones.

4. CONCLUSION

With the help of I.M.P.A.K.T. prototype, we proposed a tractable algorithm to perform an automatic semantic-based team composition and showed how it is possible to reproduce over a relational database the same problem expressiveness of an ontology-based representation. The approach is based on candidate profiles and project activities described in $\mathcal{ALE}$(D) DLs. It is noteworthy that the implemented service exploits standard-SQL querying expressiveness to simulate the proper reasoning procedures, thus obtaining reasonable results in terms of scalability and turnaround times. Moreover the composition strategy is general purpose enough to adopt it in constraint-based environments containing semantically annotated resources. Currently, we are investigating both a full database optimization and a more comprehensive experimental

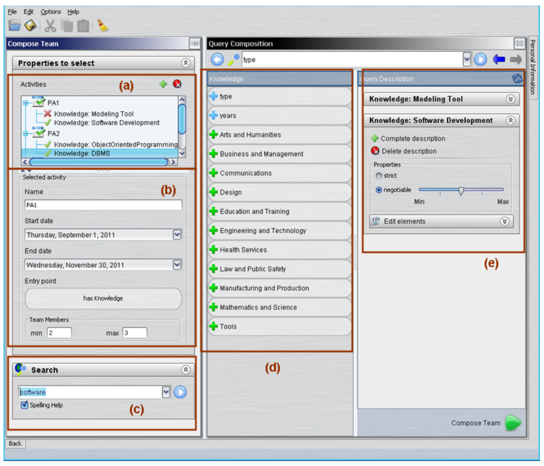

Figure 5: Activities definition GUI

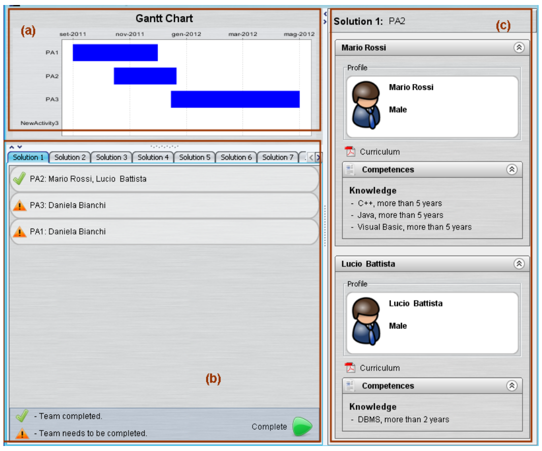

Figure 6: Team Composition results GUI

evaluation of the algorithms used for the *Team Composition* process.

5. ACKNOWLEDGMENTS

The authors acknowledge partial support of Apulia Region Strategic Project PS_025 and of ECTP INTERREG project GAIA, and are grateful to Antonio Cascone for SW implementations.

6. REFERENCES

[1] F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.

[2] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI'99*, pages 96–101, 1999.

[3] F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *KI '02*, volume 2479 of *LNAI*, pages 99–113, 2002.

[4] E. Biesalski and A. Abecker. Similarity measures for skill-profile matching in enterprise knowledge management. In *ICEIS 2006*, pages 11–16, 2006.

[5] M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Commun.*, 10(3-4):137–150, 1997.

[6] J. Chomicki. Querying with Intrinsic Preferences. In *EDBT 2002*, pages 34–51. Springer-Verlag, 2002.

[7] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An Efficient SQL-based RDF Querying Scheme. In *VLDB'05*, pages 1216–1227. VLDB Endowment, 2005.

[8] S. Colucci, T. Di Noia, E. Di Sciascio, F. Donini, G. Piscitelli, and S. Coppi. Knowledge Based Approach to Semantic Composition of Teams in an Organization. In *SAC 2005*, pages 1314–1319. ACM, 2005.

[9] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. In *ICEC '04*, pages 41–50. ACM, 2004.

[10] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and A. Ragone. Semantic-based skill management for automated task assignment and courseware composition. *J.UCS*, 13(9):1184–1212, 2007.

[11] S. Colucci, E. Tinelli, F. M. Donini, and E. Di Sciascio. Automating competence management through non-standard reasoning. *EAAI*, 24(8):1368–1384, 2011.

[12] V. Delaitre and Y. Kazakov. Classifying ELH ontologies in SQL databases. In *OWLED '09*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[13] T. Di Noia, E. Di Sciascio, and F. M. Donini. Extending semantic-based matchmaking via concept abduction and contraction. In *EKAW '04*, volume 3257 of *LNAI*, pages 307–320. Springer, 2004.

[14] J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, and L. Ma. Scalable semantic retrieval through summarization and refinement. In *AAAI 2007*, pages 299–304, 2007.

[15] M. Ehrig, P. Haase, N. Stojanovic, and M. Hefke. Similarity for ontologies — a comprehensive framework. In *ECIS '05*.

[16] C. W. Holsapple and K. D. Joshi. A formal knowledge management ontology: Conduct, activities, resources, and influences: Research articles. 55(7):593–612, 2004.

[17] W. Kießling. Foundations of Preferences in Database Systems. In *VLDB'02*, pages 311–322, 2002.

[18] D. OŠLeary. Enterprise knowledge management. *IEEE Computer*, 31(3):54–61, 1998.

[19] D. OŠLeary. Using AI in knowledge management: Knowledge bases and ontologies. *IEEE Intelligent Systems*, 13(3):34–39, 1998.

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

[21] S. Staab, H. Schnurr, R. Studer, and Y. Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[22] Y. Sure, A. Maedche, and S. Staab. Leveraging Corporate Skill Knowledge - From ProPer to OntoProPer. In *PAKM*, pages 30–31, 2000.

[23] E. Tinelli, A. Cascone, M. Ruta, T. Di Noia, E. Di Sciascio, and F. M. Donini. I.M.P.A.K.T.: An Innovative Semantic-based Skill Management System Exploiting Standard SQL. In *ICEIS 2009*, pages 224–229, 2009.

[24] D. Trastour, C. Bartolini, and C. Priest. Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In *WWW '02*, pages 89–98. ACM, 2002.

[25] E. P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.