Robust Query Processing for Personalized Information Access on the Semantic Web

Peter Dolog[1], Heiner Stuckenschmidt[2], Holger Wache[3]

[1] L3S Research Center, Hannover, Germany, dolog@l3s.de
[2] Universität Mannheim, Germany, heiner.stuckenschmidt@uni-mannheim.de
[3] Vrije Universiteit Amsterdam, The Netherlands, holger@cs.vu.nl

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-constraint 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.

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.
- The data is semi structured. Sources do not have to describe all aspects of the information resources.
- There is no fixed integrated schema. Each source can have its own schema, sources may make partial use of different available schemas.

With the increasing popularity of RDF as a representation language in domains such as medicine [17] or e-leaning [4] 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. Based on the assumed kind of mismatch between the users intention and the formulated query there are different techniques used. We consider two basic mechanisms of cooperative query processing, *query refinement* and *query relaxation* which are briefly presented in the following.

1.1 Query Refinement and Relaxation

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, for example in terms of the type of the objects she wants to retrieve and maybe one or two properties. Taking an example from the domain of e-learning, the user might be able to specify that she is looking for a lecture on the Java Programming Language. Learning resources, however, are often annotated with a fair amount of metadata that specifies important information such as the assumed level of expertise and required previous knowledge. In order to select learning resources that are suited for the user, these additional properties have to be specified in the query as well. Dolog et al [4] show that this information can be included into a user query based on a user profile. They describe a method for automatically refining queries with information from the user profile thereby enabling a pre-selection of query answers.

A problem of the automatic refinement of queries lies in the fact that it often overshoots the target instead of too many results an automatically refined query often returns no result at all, because none of the resources exactly matches the users needs. A possible solution to this problem is to successively relax the constraints imposed in the refinement step. Different Techniques for relaxing queries have been proposed in the database area. Gaasterland et al [6] provide a unifying view on different relaxation techniques in terms of replacing subexpressions in the query. In previous work we described an approach for relaxing conjunctive queries over description logic knowledge bases by removing conjuncts from the query in a particular order [16,18] .

1.2 Contributions

In this paper, we build upon existing work on query refinement for personalized information access and extend it in the following ways:

- We describe a framework for information access that combines query refinement and relaxation in order to provide robust, personalized access to heterogeneous RDF data.
- We propose an implementation of the framework in terms of conditional rewriting rules for RDF query patterns.
- We discuss the application of the framework in the context of an existing e-learning system.

2 Background

User queries to the open learning environment will consist of one or several keywords related to the topic the user wants to learn about. The result is a list of learning resources including information about the subject and the title of the resource as well as a description of the content. In order to produce this list, the user request is translated into a query an RDF query language that matches the metadata use to describe learning resources in the system. Figure 1(a) shows the query corresponding to a user request for ”inference engines” in SeRQL[1] syntax .

```
SELECT * FROM
  {Resource} subject {Subject},
  {Resource} title {Title},
  {Resource} description {Description},
WHERE
  Subject Like "inference engines"
```

(a) Basic Query

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

(b) Refined Query

Fig. 1. Rewriting-based Query Refinement

In a second step, the general query shown in figure 1(a) is adapted to better reflect the learning preferences of the user. In this step, the query is refined by extending it with additional constraints that are derived from the user profile. This is done by extending the path expression in the FROM and by adding variable assignments in the WHERE part of the query. Typical additions to a query are a restriction of the language of resources to the preferred language of the user and a general constraint demanding that the user must have all competencies that are required for understanding the resource.

Figure 1(b) shows the result of refining the general query from figure 1(a) with language and competence constraints.

In practice it turns out that the approach of personalization by query refinement suffers from serious problems. In fact problems occur in both steps of the query formulation process. The first problem already occurs when the basic query is formulated. In our open learning repository, this query does not return any result despite the fact that there are 8 resources on the subject. The reason for this is that only about 10% of all resources are completely annotated with subject, title and description. Unfortunately, all 8 potential answers miss at least one of these properties and are therefore not returned as an answer. This problem can be reduced but *making predicates or triples optional in the query*.

Another problem lies in the fact, that 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 ”Lernen” (German for ”learning”) no resources are returned despite the fact that there are resources for instance about Bayesian learning and learning in case based reasoning. The problem is here that in the case of the first resource the term learning only occurs in the title, but not the subject. In the case of the second resource, the term only occurs in the description and is mentioned neither in the subject nor the title of the resource. This problem can be used by *replacing predicates in patterns by other ones* based on domain knowledge.

[1] http://www.openrdf.org/doc/sesame/users/ch06.html, We omit namespaces for the sake of readability

We can observe the similar problems in connection with the refinement of the general query based on the user profile as the competence of a user is often defined in terms of learning resources that were successfully mastered by the student. If the subject of previous courses do not exactly match the requirements for a new resource, thus source is excluded from the results even if the topics are very similar. A mechanism for *replacing values in patterns* in query restrictions should be provided to solve the problem by replacing required competence by a similar or more general one. A more general mechanism for approaching this kind of problem is to *replace values by variables and restrict the value of the variable using an appropriate predicate* in the WHERE part of the query.

Further Problems arise from the inflexible nature of the rewriting mechanism that instantiates variables with the preferred value and leaves no room for taking the second best choice if the available resources are for example not in the preferred language, the user does not have all but most of the required competencies or the competencies of the user are not the same but very similar to the required ones. We will come back to these examples when we discuss our solution to the problem.

3 Rewriting RDF Queries

We propose an approach for query rewriting based on Event-Condition-Action (ECA) rules (see e.g. [13]) to solve the problem of over-constraint queries. This rewriting relaxes the over-constraint query based on rules and in order defined by events and conditions. 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 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 [8]. A query over these resources is formulated as 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 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 query Q is defined as the tuple $\langle M_Q, O_Q P_Q\rangle$ with $M_Q, O_Q \in \mathcal{TR}$ and $P_Q \subseteq \mathcal{P}$ where M_Q is the set of mandatory pattern (patterns that have to be matched by the result, O_Q is a set of optional pattern (patterns that contribute to the result but do not necessarily have to match the result) and $\mathcal{P}$ is the set of predicates with name $\mathcal{PN}$, defined over $\mathcal{T}$, and $\mathcal{V}$.*

The triple patterns M_Q in a query Q determine those ground triples where a substitution τ exists (τ may also exist for a subset of patterns in O_Q). Formally a substitution

τ is a list of pairs (X_i, T_i) where each pair tells which variable X_i has to be replaced by $T_i \in \mathcal{T} \cup \mathcal{V}$. Applied to a query, the substitution τ replaces variables in TR_Q with appropriate terms. If $\tau(M_Q)$ is equal to some ground triples then the substitution is valid. All valid substitutions for M_Q plus existing substitutions for O_Q constitute answers to the query. The predicates P_Q restrict these substitutions additionally because only those bindings are valid answers where the predicates, i.e. $\tau(P_Q)$, are also satisfied. The predicates define additional constraints for the selection of appropriate triples. Using this abstract definition, the query in figure 1(a) 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 $subject, title, description, \text{“}inferenceengines\text{”} \in \mathcal{T}$ 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 $Resource, Subject, Title, Description, R1, R2, R3 \in \mathcal{V}$, $subject, title, description, \text{”}inferenceengines\text{”} \in \mathcal{T}$ and $like, = \in \mathcal{PN}$. This 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 we will refer in the following 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) [3,12] 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 2 (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.*

For conditions the same constructs as for queries are used where the possible results are also constrained by predicates. Patterns and replacements formally have the same structure like 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(a) with the language preference of the user 42.

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

where $languagePrefernce$ is a predicate which looks in his user profile for the language preference of $User$ who is in our case user 42.

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 rules 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 an substitution.

Definition 3 (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 [1] 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 4 (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 $M'_Q \subseteq M_Q$, $O'_Q \subseteq O_Q$ and $P'_Q \subseteq P_Q$ 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 $M^R_Q = (M_Q \setminus M'_Q) \cup \theta(TR_{RE})$, $O^R_Q = (O_Q \setminus O'_Q) \cup \theta(TR_{RE})$ and $P^R_Q = (P_Q \setminus P'_Q) \cup \theta(P_{RE})$ with $RE = \langle TR_{RE}, P_{RE}\rangle$.

Informally spoken, 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:

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

Please note that the language preference of user 42 is “de” which means German.

4 Domain-Dependent Relaxation

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 or course arbitrary combinations of these actions:

- *Making Patterns optional* — this provides a query which considers a situation that some of the patterns do not have to appear in metadata. A query then gives also results where particular predicate relaxed to an optional predicate does not occur;
- *Replacing Value* — this provides a query where particular predicate value is replaced with another value or a variable. Taxonomies may be used to provide siblings, more general terms, and so on;
- *Replacing Patterns/Predicate* — this provides a query where particular triple resp. predicate in restrictions is replaced by another triple resp. predicate. A domain knowledge is employed for this purposes. For 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 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 exact 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) This predicate is defined in the context of a specific user 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 predicate to be non-transitive. The concrete implementation depends on the chosen representation of the domain and the user profile. In the following, we show how domain and user specific relaxations can be specified using these predicates.

Domain Preferences For example, in the learning environment a user searches a resource with a specific subject. But if there is no resource with that subject then we would like to relax the query that the subject term can also appear in the title of a resource description. This strategy can be derived from a domain preference stating that the "subject" relation has the highest priority as it can be assumed to most precisely reflect the content of a resource followed by the "title" and finally the "description" relation. For the actual relaxation process each of these relations is implemented by a rewriting rule like the following:

$$\begin{aligned}
PA &= (\{(Resource, subject, Subject)\}, \emptyset, like(Subject, Value) \\
RE &= (\{(Resource, R, New)\}, \{(Resource, subject, Subject)\}, \\
&\quad \{like(New, Value)\}) \\
CN &= \{domain - preferred - over(subject, R)\}
\end{aligned}$$

The triple {`Resource`} `subject` {`Subject`} looks for any resource `Resource` with subject `Subject`. The predicate *like* constrains the variable `Subject` to the user's term (`Value`), i.e. the subject the user is looking for. If a query contains such an triple and such an predicate then the rewriting rule is applicable.

The replacement part of the rule defines how the matched triple and predicate has to be replaced. The subject Pattern is replaced by the second triple {`Resource`} `R` {`New`}. The pattern about the subject of the resource is made optional because the reason that the original query did not produce any result might have been that the relevant resources did not have any subject.

The rewriting rule can be applied to the query in Figure 1b). The result is shown in Figure 2.

```
SELECT * FROM
  [{Resource} subject {Subject}],
  {Resource} title {New},
  {Resource} title {Title},
  {Resource} description {Description},
  {Resource} language {Language},
  {Resource} requires {} subject {Prerequisite},
  {User} hasPerformance {} learning_competency {Competence}
WHERE
  New Like "*inference engines*",
  Prerequisite = Competence,
  Language = de,
  User = user42
```

Fig. 2. Relaxed Query

This rule can easily be generalized to a rule that applies to any kind of preferences between relations in query patterns. In this case the concrete relation subject has to be replaced by an appropriate variable. In this case, the rewriting process is completely controlled by the definition of preferences between relations and values.

User Preferences Another kind of relaxation is the rewriting the over-constrained query according to the knowledge about the user. In the learning scenario the user might prefer learning resources in German but Dutch may also be okay. This knowledge is used to refine the query, i.e. looking for resources in German. However if there is no resources in German then the query can be relaxed according to user's second preference.

In contrast to the domain preferences mentioned in the last section that can be specified inside the application, these preferences can be different for each user. As a consequence we have to provide an interface where each user can specify his or her personal preferences that can then be stored in the user profile. A user interface for that is very simple, a slider is provided next to each item at a user interface for specifying an importance of the predicate for a user. The default slider positions are provided according to a default domain knowledge. Using our general environment model, these preferences can be used in the same way as domain preferences once they have been entered by the user. The use of the corresponding preference relations in the rewriting process is illustrated by the following example.

$$\begin{aligned} PA &= (\{(Resource, language, Language), \emptyset, \{Language = L1\}) \\ RE &= (\{(Resource, language, Language), \emptyset, \{Language = L2\})\} \\ CN &= \{user-preferred-over(L1, L2)\} \end{aligned}$$

The rule tries to relax user's first language preference to his second reference as stored in his profile. The value of the first preference is replaced by the value of the second preference. Assuming that user-preferred-over(de,nl) holds, the result of applying this rewriting to the query in figure 2(b) is the query shown in figure 3

```
SELECT * FROM
  [{Resource} subject {Subject}],
  {Resource} title {TMPTitle},
  {Resource} title {Title},
  {Resource} description {Description},
  {Resource} language {Language},
  {Resource} requires {} subject {Prerequisite},
  {User} hasPerformance {} learning_competency {Competence}
WHERE
  TMPTitle Like "*inference engines*",
  Prerequisite = Competence,
  Language = nl,
  User = user42
```

Fig. 3. Example of rewritten Query

The above rewriting rule can easily be generalized to a rewriting for any value preference. The condition then would be that the environment items of such preferences must not refer to the subject predicate `language` but only to the same predicate (which is represented as a variable). So a variable instead of the literal `language` in the condition yields into a general rewriting rule for value preferences.

5 Relaxation Strategies

As we have seen in the examples above, different modifications can be combined either in the same rewriting rules or by successive application of re-writing rules on the result of the previous one. In the example above the order in which the rules are applied does not matter as the changes are independent, we can easily see, however, that sets of re-writing rules do have have to be confluent. If we add a rule that remove the subject pattern, the language preference can still be changed independently from others, but we have to decide whether to remove the subject pattern or whether to ask for titles that contain the topic instead. Different techniques have been proposed to cope with this situation. We can distinguish the following general strategies:

User Interaction As described in [11] possible re-writings and the resulting queries can be represented as a graph structure. This representation can be used to guide the user through the space of re-writings by presenting all possible rewriting and the corresponding answers to the user letting her decide which way to proceed.

Heuristic Search On a more general level, the process of query refinement can be formulated as a heuristic search problem. The search space consists of all possible queries and re-writing rules are used to navigate in the search space. Different search strategies can be employed to determine the best sequence of re-write operations. Heuristics used range from the number of results, the degree of ambiguity of the query [14] or the effort needed to recompute the result of the new query from the result of the previous one [15]. A general problem of heuristic search approaches to relaxation strategies is the lack of an appropriate termination criterion that determines when the search has reached a goal state. This problem can be addressed by limiting the search depth and picking the solution with the best heuristic evaluation. Another possibility is to implement an interactive search paradigm where the system evaluates different alternatives using heuristic search and lets the user decide with of the sequences to use.

Divide and Conquer In the context of relational databases, divide and conquer techniques have been investigated that process different possible relaxations in parallel and merge the results of these parallel evaluations. Promising approaches of this type are top-k methods [5,7,2] and skylining algorithms [10,9]. Top-k needs a function which associates a ranking number with each answer. The k best answers)with respect to this function) across all possible relaxations are returned. Obviously, the function should operate independently from the relaxation in order to select answers from different relaxations. Skylining assume that the data is organized according to several independent dimensions. It tries to return the best answers according to each dimension. In this context the dimensions are the different possibilities of relaxations. Skylined relaxation returns the best answer from each possible relaxation.

6 Implementation

We have implemented and tested the relaxation approach described in this paper in a prototypical fashion. In particular, we defined a PROLOG based language for specifying and operationalizing re-writing rules. The corresponding languages operates on RDF queries specified in SeRQL, but can easily be adapted to any other RDF query language. Specifications of the rewriting rules mentioned in the last section are shown in the appendix. We tested the rewriting approach on an existing e-learning data that can be accessed via the Sesame System. Currently, the relaxation approach is being integrated into the distributed e-learning system described in [4].

7 Conclusions and Further Work

In this paper, we have proposed a framework for query relaxation to provide personalized information access to resources on the semantic web. The framework is based on the event-condition-action (ECA) paradigm where events are matching patterns, conditions are based on ordering between concepts of common sense domain knowledge and user preferences, and actions are the replacements for relaxing a query. The relaxation is based on the term rewriting principles enhanced with conditions provided by the ECA paradigm. This integration is a contribution to the term rewriting domain. The relaxation is controlled by conditions from domain and user preference ontology. The order is given by importance of predicates and values in the ontology for environment preferences, user profile, and common sense domain knowledge. This makes the approach very well suitable for the access to metadata on the semantic web as the domain knowledge helps to overcome the fact of heterogeneity and differences in how the metadata are authored on the semantic web.

In our further work, we would like to concentrate on the algorithms for determining termination of relaxation. We have considered several strategies in this paper but it requires further studies to give a recommendation how to decide among them. We also would like to experiment with different user preference models and how they contribute to the relaxation process. Last but not least, user preference elicitation methods and techniques needs to be studied to get as accurate user preferences as possible to support personalized access to information on the semantic web.

7.1 Acknowledgments

This work is partly supported by KNOWLEDGEWEB (IST-2004-507482) and PROLEARN (IST-2004-507310), EU funded projects in the semantic web and technology-enhanced learning areas.

References

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

2. Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data EngineeringE*, pages 369–. IEEE Computer Society, 2002.
3. Stefano Ceri. A declarative approach to active databases. In Forouzan Golshani, editor, *ICDE*, pages 452–456. IEEE Computer Society, 1992.
4. 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.
5. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the twentieth ACM SIGMOD- SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, New York, NY, USA, 2001. ACM Press.
6. Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992.
7. Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 419–428. Morgan Kaufmann, 2000.
8. Pat Hayes. Rdf semantics. Recommendation, W3C, 2004.
9. 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.
10. 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.
11. A. Motro. Flexx: A tolerant and cooperative user interface to database. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–245, 1990.
12. 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.
13. N. Paton. *Active Rules in Database Systems*. Springer, 1999.
14. 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.
15. H. Stuckenschmidt. Similarity-based query caching. In *th Internationa Conference on Flexible Query Answering System FQAS*, Lecture Notes in Artificial Intelligence, Lyon, France, 2004. Springer Verlag.
16. H. Stuckenschmidt and F. van Harmelen. Approximating terminological queries. In *Proceedings of the Fifth International Conference on Flexible Query Answering Systems FQAS 2002*, Lecture Notes in Artificial Intelligence, Copenhagen, Denmark, 2002. Springer Verlag.
17. Heiner Stuckenschmidt, Anita de Waard, Ravinder Bhogal, Christiaan Fluit, Arjohn Kampman, Jan van Buel, Erik van Mulligen, Jeen Broekstra, Ian Crowlesmith, Frank van Harmelen, and Tony Scerri. Exploring large document repositories with rdf technology - the dope project. *IEEE Intelligent Systems*, 2004. to appear.
18. Holger Wache, Perry Groot, and Heiner Stuckenschmit. Scalable instance retrieval for the semantic web by approximation. In *Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE'05)*, 2005.

Appendix

```
PATTERN
  {Resource} subject {Subject}
  WHERE
  Subject Like Value^^xsd:string,
REPLACE-BY
  [{Resource} subject {Subject}],
  {Resource} R {T}
  WHERE
  T Like NEW
WITH
  domain-preferred-over(subject,R)
  NEW = concat("*",concat(Value,"*"))
```

Fig. 4. Re-writing rule for replacing subject with title

```
 PATTERN
  {Resource} language {Language}
  WHERE
  Language = L1,
REPLACE-BY
  {Resource} language {Language}
  WHERE
  Language = L2,
WITH
  user-preferred-over(L1,L2)
```

Fig. 5. Specification of the Re-writing rule for langauge preferences