Reasoning in resource-constrained environments: a matchmaking engine over relational Knowledge Bases

Eufemia Tinelli[1], Francesco M. Donini[2], Michele Ruta[1], and Eugenio Di Sciascio[1]

[1] Politecnico di Bari, via Re David 200, I-70125, Bari, Italy
{e.tinelli,m.ruta,disciascio}@poliba.it
[2] Università della Tuscia, via S. Carlo 32, I-01100, Viterbo, Italy
donini@unitus.it

Abstract. We present a framework for logic-based matchmaking on ALN ABoxes stored in a relational database. The proposed approach allows both non-standard reasoning and subsumption check be performed only via standard SQL queries. Main contribution is in the SQL implementation of the following features: (i) compliance with four match classes (*i.e.*, exact, full, partial and potential); (ii) rank computation for each matching outcome and (iii) preferences management in the user query. Performance evaluation carried out on a PostgreSQL 8.4 engine reports reasonable results in terms of scalability and turnaround times for large scale data sets.

1 Introduction

Benefits introduced by semantic technologies are well-known in a number of frameworks where simplistic keyword-based searches are not enough. Inference services, both standard and non-standard [11], allow to match requests and resources based on the actual meaning of their descriptions and –more interesting– to provide classification and logic-based ranking. Beyond obviously good matches, such as *exact* or *full* ones, we deem so called *potential or intersection* matches (where requests and supplied resources have something in common and no conflicting characteristics) as more interesting and useful from the user perspective. *Partial or disjoint* matches (where requests and supplies have some conflicting features) can also be considered worthwhile in all scenarios when nothing better exists. In those cases, one can be interested in understanding the conflict degree between perspective matching descriptions. What usually prevents a widespread usage of semantic approaches is that they require heavy computational capabilities, and response times are often unacceptable in common applications as soon as real (or realistic) data sets have to be faced. Furthermore, current systems usually allow a requester only to express her mandatory requirements and there is no possibility to grade user preferences in a more fine grained way. The problem of finding efficient reasoning strategies has been widely studied (see [8, 20, 14] among others). Basically, Knowledge Compilation [7] has been employed for making computationally acceptable the reasoning, splitting query answering in two phases: (i) 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*).

This paper presents an automated matchmaking framework, which exploits Knowledge Representation (KR) and reasoning techniques as well as Description Logics (DLs) formalisms, to retrieve the *best* supplied resources w.r.t. a user request, ranked according to the semantic distance from the request itself. Knowledge Bases (KBs) –stored in a relational database– are used, so that inferences are performed via standard SQL queries. The proposed matchmaker leverages KB pre-processing to reduce on-line reasoning overhead. Relevant provided features include: (i) it copes with several match classes; (ii) it allows to assign a relevance degree to each feature in the user query and (iii) it is able to return a logic-based explanation of the ranking results. The paper presents both the modeling approach allowing to translate a given KB into the reference relational database and the incremental building of SQL sub-queries allowing to matchmake and rank results. An experimental evaluation –using PostgreSQL 8.4 DBMS– has been carried out, showing the effectiveness of the proposal and its scalability. Matchmaker performances have been compared with the ones provided by *MaMaS-tng*[3] reasoner with reference to the same set of non-standard inference services [10].

The remainder of the paper is organized as follows. In the next section, a survey of most significant related work is presented; subsequently, Section 3 introduces the proposed framework and approach and Section 4 reports on a performance evaluation of the implemented approach. Conclusions and future research directions close the paper.

2 Background

Several systems and approaches have been presented in literature, where database technology is used to both persistently store knowledge and make scalable queries on it [5, 18]. They are mainly classified according to the language (*i.e.*, RDF(S) [4] or OWL [5]) they adopt for defining ontologies. In what follows, most relevant frameworks will be surveyed to allow a comparison with the approach we propose here.

Oracle Spatial 11g[6] is the first enterprise-oriented, scalable and reliable data management platform for RDF-based applications. It supports query answering for RDF(S) and OWLPrime. Based on a graph data model, RDF triples are made persistent, indexed and queried, similarly to other object/relational data types. *Owlgres*[7] is a DL-Lite [9] reasoner implementation for PostgreSQL. A distinguishing feature is that, along with standard inferences (*e.g.*, subsumption), it supports conjunctive query answering over ABoxes in a secondary storage (typically an RDBMS) so coping with large datasets. A comparable system using RDBMS to deal with large sets of data is *QuOnto*[8], a DL-Lite reasoner providing consistency check and conjunctive query replying services. Neither QuOnto nor OWLgres return a ranked list of results. Further ontology storage systems –such as *DLDB* [19] and *Sesame on PostgreSQL* [6]– adopt binary tables, one

[3] http://sisinflab.poliba.it/MAMAS-tng/
[4] http://www.w3.org/TR/rdf-primer/
[5] http://www.w3.org/TR/owl2-overview/
[6] http://www.oracle.com/technology/tech/semantic_technologies/index.html
[7] http://pellet.owldl.com/owlgres/
[8] http://www.dis.uniroma1.it/~quonto/

for each class in the TBox; whereas SOR (Scalable Ontology Repository) [17] exploits four kinds of tables for managing OWL-Lite constructs: atomic tables (for primitive concepts and properties), TBox axiom tables, ABox fact tables and class constructor tables. But the most popular and recent OWL storage is OWLIM [15]. It is a Sesame plug-in able to add a robust support for the semantics of RDFS, OWL Horst and OWL2 RL. A possible optimization is obtained by caching the classification hierarchy in the database as it is implemented in *Instance Store* (***iS***) [4], an engine for reasoning over OWL KBs specifically adopted in biomedical-informatics. A highly-scalable OWL reasoner is SHER (Scalable Highly Expressive Reasoner) [13] enabling conjunctive query answering. It supports a subset of OWL-DL excluding nominals, and it relies on an indexing technique of ABox instances in the database. SHER embeds Pellet to infer implicit information from indexed data and to obtain explanations for inconsistencies. PelletDB[9] provides an OWL 2 reasoning system specifically built for enterprise semantic applications. It combines Pellet's OWL capabilities and scalable native reasoning of Oracle Database 11g so ensuring performance improvements w.r.t. to the use of such technologies separately. Differently from the previous approaches, the most widespread DL-reasoner, *i.e.*, KAON2[10], does not implement the tableaux calculus, but it reduces a SHIQ(D) knowledge base to a disjunctive datalog program. An inference engine for answering conjunctive queries has been so developed applying well-known deductive database techniques.

All the cited systems, although often allow an expressiveness greater than the one enabled by the 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 a logic-based ranked list of results and the possibility to implement a semantic explanation of outcomes.

3 Proposed Approach

Description Logics are the reference formalisms we adopt in this paper. In particular, we refer to (a syntactic variant of) $\mathcal{ALN}$, whose allowed constructs are: conjunction $C \sqcap D$, universal quantification $\forall R.C$, and unqualified number restriction $(\geq\ nR), (\leq\ nR)$. A simple terminology $\mathcal{T}$ is hypothesized which contains inclusion axioms $A \sqsubseteq C$, concept definitions $A = C$, and disjointness axioms $A \sqcap B \sqsubseteq \bot$. If both the requested and the supplied resources are expressed in $\mathcal{ALN}$ w.r.t. an ontology $\mathcal{T}$, it is possible to exploit their formal semantics during the classification and matching processes. Recall that (see [12, 16] for further details) given a TBox $\mathcal{T}$, a *match degree* between a request D and a supplied resource C (both expressed w.r.t. $\mathcal{T}$) can be evaluated as:

– Exact. All the features requested in $\mathcal{D}$ are exactly provided by $\mathcal{C}$, and vice versa—in formulae, $\mathcal{T} \models \mathcal{D} \Leftrightarrow \mathcal{C}$.

– Full-Subsumption. All the features requested in $\mathcal{D}$ are contained in $\mathcal{C}$—in formulae, $\mathcal{T} \models \mathcal{C} \Rightarrow \mathcal{D}$.

– Potential-Intersection. There is a nonempty intersection among the features offered in $\mathcal{C}$ and the ones requested in $\mathcal{D}$—in formulae, $\mathcal{T} \not\models \neg(\mathcal{D} \sqcap \mathcal{C})$.

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

[10] http://kaon2.semanticweb.org/

– Partial-Disjoint. Some features requested in $\mathcal{C}$ are conflicting with some other ones offered in $\mathcal{D}$—in formulae, $\mathcal{T} \models \neg(\mathcal{D} \sqcap \mathcal{C})$.

The proposed approach implements all the above match types. However, it is possible to add further user-oriented match classes via the incremental building of match requests by means of SQL sub-queries. Concepts are normalized according to the *Concept-Centered Normal Form* (CCNF), [1, Ch.2], through the recursive application of the formulas in Figure 1, until no rule is applicable at every nesting level.

TBox reduction	Concept reduction	$\bot$-reduction
	$\forall\rho.(D \sqcap E) \rightarrow \forall\rho.D \sqcap \forall\rho.E$	
$A \rightarrow A \sqcap C$	$(\geq nR) \sqcap (\geq mR) \rightarrow (\geq nR)$	$\forall\rho.(\geq nR) \sqcap \forall\rho.(\leq mR) \rightarrow \forall\rho.\bot$
if $A \sqsubseteq C \in \mathcal{T}$	if $n > m$	if $n > m$
$A \rightarrow C$	$(\leq nR) \sqcap (\leq mR) \rightarrow (\leq nR)$	$\forall\rho.(\forall R.\bot) \sqcap \forall\rho.(\geq nR) \rightarrow \forall\rho.\bot$
if $A = C \in \mathcal{T}$	if $n < m$	$\forall\rho.A \sqcap \forall\rho.B \rightarrow \forall\rho.\bot$
	$\forall R.\bot \rightarrow \leq 0R$	
		where A and B are disjoint concept names, *i.e.*, $A \sqcap B \sqsubseteq \bot \in \mathcal{T}$.

Fig. 1. Rules for CCNF. The symbol ρ is a sequence of role names $\rho = R_1 \cdots R_n$, so that $\forall\rho.C$, means $\forall R_1.(\ldots(\forall R_n.C)\ldots)$. We include the case $\rho = \varepsilon$ (empty sequence), when $\forall\rho.C$ is just C.

The proposed classification is based on a role-free ABox, where each assertion $C(a)$ means that supply a offers features C. Of course, each individual a is involved in one assertion only, while the same features C could be offered by more than one supply. To store a supply $C(a)$ in a database, we divide a C in four groups of conjuncts $C_n \sqcap C_\sharp \sqcap C_{\forall.n} \sqcap C_{\forall.\sharp}$, being C_n the concept names, $C_\sharp$ the number restrictions, $C_{\forall.n}$ the conjuncts of the form $\forall R_1.(\ldots(\forall R_n.A)\ldots)$ and $C_{\forall.\sharp}$ the conjuncts of the form $\forall R_1.(\ldots(\forall R_n.D)\ldots)$ where D is a number restriction.

A proper design of the Entity-Relationship (E-R) model is a fundamental prerequisite to correctly store both ABox instances and all the TBox $\mathcal{T}$ axioms to be used in the further reasoning stages. In the provided model: (i) entities are chosen in a way to describe all the basic information elements used in the matchmaking process; (ii) numerical features (*e.g.*, *price* or *quantity*) could be very useful in several scenarios (*e.g.*, e-commerce) but they are not closely related to the semantic description of a resource; anyway as such resource information are structured by definition, they will be more easily managed directly by the DBMS. They are named *structured conditions*. Once a concept C has been put in CCNF, the assertions $C(a)$ will be stored in the database, by assigning identifiers to given elements of the syntactic tree of C, and then linking such identifiers by suitable database relations. The logic model for the database storing conjuncts of the normalized form is reported in Figure 2. As an example, Table RESOURCE stores data related to a given resource whereas Table DL_ASSERTION stores the individual describing a resource along with data expressing both cardinality and type of normalized elements. Tables CONCEPT_NAME, NUMBER_RESTRICTION, UNIV_NAME and UNIV_NUMBER respectively store the conjuncts C_n, $C_\sharp$, $C_{\forall.n}$ and $C_{\forall.\sharp}$ of C. A nesting level will be assigned based on how many $\forall$-quantifiers have a given concept C in their scope. For example, $\forall R.C$ has a nesting level 1, $\forall R.\forall S.A$ has nesting level 2, and so on. The attribute `level` of both Table UNIV_NAME and Table UNIV_NUMBER, refers to the assigned nesting degree. Moreover, the attribute `r_type` allows to dis-

$CONCEPT\_NAME(\underline{id\_name}, name)$

$DISJOINT(\underline{id\_name}, id\_name\_disj)$

$NUMBER\_RESTRICTION(\underline{id\_number}, role, r\_type)$

$UNIV\_NAME(\underline{id\_univ\_name}, role\_list, id\_name, level)$

$UNIV\_NUMBER(\underline{id\_univ\_number}, role\_list, id\_number, level)$

$RESOURCE(\underline{id\_resource}, \cdots structured\_conditions \cdots)$

$DL\_ASSERTION(\underline{id\_assert}, owl, n\_name, n\_number, n\_univ\_name, n\_univ\_number, id\_resource)$

$ASSERT\_CONCEPT\_NAME(\underline{id\_assert, id\_name})$

$ASSERT\_NUMBER\_RESTRICTION(\underline{id\_assert, id\_number}, value)$

$ASSERT\_UNIV\_NAME(\underline{id\_assert, id\_univ\_name})$

$ASSERT\_UNIV\_NUMBER(\underline{id\_assert, id\_univ\_number}, value)$

Fig. 2. DataBase logic model

concept_name

id_name	*name*
1	A
2	B
...	...

number_restriction

id_number	*role*	*r_type*
1	R	min
2	R	max
3	T	min
4	T	max
...	...	...

univ_name

id_univ_name	*role_list*	*id_name*	*level*
1	R.S	2	2
...	...	...	...

univ_number

id_univ_number	*role_list*	*id_number*	*level*
1	R	4	1
...	...	...	...

assert_concept_name

id_assert	*id_name*
100	1
...	...

assert_number_restriction

id_assert	*id_number*	*value*
100	1	3
...	...	...

assert_univ_name

id_assert	*id_univ_name*
100	1
...	...

assert_univ_restriction

id_assert	*id_univ_number*	*value*
100	1	6
...	...	...

Fig. 3. Tables filled to store $C(a)$ with $id\_assert = 100$

tinguish numeric restriction cardinalities: $r\_type = max$ (resp. $r\_type = min$) states a $\leq n\ R$ (resp. $\geq n\ R$) restriction. Finally, actual data in individual descriptions are also stored in tables (whose name starts with ASSERT). They link the assertion identifier to its atomic conjuncts storing also numeric values of restrictions for elements in the form $C_\sharp$ and $C_{\forall.\sharp}$. Hence, if the system assigns to $C(a)$ identifier the value 100 and the normalized concept C contains the following conjuncts: A, $\geq 3\ R$, $\forall R.\forall S.B$ and $\forall R. \leq 6\ T$, then the system fills the tables in Figure 3. The presented modeling approach translates an assertion $C(a)$ of size n into $c \cdot n$ database tuples, so it increases the storage size, almost linearly. Nevertheless, such a drawback is largely repaid in terms of flexible match classes management, quick logic-based ranking and explanation of results through enumeration of additional, missing and fulfilled features[11].

3.1 Match classes and ranking function

This subsection reports on queries needed for extracting resources $C_1, C_2, \ldots$ in an exact/full/partial/potential correspondence with a user request D. Queries are incrementally built, according to both number and type of atomic elements composing the

[11] The extraction of conflicting characteristics has not been implemented yet because we do not cache partial matches, exploiting them just as intermediate results.

$$disj(A, B) \quad (1)$$
$$\forall R. \ldots \forall S. \forall T. A \quad (2)$$
$$\forall R. \ldots \forall S. \forall T. B \quad (3)$$
$$\forall R. \ldots \forall S. \exists T \quad (4)$$
$$\forall R. \ldots \exists S \quad (5)$$
$$\ldots \quad (6)$$
$$\exists R \quad (7)$$

Fig. 4. The unsatisfiability pattern in $\mathcal{ALN}$.

description as well as on user constraints. In what follows, we assume that requests D are already in CCNF.

An **Exact match** happens when request and supplied resources are logically equivalent, hence both the so-called structured conditions and all the atomic elements have to correspond, while `n_name`, `n_number`, `n_univ_name` and `n_univ_number` attributes must be equal. In fact, in order to detect an exact match the supply must have exactly the same features of the request and nothing else. As **Full Match** queries simply aim to detect subsumption relationships, we do not deal with them here. On the contrary, we will focus on **Partial** and **Potential Match**, which are strictly related. Actually, a Potential Match is simply a not Partial one. A resource C is a **Potential Match** for a given request D if they do not have conflicting features (*i.e.*, $C \sqcap D \neq \bot$). In case of conflicts, the subset containing not allowed features is the **Partial Match** outcome. The Potential Match results can be obtained by retrieving all the stored supplies excluding Partial Matches. A Partial Match between a resource C and a request D amounts to check whether $C \sqcap D$ is unsatisfiable and why, and such a test in $\mathcal{ALN}$ amounts to check the presence in $C \sqcap D$ of the pattern outlined in Figure 4. There $disj(A, B)$ denotes either two disjoint names, or two incompatible number restrictions, and $\exists R$ denotes a concept in the form $(\geq\ n\ R)$ for some $n > 0$. For roles S, T ans so on the same conditions hold. However, in the proposed approach, such a pattern is split between the database tuples representing $C(a)$, and the SQL query Q_D representing D. Intuitively, for every subconcept of D in the form (2), Q_D looks in the DB for tuples representing those subconcepts of C in the form (3)–(7) which are *not* already in D. Since the selection of the correct pattern to search is leaded by D, the worst case is represented by a request D containing a subconcept C in the form (2) with a role depth n whereas no other subconcept in the form (3)–(7) belonging to the same C pattern is in D. In this case, the $n + 1$ missing subconcepts, required to determine an unsatisfiability pattern for C, have to be looked up in the DB. In particular, one SQL WHERE condition is built in Q_D for each subconcept to search.

To better clarify user request translation into the SQL standard query, a toy example of worst case search, is briefly reported, in accordance with the pattern in Figure 4. Let us suppose a normalized request D - $\forall R. \forall S. A$ ($n = 2$) and two normalized supplies: C_1 - $\forall R. \forall S. B \sqcap \forall R. (\geq\ 1\ S) \sqcap (\geq\ 1\ S)$, C_2 - $(\geq\ 1\ S)$. In order to retrieve a potential match, we have to detect the partial matches *i.e.*, instances represented by tuples in the form (3)–(7), and to discard them from the final results set. As above mentioned, three

WHERE conditions are needed. The SQL query retrieving partial matches w.r.t. D is reported hereafter:

```
SELECT id_assert
FROM assert_univ_name A NATURAL JOIN univ_name
WHERE (level=2 AND role='R.S' AND id_name IN (SELECT id_name_disj
                                              FROM disjoint NATURAL JOIN concept_name
                                              WHERE name='A'))
AND (EXISTS(SELECT *
            FROM (assert_univ_number NATURAL JOIN univ_number)
             NATURAL JOIN number_restriction
            WHERE id_assert=A.id_assert
            AND role_list='R' AND role='S' AND r_type='min' AND value>=1))
AND (EXISTS(SELECT *
            FROM assert_number_restriction NATURAL JOIN number_restriction
            WHERE id_assert=A.id_assert
            AND role='R' AND r_type='min' AND value>=1))
```

Since the previous query returns the supply C_1, the potential matches set is only composed by supply C_2. Moreover, C_2 has $\forall R.\forall S.A$ as missing features (*explanation process*) and a rank equal to 0 as explained in the following (*ranking process*). For the Potential Match results, the logic-based ranking is obtained implementing the ranking function in [10] by aggregating tables with match results. The basic idea is to compute the semantic distance between the normalized forms of both the user request D and the retrieved supply C. To this purpose we introduce 4 tables named CONCEPT_NAME_SCORE, NUMBER_RESTRICTION_SCORE, UNIV_NAME_SCORE and UNIV_NUMBER_SCORE corresponding to the structure of tables CONCEPT_NAME, NUMBER_RESTRICTION, UNIV_NAME and UNIV_NUMBER respectively, enhanced by the attribute `score`. In fact, they store D features with the related user preference (a value between 1 and 5) and, if the user does not set scores for requested features, the matchmaker considers the default value 1. In particular, the results ranking is calculated via the formula (1) rank=(no. fulfilled features of C)/(no. features of D) in case no scores have been set and, as preliminary investigation, via the formula (2) rank=(score sum for fulfilled C features)/(scores sum for D features) otherwise.

4 System and Performance Evaluation

The proposed matchmaker acts as a Java application. A prototypical testing GUI has been developed in order to enable users: 1) to edit/import the request directly in OWL or in DIG [3] (which is more compact); 2) to weigh each normalized concept in the request; 3) to choose the match class to search for and 4) to show the ranked list of results. Experiments have been carried out exploiting an Intel Core i3 PC, equipped with 4 GB RAM. System evaluation goals were: (i) *approach outcome and scalability* –even if existing OWL benchmarks allow a comprehensive evaluation of most common reasoner capabilities [22, 21], unfortunately none is able to execute non-standard services we refer here. Hence, in order to evaluate both matchmaker correctness and performance, only a strict comparison with MaMas-tng results can be carried out; (ii) *data complexity* –a given query is chosen and the system behavior has been evaluated as a function of dataset size; (iii) *expression complexity* –a given dataset is chosen and the system behavior has been evaluated as a function of the execution time of arbitrarily selected queries.

Dataset. In accordance with the goals and assumptions in Section 3, we will use two different domain ontologies: 1) the "Clothing" one (composed by 157 classes and 18 roles) and 2) the "Hotel" one (composed by 68 classes and 12 roles). The former has many concept names whereas the latter has many concept descriptions. Following the "Hotel" ontology structure, it is possible to define individuals with roles nesting level generally higher than the ones of the "Clothing" ontology. Moreover, we have implemented a synthetic KB instances generator, able to automatically build satisfiable instances referred to a given ontology. In this way, we can build data sets having different size, ranging from 100 to 10000 individuals, and instances with a given structure (*i.e.*, number of concept names, number of restrictions, etc.). Finally, several queries have been defined for each knowledge domain. Due to lack of space, we only report on the retrieval times for two queries of average expressiveness respectively referred to the "Clothing" and the "Hotel" ontology:
Q_1 - *"I'm looking for a medium size bluejeans with five pockets and a casual style suitable for spring climate, for both young and adult people"* classified as `n_name=5`, `n_number=18` and `n_univ_name=10` in its normalized form;
Q_2 - *"I'm looking for a twin bed room with some included options (specifically, air conditioning and high speed Internet connection) in a four star hotel near Termini Station in Rome"* classified as `n_name=1`, `n_number=3`, `n_univ_name=10` and `n_univ_number=4` in its normalized form.

Data and expression complexity. The application has been tested by means of several queries with different expressiveness applied to several data sets in order to obtain a comprehensive evaluation of the approach. Our tests measure the retrieval time calculated as average time over ten repetitions. Tests have been performed composing both requests with few generic features and requests including more features with an higher specificity (*e.g.*, similar to the previous ones). Results show that retrieval times moderately increase addressing to the system more complex queries. For this reason, Figure 5 only reports on retrieval times for the requests Q_1 and Q_2. Times have been computed also considering the request normalization process. From the performance comparison standpoint, MaMaS-tng reached via its DIG interface based on HTTP Post has been compared with our relational knowledge based matchmaker running on a remote PostgreSQL server. All tests are reported in Figure 5. Note that the retrieval time difference –given the same instance number for the ontologies– is due to the different complexity of them, as said before.

Moreover, tests have proved that retrieval time of Potential Match (with and without ranking) are higher than the ones of the other match classes (as expected) whereas Exact Match and Full Match have comparable retrieval times. In fact, Potential Match requires a more complex structure of SQL sub-queries and it deals with a higher number of intermediate results (*i.e.*, tuples). Retrieval times for "Clothing" dataset of 10000 instances are justified by the presence of potential matches only by construction. Basically, it can be concluded that retrieval times linearly increase with the data size, in case of up to 5000 individuals more or less. Such outcomes are justified by the higher number of returned instances when datasets increase and –on the other hand– they suggest a proper table partition of the database is needed. The approach scalability is proved by the comparison with retrieval times produced by MaMaS-tng reasoner. In particular, our

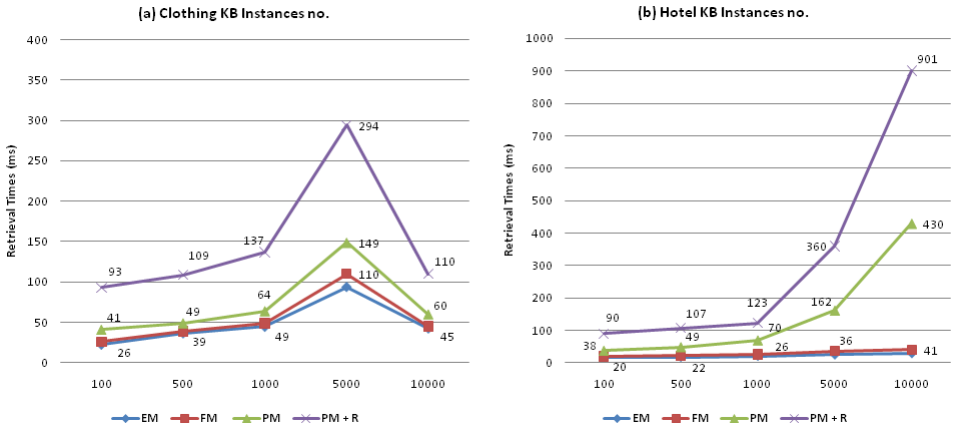

Fig. 5. Proposed system retrieval times (in ms) - [E,F,P]M=[Exact, Full, Potential] Match

higher retrieval time (*i.e.*, Ranked Potential Match - PM+R) as been used as baseline for the further comparison with MaMaS-tng.

Approach outcome. As said, MaMas-tng has been used as comparison term to evaluate output correctness. Results show that the matchmaker proposed here retrieves the same ranked list of results for each match class. The ranking assigned to each potential result has been computed both by MaMaS-tng (using $rankPotential$ [12] algorithm) and by the proposed system (using the default values for the request features weights). Best results for MaMaS-tng have a semantic distance w.r.t. the request equal to 0. So for a significant comparison, we have re-computed the previous ranking formula as: $rank\_value = num_D - num_C$, where num_D refers to request features whereas num_C sums supply features matching the requested ones. Table 1 reports on MaMaS-tng performance on the same datasets and the same queries used for results in Figure 5. Given a request D and a supplied resource C, MaMaS-tng allows to determine the match type ($matchType(D, C)$) –see ask $mT(D, C)$ in Table 1– and to calculate a ranking value ($rank(D, C)$) –see ask $r(D, C)$ in Table 1. It does not provide functions to retrieve all the individuals satisfying a requested match class as implemented in the matchmaker proposed here. So, in order to compare the matchmakers performance, it has been considered the ranked potential match computation, which corresponds to the previous two *asks* for MaMaS-tng (see Table 1 for details).

Table 1. MaMaS-tng retrieval times (in ms) for both "Clothing" and "Hotel" ontologies

PM+R=Potential Match and Ranking

Clothing	**100**	**500**	**1000**	**5000**	-	**Hotel**	**100**	**500**	**1000**	**5000**
PM + R	*93*	*109*	*137*	*294*	-	*PM + R*	*90*	*107*	*123*	*360*
r(D,C)	19771	112934	265811	N/A	-	r(D,C)	11624	54434	106347	1205115
mT(D,C)	20488	115811	269208	N/A	-	mT(D,C)	23040	101258	219400	2382376

Basically, a shallow examination of results shows highest loading times obtained with the proposed matchmaking approach. Nevertheless, it has to be noticed –as mentioned in Section 3– that the proposed approach includes a time-consuming pre-processing phase. So, knowledge bases loading times are obviously higher than in case of MaMaS-tng (see Table 2 where $M - tng$ column refers to MaMaS-tng and DB one is about our approach). Anyway, the KB loading is an off-line and *una tantum* process, performed once when the system is set and not repeated during reasoning phases. Moreover, if the TBox has not been modified then it is possible to store incrementally only new instances, drastically reducing load times. It has to be also said that, MaMaS-tng is not able to load large KBs (*i.e.*, for "Clothing" ontology, previewed 5000 ABox instances cannot be uploaded).

Table 2. Knowledge bases loading times (in ms) for both "Clothing" and "Hotel" ontologies

Clothing	**100**	**500**	**1000**	**5000**	–	**Hotel**	**100**	**500**	**1000**	**5000**
M-tng	995	4057	9599	N/A	–	M-tng	529	2858	5553	239775
DB	77605	410819	856286	4213688	–	DB	52358	333520	601448	3409431

5 Conclusion and Future Work

Motivated by the need to efficiently cope with large datasets in semantic matchmaking, we presented a logic-based framework exploiting a flexible knowledge modeling. A user request is structured as set of normalized features also weighted according to the relevance assigned by the user. By exploiting only SQL queries, the system is able to detect resources falling in several match classes also ranking results. Current implementation refers to $\mathcal{ALN}$, although as pointed out in [2] renewed interests in light-weight DLs for large ontologies and non-standard services has been observed, in order to successfully use semantic technologies in real-world applications.

Preliminary performance evaluation on various datasets show an efficient behavior also considering that optimization techniques such as the transitive closure modeling and the implementation of table partitioning have not been implemented yet. Future work aims at testing further devised strategies for score calculation along with a full optimization of the database and at evaluating performance with other existing OWL-DL storage engines with reference to comparable match classes, *i.e.*, exact and full.

6 Acknowledgments

The authors acknowledge partial support of Apulia Region Strategic Project PS_125 and the reviewers for useful comments and suggestions.

References

1. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook, 2nd edition. Cambridge University Press (2007)

2. Baader, F.: What's new in description logics. Informatik-Spektrum pp. 1–9 (2011), 10.1007/s00287-011-0534-y
3. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In: DL'03. CEUR Workshop Proceedings, vol. 81 (2003)
4. Bechhofer, S., Horrocks, I., Turi, D.: The OWL Instance Store: System Description. In: CADE '05. pp. 177–181 (2005)
5. Bock, J., Haase, P., Ji, Q., Volz, R.: Benchmarking OWL Reasoners. In: ARea Workshop at ESWC 2008. CEUR-WS, Vol 350 (2008)
6. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: ISWC '02. pp. 54–68 (2002)
7. Cadoli, M., Donini, F.M.: A survey on knowledge compilation. AI Commun. 10(3-4), 137–150 (1997)
8. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data Complexity of Query Answering in Description Logics. In: KR-06. pp. 260–270 (2006)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. J. of Automated Reasoning 39(3), 385–429 (2007)
10. Colucci, S., Di Noia, T., Pinto, A., Ragone, A., Ruta, M., Tinelli, E.: A Non-Monotonic Approach to Semantic Matchmaking and Request Refinement in E-Marketplaces. Int. J. on Electronic Commerce 12(2), 127–154 (2007)
11. Di Noia, T., Di Sciascio, E., Donini, F.M.: Semantic Matchmaking as Non-Monotonic Reasoning: A Description Logic Approach. J. of Artificial Intelligence Research 29, 269–307 (2007)
12. Di Noia, T., Di Sciascio, E., Donini, F.M., Mongiello, M.: A System for Principled Matchmaking in an Electronic Marketplace. Int. J. on Electronic Commerce 8(4), 9–37 (2004)
13. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Efficient Reasoning on Large SHIN Aboxes in Relational Databases. In: SSWS '09. pp. 110–124 (2009)
14. Doyle, J., Patil, R.S.: Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the Utility of Representation Services. Artificial Intelligence 48(3), 261–297 (1991)
15. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - A Pragmatic Semantic Repository for OWL. In: WISE. vol. 3807, pp. 182–192. Springer (2005)
16. Li, L., Horrocks, I.: A Software Framework for Matchmaking Based on Semantic Web Technology. Int. J. on Electronic Commerce 8(4) (2004)
17. Lu, J., Ma, L., Zhang, L., Brunner, J.S., Wang, C., Pan, Y., Yu, Y.: SOR: a Practical System for Ontology Storage, Reasoning and Search. In: VLDB '07. pp. 1402–1405. VLDB Endowment (2007)
18. del Mar Roldan-Garcia, M., Aldana-Montes, J.F.: A Survey on Disk Oriented Querying and Reasoning on the Semantic Web. In: ICDEW'06. pp. 58–65. IEEE Computer Society (2006)
19. Pan, Z., Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: PSSS1. vol. 89, pp. 109–113. CEUR-WS.org (2003)
20. Schaerf, M., Cadoli, M.: Tractable reasoning via approximation. Artif. Intell. 74(2), 249–310 (1995)
21. Thakker, D., Osman, T., Gohil, S., Lakin, P.: A pragmatic approach to semantic repositories benchmarking. In: The Semantic Web: Research and Applications, vol. 6088, pp. 379–393. Springer (2010)
22. Weithner, T., Liebig, T., Luther, M., Bhm, S.: Whats Wrong with OWL Benchmarks. In: SSWS 2006. pp. 101–114 (2006)