Querying the Semantic Web with Preferences

Wolf Siberski[1], Jeff Z. Pan[2], and Uwe Thaden[1]

[1] L3S and University of Hannover, Hannover
{siberski,thaden}@l3s.de
[2] University of Aberdeen
jpan@csd.abdn.ac.uk

Abstract. Ranking is an important concept to avoid empty or overfull and unordered result sets. However, such scoring can only express total orders, which restricts its usefulness when several factors influence result relevance. A more flexible way to express relevance is the notion of preferences. Users state which kind of answers they 'prefer' by adding soft constraints to their queries.
Current approaches in the Semantic Web offer only limited facilities for specification of scoring and result ordering. There is no common language element to express and formalize ranking and preferences. We present a comprehensive extension of SPARQL which directly supports the expression of preferences. This includes formal syntax and semantics of preference expressions for SPARQL. Additionally, we report our implementation of preference query processing, which is based on the ARQ query engine.

Keywords preferences, query language, semantic web

1 Introduction

With the abundance of available information, the issue of information filtering becomes more and more pressing. Instead of receiving empty or possibly huge and unordered result sets, users want to get just a manageable set of 'best' answers, which satisfy the query best, even if there are no exact matches.

As widely established in information retrieval and other areas, ranking has shown to be useful to improve the quality of result lists. As more and more Semantic Web applications emerge, this aspects gains importance for the information available in that context. However, the current Web solutions for 'best' answers are not easily applicable to this new context. User queries usually consist of a set of words that have to appear in the document and/or in some metadata of the documents. The support for structured search is very limited; only very first steps in the direction of integrating structured information, such as taxonomies, have been taken.

On the other hand, the benefit of introducing the 'best match' notion has already been identified for several Semantic Web applications (e.g., [1–3]). For example, Bibster [3] allows to search for publications by topic and ranks results according to their similarity to the requested topic. However, the preferences used in these systems typically apply to specific properties, and hard-coded, unmodifiable scoring functions are used.

The same issue has been tackled in database research in the last years. Top-k queries have been introduced which allow to identify the 'best matches' according to a numerical score [4]. Skyline queries have extended this notion to contexts where multiple independent scores have to be taken into account [5]. The most general notion developed in the database area is the notion of *preference-based querying* [6, 7], where logic formulas can be used to specify which items are preferred.

Preference queries are based on the observation that expressions of the form "I like A more than B" are easily stated by users when asked for their wishes. For example, when buying a car, it is easy for one to say which colors he prefers, that he likes cars more for which he has to pay less, that he likes automatic transmission more than manual gear change, etc. Therefore, it should be optimal if a query engine can derive best matches directly from such preference expressions.

The notion of preference is very important in the Semantic Web context, too. Actually, we show in Section 2 that the motivating example from the seminal Semantic Web article [8] written by Tim Berners-Lee et al. can in fact be easily interpreted as preference-based search. A variety of potential Semantic Web applications can benefit from preference queries, e.g. advanced document search or service matchmaking (cf. Section 6).

Therefore, we propose to add preference-based querying capabilities to Semantic Web query languages. As SPARQL is currently the most important of these query languages, we have used it as basis to formally integrate and implement such capabilities as language extension.

2 Motivating Example

In this section we revisit the motivating scenario from [8] in detail. We use this example to show how preferences fit into the Semantic Web vision, and what is needed to specify preferences as part of a query in an informal fashion.

Let us first summarize the scenario: Lucy and Pete are looking for suitable appointments at a physical therapist for their Mom[3]. They have some hard constraints for their search with respect to therapist rating, location, etc., which are not relevant in our context. We therefore only keep the constraint that the therapist's rating must be *very good* or *excellent*.

When Pete sees the first answer to this search, it turns out that there are also some soft constraints he did not consider yet. Therefore, he has to reformulate his query with "stricter preferences". The following preferences can be identified:

1. prefer a nearer therapist over one more far away.
2. prefer *excellent* therapists over *very good* ones.
3. prefer an appointment which does not overlap with the rush hour.
4. prefer appointments with a late starting time over early ones, to avoid the necessity to leave during the work hours.

If these preferences would be expressed as hard constraints, this would most likely lead to an empty result set, because it happens rarely that a result matches exactly to the

[3] Note that here we do not aim at providing an agent environment as sketched in [8].

optimal values with respect to each single preference in the query. The usual case is a trade-off, i.e. results optimal with respect to one dimension tend to have disadvantages in other dimensions. Therefore, a user would have to adapt to the system and relax his query manually (typically by try and error), until some suitable results are found.

Furthermore, in multidimensional queries a user normally is not able to prioritize his different preferences, since he does not know how this will affect the outcome. Is it more important to to have a nearby appointment or is it more important to avoid rush-hour? Is it more important to have an excellent therapist, or more important to get a late appointment? Typically these trade-offs are not weighed by the user in advance, but only when he sees the different options with their concrete advantages and disadvantages. Therefore, it has to be possible to specify multiple (independent) preference dimensions. Note that with current Semantic Web query languages such as SPARQL this is not possible (cf. Section 3.2).

To specify the mentioned preferences, we need atomic preference expressions and facilities for combination. For atomic preferences, two types can be distinguished:

- *Boolean preferences* expressed by a boolean condition (preference 2 and 3 from the example). Results satisfying that condition are preferred over results which do not satisfy it.
- *Scoring preferences* specified by a value expression (preferences 1 and 4) . Results for which this expression leads to a higher value are preferred over results with a lower value (rsp. the other way round).

While we do not want to force the user to prioritize all of his preferences, for some preferences it might be desired to specify priorities. For example, it might be more important to Pete to avoid rush hour than to get a late appointment. Therefore we need two different ways to combine preferences, one for *independent* preferences and one for *prioritized* ones.

Now we take a look at what results a user would actually expect for given preferences. To simplify the presentation, we omit some hard constraints and preferences of the example, and continue with a reduced query:

Return all excellent and very good therapists, with the following preferences:

Prefer excellent therapists over very good ones (preference 2).

Prefer appointments outside rush hour over appointments overlapping it (preference 3).

Prefer the later appointment over an earlier one, if both are equal with respect to rush hour (preference 4).

Note that this removes no complexity with respect to preference specification.

A sample knowledge base on which the query can be executed is shown in Figure 1. Physical therapists have an associated rating and associated appointments. Appointments have start and end time.

Based on this knowledge base, let us analyze what a user would expect as results. Definitely, he does not want to get any result A which is worse than another one B with respect to one preference dimension, and not better in any other dimension. If this is the case, we say that A is *dominated* by B. The interesting results are therefore those which are *not* dominated by others.

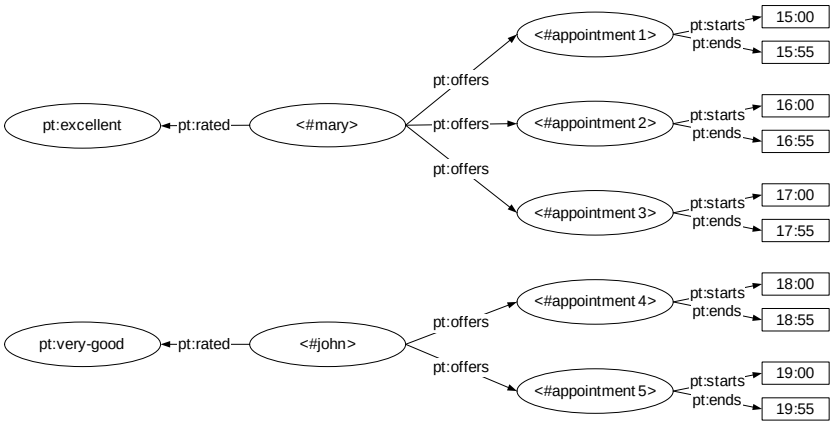

Fig. 1. Example Knowledge Base

We assume that rush hour is from 16:00 to 18:00. Then, as Figure 2 shows, the non-dominated results are *appointment1* and *appointment5*. *appointment1* is in every preference dimension better or equal to *appointment2* and *appointment3*. The same applies to *appointment5* with respect to *appointment4*. The hatched regions denote the domination areas of these results: all answers lying in these areas are dominated and thus not optimal. On the other hand, *appointment1* and *appointment5* can't dominate each other, because *appointment1* is better with respect to rating, but *appointment5* is superior with respect to appointment time. Therefore, these two should be returned as result to the user.

We will show in Section 3.2 what part of the requirements derived from the example SPARQL can cover, and pick up the scenario to illustrate our proposed language extension in Section 4.

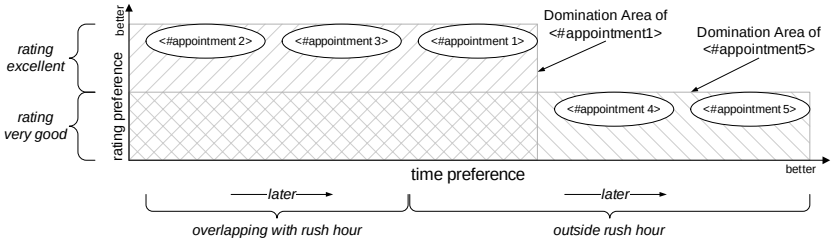

Fig. 2. Appointment Preference Relations

3 Background

3.1 Querying with Preferences

Preferences have one of their origins in decision theory, as a way to support complex, multifactorial decision processes [9]. Another important source are personalized systems (e.g. [10]), where preferences capture a users likings and dislikes. In databases, this thread was picked up by Lacroix and Lavency [11].

Following Chomicki [7], we distinguish between quantitative and qualitative approaches to preferences. In quantitative approaches, each preference is associated with an atomic scoring function, and combination operations are used to compute a score for each result tuple [12]. This restricts the approach to total orderings of result tuples. Top-k queries return the k best matches according to such a score [4]. A formal extension of relational algebra by a specific top-k operator has been proposed in [13]. The qualitative approach is more general than the quantitative one. It does not impose a total order on the result tuples, but allows treating preferences independently, which results in a partial preference order. For relational databases, the qualitative approach has been formalized independently by Kießling [6] and Chomicki [7].

In the following we rely on Chomicki's preference query formalization [7]. In this extension to relational algebra, preferences are expressed as binary relations between tuples from the same database relation. The central concept is the notion of domination (as introduced informally in the previous section).

Definition 1. *Given a relation schema $R(A_1, \ldots, A_n)$ such that U_i, $1 \leq i \leq n$, is the domain of the attribute A_i, a relation $\succ$ is a* preference relation *over R if it is a subset of $(U_1 \times \cdots \times U_n) \times (U_1 \times \cdots \times U_n)$. A result tuple t_1 is said to be dominated by t_2, if $t_1 \succ t_2$.*

We restrict this very general notion to relations that are defined by so-called *intrinsic preference formulas*, first order logic expressions in which a limited set of constraint operators occur.

Definition 2. *Given a relation schema R, an intrinisic preference formula $C(t_1, t_2)$ is a first order formula over two tuples of R which only uses equality and rational order ($<$, $>$) constraints. Such a preference formula C defines a preference relation $\succ_C$: $t_1 \succ_C t_2 \equiv C(t_1, t_2)$.*

For a more convenient notation, we introduce an additional operator to denote incomparability between two result tuples.

Definition 3. *Given a preference formula C and two tuples t_1 and t_2, the incomparability operator $\sim_{\mathsf{C}}$ is defined as*

$t_1 \sim_{\mathsf{C}} t_2 \equiv t_1 \not\succ_{\mathsf{C}} t_2 \wedge t_2 \not\succ_{\mathsf{C}} t_1$.

If t_1 either dominates t_2 or is incomparable with it, this is denoted as

$t_1 \succeq_{\mathsf{C}} t_2 \equiv t_1 \succ_{\mathsf{C}} t_2 \vee t_1 \sim_{\mathsf{C}} t_2$.

Now we can define the new operator, called *winnow operator*, that selects all non-dominated objects from a set of tuples.

Definition 4. *If R is a relation schema and C a preference formula defining a preference relation $\succ_C$ over R, the winnow operator ω_C is defined as $\omega_C(R)$, and for every instance r of R:*

$$\omega_C(r) = \{t \in r | \neg \exists t' \in r.\ t' \succ_C t\}.$$

ω_C therefore selects all non-dominated objects from a set of tuples. In Section 4, we show how to apply these concepts for our extension of SPARQL.

3.2 Ontology Querying

An *ontology* [14] typically consists of a set of important classes, important properties, and constraints about these classes and properties. An ontology language provides some constructors to construct class and property descriptions based on named classes and properties, as well as some forms of axioms about classes, properties and individuals. For example, RDFS [15] provides some axioms (such as domain and range axioms), but no class or property constructors. OWL DL [16] provides class constructors (e.g. conjunction $C \sqcap D$ and number restriction $\leqslant n\, R$), property constructors (e.g. inverse properties R^-) and more kinds of axioms (such as individual equality axioms $\mathtt{a} \approx \mathtt{b}$) than RDFS. Furthermore, OWL DL distinguishes *individual* properties (properties relating individuals to individuals) from *datatype* properties (properties relating individual to data literals). Data literals are literal forms of data values. Due to space limitation, the reader is referred to [15] and [16] for details of the RDFS and OWL DL languages, respectively.

A conjunctive query (CQ) q is of the form

$$q(X) \leftarrow \exists Y.conj(X, Y, Z)$$

or simply $q(X) \leftarrow conj(X, Y, Z)$, where $q(X)$ is called the head, $conj(X, Y, Z)$ is called the body, X are called the distinguished variables, Y are existentially quantified variables called the non-distinguished variables, Z are individual names or data literals, and $conj(X, Y, Z)$ is a conjunction of atoms of the form $C(v)$, $r(v_1, v_2)$, $s(v, t)$, or $\mathtt{E}(t_1, \ldots, t_n)$, where $C, r, s, \mathtt{E}$ are respectively classes, object properties, datatype properties and datatype built-ins, $v,\ v_1$ and v_2 are *individual* variables in X and Y or individual names in Z, and $t,\ t_1, \ldots, t_n$ are *data* variables in X and Y or data literals in Z. As usual, an interpretation $\mathcal{I}$ satisfies an ontology $\mathcal{O}$ if it satisfies all the axioms in $\mathcal{O}$; in this case, we say $\mathcal{I}$ is a model of $\mathcal{O}$. Given an evaluation $[X \mapsto S]$, if every model $\mathcal{I}$ of $\mathcal{O}$ satisfies $q_{[X \mapsto S]}$, we say $\mathcal{O}$ entails $q_{[X \mapsto S]}$; in this case, S is called a *solution* of q. A solution sequence $\mathbf{S} = (S_1, \ldots, S_n)$ is a list of solutions. A disjunctive query (DQ) is a set of conjunctive queries sharing the same head.

SPARQL SPARQL [17] is a query language (W3C candidate recommendation) for getting information from such RDF graphs. It introduces a notion of *E-entailment regime*, which is a binary relation between subsets of RDF graphs. The default SPARQL setting is simple entailment [18]; examples of other E-entailment regime are RDF entailment [18], RDFS entailment [18] and OWL entailment [18].

SPARQL provides *solution modifiers* which allow to transform the solution list derived from a CQ in several ways. The following solution modifiers are available: Distinct, Order, Limit and Offset. Here is the SPARQL syntax for the last three solution modifiers.

```
SolutionModifier::= OrderClause? LimitClause? OffsetClause?
OrderClause     ::= 'ORDER' 'BY' OrderCondition+
OrderCondition  ::= ( ( 'ASC' | 'DESC' ) '(' Expression ')' ) |
                    ( FunctionCall | Var | '(' Expression ')')
LimitClause     ::= 'LIMIT' INTEGER
OffsetClause    ::= 'OFFSET' INTEGER
```

Distinct The Distinct solution sequence modifier D (used in the SELECT clause) ensures solutions in the sequence are unique; i.e., $\mathtt{D}(\mathbf{S}) = \mathbf{S}' = (S'_1, \ldots, S'_k)$ so that $\{S'_1, \ldots, S'_k\} \subseteq \{S_1, \ldots, S_n\}$ and $S'_i \neq S'_j$ for all $1 \leq i < j \leq k$.

OrderClause The Order solution sequence modifier O applies ordering conditions to a solution sequence, and thus provides a limited form of preference expressions. An ordering condition can be a variable or a function call, and it can be explicitly set to ascending or descending by enclosing the condition in ASC() or DESC() respectively.[4] In general, an expression is a disjunctive normal form of numeric expression (see [17] for details) but typically is a variable. Given an order condition C, we have $\mathtt{O}(\mathbf{S}, \mathsf{C}) = \mathbf{S}' = (S'_1, \ldots, S'_n)$ so that $\{S'_1, \ldots, S'_n\} = \{S_1, \ldots, S_n\}$ and $S'_i \succeq_{\mathsf{C}} S'_j$ or $S'_i \sim_{\mathsf{C}} S'_j$ for all $1 \leq i < j \leq n$. We say that S'_i dominates S'_j w.r.t. C if $S'_i \succ_{\mathsf{C}} S'_j$ holds. The semantics of multiple order conditions (`ORDER BY` C_1, C_2, ...) are treated as prioritised composition (cf. 4.2):

$$S'_i \succ_{\mathsf{C}_1,\mathsf{C}_2} S'_j \equiv S'_i \succ_{\mathsf{C}_1} S'_j \vee (S'_i \sim_{\mathsf{C}_1} S'_j \wedge S'_i \succ_{\mathsf{C}_2} S'_j)$$

i.e., ordering according to C_2 unless C_1 is applicable. To sum up, with the OrderingClause SPARQL supports only unidimensional (prioritized) composition of ordering expressions.

LimitClause The Limit solution sequence modifier L puts an upper bound m on the number of solutions returned; i.e., $\mathtt{L}(\mathbf{S}, m) = \mathbf{S}' = (S_1, \ldots, S_k)$ where $k = m$ if $n \geq m$ and $k = n$ otherwise.

OffsetClause The Offset solution sequence modifier OS causes the solutions generated to start after the specified number of solutions; i.e., $\mathtt{OS}(\mathbf{S}, m) = \mathbf{S}' = (S_m, \ldots, S_n)$, where $m \leq n$, and $\mathtt{OS}(\mathbf{S}, m) = \mathbf{S}' = ()$, otherwise. The combination of the Order, Limit and Offset solution sequence modifiers can result in returning partial results.

Example Now let us take a look at what we can achieve with respect to the example from Section 2 using the current solution modifiers. As we cannot specify independent preferences, we have to decide for either rating preference or time preference. Here, we show the query for the latter:

[4] The default is ascending.

```
PREFIX pt: <http://physical-therapists.org/schema>

SELECT ?t ?app ?start ?end ?rating
WHERE ?t pt:offers-appointment ?app .
    ?t pt:rating ?rating .
    ?app pt:starts ?start .
    ?app pt:ends ?end .
    ?t pt:has-rating ?rating
FILTER (?rating = pt:very-good || ?rating = pt:excellent) .
ORDER BY DESC(?end <= '16' || ?start >= '18' ) DESC(?start)
```

As we can see, expression of prioritized preferences is possible using several order conditions. In contrast to the discussion in Section 2, the shown query will also return dominated appointments, but only at the bottom of the solution list.

4 Preference-based Querying for SPARQL

In this section, we will introduce our formal extension of SPARQL solution modifiers to support the kind of preference that we need in ontology querying answering. For illustrative purposes we start with an informal description of our sample preference query according to the proposed extension:

```
 1 SELECT ?t, ?app
 2 WHERE {?t pt:offers-appointment ?app .
 3        ?t pt:has-rating ?rating .
 4        ?app pt:starts ?start .
 5        ?app pt:ends ?end .
 6 FILTER (?rating = pt:very-good || ?rating = pt:excellent)}
 7 PREFERRING
 8        ?rating = pt:excellent
 9     AND
10        (?end <= '16:00' || ?start >= 18:00)
11          CASCADE HIGHEST(?start)
```

Line 1–6 of the query contains the solution pattern and hard constraints, defined as usual. The PREFERRING keyword on line 7 starts the preference definition. Line 8 specifies that results where *?rating* = *pt:excellent* is true are preferred over the ones where this is not the case. The 'AND' keyword (line 9) is used to separate independent preference dimensions. The *avoid rush hour* preference is expressed in line 10, and line 11 contains the the *late appointment* preference. The 'CASCADE' keyword expresses that the left-hand preference (*avoid rush hour*) takes priority over the right hand preference (*late appointment*).

4.1 The *Preferring* Solution Sequence Modifier

Now we extend SPARQL with a new *Preferring* solution sequence modifier, in order to facilitate the representation of preference motivated by the examples presented in Section 2. Our extension covers the following two features:

1. Skyline queries: find all the solutions that are *not* dominated by any other solutions.
2. Soft constraints: Preferably return only the solutions that satisfy all the (hard and soft) constraints; otherwise, relax some or all soft constraints and return only the best answers.

In our extension, preference is a first-class construct in the query language. The extended SPARQL syntax is listed below.

```
SolutionModifier ::= PreferringClause? OrderClause? LimitClause?
                     OffsetClause?
PreferringClause ::= 'PREFERRING' MultidimensionalPreference
MultidimensionalPreference ::= CascadedPreference
                     ('AND' CascadedPreference)*
CascadedPreference ::= AtomicPreference
                     ('CASCADE' AtomicPreference)*
AtomicPreference ::= BooleanPreference
                     | HighestPreference | LowestPreference
BooleanPreference::= Expression
HighestPreference::= 'HIGHEST' Expression
LowestPreference ::= 'LOWEST' Expression
```

Intuitively, users can specify preferences that do not overwrite each other, by using the Preferring clauses with the definitions independent preference separated by the 'AND' construct. In each of these dimensions, atomic preferences can be nested using the 'CASCADE' construct. Here, the leftmost part of the preference expression is evaluated first, and only if two solutions are equal with respect to this part, the next atomic preference expression is evaluated.

4.2 Semantics of the *Preferring* Modifier

Formally, we define the semantics of atomic and combined preference relations, as follows:

Boolean preferences Boolean preferences are specified by a boolean expression BE. For any solutions S_i and S_j, the domination relation for such a preference, $\succ_{\mathsf{C_{BE}}}$ is defined as

$$S_i \succ_{\mathsf{C_{BE}}} S_j \quad \equiv \quad \mathsf{BE}(S_i) \wedge \neg \mathsf{BE}(S_j).$$

Scoring preferences They are specified by an expression which evaluates to a number or a value in other SPARQL domains that have total ordering. For such an ordering $<$ and any solutions S_i and S_j, the domination relation $\succ_{\mathsf{C_{LOWEST},<}}$ is defined as

$$S_i \succ_{\mathsf{C_{LOWEST},<}} S_j \quad \equiv \quad S_i < S_j,$$

and $\succ_{\mathsf{C_{HIGHEST},<}}$ is defined as

$$S_i \succ_{\mathsf{C_{HIGHEST},<}} S_j \quad \equiv \quad S_j < S_i.$$

Multidimensional Preferences For any solutions S_i and S_j, the domination relation to combine independent preferences $\succ_{[\mathsf{C}_1 \text{ AND } \mathsf{C}_2]}$ is defined as

$$S_i \succ_{[\mathsf{C}_1 \text{ AND } \mathsf{C}_2]} S_j \equiv S_i \succeq_{\mathsf{C}_1} S_j \wedge S_i \succeq_{\mathsf{C}_2} S_j \wedge (S_i \succ_{\mathsf{C}_1} S_j \vee S_i \succ_{\mathsf{C}_2} S_j).$$

Intuitively, this says that S_i is dominated by S_j in neither C_1 nor C_2, and that S_i dominates S_j in either C_1 or C_2.

CascadedPreference For any solutions S_i and S_j, the domination relation to combine prioritized preferences $\succ_{[\mathsf{C}_1 \text{ CASCADE } \mathsf{C}_2]}$ is defined as

$S_i \succ_{[\mathsf{C}_1 \text{ CASCADE } \mathsf{C}_2]} S_j \equiv S_i \succ_{\mathsf{C}_1} S_j \vee (S_i \sim_{\mathsf{C}_1} S_j \wedge S_i \succ_{\mathsf{C}_2} S_j)$.

With these definitions, we can now define the preferring solution modifier PS: Given a domination relation C, $\mathtt{PS}(\mathbf{S}, \mathsf{C}) = \mathbf{S}' = (S'_1, \ldots, S'_n)$ so that, for any $S'_i \in \mathbf{S}'$, there exists no $S_j \in \mathbf{S}$ such that $S_j \succ_{\mathsf{C}} S'_i$. Thus, the solution modifier PS gives us exactly the non-dominated solutions in $\mathbf{S}$.

Depending on the given preferences and solutions, PS may deliver just one (the best) solution. We iteratively define the next best solutions as follows:

$\mathtt{PS}^1(\mathbf{S}, \mathsf{C}) = \mathtt{PS}(\mathbf{S}, \mathsf{C})$

$\mathtt{PS}^{n+1}(\mathbf{S}, \mathsf{C}) = concat\,(\mathtt{PS}^n(\mathbf{S}, \mathsf{C}), \mathtt{PS}(\mathbf{S} \setminus \mathtt{PS}^n(\mathbf{S}, \mathsf{C}), \mathsf{C}))$

When combined with the LIMIT k solution modifier, n is selected such that $|\mathtt{PS}^n(\mathbf{S}, \mathsf{C})| > k$.

5 Implementation

As a proof of concept, the SPARQL implementation ARQ [19] has been extended. ARQ is based on a query operator approach, where an operator class is implemented for each solution modifier. This architecture allows to plug in additional solution modifiers easily. Query processing in ARQ is a three-stage process (see 'Query Engine' in Fig. 3):

First, the query is *parsed* and converted into an internal representation. To enable preference handling for this step, productions according to the syntax specified in the previous section have been added to the parser. Preference expression classes which are responsible for evaluating the different preference constructs have been implemented. The extended parser instantiates objects from these classes and assembles them to a preference relation representation (see right-hand side of Fig. 3).

Second, ARQ creates a *query plan* for each incoming query, consisting of accordingly chained operators. Such an operator for preference handling, has been added which which contains the algorithm for determining dominating objects, based on the given preference relation representation. The structure of an example query plan is shown in the middle of Fig. 3. The planning algorithm has been extended to insert the preference operator into the plan if a preference clause is present.

Finally, the query is *executed* on the given knowledge base. During this execution, the preference operator filters all dominated solutions. In our prototype, we use the BNL (Blocked Nested Loop) algorithm [5] for this purpose. The computation of the preference relation is delegated to its representation which was generated during query planning.

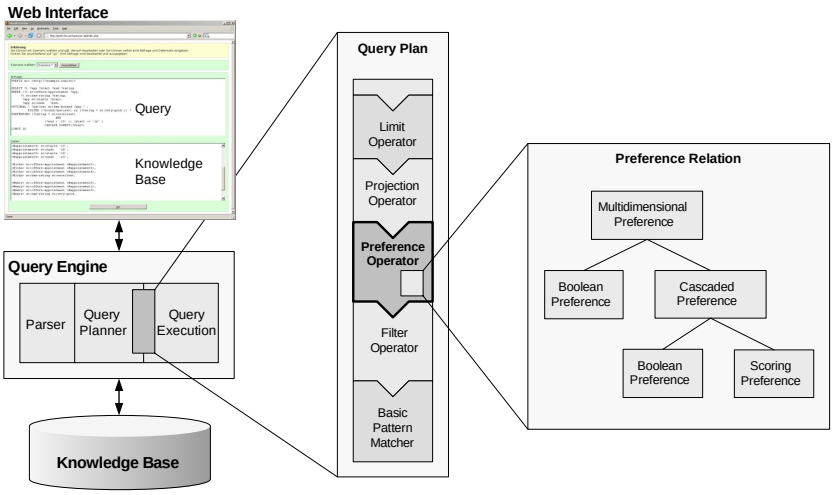

Fig. 3. ARQ Query Engine with Sample Query Plan and Preference Expression

The ARQ query engine interface can be used by applications as before, and we have implemented a Web interface to test applications of our extension which calls the modified query engine to evaluate preference queries.[5]

6 Related Work

Most of the approaches that deal with ranking in the Semantic Web offer very specific (hard-coded) ways to specify some of the properties and scoring functions, which are not editable by the users.

Bibster [3] is a system for storing and sharing information about publications. It allows to search for publications by topic and uses a very specific and unmodifiable preference function, which makes it impossible to define scores and constraints on arbitrary properties.

More domain dependent is the Textpresso-system [1], a system that allows for ontology-based search for biological literature. Textpresso focuses on optimized ontology-creation for this domain. Querying can be done combining fulltext and concept search, i.e., using known associations or the combination of concepts. Explicit rankings are not definable.

Aleman-Meza et al. [20] present a ranking approach which is based on measuring complex relationships. Two entities are related (semantically associated) if there is at least one binding property. They present several ways to rank the complex relationships, but also do not propose a query language extension. A more general approach is the Corese Search Engine [2], a search engine based on conceptual graphs. It is based

[5] available at `http://prefs.l3s.uni-hannover.de`

on RDF, and its expressivity is comparable to RQL or SqishQL. The extension for approximate search is not done by extending one of the existing query languages but is designed as a completely new language.

A more flexible solution is proposed in [21]. Here, the way in which a result of a query is derived is used to rank the results of that query (based on how the results "relate"). The relevance is defined on the level of the relation instances, while the results are a set of concept instances. The scoring functions used are in IR-style, but not definable by the user.

The only approach known to the authors that also extends a query language is Imprecise RDQL [22]. This approach introduces the concept of similarity to enable ranking on arbitrary properties. Their idea of similarity joins is based on the work of [23]. The specification of soft constraints is still rather limited: IMPRECISE defines the variable that shouldn't be matched exactly. The measure to be used for an imprecise variable is specified by the SIMMEASURE clause, but the measures which can be used are constrained to a set of predefined metrics which the authors defined in a library. Furthermore, like the previous approaches Imprecise RDQL offers only scoring for one dimension.

Often, users won't express their preferences directly, but the application might infer preferences from a user profile or other context information, and amend an explicit query accordingly (e.g. [24]). Various techniques for preference mining and elicitation have already been developed, e.g. [25, 26], which can be used in Semantic Web applications as well.

7 Conclusion

In this paper we showed that ranking and preferences as established concepts in relational databases also play an important role in querying the Semantic Web. We discussed why preferences are needed and how this concept can be transferred to Semantic Web query languages such as SPARQL. The presented formal model can be used as unifying framework for of a wide variety of ranking specifications. Finally, we described our ARQ implementation of the SPARQL extension. Thus, the solution presented here provides the basis for combining the strengths of logic-based precise querying and benefits of ranking-based retrieval.

References

1. Müller, H., Kenny, E., Sternberg, P.: Textpresso: An ontology-based information retrieval and extraction system for biological literature. PLoS Biol **2** (2004)
2. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.: Querying the semantic web with corese search engine. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI). (2004) 705–709
3. Haase, P., Broekstra, J., Ehrig, M., Menken, M., Mika, P., Olko, M., Plechawski, M., Pyszlak, P., Schnizler, B., Siebes, R., Staab, S., Tempich, C.: Bibster – a semantics-based bibliographic peer-to-peer system. In: Proceedings of 3rd International Semantic Web Conference (ISWC). (2004) 122 – 136

4. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Santa Barbara, California, USA (2001)
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany (2001) 421–430
6. Kießling, W.: Foundations of preferences in database systems. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China (2002) 311–322
7. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. **28** (2003) 427–466
8. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (2001)
9. Fishburn, P.C.: Utility Theory for Decision Making. Wiley, New York (1970)
10. Riecken, D.: Introduction: personalized views of personalization. Commun. ACM **43** (2000) 26–28 (Introduction to Special Issue on Personalization).
11. Lacroix, M., Lavency, P.: Preferences; putting more knowledge into queries. In: Proceedings of 13th International Conference on Very Large Data Bases (VLDB), Brighton, UK (1987) 217–225
12. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), Dallas, TX, USA (2000) 297–306
13. Li, C., Soliman, M.A., Chang, K.C.C., Ilyas, I.F.: Ranksql: Supporting ranking queries in relational database management systems. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway (2005) 1342–1345
14. Uschold, M., Gruninger, M.: Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review (1996)
15. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema (2004) W3C recommendation, http://www.w3.org/TR/rdf-schema/.
16. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax (2004) W3C Recommendation, http://www.w3.org/TR/owl-semantics/.
17. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF (2006) W3C Candidate Recommendation, http://www.w3.org/TR/rdf-sparql-query/.
18. Hayes, P.: RDF Semantics (2004) W3C recommendation, http://www.w3.org/TR/rdf-mt/.
19. Seaborne, A.: An open source implementation of SPARQL (2006) WWW2006 Developers track presentation, http://www2006.org/programme/item.php?id=d18.
20. Aleman-Meza, B., Halaschek-Wiener, C., Arpinar, I.B., Ramakrishnan, C., Sheth, A.P.: Ranking complex relationships on the semantic web. IEEE Internet Computing **9** (2005) 37–44
21. Stojanovic, N.: An approach for defining relevance in the ontology-based information retrieval. In: Proceedings of the International Conference on Web Intelligence (WI), Compiegne, France (2005) 359–365
22. Bernstein, A., Kiefer, C.: Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In: 21th Annual ACM Symposium on Applied Computing (SAC), New York, NY, USA, ACM Press (2006)
23. Cohen, W.W.: Data integration using similarity joins and a word-based information representation language. ACM Trans. Inf. Syst. **18** (2000) 288–321
24. Dolog, P., Henze, N., Nejdl, W., Sintek, M.: The personal reader: Personalizing and enriching learning resources using semantic web technologies. In: Proceedings of the Third International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH), Eindhoven, Netherlands (2004) 85–94

25. Sai, Y., Yao, Y., Zhong, N.: Data analysis and mining in ordered information tables. In: Proceedings of the International Conference on Data Mining (ICDM), San Jose, CA, USA (2001) 497–504
26. Blum, A., Jackson, J.C., Sandholm, T., Zinkevich, M.: Preference elicitation and query learning. Journal of Machine Learning Research **5** (2004) 649–667