Int J Digit Libr (2004) 4: 60–63 / Digital Object Identifier (DOI) 10.1007/s00799-003-0068-2

Rule-based query personalization in digital libraries

Georgia Koutrika, Yannis Ioannidis

Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ilissia, Athens 15784, Greece
e-mail: {koutrika, yannis}@di.uoa.gr

Published online: 23 July 2004 –

Abstract. Searching a digital library is typically a tedious task. A system can improve information access by building on knowledge about a user acquired in a user profile in order to customize information access both in terms of the information returned in response to a query (*query personalization*) as well as in terms of the presentation of the results (*presentation personalization*). In this paper, we focus on query personalization in digital libraries; in particular, we address structured queries involving metadata stored in relational databases. We describe the specification of user preferences at the level of a user profile and the process of query personalization with the use of query-rewriting rules.

Keywords: Personalization – Preferences – Profile-based query rewriting

1 Introduction

Digital libraries exist in many forms and are of various types [4]. Speaking in terms of stored information, digital libraries are essentially collections of unstructured and structured data. Unstructured data involve different kinds of documents, where the term *document* is used in the broadest possible sense to denote articles, movies, books, etc. Structured data typically involve metadata about documents, e.g., metadata about movies include information about titles, actors, directors, etc. that may be stored in a database. At the level of information access, one of the basic functions a digital library serves is the ability to perform searches on the underlying collection. Searches may involve information incorporated in a document (e.g., keywords appearing in text documents) as well as metadata (e.g., the author of a document); moreover, searches may be unstructured (e.g., a set of keywords such as "*Databases*" and "*Ullman*") or structured (e.g., author="*Ullman*" and year=*1990*).

In practice, users often get frustrated while searching a digital library because of the substantial personal effort needed to locate information of interest to them. For this reason, current information systems build on knowledge about user characteristics to provide customized responses. Unfortunately, such behavior is absent from database systems, which always provide the same response to everyone.

Example: Consider the database schema below with information about movies stored in a collection (attributes are self-explanatory).

Movies	(mid, title, type, year, plot, did)
Cast	(mid, aid, role)
Actors	(aid, aname, birth, sex, nationality, nominated)
Directors	(did, dname, birth, nationality, nominated)

Moreover, consider two users, Julie and Peter, both inquiring about movies released in 2002. Typically, this is done through some simple interface; thus their request is:

Q: Movies(_, _, _, *2002*, _, _).

However, these users have different tastes in movies. Julie likes comedies and thrillers and Peter prefers sci-fi movies, and he is a fan of Woody Allen. These preferences could be stored in a *user profile* and the system could automatically integrate them into the initial, predefined query, saving human effort (on the part of a user as well as a programmer). Julie would be more pleased with the results of the following query:

Q': Movies(_, _, "*thriller*", *2002*, _, _) *or*
Movies(_, _, "*comedy*", *2002*, _, _).

On the other hand, Peter would prefer the results of this query:

Q": Movies(_, _, "*sci-fi*", *2002*, _, _) *or*
(Movies(X, _, _, *2002*, _, _), Cast(X, Y, _),
Actors(Y, "*W. Allen*", _, _, _, _)).

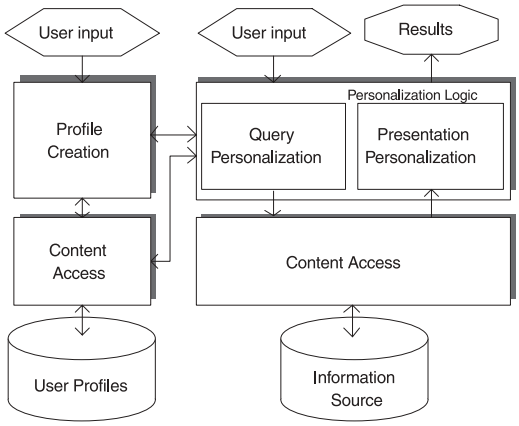

Fig. 1. Architecture of a personalization system

The general architecture of a system providing such personalized access is depicted in Fig. 1 and includes several modules surrounding the traditional *Content Access* module. The system keeps a repository of user information (*User Profiles*) that is either inserted explicitly by the user or collected implicitly by monitoring user interaction with the system (*Profile Creation*). This profile information, required by the system to implement its *personalization logic*, is integrated with an incoming request both during content selection (*Query Personalization*) as well as result presentation (*Presentation Personalization*) in order to personalize the overall user experience.

In this paper, we focus on query personalization in digital libraries; in particular, we address structured queries involving metadata stored in relational databases. First, we describe how user preferences are captured in a user profile as query rewriting rules with assigned weights that indicate user interest. We treat query personalization as a query-rewriting problem and provide an algorithm that produces a personalized version of any query by combining several possible rewritings of the initial query based on the rules specified in the user profile.

2 Related work

User preferences may be of various forms [2], e.g., mandatory (i.e., which must be definitely satisfied) or optional (i.e., wishes), conditional or unconditional, single-value or multivalue, etc. There is extensive research on preferences in logic (e.g., [3]), but only a few papers ([1, 2, 5, 6]) address this issue in the context of databases. These efforts focus on the expression of short-term user preferences at the query level, whereas we deal with the specification of long-term preferences stored in user profiles. At query time, stored preferences that are relevant to an incoming request are dynamically identified and integrated into the latter, producing a *personalized* query that is executed instead of the initial one. In our work, query-rewriting rules are used to express in a uniform way mandatory and optional user preferences of various forms, e.g., conditional and unconditional preferences, single-value and multivalue, etc. [2]. The dynamic construction of personalized queries, i.e., query personalization, is then treated as a query-rewriting problem.

3 User profile

We concentrate on the relational model for our presentation. Our approach to personalization is based on maintaining for every user a profile that stores *query-rewriting rules*. A weight w is assigned to each rule r, which is a real number in the range [0,1] and indicates the significance of the rule. Mandatory user preferences are expressed as rules assigned a weight equal to 1; user preferences are represented by rules assigned a weight in the range (0,1), where a high weight-value indicates strong preference.

Using a logic-based language, a query-rewriting rule r is represented as:

$$\mathrm{H} \longleftarrow \mathrm{B}\,,$$

where H and B denote the head and body of the rule, respectively, and represent queries. The symbol $\leftarrow$ denotes replacement of the query expressed in the head of the rule with the query expressed in the body of the rule. Note that even if the body of the rule contains the head, no recursion is implied, but simply a one-time application.

A query is expressed as

$$\mathrm{L}_1, \ldots, \mathrm{L}_n.$$

Each L_i is a literal of the form $p_i(t_1, \ldots t_{ki})$ such that p_i is a predicate symbol and t_j are terms, either constants or variables. Variables are denoted with their initial letter capitalized. Underscores (_) denote "don't care" variables. We use the term *general rule* if a rule does not contain any constants and the term *specific rule* if it does.

As mentioned earlier, different types of preferences can be expressed with the use of rewriting rules. For example, consider the user profile depicted in Fig. 2 regarding the movie database. Rule r_1 represents a general mandatory preference that a request for movies should always consider starring actors. Rules r_2 and r_4 represent strong preferences for comedies and thrillers, respectively. Rule r_3 represents a conditional preference, i.e., if the movie is a thriller, then Al Pacino is favored. Finally, rule r_5 represents a weaker preference on Italian actors. This is an example of a multivalue preference.

4 Query personalization

Query personalization proceeds in two steps. First, a set of possible queries is generated by different rewritings of the initial user query based on the user profile. Then, these queries are combined into a single query; this is the

r_1, $w_1 = 1$	Movies(_, _, _, _, _, _) ← Movies(X, _, _, _, _, _), Cast(X, Y, _), Actors(Y, _, _, _, _, _)
r_2, $w_2 = 0.9$	Movies(_, _, Type, _, _, _) ← Movies(_, _, '*comedy*', _, _, _)
r_3, $w_3 = 0.87$	Movies(_, _, '*thriller*' , _, _, _) ← Movies(X, _, '*thriller*', _, _, _), Cast(X, Y, _), Actors(Y, '*Al Pacino*', _, _, _, _)
r_4, $w_4 = 0.8$	Movies(_, _, Type, _, _, _) ← Movies(_, _, '*thriller*', _, _, _)
r_5, $w_5 = 0.5$	Actors(_, _, _, Sex, Nationality, _) ← Actors(_, _, _, '*male*', '*Italian*', _)

Fig. 2. An example of a user profile

personalized query that will be evaluated on behalf of the user. Below we describe each step in more detail.

A query Q' is a rewriting of query Q if there is a subset of rules $\boldsymbol{R} = \{r_i | i = 1 \ldots n\}$ defined in the user profile that produce Q' if applied successively to Q. At each step, a rule r_i is applied to the query Q_{i-1} produced at the previous step and produces Q_i provided that the query contains the head of the rule (this is known as *query subsumption* for the form of rules used here). At the final round, $Q_n = Q'$ is produced whose weight is a function of the weights of the rules that were used. Note that, for Q' to be meaningful, $\boldsymbol{R}$ must include at least one specific rule.

For example, assume a query Q: Movies(_, _, _, _, _, _). In this case, multiple rules can be used, namely, r_1, r_2, r_3, r_4, resulting in more than one possible rewriting of the query. Suppose we choose rule r_1; then the query is rewritten as

Q_1: Movies(X, _, _, _, _, _), Cast(X, Y, _),
Actors(Y, _, _, _, _, _).

Subsequently, we can apply any of the five rules to rewrite this new query; we choose rule r_5, which constructs the following query:

Q_2: Movies(X, _, _, _, _, _), Cast(X, Y, _),
Actors(Y, _, _, "*male*", "*Italian*", _).

Thus Q_2 is a possible rewriting of Q by applying successively rules r_1 and r_5.

The algorithm for the construction of possible query rewritings for a query Q is sketched in Fig. 3. The algorithm's inputs are a query Q, a user profile U, and a criterion C_R. The latter determines (implicitly or explicitly) the number of rewritings of Q that will be generated. For example, such a criterion may provide the maximum number of rules that may be used for query rewriting or a minimum weight that a rule should have in order to be considered for query rewriting. The algorithm's output is a set of queries $\boldsymbol{Q}_R$ produced by different rewritings of query Q.

The algorithm maintains an ordered queue $\boldsymbol{QP}$ of queries generated so far. First, it produces a query Q' that is a rewriting of Q based on these rules that express mandatory requirements and are *applicable* to Q. Then, the algorithm proceeds to apply rules that represent preferences. In each round, it selects from QP a query Q' with the maximum weight. If Q' does not satisfy the input criterion C_R, then the algorithm stops; otherwise, it considers all rules r_i that are *applicable* to Q' in decreasing order of weight. Each rule r_i is applied to Q' producing a query Q_i; if this new query satisfies the input criterion, then it is placed into $\boldsymbol{QP}$. If the algorithm constructs a query Q_i that does not satisfy the criterion, then it does not consider any more rules for rewriting Q' and the latter is placed in the set of possible rewritings that form the output. In this way, the algorithm constructs queries that are *maximal* rewritings of Q, i.e., queries that satisfy the interest criterion C_R, and there is no *applicable* rule that can rewrite them into new ones that still satisfy C_R. We should also note that the algorithm is complete, i.e., it generates all queries that are maximal query rewritings of Q.

```
Input: a query Q, a criterion C_R, a user profile U
Output: a set of possible query rewritings Q_R
{
   QP: an ordered queue of queries
  /*rules are considered in decreasing order of weight */
   Rewrite Q as Q' based on applicable mandatory rules
   For each rule r_i in U applicable to Q' in U {
         Apply r_i to Q and produce Q_i
         add Q_i into QP
   }
   While QP not empty do {
      Get head Q' from QP
      If(C_R is satisfied for Q') then {
        For each rule r_i in U applicable to Q' {
          apply r_i to Q' and produce Q_i
          If (C_R is satisfied for Q_i) then add Q_i into QP
          else add Q' into QP and break
      }}
      else stop
   }}
```

Fig. 3. Construction of Query Rewritings

A rule is *applicable* to a query Q' provided the following conditions hold:

Relevance Condition:

(RC1) The query contains the head of the rule.

Consistency Conditions:

(CC1) The rule contains in its body no other part of the query except for its head.

(CC2) The rule has a weight lower than (or equal to) that of the rules already used for the construction of the query.
(CC3) The rule does not contain an evaluation of a variable already evaluated in Q'.

(Variable evaluation is defined as replacement by a constant.) Consistency conditions are necessary for avoiding repetitions, conflicts, and cycles. For example, assume the trivial case where there are two rules r_1 and r_2 and a query Q such that

$r_1 : Q \longleftarrow Q, Q_1$
$r_2 : Q_1 \longleftarrow Q,$

i.e., (CC1) does not hold. In this case, applying r_1 and then r_2 produces $Q \leftarrow Q$, which is a meaningless rewriting.

Now, assume there are two rules, r_3 and r_4, in decreasing order of weight and a query Q such that

$r_3 : Q \longleftarrow Q, Q_1$
$r_4 : Q \longleftarrow Q, Q_2,$

and (CC2) does not hold. In this case, applying r_3 and then r_4 produces a possible rewriting $Q \leftarrow Q, Q_2, Q_1$. Applying r_4 and then r_3 produces a possible rewriting $Q \leftarrow Q, Q_1$, Q_2, which is essentially the same as in the first case.

(CC3) is necessary for avoiding replacement of a higher (specific) preference with a lower one.

Example: Returning to our example, consider Julie's initial request, expressed as Q: Movies(_, _, _, *2002*, _, _) and a criterion that requires that only rules with weight greater than 0.85 may be applied. Then, the algorithm proceeds as follows:

a. Rule r_1 is mandatory and applicable. Thus, Q is rewritten as Q' as follows:

Q': Movies(X, _, _, *2002*, _, _), Cast(X, Y, _),
Actors(Y, _, _, _, _, _).

Then, rules expressing preferences are considered.

b. Rules r_2, r_3 both have weights greater than 0.85 and are applicable. Consequently, two possible rewritings of Q' are the following:

Q_1: Movies(X, _, "*comedy*", *2002*, _, _), Cast(X, Y, _),
Actors(Y, _, _, _, _, _).
Q_2: Movies(X, _, "*thriller*", *2002*, _, _), Cast(X, Y, _),
Actors(Y, "*Al Pacino*", _, _, _, _).

c. For Q_1 we consider only rules below r_2 (due to (CC2)), which has already been applied. Only rule r_3 satisfies the criterion on weight, but it cannot be used due to (CC3). Also, no rewriting is possible for Q_2.

Consequently, as depicted in Fig. 4, the possible queries produced by rewriting Q are Q_1 and Q_2.

The queries that are produced by the different rewritings of the initial query are combined to form the final personalized query to be executed. The simplest case of combination is the disjunction of all these queries. Considering the example query on movies, the disjunction of the derived queries generates the following final, personalized query:

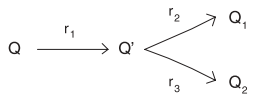

Fig. 4. Query rewriting for the example query Q

(Movies(X, _, "*comedy*", *2002*, _, _)) *or*
(Movies(X, _, "*thriller*", *2002*, _, _), Cast(X, Y, _),
Actors(Y, "*Al Pacino*", _, _, _, _).

Other combinations are clearly possible (e.g., conjunction of all rewritings) and are currently under investigation.

5 Conclusions and future work

We have described the specification of user requirements and preferences at the level of a user profile and the process of query personalization with the use of query-rewriting rules. An algorithm for query personalization was briefly outlined that produces a set of possible queries based on maximal rewritings of a query while avoiding conflicts and cycles. The queries produced are then combined with disjunction. Future work includes elaboration of the algorithm in order to cope with other kinds of conflicts that may arise when trying to combine user requirements. The issue of conflicts is very important in query personalization, and little work has been done in this area. Moreover, other interesting issues exist, including investigation of other forms of query combination for queries produced by different rewritings of an initial query, ways to rank personalized results based on the weight of the preferences they satisfy, and delivery of top-n answers.

References

1. Agrawal R, Wimmers EL (2000) A framework for expressing and combining preferences. In: Proceedings of the ACM SIGMOD international conference on management of data, Dallas, 16–18 May 2000, pp 297–306
2. Chomicki J (2002) Querying with intrinsic preferences. In: Proceedings of the 8th international conference on extending database technology, Prague, Czech Republic, 25–27 March 2002, pp 34–51
3. Delgrande JP, Schaub T, Tompits H (2000) Logic programs with compiled preferences. In: Proceedings of the European conference on artificial intelligence, Berlin, 20–25 August 2000, pp 464–468
4. Digital Libraries (1995) Commun ACM 38(4)
5. Govindarajan K, Jayaraman B, Mantha S (2001) Preference queries in deductive databases. New Generation Comput 19(1):57–86
6. Hristidis V, Koudas N, Papakonstantinou Y (2001) PREFER: a system for the efficient execution of multiparametric ranked queries. In: Proceedings of the ACM SIGMOD international conference on management of data, Santa Barbara, CA, 21–24 May 2001, pp 259–270