September 2001

Preference XPATH: A Query Language for E-Commerce

Werner Kießling
University of Augsburg, kiessling@informatik.uni-augsburg.de

Bernd Hafenrichter
University of Augsburg, hafenrichter@informatik.uni-augsburg.de

Stefan Fischer
University of Augsburg, fischer@informatik.uni-augsburg.de

Stefan Holland
University of Augsburg, holland@informatik.uni-augsburg.de

In: Buhl, Hans Ulrich, u.a. (Hg.) 2001. *Information Age Economy*; 5. Internationale Tagung Wirtschaftsinformatik 2001. Heidelberg: Physica-Verlag

ISBN: 3-7908-1427-X

Preference XPATH: A Query Language for E-Commerce

Werner Kießling, Bernd Hafenrichter, Stefan Fischer, Stefan Holland

University of Augsburg

Abstract: – We present a new XML-based search technology that enables users to formulate complex customer or vendor preferences, which typically occur within e-commerce applications. Preferences are modeled in a natural way by partial orders. Since our semantics of multi-attribute preferences implements the Pareto-optimality principle, Preference XPATH queries avoid both the unwanted "empty-result"-effect and the flooding-effect with lots of irrelevant query results. If perfect matches are not available best-possible alternatives are found instead. We have extended the XML query language XPATH by the capability to formulate preferences as soft selection conditions. As our extensions are fully compatible with the XPATH standard, both hard and soft selection conditions become now available to any XML-based e-commerce application. Several e-shopping examples show how easy and elegant it is to transform customer wishes into Preference XPATH queries. Our prototype implementation is smoothly integrated with the XML database system Tamino of Software AG. Moreover, we show how Preference XPATH can be used within the XML query language QUILT. It even merges with XML style sheets (XSLT) and the XML pointer language (XPointer). Thus with Preference XPATH powerful personalized search engines and match-making processes for B2C and B2B can be implemented completely inside the XML framework.

1 Introduction

In e-commerce XML [BPSM00] has become a very important standard for storing, presenting or exchanging data. XML documents can be classified into text-oriented or data-oriented documents. *Text-oriented* documents are usually text mixed with markup to point out the structure of the underlying text [Robi99]. For this document class information retrieval methods can be adopted to the needs of XML. In [Fuhr00], e.g., an approach is described which extends the query language XQL (see [RLS98]) for use in information retrieval. In *data-oriented* XML documents XML is used as vendor independent interchange format between applications and seems to become a major standard for data exchange on the web ([ABS00]). XML can be used as an integrated view on different kinds of data

sources. For this document class attribute search methods that can deal with customer and vendor preferences of e-commerce applications must be supported. Basically exist three main approaches for attribute search:

- The naive way of translating each preference into a *hard* selection condition.
- Translation of preferences into *soft* selection conditions and application of some ranked query model, using *numerical* scores and some weighted combination functions.
- Translation of preferences into *soft* selection conditions, modeling preferences *qualitatively*, e.g. as *partial orders*.

In practice, mixtures between hard and soft selection conditions always occur and should be supported as well. As experienced with the commercial product Preference SQL [Data99], partial orders are a good foundation for an e-commerce query language. Hence we shall adopt this promising approach and will show how to transfer it from an SQL environment into the XML realm.

In *section 2* we recapture the Preference SQL approach by explaining how to model preferences as partial orders. *Section 3* describes the design and syntax of our Preference XPATH query language and includes some characteristic examples from B2C e-shopping. In *section 4* we present a smooth way to implement Preference XPATH as an extension to an existing XPATH system like, e.g., the XML-database Tamino from Software AG. How Preference XPATH can be mixed with other XML-languages like XSL and QUILT is the topic of *section 5*. Finally, in *section 6* we give a summary and outlook of our work.

2 Preference Modeling for E-Commerce

Now we describe the basic idea of modeling preferences as partial orders and their underlying theory. *Partial orders* (i.e. reflexive, anti-symmetric and transitive binary relations on a given set of objects) are known to be compatible with relational and deductive database technology ([KKTG95]). In [KiGü94] it was already shown, how declarative query languages can accommodate partial orders to express user-definable preferences. Since then this idea was carried over to the commercial product *Preference SQL* [Data99] which is a development tool for personalized search engines in e-commerce applications. Here we will adapt these ideas to the XML environment of XPATH. In Preference SQL each preference is modeled as a partial order. For the first version of Preference XPATH we have realized roughly the same modeling capabilities as for Preference SQL. Further extensions of Preference XPATH, which are possible due to the richer data model of XML compared to standard SQL, will be addressed in the summary section.

Partial orders model semantic relationships of the type "I like *A* more than *B*", hence being of *qualitative* nature. This kind of expressing preferences is familiar to everybody - not only to technical specialists, juggling with numerical scores and weights. For this very reason we consider it as an appropriate way to express preferences for personalized product searches or match-makings in B2C or B2B. Preferences are distinguished into base preferences and combined preferences.

2.1 Base Preferences

A *base preference* is a partial order $\langle V, \rangle$ based on a set V of values of one particular data type. In this paper we restrict ourselves to the atomic XPATH data types `String`, `Boolean` and `Number`. Base preferences themselves are either explicitly definable by the user or they can be pre-defined.

2.1.1 User-Definable Explicit Preference

Let's assume Julia wants to buy compact discs at an e-shop. Her decision depends (among other things) on her personal preferences among music categories of the CD's available. Her preference is expressed as explicitly listed partial order as shown in figure 1 (an arrow representing a better-worse relationship).

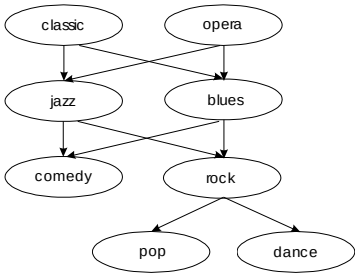

Figure 1 : Base preference as a used-definable explicit partial order

Thus Julia prefers classic and opera music most. If this perfect choice is not available, she prefers jazz or blues next. Comedy and rock are appropriate if none of the mentioned categories are available. In the worst case none of the preferred categories are available. Only then other CD's would qualify.

2.1.2 Predefined Base Preferences

Explicit preferences are a powerful means to specify preferences in finite domains. However, in many situations this can become very tedious. Therefore a set of predefined base preferences will be supplied, all of which can be described by partial orders. From the experiences gained with Preference SQL in typical e-commerce

domains like Internet portals for flight booking, car sales, real estate brokerage and many more, the subsequent choice was found as very appropriate.

- **"Around" Preference**

Suppose Julia wants to buy a car, the price should be around $10.000. It is unlikely that she finds an exact match, but there may be a lot of cars that come very close to this value. The "around" preference is used if an exact value is not a must, alternatively closest matches are acceptable. Values closer to the exact match are better than others.

- **"Interval" Preference**

Julia wants to buy a book within a price range of $10 up to $20. An "interval" preference rates all such books as equally good, books outside this range are considered worse. Analogously to the "around" preference, books outside but very close to the interval borders are better than those farther away. As special cases, one-sided intervals are supported (i.e., one interval boundary may be unspecified).

- **"Extremal" Preference**

Michael wants to buy a book, preferring the cheapest one. In terms of our preference model this means minimizing the price attribute. "Extremal" preferences are used to minimize or maximize an attribute. Values closer to the extremal value are preferable to those farther away.

- **"Positive" Preference**

This time Julia wants to buy a car, preferring the colors red and blue. If cars with either of these colors are in stock, they will be offered to Julia. Otherwise cars with different colors are an alternative choice. This type of preference is called "positive" preference. Any value in the positive set is rated better than the rest.

- **"Negative" Preference**

"Negative" preferences behave opposite to "positive" preferences. Each value not contained in the negative set is preferred. If Julia wanted to buy a car and disliked the colors magenta and cyan, cars with a different color are preferable.

Additionally we support so-called **"**pos/neg" preferences and "pos/pos" preferences, combining the properties of "positive" and "negative" preferences.

2.2 Combination of Preferences

In practice preferences and purchase decisions are rarely 1-dimensional, but much more complex in general. To construct more complex preferences, base preferences can be combined by two principal methods: prioritization and cumulation.

2.2.1 Cumulation of Preferences

Cumulation treats several preferences on different attributes as *equally important*: Object x is better than object y, if x is better than y according to at least one preference and at least equally good with respect to the other preferences.

This corresponds to the well-known *Pareto optimality* principle, which has been applied and studied extensively in particular for multi-attribute decision problems (see e.g. [KeRa76]).

More formally, let's assume the partial orders $\langle X_i, \ _i \rangle$, $1 \le i \le n$, $n \ge 2$. Cumulation is defined as the coordinate-wise order of the cartesian product $X = X_1 \times ... \times X_n$. Let $x = (x_1,...,x_n)$, $y = (y_1,...,y_n) \in X$:

$$x > y \ \Leftrightarrow \ \exists i \in 1, ..., n: \ x_i >_i y_i \ \wedge \ (\forall j \neq i: x_j \ _j \ y_j \ )$$

Cumulation is known to be a partial order itself (see [DaPr90, p. 18]).

Assume that Jutta is looking for a car. Ideally, it should display the following equally important characteristics: to be one year old, to be a BMW and to be a roadster. Thus Jutta's preference translates into a cumulation of three proper base preferences. The Pareto principle guarantees that exactly the set of *best matching* cars is found. Obviously, an existing perfect match (a one year old BMW roadster) would dominate all other cars. If not available best alternatives would be searched for Jutta automatically. Thus the Pareto-optimality semantics for multi-attribute preferences avoids the often occurring and embarrassing "empty-result" effect of many Internet search engines, if no exact matches are available. Moreover, it also avoids the annoying flooding effect with a lot of irrelevant results, because worse objects (i.e. object that are subsumed by better ones) are filtered out on the fly.

2.2.2 Prioritization of Preferences

Prioritization treats preference p_1 as *more important* than preference p_2, which in turn is *more important* than p_3, etc. , up to preference p_n:

- Object x is better than object y, if x is better than y according to p_1.
- If x and y are equivalent according to p_1, then p_2 will decide which one is better, etc.

More formally, let's assume the partial orders $\langle X_i, \ _i \rangle$, $1 \le i \le n$, $n \ge 2$. Prioritization is defined like a lexicographic order of strings of the cartesian product $X = X_1 \times ... \times X_n$. Let $x = (x_1,...,x_n)$, $y = (y_1,...,y_n) \in X$:

$$x > y \ \Leftrightarrow \ \exists \ i \in 1, ..., n: (\forall_{k = 1 \ ... \ i-1} : x_k \sim_k y_k) \ \wedge \ x_i > y_i$$

Hereby $\sim_k$ denotes equivalent objects correlated to be equally important in the partial order $\langle X_k, \ _k \rangle$. Again, prioritization forms a partial order (see [DaPr90, p.

19]). Therefore cumulation and prioritization are construction operators for complex preferences that can even be applied orthogonally.

By example let's assume that Michael searches for a book. For him the most important attributes are author and title. Less important he wants to spend as little money as possible. Prioritization can be used to express this situation by cumulating the author and title preferences as most important preference p_1, followed by an "extremal" preference p_2.

3 Design of Preference XPATH

The primary construct of XPATH is an expression (see [ClDe99] for details). The result of an expression can be objects of the types `Nodeset`, `Boolean`, `Number` or `String`. The type `Nodeset` is a set of nodes that can be, e.g., XML-element or XML-attribute. For a complete enumeration of the possible type of nodes see [Cowa00]. As shown before, preferences are computed over a set of values based on a partial order. So the object type `Nodeset` is the proper starting point within XPATH for the integration of preferences.

3.1 Syntax of Preference XPATH

As specified in [ClDe99] the non-terminal production `LocationPath` returns an object of type `Nodeset`. A `LocationPath` consists of one or more `LocationStep`'s. The result of each `LocationStep` is used as input for the next `LocationStep`. A single `LocationStep` is defined as follows:

```
LocationStep : axis nodetest (predicate)*
```

The three constituents of a `LocationStep` are:

- an `axis`, which specifies the tree relationship between the nodes selected by the location step and context node,
- a `nodetest`, which specifies the node type and expanded-name of the nodes selected by the location step, and
- zero or more `predicate's`, which use arbitrary expressions for further refinements of nodes selected by the location step.

Preferences can be regarded as special kind of soft filter expressions. They take a set of values and refine this set by dropping the non-maximum elements. We decided to enhance XPATH by the introduction of a second type of predicate, namely a `preference`. To delimit a *hard selection* (i.e. `predicate`) XPATH

uses the symbols '[' and ']'. In contrast, for *soft selections* (i.e. `preference`) we introduce '#[' and ']#'. Now the production

```
LocationStep    : axis nodetest predicate*
```

in the current XPATH standard is rewritten as follows:

```
LocationStep    : axis nodetest (predicate|preference)*
preference      : '#[' prioritization | cumulation ']#'
prioritization  : base_preference ( 'prior' 'to'
                  base_preference)*
cumulation      : base_preference ( 'and'
                  base_preference)*
base_preference : named_pref | unnamed_pref
named_pref      : name '(' expr ')'
unnamed_pref    : '(' expr ')' pref_part
pref_part       : around | extremal | interval| pos_neg
                  | positive | negative | pos_pos
around          : 'around' number
extremal        : 'maximal' | 'minimal'
interval        : 'between' number 'and' number
pos_pos         : 'in' LiteralList 'or' LiteralList
positive        : 'in' LiteralList
negative        : 'not' 'in' LiteralList
pos_neg         : 'in' LiteralList 'not' 'in'
                  LiteralList
LiteralList     : '(' '"'token'"'(',' '"'token'"')* ')'
```

Every `LocationStep` is composed of zero or more hard filter predicates or soft preferences. The result set of one `predicate` or `preference` becomes the input of the next `predicate` or `preference`. A `preference` itself is either a cumulation or prioritization. Both consist of base preferences. Every base preference is definable using a named or an unnamed syntax. Both use a standard XPATH expression (`expr;` see [ClDe99] for details) to calculate their input arguments. Named preferences `named_pref` are assumed to be defined and stored within a preference repository. They are referenced using a unique identifier (`name`). On the other hand, unnamed preferences `unnamed_pref` are defined within the Preference XPATH query itself.

As stated in the definition of `LocationStep` there is now a choice between `predicate` or `preference` to express a hard or soft selection condition, respectively. Because base preferences use a XPATH query (`expr`) to select the related nodes within the input node, Preference XPATH fits extremely well into the overall design of XPATH. This completes the presentation of the syntax of Preference XPATH. A specification of its formal *semantics* is, however, beyond the scope of this paper. For the interested reader let us give a very short sketch:

Since each preference is a partial order, the theory of subsumption under partial orders in deductive databases is applicable [KKTG95]. From the model-theoretic point of view query results are subsumption models in the sense of [KKTG95], which define the *declarative* semantics. An equivalent *operational* semantics is gained by applying fixpoint theory with subsumption. A query result contains exactly all maximal elements of the given partial order.

3.2 Sample Preference XPATH Queries

We will exemplify the ease of use of Preference XPATH in a typical B2C e-shopping scenario for used cars. The following part of a DTD defines the structure of the our sample XML database.

```
<!ELEMENT CARS (CAR)*>
<!ELEMENT CAR EMPTY>
<!ATTLIST CAR
       ident          ID         #REQUIRED
       price          CDATA      #IMPLIED
       mileage        CDATA      #IMPLIED
       horsepower     CDATA      #IMPLIED
       fuel_economy   CDATA      #IMPLIED
       color          CDATA      #IMPLIED>
```

Most attributes are rather obvious. `Ident` is used to identify each car exactly. `Fuel_economy` expresses how far you can get with a fixed amount of fuel; higher values are better. `Mileage` gets for lower values.

Query 1: Michael wishes a vehicle. It ***must*** be a car and ***should*** be ***maximum*** energy efficient, and ***equally important,*** it ***should*** have ***maximum*** horsepower.

Michael's preference straightforwardly translates into cumulation of two "extremal" preferences.

```
/CARS/CAR #[(@fuel_economy) maximal and          [Q1]
           (@horsepower) maximal]#
```

Query 2: Julia wishes a vehicle, too. Again, it ***must*** be a car. But she ***prefers*** the colors black and white. ***Next important*** is the price that ***should*** be ***less than*** $10.000. After this pre-selection she ***prefers*** a ***minimal*** mileage.

Here prioritization applies with a "positive"-preference on color first and an "extremal"-preference on price second. The third preference is applied sequentially to this result.

```
/CARS/CAR #[(@color) in ("black","white")        [Q2]
           prior to (@price) up to 10000 ]#
          #[(@milage)minimal ]#
```

After these linguistic examples let's do some query evaluation. We want to pose queries to the following XML-database:

```
<CAR ident="Kangaroo" fuel_economy="45" color="red"/>
<CAR ident="Dog" fuel_economy="35" color="red"/>
<CAR ident="Frog" fuel_economy="100" color="blue"/>
<CAR ident="Shark" fuel_economy="55" color="black"/>
<CAR ident="Cat" fuel_economy="50" color="white"/>
```

Query 3: Kathy wishes a vehicle. It ***must*** be a car. She ***prefers*** red or black color, and ***equally important*** to her is a fuel_economy of around 50.

```
/CARS/CAR #[(@color) in ("red","black") and          [Q3]
           (@fuel_economy) around 50]#
```

[Q3] uses cumulation to express a multi-attribute decision on color and fuel_economy. Since a perfect match is not available here, the cars "Kangoroo", "Shark" and "Cat" are retrieved as best-possible alternatives.

Query 4: George wants a vehicle. It ***must*** be a car. George ***prefers*** red or blue color, and ***equally important*** to him is a ***maximal*** fuel_economy.

```
/CARS/CAR #[(@color) in ("red","blue") and           [Q4]
           (@fuel_economy) maximal ]#
```

[Q4] finds a perfect match: Both color and fuel_economy are maximal for ident="Frog".

In this section we have solely shown examples of customer preferences. In reality vendors of course have their preferences, too. Both *customer* and *vendor preferences* of e-commerce applications can be modeled with partial orders, they can appear even simultaneously in a single Preference XPATH query.

4 Implementation of Preference XPATH

4.1 Rewriting Approach

We pursue a pre-processor approach to achieve a smooth integration, such that the query evaluation can happen entirely within an underlying XPATH database. To this end a Preference XPATH query is rewritten into an equivalent XPATH query using a special XPATH function, the so-called "PREFERENCE" function. This mechanism is based on the EBNF-rules 15, 19 and 20 of the XPATH-specification: Rule `[EBNF 19]` defines a `PathExpr` that is either a `LocationPath` or a filter expression (`FilterExpr`). A filter expression, as stated in rule `[EBNF 20]`, can be used to construct a function call [EBNF 15]. These two facts enable

us to rewrite every path expression into the equivalent function composition of `LocationStep`'s. For example, the `LocationStep`

`/SHOP/COMPACTDISC[@TITLE="BEST"]` can be rewritten as
`id(id(/SHOP)/COMPACTDISC)[@TITLE="BEST"].`

The function `id` is defined as identity function `id(x)=x`. For our purposes we have implemented a special XPATH-function called "PREFERENCE". Its input arguments are a `Nodeset` and a reference to a prepared preference statement. The input `Nodeset` is filtered by the `preference`, the result containing all maximum elements. In this way the query

```
/SHOP/MUSIC/COMPACTDISC [@ARTIST="Enya"]
                    #[(count(./TITLE)) more than 10]#
```

can be rewritten as:

```
PREFERENCE( /SHOP/MUSIC/COMPACTDISC[@ARTIST="Enya"],
            "(count(./TITLE)) more than 10")
```

Each Preference XPATH query can be rewritten using this technique. The "PREFERENCE" function itself has two input arguments, an input `Nodeset` and a preference statement. Optimizations for soft selections under our Pareto semantics can be encapsulated within the "PREFERENCE"-function. Since currently the expressive power of Preference XPATH is equivalent to Preference SQL, all implemented and efficient optimization algorithms of Preference SQL are transferable from the SQL environment to the XML setting. These algorithms, which are currently undisclosed, are beyond the scope of this paper. To give a brief idea, an efficient subsumption operator is required eliminating subsumed elements on the fly. Optimization heuristics like "push selection" known from SQL-databases must be generalized to "push preference", etc.

4.2 The TAMINO Prototype

Next we will discuss the prototype implementation of Preference XPATH based on the commercial database Tamino [Soft01] from Software AG. Tamino is a native XML database, supporting the query language XPATH. Tamino can be enhanced by so called "server extensions". They allow the creation of user-defined functions for the use within XPATH-expressions. Thus Tamino can easily be extended by adding different query functions to the server. As application interface Tamino provides an http interface for query processing which is implemented as an extension of the Apache web server.

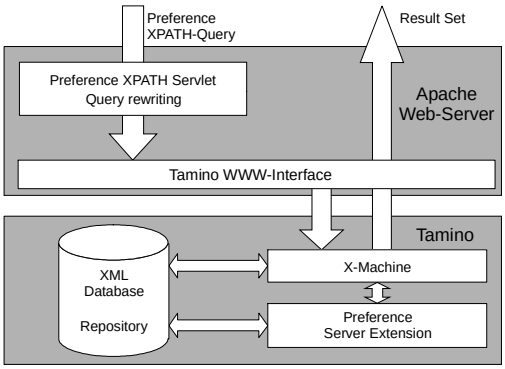

Figure 2 : Preference XPATH prototype in Tamino

In figure 2 the overall architecture is shown. We introduce a new interface layer, the "Preference XPATH-Servlet". This servlet acts as a special kind of router, guiding every user request to the Tamino interface. In case of a Preference XPATH request, the query is analyzed, rewritten into pure XPATH as presented above, then passed to Tamino and executed there by the "X-Machine". For every occurrence of the "PREFERENCE"-function the associated "Preference Server-Extension" is called. The computed result set is returned directly by the "X-Machine" to the caller.

Of course, this architectural framework is not proprietary for Tamino. Instead, it works with any XPATH engine. For example, as another prototype we have successfully used the XPATH engine of XALAN from the Apache XML Project.

5 Preference XPATH for XSLT and QUILT

The benefits of Preference XPATH are not limited to pure XPATH-engines like Tamino. Preference XPATH can be used in every application that employs XPATH expressions. The following sections show by two case studies how Preference XPATH can be applied to XSLT [Clar99] and QUILT [RCF00], both using XPATH expressions as an integral part.

5.1 Preference XPATH in XSLT

XSLT is a language for transforming XML documents into other XML documents. XSLT uses XPATH expressions to select related nodes when constructing the result document. One example is the automatic content generation for queries in e-shops. XSLT is used to transform XML source data into HTML output. This process can be *personalized* with the usage of Preference XPATH. According to the car example earlier a part of a sample style sheet may look like:

```
<xsl:apply-templates
select="./CAR #[  (@horsepower) maximal and
                  (@fuel_economy) maximal]#"/>
```

The statement `xsl:apply-templates` is used to select child nodes of the current node. Here the computed results are `CAR` elements satisfying the given preference condition best-possible.

5.2 Preference XPATH in QUILT

QUILT is another popular XML query language. It pools ideas from different query languages like XML-QL [DFFS98], XQL and XPATH. All these ideas had an impact on the design of QUILT that uses XPATH expressions to compute bindings of variables. Therefore it is possible to integrate Preference XPATH with QUILT. We adapt an example from the QUILT specification (see [RCF00]), with Preference XPATH expressions written in bold face:

```
<Result>(FOR $a IN DISTINCT
document("books.xml")//author
                  #[(text()) in ("Krisham","Baker")
RETURN <BooksByAuthor>
 <Author> $a/lastname/text() </Author>
 (FOR $b IN document("books.xml")//book[author=$a]
  RETURN $b/title #[(text()) in ("SECRET") and
                    (text()) in ("WORLD")]#)
</BooksByAuthor> SORTBY(Author))</Result>
```

This example searches for the preferred authors “Krisham” or “Baker”. The `RETURN` statement of QUILT constructs new XML output from the selected nodes. For each author element the corresponding last name is taken as result. The statements containing `FOR` ... `RETURN` select all books of the current author, preferring books with a title containing the words “WORLD” and “SECRET”.

6 Conclusion and Outlook

We proposed a new personalized search technology for e-commerce applications within the XPATH framework for XML. So far XPATH can only express hard selection conditions which are rarely appropriate to express customer or vendor wishes in B2C or B2B. Preference XPATH uses the partial order model of Preference SQL to describe user preferences.

E-commerce applications can benefit a lot from our approach, because Preference XPATH avoids both the infamous "empty result”-effect and the flooding effect

with irrelevant results of many existing search engines or match-making agents. Since we rely on the Pareto-optimality principle, the nearest match according to a user preference is computed. Preference XPATH is a syntactic enhancement of XPATH. Every Preference XPATH expression can be rewritten into an equivalent XPATH expression. Therefore every application using XPATH as part of the underlying implementation can benefit from our solution. We demonstrated the usage of Preference XPATH within XSLT and QUILT, which points a very promising way to personalize mobile content delivery. Since the XML pointer language XPointer uses XPATH, Preference XPATH is available there, too.

Our next steps will cover the entire spectrum of the XML data model. In particular we are investigating the extension of Preference XPATH to set-valued preferences (see [LeKi99] for first results) and to structural preferences. We have to consider performance issues and optimization techniques as well. In terms of applications we will interface Preference XPATH with our speaking meta-search agent COSIMA ([HFEK01]). Finally, a technological comparison of our preference modeling approach with others (e.g. with case-based reasoning CBR) may be interesting. Such a benchmark would have to consider not only issues of search precision and recall and of query performance, but many more like compatibility with industry standards, duplication of data, expenditure for preference modeling, etc.

Acknowledgments

We thank Achim Leubner and Matthias Wagner for helpful discussions, and Software AG for providing the Tamino system.

References

[ABS00] Abiteboul S., Buneman P., Suciu D., Data on the Web, Morgan Kaufmann Publishers, San Francisco, California, 2000.

[BPSM00] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E., Extensible markup language (XML) 1.0, http://www.w3.org/TR/REC-xml, October 2000.

[Clar99] Clark James: XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/1999/REC-xslt-19991116, November 1999.

[ClDe99] Clark J., DeRose S.: XML Path Language (XPath), http://www.w3.org/TR/1999/REC-xpath-19991116, November 1999.

[Cowa00] Cowan J.: XML Information Set, http://www.w3.org/TR/2000/WD-xml-infoset-20000726, July 2000.

[DaPr90] Davey, B.A.; Priestley, H.A.: Introduction to Lattices and Order, Cambridge University Press, April 1990.

[Data99] User Manual Preference SQL 1.3. Database Preference Software GmbH, 1999.

[DFFS98] Deutsch A., Fernandez M., Florescu D., Levy A., Suciu D.: XML-QL: A Query Language for XML, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, August 1998.

[Fuhr00] Fuhr, Norbert: XIRQL An Extension of XQL for Information Retrieval, SIGIR 2000, http://www.haifa.il.ibm.com/sigir00-xml/final-papers/xirql.html .

[HFEK01] Stefan Holland, Stefan Fischer, Thorsten Ehm, Werner Kießling. Gaining Customer Preferences from E-Shopping Log-Files. In proceedings 3rd Conference Information Systems in Finance 2001, Augsburg, Germany.

[KeRa76] Keeney, Ralph L.; Raiffa, Howard: Decision with Multiple Objectives: Preferences and Tradeoffs. Wiley, April 1976.

[KiGü94] Kießling, Werner; Güntzer, Ulrich: Database reasoning – a deductive framework for solving large and complex problems by means of subsumption. In 3rd Workshop on Information Systems and Artificial Intelligence, Volume 777 of LNCS, pages 118-138, Hamburg, Germany, February 1994.

[KKTG95] Köstler, Gerhard; Kießling, Werner; Thöne, Helmut; Güntzer, Ulrich: Fixpoint Iteration with Subsumption in Deductive Databases, Journal of Intelligent Information Systems, Vol. 4, pages 123 – 148, 1995.

[LeKi99] Leubner A., Kießling W: Personalized Nonlinear Ranking Using Full-text Preferences, ACM SIGIR '99, Workshop on Customised Information Delivery, Berkeley, August 1999.

[RCF00] Robie J., Chamberlin D., Florescu D., QUILT: an XML query language http://www.gca.org/papers/xmleurope2000/papers/s08-01.html, XML Europe, June 2000.

[RLS98] Robie J., Lapp J., Schach D.: XML Query Language, http://www.w3.org/TandS/QL/QL98/pp/xql.html, September 1998

[Robi99] Robie Jonathan: The Tree Structure of XML Queries, http://www.w3.org/1999/10/xquery-tree.html, October 1999.

[Soft01] Tamino, Software AG, www.softwareag.com/Tamino.