JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 6, No. 4, December 2012, pp. 243-256

Design and Implementation of a Framework for Context-Aware Preference Queries

Patrick Roocks*, Markus Endres, Alfons Huhn, and Werner Kießling
Institut für Informatik, Universität Augsburg, Augsburg, Germany
roocks@informatik.uni-augsburg.de, endres@informatik.uni-augsburg.de,
huhn@informatik.uni-augsburg.de, kiessling@informatik.uni-augsburg.de

Stefan Mandl
EXASOL AG, Nuremberg, Germany
stefan.mandl@exasol.com

Abstract
In this paper we present a framework for a novel kind of context-aware preference query composition whereby queries for the Preference SQL system are created. We choose a commercial e-business platform for outdoor activities as a use case and develop a context model for this domain within our framework. The suggested model considers explicit user input, domain-specific knowledge, contextual knowledge and location-based sensor data in a comprehensive approach. Aside from the theoretical background of preferences, the optimization of preference queries and our novel generator based model we give special attention to the aspects of the implementation and the practical experiences. We provide a sketch of the implementation and summarize our user studies which have been done in a joint project with an industrial partner.

Category: Smart and intelligent computing

Keywords: Databases; Preferences; Context awareness; Personalization

I. INTRODUCTION AND RELATED WORK

Preferences in Databases [1] – as shown by a recent survey [2] – is a well-established framework to create personalized information systems. By using well designed preference models, users can be provided with just the information they require, thereby overcoming the dreaded empty result set and flooding effect [3]. In [4] relation-algebraic aspects of database queries under changing preferences are discussed. These improvements are starting to show up in real world applications. For instance, today in the area of tourism – which is used in this paper as an example application domain – there are many portals which provide information about flights, hotels or outdoor activities (cp., Fig. 1) by parametrized queries which either result in an abundant number of items or no answer at all.

Clearly, this state of affairs is non-satisfactory for both the users and the owners of the portals. In new tourist portals (like presented in [5]), preferences allow for semantically richer queries which define a strict partial order on the items available for the purpose of selecting

Open Access http://dx.doi.org/10.5626/JCSE.2012.6.4.243 http://jcse.kiise.org

Received 12 October 2012, **Accepted** 24 October 2012
*Corresponding Author

†An earlier version of this paper has been published at DASFAA 2012. This paper is an extended version of (P. Roocks, M. Endres, S. Mandl, and W. Kießling, "Composition and efficient evaluation of context-aware preference queries," Database Systems for Advanced Applications, LNCS vol. 7239, 2012).

 pISSN: 1976-4677 eISSN: 2093-8020

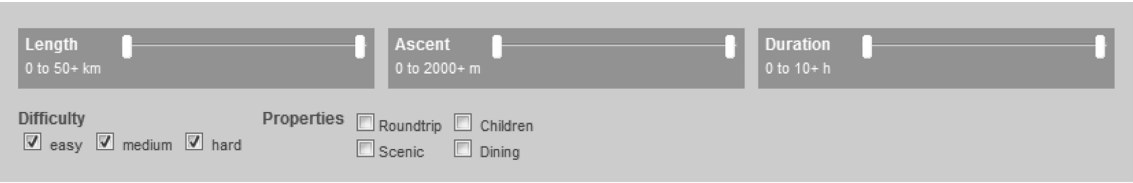

Fig. 1. Faceted search for numerical attributes of hiking tours and attribute based search for discrete attributes.

only the best items available, yielding a higher customer satisfaction.

In domains like tourism, the notion of preferences varies between users and strongly depends on users' situations. Hence, recent models of context-aware preferences emerged [6-9], which are taking into account factors like users' personalities, situation parameters (e.g., location, time, season, weather), or even options of users' acquaintances. [6] introduces CareDB, which provides scalable personalized location-based services to users based on their preferences and the current surrounding context. In particular, it deals with the problem that some contextual values may be very expensive to compute. [8] suggests a context-model for preference queries, where context is represented by a variant of description logic.

[9] suggests a discrete context model and introduces profile trees as a data structure for the context resolution problem, hence retrieving the most appropriate preferences depending on context. Similarly, [7] suggests a model for a contextual preference selection based on the idea of a situation hierarchy. These models can be seen as top-down approaches in contrast to our constructive bottom-up model. In our approach all preferences are generated dynamically instead of performing a look-up in a set of predefined preferences.

There is the open challenge to create a comprehensive framework to derive preferences from context. In this paper, we meet this challenge by presenting an approach for the context-aware preference composition, which 1) produces inductively structured preference queries that are intuitive such that they can be verified by domain experts, 2) allows for easy adaption of the composition process for the specific application domain, and 3) covers the entire path from the descriptive application model to the operational preference query language.

Throughout the paper we refer to the following example from the domain of hiking tour recommendations which informally demonstrates the lines of reasoning about context-aware preferences of domain experts (like touristic companies). Thereby, we want to model: social network (recommendations from friends), history (already visited regions by the user), and external knowledge sources (weather services, like the snow line and the altitude-dependent temperature).

EXAMPLE 1 (Running use case). Consider John, who is planning a hiking tour in the touristic region of the Bavarian Alps. Last year, he was in the "Tannheim Mountains" and a friend of his made a tour in "Walser Valley" which he was very excited about. Since John now wants to see something new, he wants to avoid regions he has already visited. At the same time, he trusts his friends and thus prefers regions recommended by friends.

As he is not very experienced, he only specifies duration for the tour, while he relies on the recommender for the ascent and length. John prefers not to hike in the snow; hence, only mountain peaks below the current snow line are preferable for him. Because it is a sunny day, the temperatures in low altitudes are too high, so the tour should mainly take place in a convenient altitude range.

John looks out for an online portal which picks out a hiking tour from a large database that perfectly matches the preferences formulated above. In case there are no "perfect tours" he accepts compromises, e.g., tours whose altitude exceeds the snow line a little bit.

The remainder of the paper is structured as follows: Section II informally presents the preference model used in this paper. Thereafter, Section III describes our context-aware preference generation process. Section IV shows how the generated preference queries can be evaluated using Preference SQL. Section V provides a conceptual view of the entire system and its implementation. Finally, Section VI summarizes our claimed contributions and outlines further research directions.

II. PREFERENCE MODELING

Preference modeling has been in focus for some time, leading to diverse approaches [1, 3, 10]. We follow the preference model from [1], which directly maps preferences to relational algebra and declarative query languages. It is semantically rich, easy to handle and very flexible to represent user preferences which are ubiquitous in our life.

A *preference* $P = (A, <_P)$ is a *strict partial order* on the domain values of the attributes A of a database relation. Strict partial order preferences can be interpreted in a very intuitive manner as personalized wishes in the form of "I like y more than x". The term $x <_P y$ is interpreted as "y is better than x according to P" where x, y are domain values of the attribute set A. The result of a preference is computed by the *preference selection*.

DEFINITION 1 (Preference selection, BMO-set). *The Best-Matching-Objects (BMO-set) of a preference P = (A, $<_P$) are all tuples from a database relation R which are maximal according to the preference order. It is computed by the preference selection operator $\sigma[P](R)$ (called winnow in [10]) and finds all the best matching tuples t for P, where t[A] is the projection to the attribute set A*

$$\sigma[P](R) := \{t \in R \mid \neg \exists t' \in R : t[A] <_P t'[A]\}$$

A preference can also be evaluated in grouped mode, given some attributes B as a subset of A.

$$\sigma[P \text{ grouping } B](R) := \{t \in R \mid \neg \exists t' \in R : t[A] <_P t'[A] \wedge t[B] = t'[B]\}$$

According to [3] this can be expressed as a preference itself:

$$t <_{P \text{ grouping } B} t' \Leftrightarrow t[A] <_P t'[A] \wedge t[B] = t'[B]$$

Preference selection offers a cooperative query answering behavior by automatic matchmaking: The BMO query result adapts to the quality of the data in the database, defeating the empty result effect and reducing the flooding effect by filtering off the worse results.

To specify a preference, a set of intuitive base preference constructors together with some complex preference constructors has been defined. Subsequently, we present some selected preference constructors used in this paper. More preference constructors as well as their formal definition can be found in [1, 3].

1) Base Preference Constructors

Preferences defined on a single attribute are called *Base preferences*. There are base preference constructors for *continuous, discrete (categorical)* and *spatial* domains, cp., [3, 11, 12]. Fig. 2 shows the "ISA"-hierarchy of several frequently occurring base preference constructors.

The $SCORE_d$ preference is the parent of all base preference constructors except EXPLICIT, cp., Fig. 2. EXPLICIT is used to create any preference that can be expressed by a finite set of "y is better than x" relationships. $SCORE_d$ allows specifying a preference using a numerical scoring function. When dealing with numerical values, it is a common practice to group ranges of scores together. Such a real-world behavior can be modeled using the *d*-parameter, which is associated with all preference constructors for continuous domains. It maps different function values to a whole-number returned by discretization which therefore might become interchangeable with other values having the same whole-number.

We now explain some categorical sub-constructors of the $SCORE_d$ preference which are often used. The parent of all categorical preferences is $LAYERED_m$, which specifies a hierarchy of partitions containing preferred domain values. The most specialized sub-constructors of $LAYERED_m$ are POS and NEG.

DEFINITION 2 (POS and NEG preference). *The discrete Positive-preference POS (A, POS-set) states that the user has a set of preferred values, the POS-set, in the domain of A. The Negative-preference constructor is the counterpart to the POS-preference, formally NEG(A, NEG-set)* := POS(*A,* dom(*A*)*NEG-set*).

EXAMPLE 2 (Running use case). Reconsider Example 1 where John is planning a new hiking tour. Since John trusts his friends he wants to avoid regions he has already visited and therefore prefers regions recommended by his friends. Using the POS and NEG preference constructors from Definition 2 these preferences can be formulated as

P_1 := POS(*recommended*, 'yes')
P_2 := NEG(*visited*, 'yes')

where *recommended* and *visited* are attributes in the database relation of hiking tours.

Now, we present some preference constructors for continuous domains.

DEFINITION 3 ($BETWEEN_d$ preference). *The continuous preference constructor* $BETWEEN_d$ *(A, low, up) expresses the wish for a value between a lower and an upper bound. If this is infeasible, values having the smallest distance to* [*low, up*] *are preferred, where the distance is discretized by the discretization parameter d.*

The $AROUND_d$ preference is a specialized $BETWEEN_d$ preference.

DEFINITION 4 ($AROUND_d$ preference). *The* $AROUND_d$ *(A, z) preference equals the* $BETWEEN_d$ *(A, z, z) preference. This means that the desired value should be z* ∈ dom(*A*).

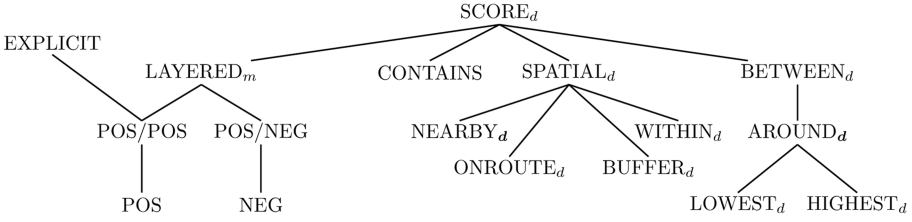

Fig. 2. Taxonomy of base preference constructors.

If this is infeasible, values near to z are preferred.

EXAMPLE 3 (Running use case). Continuing Example 1 John prefers a hiking tour with a duration of around 5 hours. Thereby, John accepts tours which take ±10% of the desired time, i.e., he uses d-parameter of 0.5 hours. Using the AROUND_d constructor from above we obtain

$$P := \text{AROUND}_{0.5}\ (duration, 5)$$

Sometimes one prefers values *more than* or *less than* a specified value. For this the MORE_THAN_d and LESS_THAN_d preference constructors can be used.

DEFINITION 5 (LESS_THAN_d, MORE_THAN_d).

a) *The* $\text{MORE\_THAN}_d\ (A, z)$ *preference equals the* $\text{BETWEEN}_d\ (A, z, \infty)$ *preference, i.e., the desired values are greater or equal to z.*

b) $\text{LESS\_THAN}_d\ (A, z)$ *is the dual preference to* $\text{MORE\_THAN}_d\ (A, z)$ *and equals* $\text{BETWEEN}_d\ (A, -\infty, z)$*; i.e., the desired values are less or equal to z.*

EXAMPLE 4 (Running use case). John does not prefer to hike in the snow, hence only mountain peaks below the current snow line of 2,400 m are preferable for him. This leads to a LESS_THAN_d preference on the *altitude* with a *d*-parameter of, let say, 200 m.

$$P := \text{LESS\_THAN}_{200}\ (altitude, 2400)$$

2) Complex Preference Constructors

If the user wants to combine several preferences into more *complex preferences*, she has to decide the relative importance of these given preferences. Intuitively, people speak of "this preference is more important to me than that one" or "these preferences are all equally important to me". Equal importance is modeled by the so-called *Pareto preference*.

DEFINITION 6 (Pareto preference). *In a Pareto preference*

$$P := P_1 \otimes ... \otimes P_m = (A_1 \times \cdots \times A_m, <_P)$$

all preferences $P_i = (A_i, <_{P_i})$, $i = 1,..., m$ *on the attributes* A_i *are of equal importance*.

The *Prioritization preference* allows the modeling of combinations of preferences that have different importance.

DEFINITION 7 (Prioritization preference). *Let* $P_i = (A_i, <_{P_i})$, $i = 1,..., m$ *be preferences. In a Prioritization preference*

$$P := P_1 \ \&\ ...\ \&\ P_m = (A_1 \times \cdots \times A_m, <_P)$$

the importance of the preferences decreases from left to right.

EXAMPLE 5 (Running use case). John prefers a hiking tour with a *duration* of 5 hours which is *more important* than all other preferences. The recommendations of his friends are *equally important* to the wish of staying below the snow line, e.g., 2,400 m. Using the complex preference constructors the example can be formulated as:

$$P_{\text{John}} := \text{AROUND}_{0.5}\ (duration, 5)\ \&\ (\text{POS}(recommended, \text{'yes'}) \otimes \text{LESS\_TRAN}_{d_a}(altitude, 2400))$$

The variable d_a is the *d*-parameter which will be used for all preferences regarding altitude. A typical value could be $d_a = 200$, implying a tolerance of 10% at a typical altitude of 2,000 m.

For the rest of this paper we also need the notion of a *preference term*.

DEFINITION 8 (Preference terms). *We define the set of preference terms F inductively:*

A base preference $P = (A, <_P)$ *is a preference term.*
If P_1 *and* P_2 *are preference terms, then the complex preferences* $P_1 \otimes P_2$ *and* $P_1\ \&\ P_2$ *are preference terms, too.*

The semantically well-founded approach of this preference model is essential for the personalized preference term composition. As a unique feature this preference model allows multiple preferences on the same attribute without violating the strict partial order. Furthermore, the *inductive* preference construction preserves strict partial order, too.

III. CONTEXT-AWARE PREFERENCE GENERATION

Based on the preference framework presented in Section II we will now present our model for creating preference terms dependent on the context. This model depends on context-based triggers which are described by a discrete situation model with few states. Compared to continuous models, such models are particularly useful when communicating with domain experts as the small number of states makes it possible to systematically consider all combinations of context states and therefore guarantee that the system behaves in the desired way for all possible context values. For example, concerning the weather, it is intuitive to distinguish between "good" and "bad". For our domain of a hiking tour recommender we introduced the states: "good", "bad" and "warning". Whereas, "bad" implies less convenient outdoor activities, the state "warning" discourages activities like hiking because of safety reasons. Dependent on the context (time of year, region) alternatives like city tours or visiting the spa could be suggested.

A. A Constructive Approach for Preference Generation

According to our running use case of a hiking tour rec-

ommender we have three different kinds of influences for the preference generation: The user input from the search mask, the user profile from the database and the contextual information from external knowledge sources. In the following we describe how they interact in our model. As described in Example 1, the weather conditions imply certain preferences. However, weather warnings do not mean that the user input is overridden. The rough concept of the preference composition always follows this prioritization-schema, where each <···>-part is called the *preference component*.

<pref_term> := <user_input> & <context> & <profile> (1)

The <pref_term> generates the preferring-clause of the resulting Preference SQL query. This schema is supported by the conclusion of [13] that users might feel a "lack of control" in context-aware systems, therefore the user input is prioritized. The same study demonstrated that users prefer context-awareness to personalization, thus the profile is less prioritized.

The gravity of the external influences depends on the user: a well specified user input, i.e., all fields of a search mask are filled out, implies that the context and the profile have only minor influence to the generated preference. If a "lazy" user leaves all fields blank, a "default request" based on the context and the profile will be generated.

In the following, the components of (1) are stepwise filled with preferences. This procedure underlines our *constructive and dynamic approach* based on the current context. It is one of the big advantages of our model that preferences are closed under prioritization and Pareto-composition, and even "seeming contradictions" keep the strict partial order property. In our running use case of the hiking tour recommender, the user input consists of the tour parameters length, duration, and ascent, unless they are not specified. The components <context> and <profile> are sub-divided:

<context> := <weather_safety> & <children> & <weather _conv> (2)

<profile> := (<social_net> ⊗ <history>) & <user_type> (3)

In the <context> component in (2) the influence of the weather is divided into <weather_safety>, i.e., the safety-relevant preferences (e.g., snow line), and <weather_conv>, the convenience-relevant (conv) preferences (e.g., convenient temperature). The <children> component is relevant for families with children and will set preferences for tours with playgrounds, etc.

In the <profile> component in (3) the recommendations of friends are contained in <social_net> and the already visited tours in <history>.

The components <user_input> in (1) and <user_type> in (3) depend on what the user filled out in the search mask and the user type. The usage of these components is shown in the following example.

EXAMPLE 6 (Running use case). Consider the tourist John who has only specified a duration of 5 hours for his tour. In our tour recommender this preference is modeled with the AROUND-constructor. Thereby, we assume a tolerance of 10% which leads to a d-value of 0.5h. Thus, we have the following user input preference:

<user_input> := $\text{AROUND}_{0.5}$ (*duration*, 5)

Depending on empirical values like average speed and condition of his user type (e.g., "tourist" or "athlete"), the system selects defaults for length and ascent according to the duration. Assume that the role of John, e.g., tourist, implies default values of 12 km for the length and 600 m for the ascent, both with a d-parameter of 10%.

<user_type> :=
$\text{AROUND}_{1.2}$(*length*, 12) ⊗ AROUND_{60}(*ascent*, 600)

The Pareto-composition of these preferences is based on the assumption that the attributes of length and ascent are of equal importance for the user. This ordering may depend on the user type, e.g., an athletic user is focused on the ascent of the tour whereas an inexperienced tourist is more confident to the duration, i.e., the preferences in our use-case also depend on the "tourist role", which is investigated in [14].

The social network and the history in (3) are Pareto-composed to take care of both aspects. In the following example we will show how conflicting preferences can be handled.

EXAMPLE 7 (Running use case). We assume the attributes *recommended* and *visited*, which are not attributes of the tour, but they represent the recommendations from the social network and the visited tours of the current user. For *recommended* we have the values "yes", "no", and "unknown", representing that the region is recommended, disadvised or that there is no or contradicting information in the social network. For the *visited* attribute we only have the values "yes" and "no" to mark the regions already visited by John. Regions which are recommended and simultaneously have not been visited are preferred.

<social_net> := POS (*recommended*, 'yes')
<history> := NEG (*visited*, 'yes')

According to the Pareto composition <social_net> ⊗ <history>, a region which was both visited and recommended is considered equal to a region which was neither visited nor recommended.

The preference model from Section II gives us the

freedom for this component-based model, where preferences with oppositional influences have a meaningful semantic interpretation in our preference model.

To fill the <context> component with context-dependent preferences, a formal model based on *context-aware generators* is introduced in the following.

B. Context-Aware Generators

The generation of preferences in our approach is context-aware in two regards: 1) preferences in the components are triggered by a discretized context, and 2) parameters of these preferences can change with the continuous context.

By this breakdown the model is kept clear. Whereas in the "discretized world" the rough structure of the preference term is decided, the "continuous world" influences the fine structure of the preference term. Thus, it is easy to see how the configuration of the recommender changes the final output.

DEFINITION 9 (Context variables, [current] situation). *Let Ω the set of continuous world states and $\omega \in \Omega$ a world state, containing user input and contextual knowledge. We define context variables by mappings*

$$v_i : \Omega \to V_i \text{ for } i \in \{1, ...,n\} =: I$$

where V_i are finite sets modeling the discretization of the context. The set $\mathbb{S} := \times_{i \in I} V_i$ represents possible situations and $s \in \mathbb{S}$ is a single situation. The aggregation of all context variables leads to the mapping

$$\text{sit} : \Omega \to \mathbb{S},\ \omega \mapsto (v_1(\omega), ..., v_n(\omega))$$

If ω is the current world state, then $\text{sit}(\omega)$ *is called the current situation. For $a \in V_i$ we define the a-induced subspaces*

$$\{v_i = a\} := V_1 \times ... \times V_{i-1} \times \{a\} \times V_{i+1} \times ... \times V_n.$$

We illustrate the concept of context variables in the following example.

EXAMPLE 8 (Running use case). To describe the current weather state of the hiking region selected by the user input, we introduce a context variable *weather* : $\Omega \to$ {good, bad,warning}. To this end, we assume ω contains information from an online weather service, which is retrieved automatically after the user submits the search request.

By *children* : $\Omega \to$ {yes, no} we describe if children are participating in the hiking tour. We assume that the user ticks this in the search mask and this information is stored in ω.

With the context variables we will trigger the rough structure of the <context> component. To form the fine structure, we introduce the concept of *ω-dependent preference terms*.

DEFINITION 10 (ω-Dependent preference terms). *Functions $f : \Omega \to M$, where M is either a finite set or the real numbers, are called ω-dependent functions. They extract a single part of the world context.*

$\mathbb{P}[\omega]$ is the set of ω-dependent preference terms. *If $p \in \mathbb{P}[\omega]$ is a base preference, then for all parameters of base preferences, ω-dependent functions may occur. Analogous to Definition 8 complex preferences which are inductively constructed from ω-dependent base preferences, are also in $\mathbb{P}[\omega]$.*

For $p \in \mathbb{P}[\omega]$ the evaluation of all ω-dependent functions in p is denoted by $p(\omega)$, where $p(\omega) \in \mathbb{P}$ holds.

EXAMPLE 9 (ω-Dependent preference terms). An ω-dependent function representing the altitude of the snow line is *snowline* : $\Omega \to \mathbb{R}$. The term LESS_THAN_{d_a}(*alt_max*, *snowline*(ω)) is an ω-dependent base preference, where *alt_max* is the attribute for the maximal altitude of the tour.

The interplay of the rough and the fine structure of preference generation is done by the *context-aware generators* introduced next.

DEFINITION 11 (Context-aware generator). *We define context-aware generators as tuples*

$$g = (S, p) \in \mathbb{G}, \quad \mathbb{G} := \mathcal{P}(\mathbb{S}) \times \mathbb{P}[\omega]$$

where $S \in \mathcal{P}(\mathbb{S})$ ($\mathcal{P}$ denotes the power set) is the set of associated situations and $p \in \mathbb{P}[\omega]$ is the associated ω-dependent preference term. A generator $g = (S, p)$ is called active, if $\text{sit}(\omega) \in S$.

EXAMPLE 10 (Running use case). To realize the components <children> and <weather_safety> we define the following generators:

$g_{\text{ch}} := (\{children = \text{yes}\}, \text{POS}(difficulty, \text{'easy'}))$
$g_{\text{ws},1} := (\mathbb{S}, \text{LESS\_THAN}(alt\_max, snowline(\omega)))$
$g_{\text{ws},2} := (\{weather = \text{bad}\}, \text{LESS\_THAN}_{d_a}(alt\_max, \min(snowline(\omega), 1500)))$
$g_{\text{ws},3} := (\{weather = \text{wrn}\}, \text{POS}(activity, \text{'CityTrail'}) \ \&\ \text{LESS\_THAN}_{d_a}(alt\_max, \min(snowline(\omega), 1500)))$

Thereby wrn is short for warning. Whereas $g_{\text{ws},1}$ is always active (the situation set is $\mathbb{S}$), the generators g_{ch}, $g_{\text{ws},2}$, and $g_{\text{ws},3}$ are restricted to situations depending on the weather and the presence of children. The sets of generators $\{g_{\text{ch}}\}$ and $\{g_{\text{ws},i} \mid i = 1, 2, 3\}$ are associated with <children> and <weather_safety>, respectively.

C. Constructing Preference Terms from Generators

Now we have components and associated sets of generators. The next step is to select the active generators and to create a preference term from them. To this end we have to specify how the preferences of the generators

shall be Pareto-composed or prioritized if there is more than one generator associated with a component.

Assume an ordering of the generators representing the *importance*. For a set of generators $G \subset \mathbb{G}$ we realize this by a function $\pi : \Omega \times G \to \mathbb{N}$, where smaller values stand for a higher importance. The π is ω-dependent, which primarily means a dependency of the user type. This concept of user type dependent preferences was illustrated in Example 6.

In order to transform a set of generators to a preference term, first, the active generators are determined (Definition 12). With Definition 13 we realize the principle that preferences created by more important generators are prioritized, and Pareto-composed if the according generators are equally important.

Definition 12 (Restriction to situation, active generators). Consider a set of generators $G \subset \mathbb{G}$. *The restriction to a situation* $s \in \mathbb{S}$ *is defined as*

$$G|_s := \{(S, p) \in G \mid s \in S\}$$

and with $G|_{\text{sit}(\omega)}$ *we restrict G to the set of active generators.*

Definition 13 (Preference generation). *For a set of generators* $G \in \mathbb{G}$, *an ordering* $\pi : \Omega \times G \to \mathbb{N}$, *and* $\omega \in \Omega$ *we define the preference generation:*

$$\text{pref}(G, \pi, \omega) := \mathop{\&}_{i \geq 1} (\otimes\{p(\omega) \mid (p, S) \in G|_{\text{sit}(\omega)}, \pi(\omega, g) = i\})$$

where $\otimes\{p_1, ..., p_n\} := p_1 \otimes ... \otimes p_n$ *and* $\&_{i=1}^{m}\ q_i := q_1 \&...\& q_m$ *for* $p_1, ..., p_n, q_1, ..., q_m \in \mathbb{P}$. *The configuration of a component is described by the tuple* (G, π).

Preference terms generated with the pref-function from Definition 13 – also known as *p-skylines* [15] – could be very long if many generators are active. But there may be generators which are more "appropriate" for the current situation than others. The measure for this is the inclusion order in the set of associated situations. Formally $g = (S, p)$ is more appropriate than $g' = (S', p')$ for the current situation t if and only if $t \in S \subsetneq S'$. Hence, generators which are more specialized to the current situation are preferred to less specialized ones. This unveils a nice analogy to the "best matches only" principle, but now we look out for the *best matching generators*.

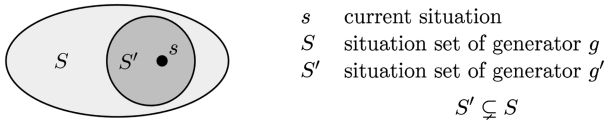

Fig. 3. Graphical illustration of the best matching generators principle.

Definition 14 (Best matching generators). *For* $G \in \mathbb{G}$ *and* $\omega \in \Omega$ *we define*

$$\text{best\_gen}(G, \omega) := \{(S, p) \in G|_{\text{sit}(\omega)} \mid \nexists\ (S', p') \in G|_{\text{sit}(\omega)} : S' \subsetneq S\}$$

The translation from components with configuration (G, π) *to a preference term using the best-matching-generators principle is given by* $\text{pref}(\text{best\_gen}(G, \omega), \pi, \omega)$.

In Fig. 3, the situation sets of two generators are illustrated. The best_gen function applied to $\{g, g'\}$ returns just g' for $s = \text{sit}(\omega)$, because S' is more specialized to the current situation.

A similar concept also occurs in [9]. But since we use this concept only to *reduce* the number of active generators, we allow several active generators. They may also produce contradicting preferences or more than one preference on the same attribute.

Example 11 (Running use case). In Example 10 using Definition 14 only one generator is in the set $\text{best\_gen}(\{g_{ws,i} \mid i = 1,2,3\})$. Due to $\{weather = x\} \notin \mathbb{S}$ for $x \in \{\text{bad, warning}\}$ we obtain:

$$\text{best\_gen}(\{g_{ws,i} \mid i = 1,2,3\}) = \begin{cases} g_{\text{ws},3} & \text{if } \text{sit}(\omega) \subseteq \{weather = \text{warning}\} \\ g_{\text{ws},2} & \text{if } \text{sit}(\omega) \subseteq \{weather = bad\} \\ g_{\text{ws},1} & \text{otherwise} \end{cases}$$

D. The Context Model in the Use Case

To apply the context model to Example 1 from the introduction we are still missing some definitions. We have generator sets for all the components of the <context> and <profile> part except <weather_conv>. We will omit this component for brevity, which is intended to generate preferences on altitude ranges with convenient temperatures, quite similar to the <weather_safety> component in Example 10.

We exemplify the formal notation of the generators for <user_input> and <user_type>, informally described in Example 6.

Example 12 (Running use case). Assume the context variable $d\_set : \Omega \to \{0,1\}$ which tells us whether the user has set the value for the tour duration. Let $inp\_dur : \Omega \to \mathbb{R}^+$ the ω-dependent function representing the duration value selected by the user. Finally $usertype : \Omega \to \{\text{tourist, athlete,...}\}$ is the context variable specifying the user type. We define the following generator for the <user_input> component:

$$g_{\text{dur\_input}} := (\{d\_set = 1\}, \text{AROUND}_{0.1 \cdot inp\_dur(\omega)}(inp\_dur(\omega)))$$

For the <user_type> component we define the following generators:

$$g_{\text{dur\_default},1} := (\{d\_set = 0, usertype = \text{tourist}\}, \text{AROUND}_{0.5}(5))$$

$g_{\text{dur\_default},2} :=$
$(\{d\_\ set = 0, usertype = \text{athlete}\}, \text{AROUND}_{0.8}(8))$

In an analogous manner we define generators for ascent and length. These definitions produce the desired user input and the user type dependent input.

Thus, the concept of generators is "reused" for this component: By allowing that the input from the search mask is also stored in ω and by defining context variables as in the example above the concept of generators helps us to formally specify the user model for the input.

In the following example we subsume our results and demonstrate how the entire preference term is composed.

EXAMPLE 13 (Running use case). According to the previous examples and the assumption that convenient weather conditions imply a minimal altitude (attribute *alt_ min*) higher than 1,200 m (because it is too hot below), here, we present the entire preference term created by our model:

$$\langle \textbf{pref\_term} \rangle = \underbrace{\text{AROUND}_{0.5}(duration,\ 5)}_{P_1 := \langle \text{user\_input} \rangle}\ \&$$

$$\underbrace{\text{Less\_THAN}_{d_a}(alt\_max,\ 2400)}_{P_2 := \langle \text{weather\_safety} \rangle}\ \&$$

$$\underbrace{(\text{POS}(recommended,\ \text{'yes'})}_{P_3 := \langle \text{social\_net} \rangle} \otimes \underbrace{\text{NEG}(visited,\ \text{'yes'}))}_{P_4 := \langle \text{history} \rangle}\ \&$$

$$\underbrace{\text{More\_THAN}_{d_a}(alt\_min,\ 1200)}_{P_5 := \langle \text{weather\_conv} \rangle}\ \&$$

$$\underbrace{(\text{AROUND}_{1.2}(length,\ 12) \otimes \text{AROUND}_{60}(ascent,\ 600))}_{P_6 := \langle \text{uesr\_type} \rangle}$$

Thereby we use the rough structure from Eqs. (l)-(3), P_1 and P_6 are generated according to Example 6, P_3 and P_4 are explained in Example 7, and P_2 is from Example 10. Also, P_5 is mentioned above in this example.

With this we have defined a model which is applicable for our problem of designing a hiking tour recommender. But we still have one drawback with the assumption that all the external information in ω is available for the recommender. This may not always be the case.

For example, we assumed we have access to the complete weather forecast to determine the values of our context variables. However, what if the user did not specify a region for the activity or too big of a region to obtain a reliable weather forecast? Or what shall the recommender do if the weather service is unavailable? This can be modeled by replacing the current situation $\text{sit}(\omega) \in \mathbb{S}$ by a set of current situations $\text{sit}(\Omega_0) \subseteq \mathbb{S}$ with $\Omega_0 \subseteq \Omega$ where all weather states are contained, formally $\{weather = x\} \not\subseteq \text{sit}(\Omega_0)$ for all $x \in \{\text{good, bad, warning}\}$. In the formal model of the generators one must only change all occurrences of the expression $\text{sit}(\omega) \in S$ by $\text{sit}(\Omega_0) \subseteq S$, which is straightforward but will not be calculated for brevity. The implication of this is that all the weather dependent generators are not active.

IV. CONTEXT-AWARE PREFERENCE QUERY EVALUATION

The complex preference terms generated in Section III must be evaluated efficiently to retrieve the best-matching objects for the user. Therefore, we discuss in this section the Preference SQL system which provides a query language for evaluating preference queries. Moreover, we present some optimization techniques for preference optimization and present some practical performance benchmarks.

A. Preference SQL

1) Preference SQL Syntax:

Preference terms can be transformed into Preference SQL, an extension of standard SQL for preference query evaluation on database systems, cp., [11].

Syntactically, Preference SQL extends the `SELECT` statement of SQL by an optional `PREFERRING` clause. A preference query selects all best matching tuples, i.e., tuples that are not dominated by other tuples. The preference SQL currently supports most of the SQL-92 standard as well as many base and complex preference constructors. For a full overview we refer to [1, 3, 11]. For our goals, a Preference SQL query block has the following schematic design:

```
SELECT        ...  <selection>
FROM          ...  <table_reference>
WHERE         ...  <hard_conditions>
  PREFERRING  ...  <soft_conditions>
  GROUPING    ...  <attribute_list>
  TOP         ...  <number>
  BUT ONLY    ...  <but_only_condition>
  HAVING      ...  <hard_conditions_for_groups>
ORDER BY      ...  <attribute_list>
LIMIT         ...  <number>
```

Fig. 4. Example of a possible schematic design of a Preference SQL query block.

The keywords `SELECT`, `FROM`, `WHERE`, and `ORDER BY` are treated as the standard SQL query keywords. The `PREFERRING` clause specifies a preference by means of the preference constructors given in Section II and [11]. `GROUPING` allows a grouped preference selection, cp., Definition 1. Additionally, `GROUPING` contains the `GROUP BY` functionality as in the standard SQL if aggregate functions (such as sum(...), count(...), etc.) occur. `HAVING` allows for hard conditions for groups as in the standard SQL. Further keywords such as `BUT ONLY`, `TOP` and `LIMIT` are provided to modify the preference evaluation. `BUT ONLY` is used for the definition of post-filters, and `TOP` and `LIMIT` to regulate the size of the result set. After specifying a preference it is evaluated on the result of the hard conditions stated in the `WHERE` clause, return-

ing the BMO-set. Empty results can only occur if the WHERE clause returns an empty result. For more details we refer to [11] and [16].

The final preference term from Example 13, e.g., can be formulated in Preference SQL as demonstrated in Fig. 5.

A Prioritization is expressed using PRIOR TO, whereas AND in the PREFERRING clause denotes a Pareto preference. The preference constructors AROUND, LESS THAN and MORE THAN are already known from Section II and III. Note that the second argument of these preference constructors denotes the *d*-parameter. The P_i in the comments of the query corresponds to the preference terms in Example 13.

The query is executed on a productive tour database provided by Alpstein Tourismus GmbH (http://www.alpstein-tourismus.com). It returns all hiking tours in the Bavarian Alps which correlated best with the preference term from Example 13. All hiking tours are stored in the relation tour. There is a connection with the relation userregion, because it holds personalized information about recommendations and visited tours, cp., Example 7.

2) Preference SQL System Architecture:

While the first prototype [17] used query rewriting to standard SQL, the current implementation of Preference SQL (since 2005) is a middleware component between the client and database which performs the algebraic and cost-based optimization of preference query evaluation, (Fig. 6).

The "top of the database" approach allows a seamless, flexible and efficient integration with standard SQL back-end systems using a Preference SQL Java database connectivity (JDBC) Driver. This enables the client application to submit Preference SQL queries through familiar SQL clients. The Preference SQL middleware parses the query and performs preference query optimization as subsequently described. Those parts of an optimized query execution plan, which directly correspond to SQL-92, are handed over to the attached SQL database system for evaluation. The (grouped) preference selection operator together with novel algebraic and cost-based query optimization methods are dealt within the Preference SQL middleware.

This trend is strengthened by [18], which even suggests implementing numerical preferences tightly in the relational database engine. According to [2] Preference SQL is currently the only comprehensive approach which implements a general preference query model.

```
SELECT t.id, t.name
FROM   tour t, userregion u
WHERE  t.main_region = 'Bavarian␣Alps'
  AND  t.subregion_id = u.subregion_id
  AND  u.user_name = 'John'
PREFERRING -- P1
         t.duration AROUND 5, 0.5
  PRIOR TO -- P2
         t.alt_max LESS THAN 2400, 200
  PRIOR TO -- P3 ⊗ P4
         (u.recommend = 'yes'      AND
          u.visited  != 'yes')
  PRIOR TO -- P5
         t.alt_min MORE THAN 1200, 200
  PRIOR TO -- P6
         (t.length AROUND 12, 1.2  AND
          t.ascent AROUND 600, 60);
```

Fig. 5. Preference SQL query for the preference term in Example 13.

B. Optimization of Preference Queries

A first view on the query in Fig. 5 leads to the assumption that the evaluation of this query on a database system is very inefficient. Executing such queries on large data sets makes an optimized execution necessary in order to maintain small runtimes. For this, Preference SQL implements the accumulated vast query optimization knowhow from relational databases as well as additional optimization techniques to cope with the preference selection operator for complex strict partial order preferences. Basically, in its current version a Preference SQL query is processed as follows:

1) The query is parsed and transformed into an initial operator tree annotated by preference relational algebra, which comprises classical relational algebra plus the preference selection operator [3].
2) Then heuristics for the algebraic operator tree transformation are applied. For this purpose, the familiar principles known from standard relational databases

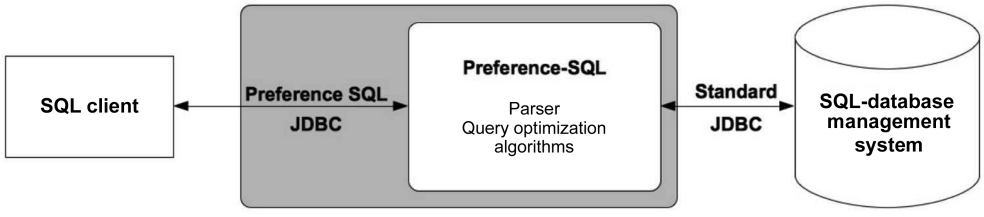

Fig. 6. System architecture of Preference SQL. JDBC: Java database connectivity.

had to be augmented by new transformation laws for preference relational algebra (see [10, 19, 20] for details).

3) For efficient evaluation of the preference selection operator in our middleware, special efficient algorithms had to be implemented. For instance, for Pareto/skyline queries this repertoire includes the Hexagon algorithm [21] and LESS [22]. The algorithm selection is guided by a cost-based query execution model.
4) Likewise, guided by a cost-based model, the preference query optimizer determines suitable sub-trees of the final optimized operator tree for offloading them to the back-end SQL system. Those sub-trees are re-translated into SQL and sent via JDBC to the attached back-end afterwards [11].
5) The entire result of the query is assembled in our middleware and returned to the requesting client by the JDBC driver of the Preference SQL.

For example, the Preference SQL rule based query optimizer transforms the query from Fig. 5 into the operator tree depicted in Fig. 7.

The optimizer applies among well-known optimization rules the law *L8: Split Prioritization and Push over Join* published by [19]. This law splits the large prioritization preference from Example 13 and pushes $\sigma[P_1 \,\&\, P_2]$ over the join ($\bowtie$). Since the preference selection $\sigma[P_1 \,\&\, P_2]$ may eliminate tuples from the relation tour which are necessary for the join on the subregions, a semi-join ($\ltimes$) on `tour` and `userregion` is necessary. Finally, *a grouped preference selection* after the join operation leads to an optimized operator tree of this context-aware preference query. Note that we omit the *Push-Projection* optimization law for a better reading. Afterwards, a cost-based algorithm selection is applied for efficient Pareto and prioritization evaluation. Such a preference query processing leads to a very fast retrieval of the best-matching objects concerning the users preferences. For further details we refer to [11].

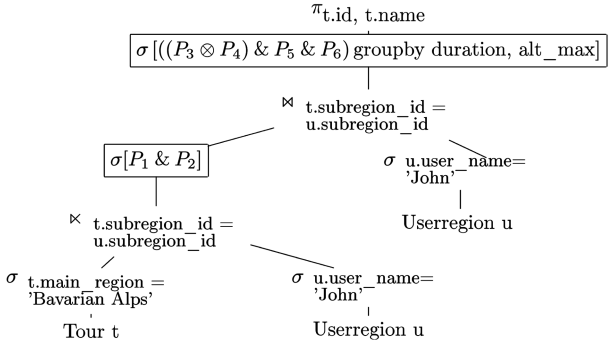

Fig. 7. Optimized operator tree of the Preference SQL query in Fig. 5.

C. Practical Performance Tests

We now present some practical runtime benchmarks for the evaluation of context-aware preference queries.

In our benchmarks the Preference SQL operates as a Java framework on top of a PostgreSQL 8.3 database. It stores the already mentioned real-world database of Alpstein Tourismus GmbH. The relations `tour` and `user-region` used in this paper contain approximately 48,000 rows and 200 rows, respectively. We build groups of three queries each with a specified *main region* as a hard constraint, e.g., *Bavarian Prealps, Bavarian Alps*, and *Alps*. For each query in a group we generated different preference terms to show the influence of context-aware preference generation. Furthermore, the hard constraint on the main region leads to a different number of tuples (the basic set) for preference evaluation. The runtimes are depicted in Fig. 8.

Query 4 corresponds to the preference query used in this paper, cp., Fig. 5. It can be evaluated in less than 1 second. The hard selection on the main region of the Bavarian Alps as well as the early computation of the P_1 & P_2 preference (cp., Fig. 7) leads to a preference selection on 1,650 tuples after joining. Queries 5 and 6 are restricted to the same main region but have different preference terms.

Query 7 is similar to Query 1. However, the basic set (about 16,000 tuples) is changed because of a different main region, namely the complete Alps. The runtime is about 3 seconds, which is fast enough for context-aware recommender applications, e.g., in the domain of hiking tours.

The evaluated queries are only a small selection of the queries occurring in our use case, but they are representative for our performance benchmarks. The runtime of these queries show that our approach of a context-aware preference query composition is not only intuitive but

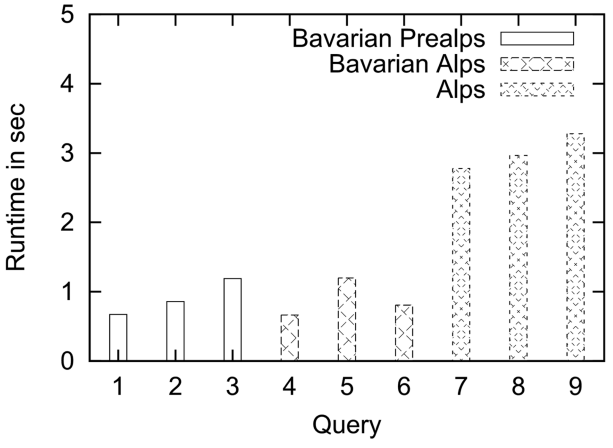

Fig. 8. Runtime for different context-aware preference queries.

also very efficient in computation of results on a productive database.

V. IMPLEMENTATION OF THE MODEL IN AN INDUSTRIAL PROJECT

In the previous sections we have described the theoretical foundations of our context model and the practical experiences with it according to our running use case from the introduction. In this section we describe our experiences with the context-model while implementing it in a joint project together with our industrial partner Alpstein Tourismus GmbH.

We present a sketch of the implementation of the context-model itself and also the entire system, including interfaces to the database and the backend for the domain experts. They are responsible for the content of the context-model, i.e., the preferences and the associated situation sets, organized in the context-aware generators. For example it is the task of the domain expert to adjust default values for length, duration and ascent for the different user types. It is also their task to define the context-variables which span the situation set.

A. The Entire System

The implementation is mainly sub-divided in two parts: the first part is the generation of the query. To this end the input and the model must be brought together. We call the application performing this task the *aggregator*. We describe the components of the query generation in detail:

- The front-end for the user, where the user input is taken from. The input form is shown in Fig. 9. There the user has to select a user type (unless he is not a registered user) and his preferred values for length, ascent and duration.
- If the user is registered for the portal, his user type is taken from the *profile* instead of the input form. There is also additional information in the profile, e.g., recommendations from the social network or the history of visited tours.

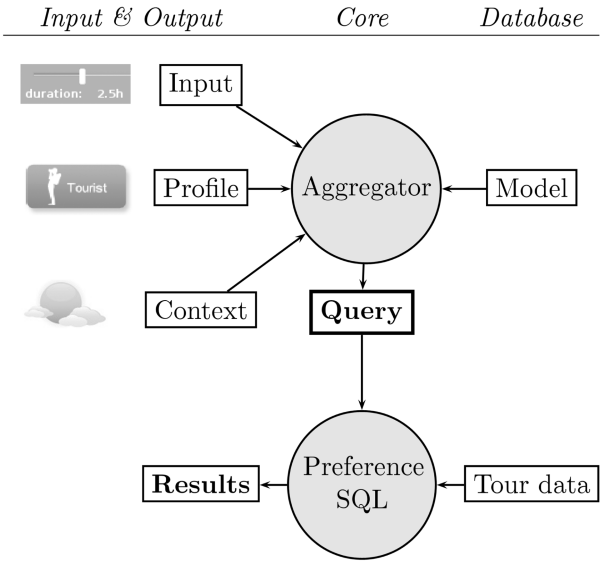

Fig. 10. The architecture of the entire system.

- The information from external knowledge sensors is subsumed by the *context*.
- The *model content* is stored in the database.
- The generation of the query, based on the components above, is done in the *aggregator*.

The next part is the evaluation of the preference query. This is done using the Preference SQL and the tour database provided by our industrial partner. After the evaluation, the results are presented to the user. The architecture of the entire system is depicted in Fig. 10.

B. Sketch of Implementation of the Context Model

In the following we provide a brief description of the main concepts for the implementation of the context model.

There are some important classes:

- *Context*, implementing a data structure for $\omega \in \Omega$ from Definition 9. All the ω-dependant functions are fields of this class. Their type is arbitrary, but often

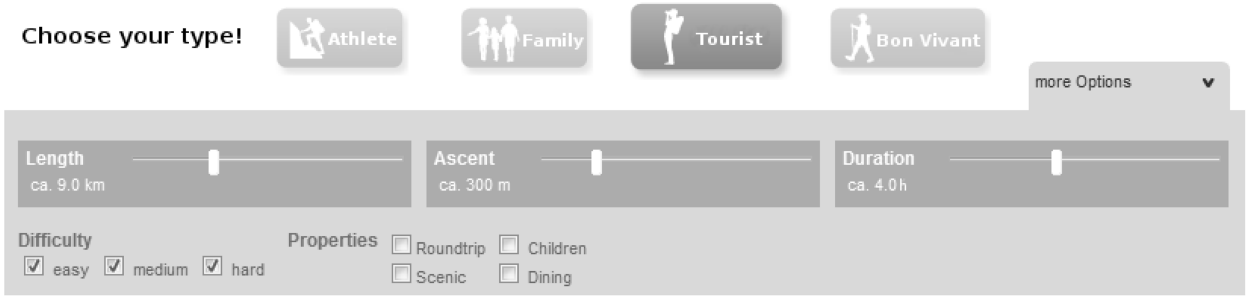

Fig. 9. Partial input mask for the hiking tour recommender.

numeric (e.g., snowline,...).

- *Situation*, implementing the v_i and sit : $\Omega \rightarrow \mathbb{S}$ from Definition 9: Every context variable is a field of this class which may "null". Thereby "null" expresses that the value of this variable is not known. Mathematically the "null" represents the set of all possible values for this context variable. Due to this, the Situation class represents not only single situations but also *situation sets*. As the context variables are discrete, the fields are usually of "enum" type, and sometimes they are also booleans for "yes/no" variables. The situation class contains methods to check if:
  - According to Definition 12: Check, if a situation is contained in the represented situation set.
  - According to Definition 13: Check, if a situation set is strictly contained in another situation set.

  Both methods obtain the fields of the situation class by the reflection techniques in Java, i.e., the methods do not have to be modified if new context variables are introduced.
- *Preference Term*, implementing Definition 10. This is an abstract class for ω-dependant preference terms. It has concrete subclasses for complex preferences and base preferences, ω-dependant attributes are realized by function classes mapping a "Context"-object to a numerical domain.
- *Situated Generator*, implementing Definition 11: This class has the attributes *Situation* and *Preference Term*; hence it implements our central concept of the context-aware generator. It contains also a numerical ranking field, which represents the π-function from Definition 13.
- Finally a main class for the program, where the algorithm from Definition 13 is implemented, i.e., the entire preference term is generated.

The content of the model (i.e., the generators and the associated situations and preferences) is stored in a database via Hibernate; hence, Java objects representing the concrete model are created automatically.

C. Qualitative User Studies

Based on the implementation we completed some qualitative user studies presented in [5]. We will recapitulate and summarize the results in the following.

The technical criterion of the tests is the runtime; the user acceptance testing includes:

- Session time of test subjects
- Behavior: success, abort or iteration
- Comments of test subjects

We compared the Faceted search without any context model, described in the introduction, with the preference search based on the context model, described in the running use case. Additionally, the results of the preference search have been analyzed by the domain experts from our industrial partner. The details of the evaluation can be found in [5]. The tour database contained approximately 48,000 tours.

In summary, we obtained the following results:

1) The run time of the generated Preference SQL queries never exceeded 5 seconds.
2) The session time of the preference search was about half of the session time with the Faceted search.
3) The test subjects expected "larger" result sets executing the preference search – due to the many preferences generated in the context-model, the result set was often quite small (<5 tours).
4) Empty results are still present in the preference search due to hard selections like geographical restrictions.
5) The flooding effect was never noticed executing the preference search. The result set mostly consists of 1–7 tours.
6) The average quality of the result set of the preference search was rated as 1.6 by domain experts – where 1 means "best" and 3 means "worst".

VI. SUMMARY AND OUTLOOK

In this paper we present a new approach to context-aware preference query composition which covers everything from context modeling to generating the preference query in a well known preference query language. Preferences, after being retrieved from a repository, are composed – on the fly – depending on context. The construction is performed by an algorithm which generates the best-suited composition of context-dependent preferences per actual situation. The algorithm relies on a discrete context model, where preferences are associated to situation sets. Since the context model as well as the context-sensitive preferences have an explicit declarative description, domain experts may easily adapt or extend the model to meet their specific requirements without any changes to the generation algorithm.

By using discretization and context dependent functional mappings, we achieve an easy to understand system that is suitable for commercial applications. A running use case from the field of commercial e-business platforms for outdoor activities is used to motivate and illustrate the various steps in the process. The performance evaluation demonstrates that the answers for the generated queries can be computed efficiently on realistic data sets. Due to the BMO principle the trial and error behavior is replaced by a one-shot behavior which significantly reduces the session time at a high level of customer satisfaction. We have sketched an implementation which proved its value in a joint project with an industrial partner. The qualitative user studies underline the practical relevance of our work.

For future research we are working on further optimizing the queries using algebraic optimizations. Furthermore, we investigate preference modeling, especially the composition of complex preferences. In the use case presented in this paper we observed that in long chains of prioritization preferences containing Pareto preferences (cp., Fig. 5) the last occurring preferences of the term are often not decisive for the query result. In some cases, users rate the results as counterintuitive. In [23] we analyzed this effect in detail and suggested a new kind of Pareto preference as a remedy.

ACKNOWLEDGMENTS

This project has been funded by the German Federal Ministry of Economics and Technology (BMWi) according to a resolution of the German Bundestag, grant no. KF2751301.

REFERENCES

1. W. Kießling, "Preference queries with SV-semantics," *Proceedings of the 11th International Conference on Management of Data*, Goa, India, 2005, pp. 15-26.
2. K. Stefanidis, G. Koutrika, and E. Pitoura, "A survey on representation, composition and application of preferences in database systems," ACM Transactions on Database Systems, vol. 36, no. 3, article no. 19, 2011.
3. W. Kießling, "Foundations of preferences in database systems," *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002, pp. 311-322.
4. J. Chomicki, "Database querying under changing preferences," *Annals of Mathematics and Artificial Intelligence*, vol. 50, no. 1-2, pp. 79-109, 2007.
5. W. Kießling, M. Soutschek, A. Huhn, P. Roocks, M. Endres, S. Mandl, F. Wenzel, and A. Zelend, "Context-aware preference search for outdoor activity platforms," University of Augsburg, Augsburg, Germany, Technical Report 2011-15, 2011.
6. J. J. Levandoski, M. E. Khalefa, and M. F. Mokbel, "An overview of the CareDB context and preference-aware database system," *IEEE Data Engineering Bulletin*, vol. 34, no. 2, pp. 41-46, 2011.
7. E. Pitoura, K. Stefanidis, P. Vassilidis, "Contextual Database Preferences," *IEEE Data Engineering Bulletin*, vol. 34, no. 2, pp. 20-27, 2011.
8. A. H. van Bunningen, L. Feng, and P. M. G. Apers, "A context-aware preference model for database querying in an ambient intelligent environment," *Proceedings of the 17th International Conference on Database and Expert Systems Applications*, Krakow, Poland, 2006, pp. 33-43.
9. K. Stefanidis, E. Pitoura, and P. Vassiliadis, "Adding context to preferences," *Proceedings of the IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007, pp. 846-855.
10. J. Chomicki, "Preference formulas in relational queries," *ACM Transactions on Database Systems*, vol. 28, no. 4, pp. 427-466, 2003.
11. W. Kießling, M. Endres, and F. Wenzel, "The preference SQL System: an overview," *IEEE Database Engineering Bulletin*, vol. 34, no. 2, pp. 11-18, 2011.
12. F. Wenzel, M. Endres, S. Mandl, and W. Kießling, "Complex preference queries supporting spatial applications for user groups," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1946-1949, 2012.
13. L. Barkhuus and A. Dey, "Is context-aware computing taking control away from the user? Three levels of interactivity examined," *Proceedings of the 5th International Conference on Ubiquitous Computing*, 2003, Seattle, WA, pp. 150-156.
14. H. Gibson and A. Yiannakis, "Tourist roles: needs and the lifecourse," *Annals of Tourism Research*, vol. 29, no. 2, pp. 358-383, 2002.
15. D. Mindolin and J. Chomicki, "Discovering relative importance of skyline attributes," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 610-621, 2009.
16. Preference SQL, http://www.preferencesql.com.
17. W. Kießling and G. Kostler, "Preference SQL: design, implementation, experiences," *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002, pp. 990-1001.
18. A. Arvanitis and G. Koutrika, "Towards preference-aware relational databases," *Proceedings of the IEEE 28th International Conference on Data Engineering*, Washington, DC, 2002, pp. 426-437.
19. B. Hafenrichter and W. Kießling, "Optimization of relational preference queries," *Proceedings of the 16th Australasian Database Conference*, Darlinghurst, Australia, 2005, pp. 175-184.
20. M. Endres and W. Kießling, "Semi-skyline optimization of constrained skyline queries," *Proceedings of the 22nd Australasian Database Conference*, Perth, Australia, 2011.
21. T. Preisinger and W. Kießling, "The hexagon algorithm for Pareto preference queries," *Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling*, Vienna, Austria, 2007.
22. P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, 2005, pp. 229-240.
23. B. Moller and P. Roocks, "An algebra of layered complex preferences," *Relational and Algebraic Methods in Computer Science, Lecture Notes in Computer Science vol. 7560*, W. Kahl and T. G. Griffin, editors, Heidelberg: Springer Berlin, pp 294-309, 2012.

Patrick Roocks

Patrick Roocks received his diploma in mathematics in 2010 at the University of Augsburg (Germany). Since then he works as a scientific assistant at the chair for Databases and Information Systems at the University of Augsburg. He is currently involved in a project in the area of parallelized preference databases. Additionally he is working on algebraic calculi for preference databases.

Markus Endres

Markus Endres is a post-doctoral researcher at the Department of Computer Science at the University of Augsburg (Germany). In addition he works as a lecturer for mathematics at the University of Applied Sciences Munich. His research interests are preference based database systems and regard several aspects of (parallel) preference query evaluation, in particular Pareto (Skyline) computation, preference query optimization and preference recommender systems. He is the author of several publications, including journals and papers in proceedings of international conferences such as VLDB, DEXA, DASFAA, ADC, FQAS, MPC, and a book on Semi-Skyline query computation.

Stefan Mandl

Stefan Mandl was a researcher and the chair for Artificial Intelligence at the University Erlangen-Nuremberg, where in 2008 he received a Ph.D. for his work on unconsidered contexts of formal representations. From 2011 to 2012 he was research assistant at the chair for Databases and Information Systems at the University of Augsburg. Currently he is a software developer at EXASOL AG Nuremberg.

Alfons Huhn

Alfons Huhn is currently a researcher and the chair for Databases and Information Systems at the University of Augsburg, Germany. He received his Ph.D. in computer science from the University of Karlsruhe in 1991.

Werner Kießling

Werner Kießling, born 1953, received his Dr. degree in 1983 from the Technical University of Munich, supervised by Prof. R. Bayer. After spending a Postdoc year with Prof. M. Stonebraker at the University of California at Berkeley he became director of R&D for MAD Intelligent Systems in Silicon Valley. Turning back to academia he held a permanent professor position at the Technical University of Munich. Since 1993 until today he is chairholder for Databases and Information Systems within the Faculty of Applied Informatics at the University of Augsburg. Prof. Kießling has worked in many areas of database systems, formerly in particular in deductive database systems. He has co-authored about 100 research papers in international conferences, journals and workshops. His current main research focus is on foundations and implementation issues of preference technology for databases (Preference SQL), and on its applications for personalized search engines, in particular for e-commerce, m-commerce and OLAP.