Relaxing RDF Queries based on User and Domain Preferences

Peter Dolog
Aalborg University, Department of Computer Science,
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark,
dolog@cs.aau.dk

Heiner Stuckenschmidt
Universität Mannheim, A5, 6, 68159 Mannheim, Germany,
heiner@informatik.uni-mannheim.de

Holger Wache
University of Applied Sciences Northwestern Switzerland (FHNW)
School of Business, Riggenbachstrasse 16, CH-4600 Olten, Switzerland
holger.wache@fhnw.ch

Jörg Diederich
Forschungszentrum L3S, Leibniz Universität Hannover,
Appelstraße 9a, 30167 Hannover, Germany,
diederich@L3S.de

November 30, 2007

Abstract

Research in Cooperative Query answering is triggered by the observation that users are often not able to correctly formulate queries to databases that return the intended result. Due to a lack of knowledge of the contents and the structure of a database, users will often only be able to provide very broad queries. Existing methods for automatically refining such queries based on user profiles often overshoot the target resulting in queries that do not return any answer. In this paper, we investigate methods for automatically relaxing such over-constrained queries based on domain knowledge and user preferences. We describe a framework for information access that combines query refinement and relaxation in order to provide robust, personalized access to heterogeneous RDF data as well as an implementation in terms of rewriting rules and explain its application in the context of e-learning systems.

Keywords: Query Relaxation, User Modeling, Preferences, ECA Rules

1 Introduction

Research in Cooperative Query answering is triggered by the observation that users are often not able to correctly formulate queries to databases that return the intended result. This is even more the case for semantic web systems based on RDF for the following reasons:

- **The data accessed often comes from different sources. The internal structure of these sources is not always known.**
- **There is no fixed integrated schema. Each source can have its own schema, sources may make partial use of different available schemas.**
- **Authors of data are not forced to stick to a given schema and often use different conventions to represent the same information.**

```
SELECT * FROM
  {Resource} subject {Subject},
  {Resource} title {Title},
  {Resource} description {Description},
  {Resource} language {Language},
  {Resource} requires {} subject {Prerequisite},
  {User} type {user}
  {User} hasPerformance {Performance},
  {Performance} learning_competency {Competence}
WHERE
  Subject Like "inferenceengines" and
  Prerequisite = Competence and
  Language = de and
  User = user42
```

Figure 1: Query extended with user preferences

With the increasing popularity of RDF as a representation language in domains such as medicine [SvHdW+04] or e-learning [DHNS04, DSKN08] this problem becomes more pressing. If RDF query languages are to be used in a large scale we have to make sure that people will be able to formulate meaningful queries. If this is not the case, we have to find ways to still provide the user with the intended results. Cooperative query processing aims at supporting the user by automatically modifying the query in order to better fit the real intention of the user. The specific problem that we address in this work is the situation where the content of the information as well as the RDF-based metadata does not have the expected format. As a motivating example, we use the domain of e-learning.

In open e-learning environments such as the one described in [DHNS04, DSKN08], learning materials from different authors and different sources are made available online through an RDF-based infrastructure. In particular, each learning resources is described by a combination of different metadata providing details about type, format and content of the material as well as information about required expertise. In order to capture this metadata, different metadata schemas including standards like Dublin Core[1] and LOM[2] as well as domain ontologies like the ACM topic hierarchy[3] are used [Bra05]. In addition, information about the expertise of the user is needed to decide whether material meets the user's level of expertise. The result is a rather complex metadata schema that needs to be queried in order to retrieve relevant material. Due to this complexity, the corresponding query is normally created by the system based on user input and known preferences. An example of such a complex query in the SeRQL query language is shown below[4].

The query asks for learning resources in German about the subject "inferenceengines" that meet the previous knowledge of the user – in particular, it claims that the subject of required units should be contained in the competence of the current user (user42). According to the metadata schema, this query should return all relevant results. When we apply this query to real data, however, it does not return any result despite the fact that there are a number of suitable resources. The problem in this case are irregularities in the way the data is described. In particular, not all the authors of resource metadata stick to the conventions that come with the metadata schema – a situation that we will encounter quite often on the semantic web. For the purpose of this paper, we will not systematically analyze the different problems we identified in the dataset but restrict ourselves to two typical examples that we will use to illustrate our approach in the remainder of the paper.

The subject assigned to a course does not always correctly summarize the content. In our test data set for example, if the user provides the keyword "inferenceengines" no resources are returned despite the fact that there are resources for instance about inference implementation and tools for inference. The problem is here that in the case of the first resource the term "inferenceengines" only occurs in the title, but not in the subject. In the case of the second resource, the term only occurs in the description and is mentioned neither in the subject nor

[1]http://dublincore.org/schemas/rdfs/
[2]http://kmr.nada.kth.se/el/ims/md-lomrdf.html
[3]http://www.cs.vu.nl/ heiner/public/SW@VU/classification.daml
[4]Namespaces have been omitted for the sake of readability

the title of the resource. Therefore, a query for resources with the subject "inferenceengines" will return no results.

Learning resources normally specify required expertise in terms of knowledge that the user should have in order to be able to understand the content of the resource. In existing data sets there are at least three different ways in which this requirement is specified. The standard way is to refer to other learning resources that should have been mastered before - this information can be taken from the user profile. There are also cases, however, where required skills are given in terms of links to a topic hierarchy or even in terms of literals containing the required skill as a string. If the user query is now automatically expanded with links to resources mastered by the user, these resources will not be found.

In this paper, we present an approach for addressing the problems querying heterogeneous RDF models using preference-based query relaxation. Our work is similar to the work of Gaasterland [GGM92b, GGM92a] on cooperative query answering in databases. In particular, we present a method for automatically relaxing over-constrained queries based on background knowledge about the domain model and user preferences, extending previous work in two directions:

- We base our work on a general relaxation framework that is designed to capture the specific properties of RDF and RDF query languages.
- We explicitly link the relaxation process to an explicit model of user preferences and domain characteristics that can be used to guide the relaxation process.

In the following, we provide an overview of a general approach for relaxing RDF queries using re-writings that we proposed in previous work. In section 3, we introduce a generic model for representing domain and user knowledge that provides the basis for guiding the relaxation process which is discussed in detail in Section 5. We also discuss the introduced model in the context of related work 5. Section 4 discusses how to elicit knowledge about a user and a domain. We briefly describe a prototypical implementation of the system in Section 6 and conclude with a discussion of achievements and open problems.

2 Relaxation by Re-writing

A possible solution to deliver users with meaningful results is to successively relax the constraints imposed in the (extended) query. Different techniques for relaxing queries have been proposed in the database area. Gaasterland et al [GGM92a] provide a unifying view on different relaxation techniques in terms of replacing subexpressions in the query.

2.1 The Re-writing Framework

We proposed in a previous work [DSW06] a rule-based query rewriting framework for RDF queries independent of a particular query language. The framework is based on the notion of triple patterns (RDF statements that may contain variables) as the basic element of an RDF query and represents RDF queries in terms of three sets:

- triple patterns that must be matched by the result (mandatory patterns)
- triple patterns that may be matched by the results (optional triple patterns).
- Conditions in terms of constraints on the possible assignment of variables in the query patterns.

Re-writings of such queries are described by transformation rules $Q \xrightarrow{R} Q'$ where Q is the original query and Q' the rewritten query. Rewriting rules consists of three parts:

- A matching pattern represented by a RDF query in the sense of the description above. Normally the matching pattern is a part of the original query Q.

- A replacement pattern also represented by an RDF query in the sense of the description above. The replacement pattern replaces the matched pattern resulting in Q'.
- A set of conditions in terms of special predicates that restrict the applicability of the rule by restricting possible assignments of variables in the matching and the replacement pattern.

A rewriting is performed in the following way: First it is checked if a rule is applicable for a query. For this purpose the matching pattern needs to match a given query Q in the sense that the mandatory and optional patterns are subsets of the corresponding parts of Q. Further the predicates in the conditions of the re-writing rule has to be satisfied. If a rule is applicable for a query then the matched patterns are removed from Q and replaced by the corresponding parts of the replacement pattern resulting in Q'.

2.2 Query Rewriting Model

Our approach is formally based on rewriting to solve the problem of over-constrained queries. This rewriting relaxes the over-constrained query based on rules. This has an advantage that we start with the strongest possible query that is supposed to return the "best" answers satisfying most of the conditions. If the returned result set is either empty or contains unsatisfactory results, the query is modified either by replacing or deleting parts of the query, or in other words relaxed. The relaxation should be a continuous step by step, (semi-)automatic process, to provide a user with the possibility to interrupt further relaxations. Before we investigate concrete relaxation strategies in the context of our example domain, we first give a general definition of the framework for re-writing an RDF query.

Each resource is annotated with an RDF description which can be seen as a set of triples [Hay04]. A query over these resources consists of triple patterns and a set of conditions that restrict the possible variables bindings in the patterns. Each triple pattern represents a set of triples. The corresponding abstract definition of a query focuses the essential features of queries over RDF; several concrete query languages are based on these ideas including SeRQL [BK04] which we use in our examples in figure 1.

Definition 1 (RDF Query) *Let $\mathcal{T}$ be a set of terms, $\mathcal{V}$ a set of variables, $\mathcal{RN}$ a set of relation names, and $\mathcal{PN}$ a set of predicate names. The set of possible triple patterns $\mathcal{TR}$ is defined as $\mathcal{TR} \subseteq 2^{(\mathcal{T}\cup\mathcal{V})\times(\mathcal{RN}\cup\mathcal{V})\times(\mathcal{T}\cup\mathcal{V})}$. A ground triple (pattern) is a triple pattern which does not contain any variable. A query Q is defined as $\langle M_Q, O_Q, P_Q\rangle$ with M_Q and $_Q \in \mathcal{TR}$ and $P_Q \subseteq \mathcal{P}$. M_Q is the set of mandatory patterns (patterns that have to be matched by the result), O_Q is a set of optional patterns (patterns that contribute to the result but do not necessarily have to match the result), and $\mathcal{P}$ is the set of predicates. A predicate has as name in $\mathcal{PN}$ and is defined over $\mathcal{T}$ and $\mathcal{V}$.*

The triple patterns M_Q in a query Q determine ground triples from a database. Informally all substitutions τ are answers to Q which maps the triple patterns M_Q to existing ground triples in the database. A substitution τ is defined a set of pairs (X_i, T_i) and applied as usual:

Definition 2 (RDF Answers) *A substitution τ is a set of pairs (X_i, T_i) with $X_i \in \mathcal{V}$ and $T_i \in \mathcal{T}\cup\mathcal{V}$. $\tau(S)$ is generated from S where each appearance of X_i is replaced by T_i for each $(X_i, T_i) \in \tau$. A substitution τ is valid for a RDF query $Q = \langle M_Q, O_Q, P_Q\rangle$ if*

- *$M = \tau(M_Q)$ where M is a set of ground triples from the database and*
- *$\tau(P_Q)$ are satisfied.*

A valid substitution τ may be extended to optional patterns O_Q, i.e. $\tau(O_Q)$ may also be equal to some ground triple in database. All valid substitutions constitute answers to the query Q.

Using these abstract definitions, the query in figure 1 without language preference on german and prerequisites on competences matched with user learner performance would be represented as

$$\begin{aligned}
M_Q &= (\{(Resource, subject, Subject),\\
&\quad (Resource, title, Title)\\
&\quad (Resource, description, Description)\},\\
O_Q &= \{\}\\
P_Q &= \{like(Subject, \text{“}inferenceengines\text{”})\}
\end{aligned}$$

where $Resource, Subject, Title, Description \in \mathcal{V}$, as well as $subject, title, description$, "$inferenceengines$" $\in \mathcal{T}$ and $like \in \mathcal{PN}$. Alternatively, we could use variables as placeholders for the relations and assign the concrete relation names to them as conditions that use the equality predicate. The corresponding definition of the example query would be

$$\begin{aligned}
M_Q &= \{(Resource, R1, Subject),\\
&\quad (Resource, R2, Title)\\
&\quad (Resource, R3, Description)\},\\
O_Q &= \{\}\\
P_Q &= \{R1 = subject, R2 = title, R3 = description,\\
&\quad like(Subject, \text{“}inferenceengines\text{”})\}
\end{aligned}$$

where $Resource, Subject, Title, Description, R1, R2, R3 \in \mathcal{V}$, $subject, title, description$, "$inferenceengi-nes$" $\in \mathcal{T}$ as well as $=$ and $like \in \mathcal{PN}$. The later representation can be seen as a normal form for queries that makes it easier to formulate re-writings in a general way. For sake of readability, in the following we will refer to the original form instead of the normal form.

Based on the abstract definition of an RDF query, we can now define the notion of a rewriting rule and rewriting process as such. We define rewriting in terms of rewriting rules that take parts of a query, in particular triple patterns and conditions, as input and replace them by different elements.

In our work, we employ the principle of rewriting rules that are inspired by ECA-rules (event-condition-action rules) [Cer92, PPW03] for continuous relaxation of user queries. A rewriting rule formally consists of three parts: a *pattern*, a *replacement* and some *conditions*. The pattern corresponds to the event, i.e. in our case an occurrence of particular triple patterns or predicates in a query. The replacement contains the terms which will substitute the matched pattern in a query; the replacement can be seen as the action in the ECA principle. Conditions constrain the rewriting and determine when particular rule can be fired because the rewriting rule can only be applied if the conditions are satisfied. These conditions can be used to define certain relaxation strategies. In particular, we will see later that conditions can be based on user preferences or background knowledge about the domain.

Definition 3 (Rewriting Rule) *A rewriting rule R is a 3-tuple $\langle PA, RE, CN\rangle$ where PA and RE are RDF queries according to Definition 1 and CN is a set of predicates.*

Conditions are the same constructs which are already introduced for queries. Conditions consist of predicates which constrain possible results. Patterns and replacements formally have the same structure as queries. They also consist of a set of triples and predicates. But patterns normally do not address complete queries but only a subpart of a query. Using this definition we can specify a rewriting rule that extends the simple query in figure 1 with the language preference of the example user `user42`.

$$\begin{aligned}
PA &= (\{(Resource, title, Subject)\}, \emptyset, \emptyset)\\
RE &= (\{(Resource, title, Subject),\\
&\quad (Resource, language, Language)\}, \emptyset,\\
&\quad \{(Language = X)\})\\
CN &= \{languagePrefernce(User, X)\}
\end{aligned}$$

where $languagePrefernce$ is a predicate which looks in the user profile of a user $User$ for the language preference. $User$ is a variable which will be bind to the id of the current user sending the query, e.g. `user42`.

While this example contained a rule for refining a query, we will see later that we can use the same mechanism for defining relaxations on a query.

In general a rewriting rule is applicable to all queries which contain the pattern at least as a part. The pattern does not have to cover the whole query. Normally it addresses some triples as well as some predicates in the query. In order to write more generic rewriting rules the pattern must be instantiated which is done by a substitution.

Definition 4 (Pattern Matching) *A pattern PA of a rewriting rule R is applicable to a query $Q = \langle M_Q, O_Q, P_Q \rangle$ if there are subsets $M'_Q \subseteq M_Q$, $O'_Q \subseteq O_Q$ and $P'_Q \subseteq P_Q$ and a substitution θ with $\langle M'_Q, O'_Q, P'_Q \rangle = \theta(PA)$.*

In contrast to term rewriting systems [BN98] the definition of a query as two sets of triples and predicates simplifies the pattern matching, i.e. the identification of the right subpart of the query for the pattern match. A subset of both sets has to be determined which must be syntactically equal to the instantiated pattern. Please note that due to set semantics, the triples and predicates in the pattern may be distributed over the query.

Now we will define how the new rewritten query is constructed with the help of the rewriting rule and pattern matching.

Definition 5 (Query Rewriting) *If a rewriting rule $R = \langle PA, RE, CN \rangle$*

- *is applicable to a query $Q = \langle M_Q, O_Q, P_Q \rangle$ with subsets $M'_Q \subseteq M_Q$, $O'_Q \subseteq O_Q$, $P'_Q \subseteq P_Q$ and substitution θ*
- *$\theta(CN)$ is satisfied,*

then the rewritten query $Q^R = \langle M^R_Q, O^R_Q, P^R_Q \rangle$ can be constructed with $M^R_Q = (M_Q \setminus M'_Q) \cup \theta(M_{RE})$, $O^R_Q = (O_Q \setminus O'_Q) \cup \theta(O_{RE})$ and $P^R_Q = (P_Q \setminus P'_Q) \cup \theta(P_{RE})$ with $RE = \langle M_{RE}, O_{RE}, P_{RE} \rangle$.

Informally speaking, if the pattern match a query and the conditions are satisfied then the matched pattern is substituted by the replacement.

Applied the above rewriting rule to the basic query we get the following refined rule:

$$\begin{aligned}
M_Q &= (\{(Resource, subject, Subject), \\
&\quad (Resource, title, Title) \\
&\quad (Resource, description, Description), \\
&\quad (Resource, language, Language)\}, \\
O_Q &= \emptyset \\
P_Q &= \{like(Subject, \text{``}inferenceengines\text{''}), \\
&\quad (Language = \text{``}de\text{''})\}
\end{aligned} \tag{1}$$

According to the rewriting rule The triple $(Resource, title, Subject)$ which is matched by PA of the rewriting rule is replaced by $(Resource, title, Subject)$, $(Resource, language, Language)$ and P_Q is extended by $(Language = \text{``}de\text{''})$. Please note that the language preference of user42 is German because of the constraint $languagePrefernce(user42, \text{``}de\text{''})$ is satisfied and "de" means German. Similarly, the query can be further rewritten to add learner performance constraints. Note also that similar process is applied in query relaxation process. Rewriting rules will then contain for example a pattern on subject replaced with a pattern on title.

2.3 Using Re-writings

In general the rewriting is a very powerful approach in order to manipulate the over-constrained query. With replacing parts of a query we can realize four types of actions and of course arbitrary combinations of these actions:

- *Making Patterns optional* — this provides a query which considers a situation that some of the resources do not have all the metadata annotations expected - e.g. some resources might not have a subject, in this case the corresponding part of the query has to be made optional. A query then gives also results where the particular predicate relaxed to an optional predicate does not occur;

- *Replacing Value* — this provides a query where a particular predicate value is replaced with another value or a variable. This can be useful to find resources that do not directly match the user interest, but are concerned with a broader or related topic. Taxonomies may be used to provide siblings, more general terms, and so on;
- *Replacing Patterns/Predicate* — this provides a query where a particular triple resp. predicate in restrictions is replaced by another triple resp. predicate. A domain knowledge is employed by this purposes. In the case of our example, if a subject query is not satisfied, it may be replaced by title query with similarity measures;
- *Deleting Patterns/Predicate* — this provides a query where a particular predicate is deleted from a query completely.

As such, these operations are independent of the application domain and the user preference. The connection to the domain and the user can be made using a set of special predicates in the condition of the rewriting rules that link the manipulation to specific aspects of the domain and the user model. In the following, we discuss two specific predicates for including information about the domain and the user into the rewriting process.

domain-preferred-over(X,Y) This predicate indicates that due to the special nature of the domain a certain relation or value is a better choice for retrieving results than another one. We can use this to relax queries by replacing a highly preferred value by a less preferred one that is more likely to deliver a result even if this result may be less exact.

user-preferred-over(X,Y,U) This predicate is defined in the context of a specific user U and indicates that the user considers a certain relation more important or prefers a certain value over another one. We can use this predicate to relax queries by replacing highly preferred values by less preferred ones or for deciding which of the predicates in a query can be relaxed more easily, because the user considers it less important.

In order to make the relaxation smooth, we consider these predicates to be non-transitive.[5] The concrete implementation depends on the chosen representation of the domain and the user profile. In the following, we discuss the implementation and use of an abstract user and domain model based on semantic web technologies and explain how re-writing conditions can be checked based on these models using an RDF query language.

3 Environment, Preference and Domain Model

In order to include knowledge about the domain of interest and the preferences of the user into the query relaxation process, we have designed a general scheme for representing relevant knowledge independent of a concrete application. This general scheme exploits the meta-modeling capabilities of RDF to define aspects of the world we can take into account in the rewriting process (compare fig. 2).

The schema follows an idea, that each environment can be generated according to an *application domain schema* used by the application. Rather than directly representing domain knowledge or user preferences it provides metaclasses that can be instantiated by existing representation schemes for information resources such as Learning Object Metadata (LOM) [Nil01] as well as metadata schemas like the Dublin Core standard [DC2], and taxonomies and ontologies used for predicate values in the information resource schemas such as ACM computing classification system [oCm02].

Environment concept can be for example linked to a field on a user interface form where the user can type a search term or it can be filled in with a class from a taxonomy. Such a generic environment schema provides us with a flexibility to describe any user environment which is based on schemas. For example, an environment concept can model a field on an entry form which is used to enter a subject term a user is searching for in the metadata. Such a field will

[5]Transitivity is ensured by the rewriting procedure itself; it does not needed to be considered for the predicates itself.

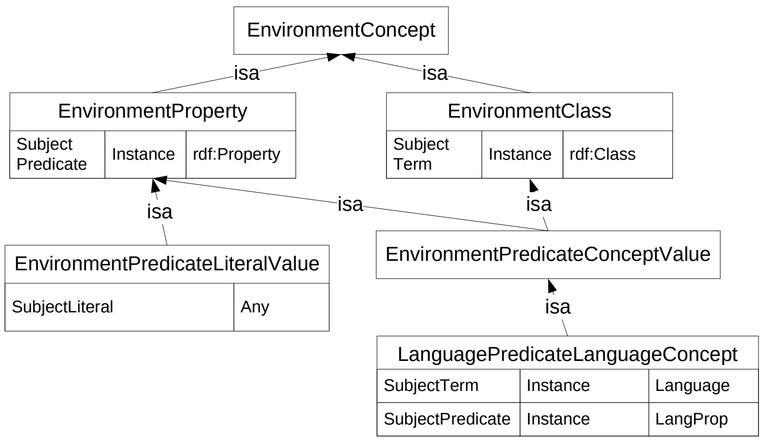


Figure 2: A Schema for Generic Environment

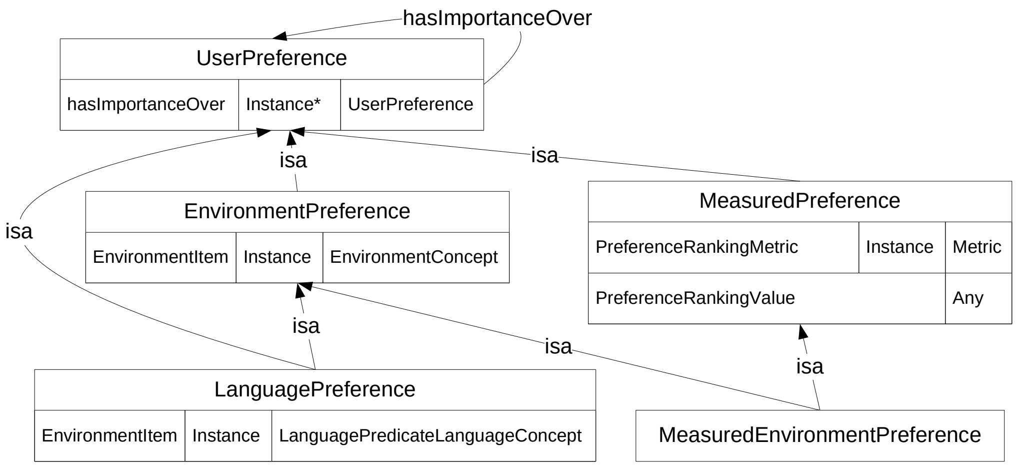


Figure 3: A Schema for Environment User Preferences

be an instance of `EnvironmentProperty` class pointing to a dc:subject predicate of the Dublin core schema. An example of combined class and predicate instance of an environment would be a predicate dc:subject with a class from a taxonomy like ACM CCS as its value.

Another advantage of such a generic environment schema is that we can refer to environment concepts from user preferences. Figure 3 depicts a schema for environment user preferences. Each user can express his level of preference for any environment concept. This is reflected by the `EnvironmentItem` property of the `EnvironmentUserPreference` class. This is a generic definition of an environment preference through an `EnvironmentConcept` class as a domain for `EnvironmentItem` attribute. Classes for environment preferences are further specialized according to which environment concept class is used to describe them. For example, an environment class for representing language predicates `LanguagePredicateLanguageConcept` from fig. 3 is used as a domain for the `EnvironmentItem` property of `LanguagePreference`. Note, that this class inherits and overloads properties from `EnvironmentProperty` as well as `EnvironmentClass`.

The level of a user preference can be expressed as a value from a metric. This is modeled by the `MesuredPreference` as a subclass of a user preference. The values from preference measures can be used to order them, i.e. to deduce the ordering relations between preference instances which is modeled by `hasImportanceOver` relation of user preference. Another alternative is to deduce preference relations from usage logs as we describe in the next section.

Besides the user preferences, we also consider schema of a user background. This is represented as a learning performance and skills gained by performing learning. We use our schema for such a learner's learning performance [DS05] where the learning performance is described by a relation to learning competence, portfolios created and certificates gained during/from learning activities which have been connected to the learning performance.

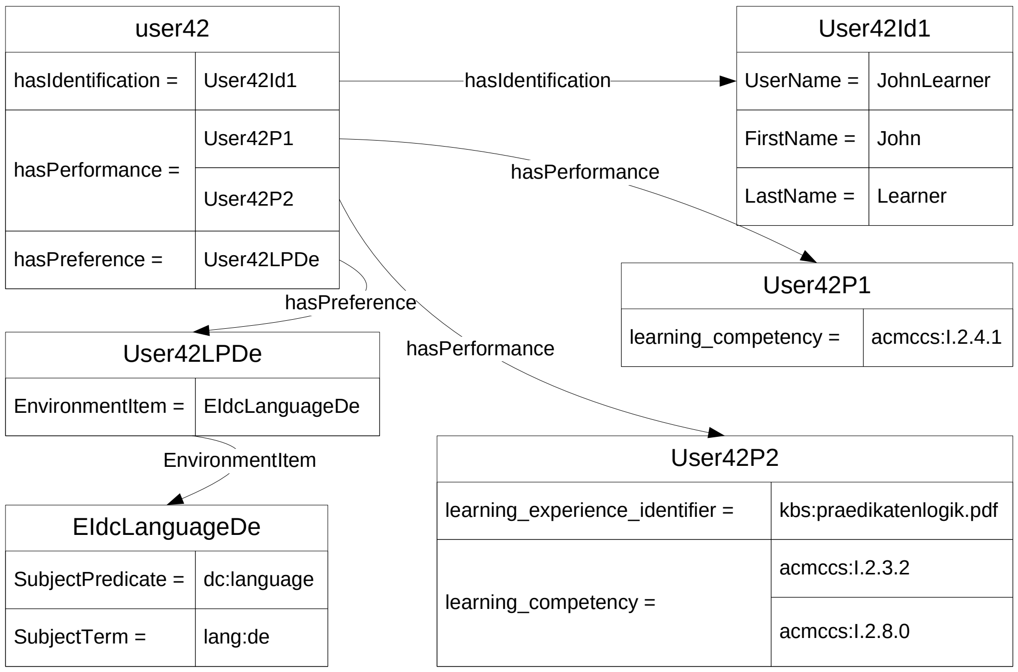


Figure 4: An Excerpt of Instance Examples for Environment User Preferences

To show a concrete instance of the environment preferences of a user, let us now consider a situation where a user John whose id is user42, as in our query from fig. 1 prefers a German language. In addition, he has attended two lectures, one on predicate logic and one on modal logic. An instance reflecting this situation described according to the environment user preference schemas is depicted in figure 4. His profile points to two performance objects: User1P1, and User1P2. The User1P1 is a performance record from modal logic lecture where user learner about the modal logic concepts (I.2.4.1 of ACM CCS). The User1P2 is a performance record from the predicate logic lecture where user learned about inference engines (I.2.3.2 of ACM CCS) and backtracking (I.2.8.0 of ACM CCS). The user42 profile also points to one preference object: User1LPDe. This is a language preference referring to a German language (lang:de).

3.1 Preference Based User Interface and Query Generation.

The user and environment preferences are utilized for user interface and initial query generation as well.

Figure 5 shows an example of a group of related fields described in RDF and generated from the user preferences together with additional descriptions of those fields. There is a group of related fields for a user input on a learning goal (topic) field. There are two options a user always have: typing a free text and selecting from a taxonomy. This is reflected by two fields within the environment concept group. The free text can be typed in three distinct fields (cardinality). Query related descriptions are predicate, operator, and preference. Query predicate gives a restriction field to which a user input is being applied. Operator (such as LIKE) gives an comparison operator to be applied to match user input with the predicate. Preference attribute is given to specify whether a user can express his interest/preference utility for further query processing and query relaxation. Similarly to the free text subject field, selection from taxonomy field points to query related attributes and a cardinality (in this case just 1). Furthermore, it

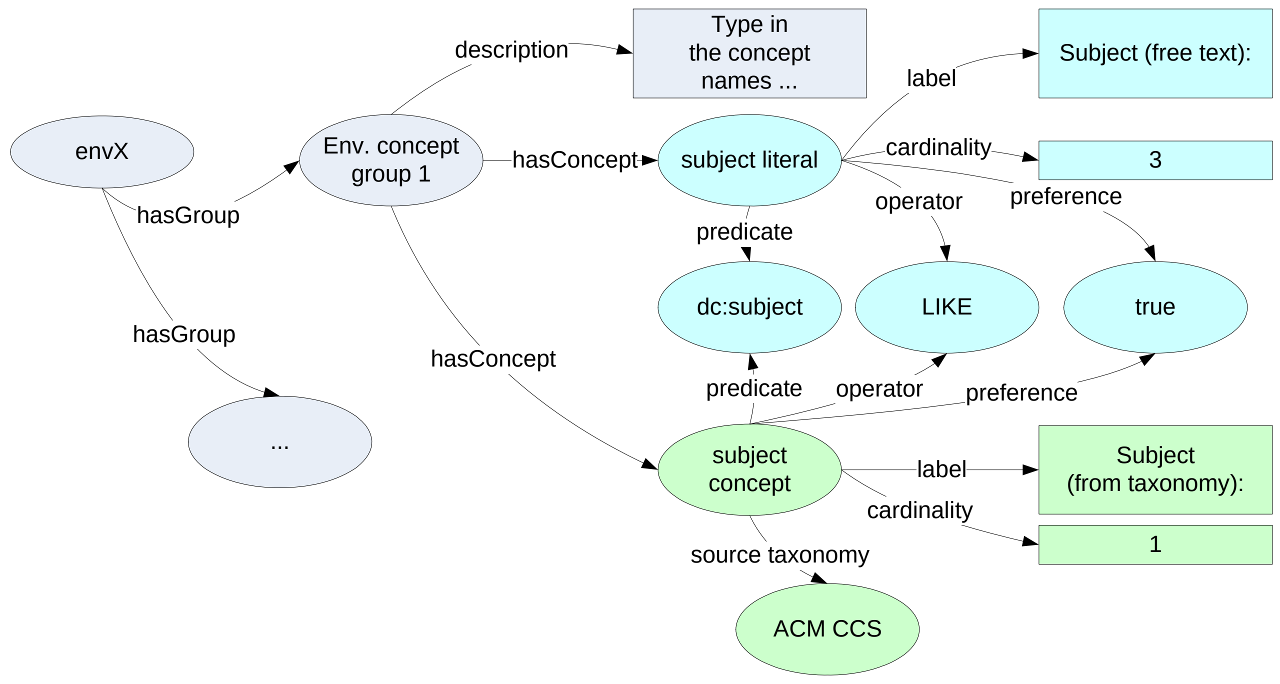


Figure 5: An Excerpt of a User Interface Environment for Query Composition

points to a default taxonomy to be used for concept generation.

A number of heuristics are used to produce such descriptions from the environment and user preferences. For example, LIKE operator is usually generated in interface description for text fields. A range operator is generated for time and date fields. For free text attributes, a triple of user interface fields is generated and one indented list is generated for taxonomical attributes. Taxonomies are suggested from pruning of resource data. If there is more than one taxonomy used in the resource data, a list with used taxonomies is generated that a user can select his own. A user can store his own environment model with own preference values. There is always a default environment which is usually used for novice users.

This model is utilized in an initial query construction process as well. The descriptions like predicate, operator and taxonomy are utilized when constructing the query restrictions. The preference values for attributes serve as an input to relaxation process. Furthermore, the user preference model also contains information on projection predicates. Before the query is submitted to the relaxation service, other preferences stored in user profile are considered as well for the initial query construction together with preference values if given.

3.2 User Preferences and User Modeling

There are two major areas of related work for preference models: preferences in databases and CP-Nets. Pioneering work on preferences in databases and their formal models have been provided in [Kie02, Cho03]. The preferences there are defined as partial orders over domains of attributes from a database schema. Preferences can be composed by different set operators such as union and intersection and so on. The composition creates another preference relation depending on the operator used to compose the preferences.

Another area where preferences have been considered is artificial intelligence. Conditional ceteris paribus represented as a network [BBHP99] or its extensions [BDSS05] are used to describe preference dependencies over predicates. They describe how certain predicates with its values depend on the other predicates leading to different outcomes based on the preference predicates.

In our rewriting approach we combine benefits of both approaches. We allow for explicit representation of preference relation over RDF property domains. As in RDF schema, properties are also defined as resources, we can consider similar relation over properties/predicates as well, thus providing unified model describing preference relations over both predicates as well as domains. In the next section we show how both, preference relations over predicates as well as domains can be learned from RDF queries in existing systems.

In addition, we integrate such preference models with user profiling. User profile contains preferences as its one dimension. Besides that, different aspects of user context and history are useful in our context such as learning performance and experience. These aspects differ from preferences as they rather represent knowledge about past user actions relevant for the environment and a user task. Formally, they can form relations or nets similar to preferences but with different semantics.

We also argue, that preferences and user profiles strongly depend on environment. Therefore we provide a generic environment model where preferences can be embedded and related to schemas and ontologies of a problem domain provided through the environment.

Preferences in databases have been integrated with traditional query languages in preference queries. The preference queries have been studied in the context of relational algebra [LL87a, Kie02, Cho03] or datalog [KG94, KKTG95]. The preference query languages developed on top of that work provide several operators which are understood by engines implementing them. We adopt a different approach. As the combination of preferences and different aspects of user profiles depend on the context and environment, we rather argue for a more flexible approach, i.e that rewriting rules, preference, and user relevant dimensions are configured when an environment is deployed. Furthermore, the rules as well as user profiles are continuously updated by learning preference over selected domain and environment models. Our approach implement the preference and user profile based query rewriting at a level above a query language engine. This allows for flexibility in query language, preference relations and user profile as well as query engine.

Our approach, relates to reasoning employed in cp-nets. However, while main cp-net reasoning target is to answer whether an outcome is preferred over another one based on the reasoning on ceteris paribus, in our approach we try to get a better outcome by modifying a query.

4 Preference Elicitation

An essential requirement for our approach to be useful in practice is the ability to deduce preferences as a basis for the rewriting rules. As described above, we distinguish between user and domain preferences. The elicitation of user preferences is an active field of research as many personalization techniques rely on correct user models. We have positioned our work with respect to the major techniques proposed in the literature and can directly be applied to our approach later. Finding and representing domain preferences is a less well investigated problem and deserves more attention. In the following, we discuss the elicitation of domain preferences in the context of a concrete target domain. On top of that, there has been a proposal for query relaxation based on the RDF Schema data model, that can also be used as a generic preference model for applications where neither domain nor user preferences are available.

4.1 Domain Preferences

In a recent article Hurtado and others have proposed a general query relaxation approach for RDF data based on the semantics of RDF schema [HPW07]. In particular, they propose a number of relaxation patterns for RDF query languages. These relaxation patterns are defined in terms of triple patterns and can therefore directly be implemented in our approach. Besides general relaxation patterns for RDF that are similar to the general rewriting possibilities mentioned previously (dropping triples) and relaxations adopted from earlier research on databases (breaking join dependencies) the authors also propose ontology-based relaxation patterns that take the schema and therefore the nature of the domain into account. In [HPW06] the following schema-based relaxations are mentioned:

Type relaxation: replacing a triple pattern (a, rdf:type, b) with (a, rdf:type, c), where (b, rdfs:subClassOf, c) follows from the model. For example, the triple pattern (?X, type, ConferenceArticle) can be relaxed to (?X, rdf:type,Article) and then to (?X, rdf:type, Publication).

Predicate relaxation: replacing a triple pattern (a, p, b) with (a, q, c), where (p, rdfs:subPropertyOf, q) follows from the model. For example, the triple pattern (?X, proceedingsEditorOf , ?Y) can be relaxed to (?X, editorOf , ?Y) and then to (?X, contributorOf , ?Y).

Predicate to domain relaxation: replacing a triple pattern (a, p, b) with (a, rdf:type, c), where (p, rdfs:domain, c) follows from the model. There are no domain declarations in Figure 1.

Predicate to range relaxation: replacing a triple pattern (a, p, b) with (b, rdf:type, c), where (p, rdfs:range, c) follows from the model. For example, the triple pattern (?X, editorOf , ?Y) can be relaxed to (?Y, rdf:type,Publication).

We consider these relaxations to be typical examples of domain preferences that can be used in our setting. The implementation of these patterns as re-writing rules is straightforward.

It turns out that in practical applications, following schema-based relaxation rules is often not enough as not all relevant preferences have a direct relation to the schema. Useful relaxations often rather depend on the nature of user queries typically posed to the system. In order to get a better idea of the impact of the nature of user queries on domain preferences, we analyzed the problem of eliciting domain preferences for the REASE system [DDM]. REASE is a repository of learning resources for the domain of semantic web technologies. The system has been developed in the context of the Networks of Excellence KnowledgeWeb and REWERSE. It currently has about 500 registered users. The system allows the user to pose keyword-based queries similar to search engines like Google. Internally, learning resources are represented by RDF metadata descriptions based on a complex schema. Details of the schema can be found in [Bra05]. This provides us with a certain degree of flexibility for mapping user queries onto the metadata description that we can exploit in the relaxation.

The REASE system already has a limited form of relaxation built into the search engine that has been designed based on extensive experiments for optimizing search results. In particular each user query is evaluated against different metadata fields using different weights for computing the aggregated result. In particular the following metadata fields are used as target for user queries

1. Title (with weight 1)
2. Description (with weight 0.7)
3. MainContributorNames (with weight 0.3)
4. OtherContributorNames, EducationalObjectives, AdditionalInformation, Curriculum and Prerequisites: (with weight0.1)

These empirically determined weights impose a preference relation over the different properties of a learning resource, that can be used in our system.

Further, in the user study reported in [DDM] we have analyzed the use of variations of a search terms and the impact on the completeness of query results. Table 1 shows the result for the case of users searching for information about problem-solving methods. The analysis shows that the users prefer the search terms 'problem solving methods' and 'psms' where the second search term will not return any result. We can use the information from the analysis and rewrite the query replacing the abbreviation 'psms' by the complete search term that is known to return results. Similar observations could be made for other search terms such as 'species' (searches for the different sublanguages of the web ontology language that are sometimes also referred to as 'layers'). This principle can be generalized by building a domain specific thesaurus

as a basis for re-writing search terms.

Query term	Percentage	Hits
problem solving method[s]	27%	2 / 2
psm[s]	22%	0 / 2
problem solving methods psm	9%	2 / 2
problem solving methods (advanced search)	7%	2 / 2
'problem solving method[s]'	6%	1 / 2
problem solving	5%	2 / 2

Table 1: Variation of Query Terms in the REASE System (from [DDM])

In summary, neither re-writing queries based on term lists and thesauri nor the schema-based rewriting of queries is new, the real benefit of our method is its ability to integrate different concrete approaches such as the ones mentioned into a common framework. In highly heterogeneous environments the combination of these different approaches is a real benefit that should not be underestimated.

5 Processing Rewritings

In previous work [DSW06] we presented the theoretical foundations of re-writing RDF queries based on sets of triple patterns. In the following, we describe the concrete implementation of this re-writing approach in SWI PROLOG. We chose SWI PROLOG as a basis for the implementation, because the declarative nature of PROLOG allows a straightforward implementation of re-writing systems. SWI PROLOG is especially suited, because its semantic web library contains many useful functions for manipulating RDF data and RDF Queries.

5.1 Rewriting Queries

As a first step in the re-writing approach, the refined user query (compare figure 1) is translated into a PROLOG representation. This is done using the functionality provided by the PROLOG-based SeRQL query engine provided by SWI PROLOG. After this translation, the query mostly consists of a list of predicates of the form `rdf(Subject, Relation, Object)` **that represent the FROM part of the query and a list of predicates of the form** `serql_compare(Operation, Argument1, Argument2)` **which represent the WHERE part of the query. These lists of predicates correspond to mandatory patterns[6] and conditions in our re-writing approach mentioned in section 2.1. Based on this representation, we can now also define re-writing rules in terms of special PROLOG predicates containing an identifier, a matching and a replacement pattern as well as conditions. The rewriting rules are applied to the PROLOG representation of the query. The result relaxed query is translated back into SeRQL using the corresponding functionality of SWI PROLOG. This is an example of a rewriting rule that addresses example 2 in terms replacing the path expression** `{Resource} requires {} subject {Prerequisite}`**, by** `{Resource} requires {Prerequisite}` **which is a less preferred but often used variation.**

```
'Prerequisite-Relaxation' @@
    pattern(where([rdf(Ressource, 'requires', Object),
                   rdf(Object, 'subject', Prerequisite)]),
            from([]))
==> replace(where([rdf(Ressource, 'requires', Prerequisite)]),
            from([]))
    && true.
```

This rule is a very specific one as it only applies in one special case and does not make use of information about preferences but contains a hard-wired strategy for relaxing a certain

[6]Currently optional patterns are not supported by the implementation, but the same machinery can be used to also include them into the rewriting.

aspect of the query. Making use of the predicates introduced in section 2.3 we can use rather generic re-writing rules that are guided by information from the environment and user model introduced in section 3.

Instead of hardcoding relaxations into rewriting rules, we can also use the environment and user model to parametize a set of generic rewriting rules. Below we have an example of such a generic rule. This rule looks for relations in query patterns that are mentioned in the user model and replaces the corresponding relation name by a less preferred relation based on information about domain preferences with respect to relations.

```
'Generic-Domain-Preference' @@
    pattern(where([rdf(Subject, Relation, Object)]),
         from([]))
==> replace(where([rdf(Subject, Relation, Object),
                   rdf(Subject, LessPreferredRelation, Object)]),
         from([]))
    && (domain_preferred_over(Relation, LessPreferredRelation)).
```

This rule is used to solve the problem in example 1. There the preferred relation is 'subject', the less preferred one 'title'. The explicit representation of both, a user model and a meta-model of the domain presented in section 3 provides us with a basis for computing the special predicates `user-preferred-over` and `domain-preferred-over` and using it to guide the query re-writing process. In particular, we can define these predicates in PROLOG using elements from the SWI RDF library to directly refer to the RDF-based representation of the user and environment model. Below we show the corresponding PROLOG predicate for computing the `domain-preferred-over` relation using in the rewriting rule above.

```
domain_preferred_over(Preference, SubPreference) :-
    rdf(EnvironmentItemInstance, 'SubjectPredicate', Preference),
    rdf(Preference, 'EnvironmentItem', EnvironmentItemInstance),
    rdf(Preference, 'hasImportanceOver', SubPreference),
    rdf(SubPreference, 'EnvironmentItem', SubEnvironmentItemInstance),
    rdf(SubEnvironmentItemInstance, 'SubjectPredicate', R),
    rdf(Preference, 'type', ClassVariable), ClassVariable != 'UserPreference',
    rdf(SubPreference, 'type', ClassVariable2), ClassVariable2 != 'UserPreference'.
```

As the relations 'subject' and the relation 'title' are in the user_prefered_over relation with respect to our example user, the original query will be rewritten and the pattern {`Resource`} `subject` {`Subject`} will be replaced by {`Resource`} `title` {`Subject`}. A second rewriting rule can be used to modify the value of 'Subject' in such a way that the query test if the title contains the value of the 'Subject' variable as a substring. The combination of these two generic rewriting rules solve the problem in example 1.

5.2 Control Strategy

As mentioned above, the main problem of the re-writing approach to query relaxation is the definition of an appropriate control structure to determine in which order the individual rewriting rules are applied to general new queries. Different strategies can be applied to deal with the situation where multiple re-writings of a given query are possible. Example are:

- User Interaction [Mot90]: possible re-writings are presented to the user who decides in which direction to proceed
- Heuristic Search [Sto03], [Stu04]: The best re-writing is determined based on the similarity of the resulting query with the original one.
- Divide and Conquer (i.e. Skylining) [LL87b, KK02]: The best results of each possible combinations of re-writings is returned.

In the current version of the system we have implemented a simple version of skylining. In particular, we interpret the problem of finding relaxed queries as a classical search problem. The search space is defined by the set $\mathcal{Q}$ of all possible queries. Each application of a rewriting rule R on a query Q is a possible action denoted as $Q \xrightarrow{R} Q'$. A query represents a goal state in the search space if it does have answers. In the current implementation we use breath-first

search for exploring this search space. Different from classical search, however, the method does not stop when a goal state is reached. Instead the results of the corresponding query are returned, goal state is removed from the set of states to be expanded and search is continued until there are no more states to be expanded. As each goal state represents the best solution to the relaxation problem with respect to a certain combination of re-writings, the goal states form a skyline for the rewriting problem and each of them is returned to the user together with the query answers. The second difference to classical search is that we do not allow the same rule to be applied more than once with the same parameters in each branch of the search tree. The only kind of rules that can in principle be applied twice are rules that add something to the query (Rules that delete or replace parts of the query disable themselves by removing parts of the query they need to match against). Applying the same rule that extend the query twice leads to an unwanted duplication of conditions in the query that do not change the query result, but only increase the complexity of query answering [GHM04].

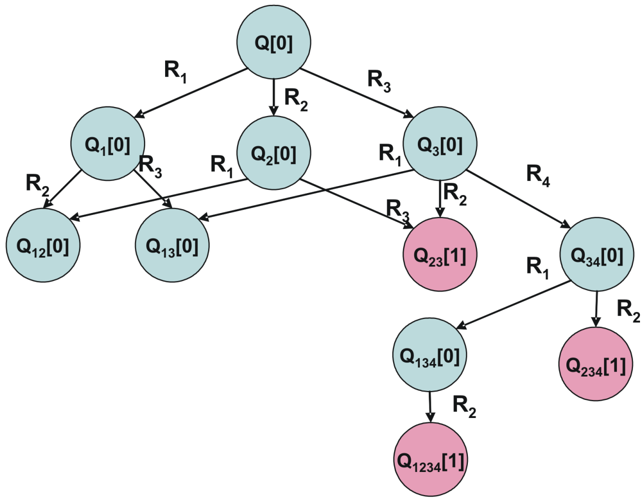


Figure 6: Search Space for example rewritings

Figure 6 illustrates the control strategy exploring the search space for the query given in Figure 1. The nodes in the graph represent queries - the number in square brackets denotes the number of answers. The edges between nodes represent re-writings. There are four possible re-writings R_1 to R_4, where R_1 relax the prerequisite requirement in the query, i.e. the first rewriting rule in Section 4.1. R_2 relax a string comparison in a SERQL query where a two strings must no longer be equal but the first string can now be a substring of the second, e.g. the string "inferenceengines" need no longer to be equal but only a substring of the title. R_3 and R_4 are two instantiations of the generic domain preferred rewriting (cf. second rule in section 4.1); R_3 replaces subject by title in the query whereas R_4 replaces title by description (cf. example 1). The search space is initialized by the original query Q. R_1 to $R3$ are all applied resulting in the rewriting queries Q_1 to Q_3. Because none of these rewritten queries return any results the rewriting process proceeds with the application of R_1 to R_3 to the rewritten queries Q_1 to Q_3. The resulting queries can be merged to Q_{12}, Q_{13}, and Q_{23} because it does not matter if first R_1 and then R_2 is applied or vice versa. Additionally R_4 is now applicable to Q_3 resulting in Q_{34} because after replacing the subject by title (by R_3) the title can be replaced by the description. Furthermore Q_{23} is a goal state because it returns one resource as answer to the (relaxed) query. Figures 5 shows also how Q_{34} is further rewritten to the new goal states Q_{234} and Q_{1234}.[7] Finally we get three re-writings Q_{23}, Q_{234} and Q_{1234} of the original query; each of them returns one learning resource.

[7]The expansion of other states is omitted.

6 Implementation

We have implemented the relaxation approach described above using state of the art semantic web technologies in the context of the European Research Networks KnowledgeWeb (Realizing the Semantic Web) and Prolearn (Network of Excellence in Professional Learning). The resulting system is an extension of the general e-learning infrastructure described in [DHNS04] and tested it on a repository of e-learning resources in the area of general computer science that have been annotated with RDF-based metadata using the schema proposed in [Bra05]. As the resources are provided by a large number of different authors, the metadata description that form the basis of search contains many of the problems mentioned above which makes this data set an ideal test case for our approach. In the following, we provide an overview of the general system architecture and explain the prototypical search interface available on the web.

6.1 System Architecture

Figure 7 shows the architecture of the implemented system. The system consists of a central component that contains the query rewriting functionality and the relaxation strategy. This component receives a user request in terms of a SeRQL query that has been generated by the user interface by automatically refining the user request based on known user preferences. This query is translated into a PROLOG representation using the functionality provided by the SWI-PROLOG semantic web library. The rewriting rules which are specified in a PROLOG knowledge base are then applied to this representation of the user query. Most of the rewriting rules contain conditions that refer to user or environment preferences or both. The corresponding conditions are evaluated over the corresponding RDF models that contain the environment and user data, again making use of the PROLOG RDF interface provided by SWI PROLOG. The result of this rewriting is a series of more general queries that are translated back into SeRQL and are issued to a Sesame RDF repository that contains the actual data. The results of these queries are returned to the user interface together with the relaxed query in order to enable the user to understand the basis on which the results were achieved. In the following we discuss the different functional components of the system in more detail.

6.2 User Interface

The prototypical search interface of the system combines user preference elicitation with a query formulation dialog. The original version of the personalized search described in [DHNS04, DSKN08] had just a query formulation for restrictions of subject of resources. We have extended the user interface with generating environment based on the environment schema, a default environment for novice users, and a user preference elicitation. A user interface of such a personalization search environment is depicted in fig. 8.

The default environment consists of items for specifying subject concepts, title, description, and language as query literals. Each of the attributes on the user interface have a preference elicitation slider. The slider is used to specify a value measuring an importance of a preference of particular attribute to a user. Internally, these numerical values are translated into qualitative relations describing a total order of importance over the different aspects. With respect to example 1 these relations specify in which order subject, title and description are considered as a source of information about the topic of a resource. A button for opening a dialog where a user can specify the value preferences and their order is provided where it is appropriate (e.g. *Subject Values Preferences* button or *Language Preferences* button). The source of values for subject preferences is in our case the ACM CCS taxonomy, used also for selecting concepts on the user dialogs. We use standard set of language identifiers as a source for values for language preferences. The value preference dialog displays a tree, a graph or a set of concepts with value labels determining the importance of the preference. When a user points to a concept, a slide bar is drilled down to change the preference importance value. If user needs to extend her restrictions, he can do that by selecting from other schema attributes which are offered when he presses *Add Attributes* button. The attributes which a user filled in on the user interface are used to construct the restriction part of queries.

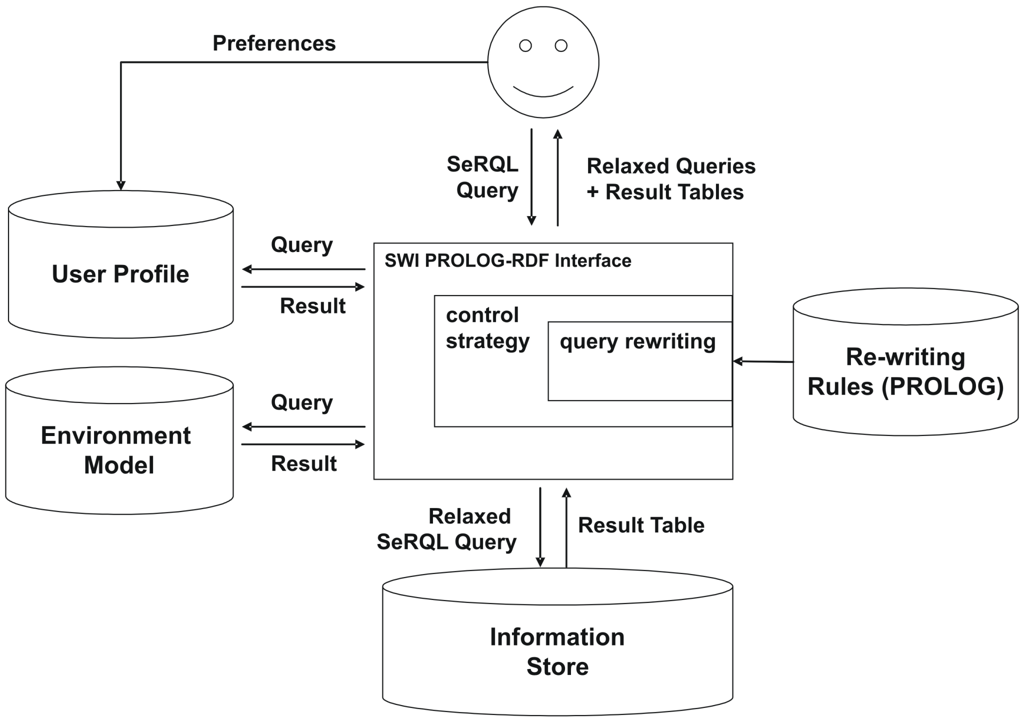


Figure 7: System Architecture

7 Conclusions

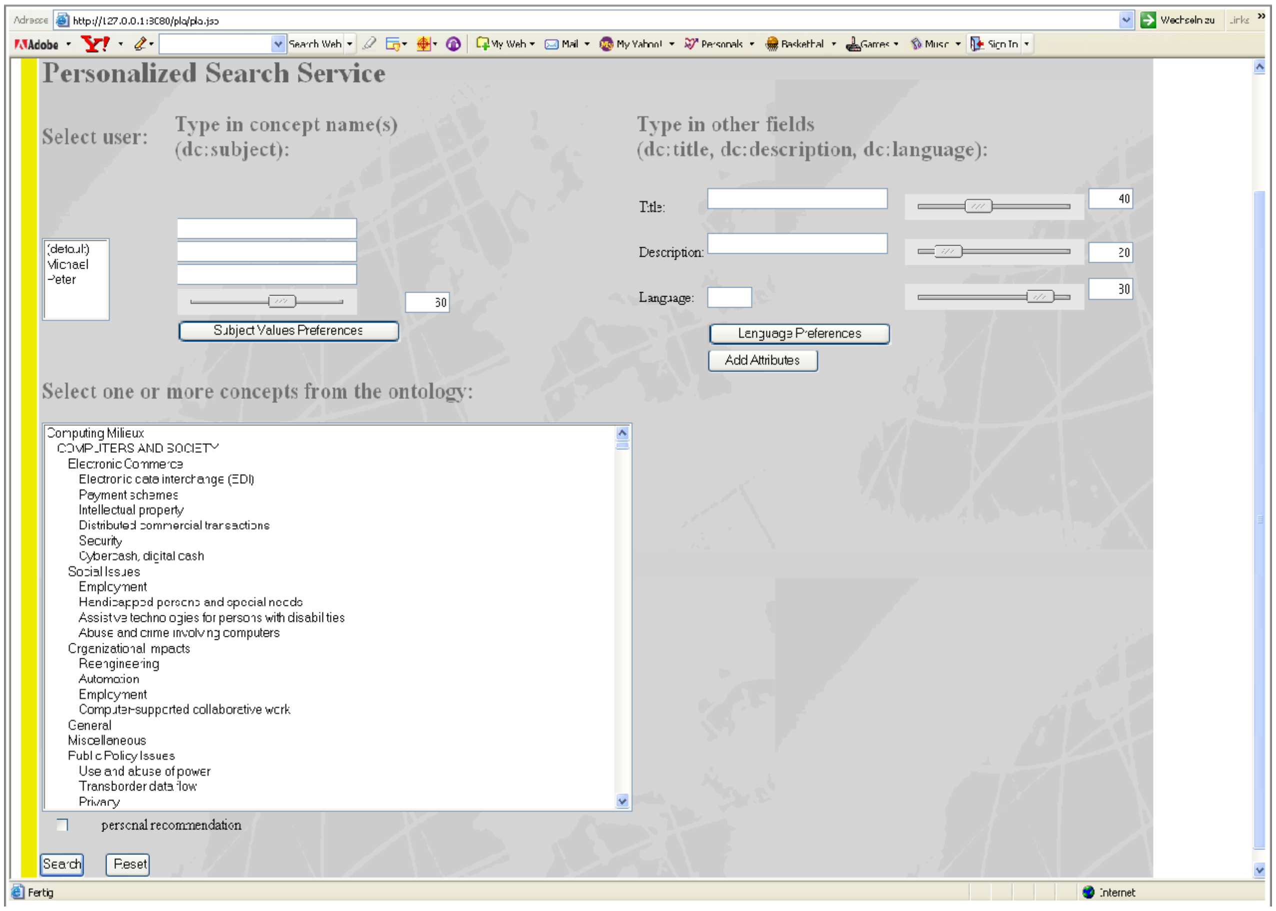
In this paper we addressed the problem of querying RDF data that shown irregularities due to multiple authorship and non-compliance to a standardized metadata schema. We illustrated this problem using an example from the e-learning domain, but we are convinced that the problem is a general one that is inherent in the idea of metadata on the semantic web. We have presented an approach for successively re-writing queries based on background knowledge. In particular, we have described how background knowledge about different alternative representations can be used to define generic relaxation rules that can be applied across different domains provided that we have a suitable environment model. We have implemented and tested the approach in the domain of e-learning using real world data about e-learning resources in computer science.

Open questions concerned with the approach are about suitable ways of acquiring the necessary information about user preferences as well as the application environment. For the case of user preferences there is a large body of work in the area of user modeling including methods for automatically learning preferences based on user behavior. The acquisition of information about the environment model will probably be more of a challenge, because it is not clear whether alternative representations of the same information can be detected by observing the user. We have shown some strategies to acquire this information from usage logs. However, this problem needs to be further studied.

In summary, we can say that we need more experience with real data and real users in order assess the effort connected with the acquisition of the knowledge necessary to successfully apply our approach in different domains. While for the domain of e-learning the benefits have been shown, this remains to be done in other domains.

References

[BBHP99] Craig Boutilier, Ronen I. Brafman, Holger H. Hoos, and David Poole. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of the*


http://127.0.0.1:8080/plq/plq.jsp
Personalized Search Service
Select user:
(default)
Michael
Peter
Type in concept name(s) (dc:subject):
Subject Values Preferences
Type in other fields (dc:title, dc:description, dc:language):
Language Preferences
Add Attributes
Select one or more concepts from the ontology:
Computing Milieux
COMPUTERS AND SOCIETY
Electronic Commerce
Electronic data interchange (EDI)
Payment schemes
Intellectual property
Distributed commercial transactions
Security
Cybercash, digital cash
Social Issues
Employment
Handicapped persons and special needs
Assistive technologies for persons with disabilities
Abuse and crime involving computers
Organizational impacts
Reengineering
Automation
Employment
Computer-supported collaborative work
General
Miscellaneous
Public Policy Issues
Use and abuse of power
Transborder data flow
Privacy
personal recommendation
Search
Reset
Fertig
Internet

Subject: 30
Title: 40
Description: 20
Language: 30

Figure 8: Prototypical Search Interface

Symposium on Uncertainty in Artificial Intelligence, 1999.

[BDSS05] Ronen I. Brafman, Carmel Domshlak, Solomon E. Shimony, and Yael Silver. Tcp-nets for preferences over sets. In *WS on Advances in Preference Handling*, 2005.

[BK04] J. Broeskstra and A. Kampman. Serql: A second generation rdf query language. In *SWAD - Europe Workshop on Semantic Web Storage and Retrieval*, Amsterdam, The Netherlands, Nov. 2004.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[Bra05] Jan Brase. *Usage of metadata.* Phd thesis, University of Hannover, 2005.

[Cer92] Stefano Ceri. A declarative approach to active databases. In Forouzan Golshani, editor, *ICDE*, pages 452–456. IEEE Computer Society, 1992.

[Cho03] Jan Chomicky. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):1–40, December 2003.

[DC2] The dublin core metadata initiative. `http://dublincore.org/`.

[DDM] Jörg Diederich, Martin Džbor, and Diana Maynard. Rease: The repository for learning units about the semantic web. *New Review of Hypermedia and Multimedia.* accepted for publication.

[DHNS04] Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Personalization in distributed e-learning environments. In *Proc. of WWW2004 — The Thirteen International World Wide Web Conference*, New Yourk, May 2004. ACM Press.

[DS05] Peter Dolog and Michael Schäfer. A framework for browsing, manipulating and maintaining interoperable learner profiles. In Liliana Ardissono, Paul Brna, and Antonija Mitrović, editors, *Proc. User Modeling 2005: 10th International Conference, UM 2005*, volume 3538 of *LNAI*, Edinburgh, Scotland, UK, July 2005. Springer.

[DSKN08] Peter Dolog, Bernd Simon, Tomaz Klobucar, and Wolfgang Nejdl. Personalizing access to learning networks. *ACM Transactions on Internet Technologies. Special Issue on Distance Education*, 8(2), May 2008.

[DSW06] Peter Dolog, Heiner Stuckenschmidt, and Holger Wache. Robust query processing for personalized information access on the semantic web. In *Proceedings of the 7th International Conference on Flexible Query Answering Systems (FQAS-06)*, 2006.

[GGM92a] Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992.

[GGM92b] Terry Gaasterland, Parke Godfrey, and Jack Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3/4):293–321, 1992.

[GHM04] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *ACM Symposium on Principles of Database Systems (PODS)*, Paris, France, June 2004.

[Hay04] Pat Hayes. Rdf semantics. Recommendation, W3C, 2004.

[HPW06] Carlos Hurtado, Alexandra Poulovassilis, and Peter Wood. A relaxed approach to rdf querying. In *ISWC'2006 — 5th International Semantic Web Conference*, Lecture Notes in Computer Science, Athens, GA, USA, November 2006. Springer-Verlag.

[HPW07] Carlos Hurtado, Alexandra Poulovassilis, and Peter Wood. Query relaxation in rdf. *Journal of Data Semantics*, 2007.

[KG94] W. Kiessling and U. Güntzer. Database reasoning—a deductive framework for solving large and complex problems by means of subsumption. In *Proceedings of the 3rd Workshop on Information Systems and Artificial Intelligence*, volume 777 of *LNCS*, pages 118–138, New York, USA, 1994. Springer.

[Kie02] Werner Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[KK02] Werner Kießling and Gerhard Köstler. Preference sql - design, implementation, experiences. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB02)*, pages 990–1001, 2002.

[KKTG95] G. Köstler, W. Kiessling, H. Thöne, and U. Güntzer. Fixpoint iteration with subsumption in deductive databases. *J. Intel. Inf. Syst*, 4:123–148, 1995.

[LL87a] M. Lacroix and P. Lavency. Preferences: Putting more knowledge into queries. In *Proceedings of the International Conference on Very Large Data Bases*, pages 217–225, Amsterdam, The Netherlands, 1987.

[LL87b] M. Lacroix and Pierre Lavency. Preferences; putting more knowledge into queries. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *Proceedings of 13th International Conference on Very Large Data Bases (VLDB87)*, pages 217–225. Morgan Kaufmann, 1987.

[Mot90] A. Motro. Flexx: A tolerant and cooperative user interface to database. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–245, 1990.

[Nil01] M. Nilsson. Ims metadata rdf binding guide. http://kmr.nada.kth.se/el/ims/metadata.html, May 2001.

[oCm02] Assosiation of Computing machinery. The acm computer classification system. http://www.acm.org/class/1998/, 2002.

[PPW03] George Papamarkos, Alexandra Poulovassilis, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *SWDB*, pages 309–327, 2003.

[Sto03] Nenad Stojanovic. On analysing query ambiguity for query refinement: The librarian agent approach. In *Conceptual Modeling - ER 2003*, volume 2813 of *Lecture Notes in Computer Science*, pages 490 – 505. Springer-Verlag, 2003.

[Stu04] H. Stuckenschmidt. Similarity-based query caching. In *6th International Conference on Flexible Query Answering System FQAS*, volume 3055 of *Lecture Notes in Artificial Intelligence*, pages 295–306, Lyon, France, 2004. Springer Verlag.

[SvHdW$^+$04] Heiner Stuckenschmidt, Frank van Harmelen, Anita de Waard, Tony Scerri, Ravinder Bhogal, Jan van Buel, Ian Crowlesmith, Christiaan Fluit, Arjohn Kampman, Jeen Broekstra, and Erik M. van Mulligen. Exploring large document repositories with rdf technology: The dope project. *IEEE Intelligent Systems*, 19(3):34–40, 2004.