Pattern Tree-based XOLAP Rollup Operator for XML Complex Hierarchies

Marouane Hachicha
Université de Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France
69676 Bron Cedex, France
Email: marouane.hachicha@univ-lyon2.fr

Jérôme Darmont
Université de Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France
69676 Bron Cedex, France
Email: jerome.darmont@univ-lyon2.fr

***Abstract*—With the rise of XML as a standard for representing business data, XML data warehousing appears as a suitable solution for decision-support applications. In this context, it is necessary to allow OLAP analyses on XML data cubes. Thus, XQuery extensions are needed. To define a formal framework and allow much-needed performance optimizations on analytical queries expressed in XQuery, defining an algebra is desirable. However, XML-OLAP (XOLAP) algebras from the literature still largely rely on the relational model. Hence, we propose in this paper a rollup operator based on a pattern tree in order to handle multidimensional XML data expressed within complex hierarchies.**

I. Introduction

In many institutions, decision-support applications require external data. In this context, the Web is a tremendous data source and Web farming [1] is more and more casual. As a consequence, a new trend toward on-line data warehousing is currently emerging, including approaches such as XML warehousing [2].

The XML language is indeed becoming a standard for representing business data [3]. Moreover, it is particularly adapted for modeling so-called complex data [4] from heterogeneous sources, and particularly the Web. Thus, many studies aim at extending the XQuery language [5] with OLAP (On-Line Analytical Processing)-like queries (grouping, aggregation, etc.) [3], [6], [7]. Such extensions should not only allow classical OLAP analyses, but also take the specificities of XML into account, e.g., ragged hierarchies [3] that would be intricate to handle in a relational environment.

In this context, we are working to propose an OLAP algebra over multidimensional XML data (XML data modeled in multidimensional way). On the long run, we are actually aiming at three objectives: contribute to define a formal framework that does not currently exist in the XOLAP [8] context; support the effort for extending the XQuery language to allow OLAP queries, especially with XML-specific operators; allow query optimization for OLAP XQueries. Native-XML DBMSs (Database Management Systems), though in constant progress, are indeed limited in term of performance and would greatly benefit from automatic query optimization, especially for costly analytical queries.

In this paper, we particularly focus on the first objective. In a previous work, we have expressed classical (structural, set and granularity-related) OLAP operators over multidimensional XML data organized in simple hierarchies [9] with the TAX XML algebra [10]. The next step is now to take XML specifics into account and propose operators for data organized in ragged, complex hierarchies. TAX, as many other XML algebras, is based on pattern trees [11] to model user queries. Further combining TAX operators to process hierarchies with unpredictable structures would require to handle combinations of many pattern trees. On the other hand, a more straightforward way to achieve our goal is to directly work at the pattern tree level and design a single, ad-hoc pattern tree. Hence, we propose a pattern tree model with advanced matching capabilities, including aggregation, grouping and ordering; and illustrate its use through a rollup operator that applies onto complex hierarchies. This marks a first step in defining a full set of pattern tree-based XOLAP operators.

The remainder of this paper is organized as follows. In Section II, we formally define pattern trees and related concepts. In Section III, we survey the pattern trees that are used in XML algebras. In Section IV, we formally define the complex hierarchies we want to handle. In Section V, we introduce our pattern tree-based rollup operator. We finally conclude this paper and discuss research perspectives in Section VI.

II. Background

We define in this section the main concepts that lie behind XML algebras, i.e., XML data trees and subtrees, pattern trees and the operations of matching and embedding.

A. XML Data Trees and Subtrees

A data tree *t* models an XML document or a document fragment. It may be defined as a triple $t = (r, N, E)$, where *N* is the set of nodes, $r \in N$ is the root of *t*, and *E* is the set of edges stitching together couples of nodes $(n_i, n_j) \in N$.

Given an XML data tree $t = (r, N, E)$ and $e \in E$ an edge connecting two nodes (n_i, n_j). $t' = (r', N', E')$ is a subtree of *t iff* the following conditions are satisfied: $N' \subseteq N$; there exists an edge $e' \in E'$ connecting two nodes (n_i', n_j') such that $n_i = n_i'$ and $n_j = n_j'$.

B. Pattern Trees

A pattern tree *pt*, also called tree pattern or tree pattern query (TPQ) [11] is a pair (t, F) where: (1) *t* is a tree (*r*,

N, *E*). An edge may either be a parent-child (*pc* for short, simple edge / in XPath) node relationship or an ancestor-descendant (*ad* for short, double edge // in XPath) node relationship; (2) *F* is a formula that specifies constraints on node values. More explicitly, F is a boolean combination of predicates on node values.

Basically, a pattern tree captures a useful fragment of XPath [12]. But it can also be seen as the translation of a user query formalized in natural language or in an XML query language such as XQuery [13]. Translating an XML query plan into a pattern tree is not a simple operation. Some XQueries are written with complex combinations of XPath and FLWOR expressions, and imply more than one pattern tree. Such queries must be broken up into several patterns trees. Only a single XPath expression can be translated into a single pattern tree. The more a query is difficult, the more its translation in pattern tree(s) is complex [14]. To this aim, starting from patterns to express user queries in a first stage, and optimizing them in a second stage is a very effective solution for XML query optimization.

C. Matching and Embedding

Answers for pattern trees (named witness trees in TAX) are formalized through one or multiple matchings. Matching a pattern tree *pt* into an XML data tree t is a function `f:pt→t` that maps nodes of *pt* to nodes of *t* such that: (1) structural relationships are preserved, i.e., if nodes (x, y) are related in t through a pc node relationship (respectively an *ad* node relationship), their counterparts $(x^{'}, y^{'})$ in pt must be related through a pc node relationship (respectively an *ad* node relationship) too; and (2) formula *F* of *pt* must be satisfied.

Embedding a pattern tree pt into a data tree t is a function $g : pt \rightarrow t$ that maps each node of pt to nodes of t such that structural relationships (*pc* and *ad*) are preserved. The difference between embedding and matching is that embedding maps a pattern tree against a data tree *structure* only, whereas matching maps a pattern tree against a data tree structure *and contents* [15]. In the remainder of this paper, we use the more general term matching when referring to mapping pattern trees against data trees.

D. Example

For comprehensibility, let us consider the XML data tree from Figure 1(a) that represents a collection of books. Root *doc* unites *books* described by their *titles*, *authors*, *editors*, *years* and *summaries*. Data trees nodes are connected by simple edges (/), i.e., *pc* relationships. Books are not necessarily described the same way. For instance, a summary may not be present in all books. Some books can be written by more than one author.

The pattern tree from Figure 1(b) selects book titles, authors, and editors. Moreover, formula F indicates that author must be different from Jill. Matching this pattern tree against the data tree from Figure 1(a) outputs the data tree (or witness tree) from Figure 1(c). Only one book is selected, since the other one ($title =$ "A dummy for a computer") is written by $author =$ "Jill", which contradicts formula F.

Finally, the *ad* relationship $\$1//\$3$ in Figure 1(b)'s pattern tree is correctly taken into account. The book element ($title =$ "A dummy for a computer") is indeed not disqualified because of its structure, but because one of its authors is Jill. It this author was Gill, the book would appear in output.

III. Pattern Trees Used in XML Tree Algebras

The aim of an XML tree algebra is to feature a set of algebraic operators to manipulate and query XML data tree structures. The output of a query formulated over a tree must also be a data tree that respects a tree structure, i.e., a pattern tree.

First XML algebras have appeared in 1999 [16] in conjunction with efforts aiming to define a powerful query language for XML [17]. Note that these XML algebras have appeared before the first specification of XQuery, which is regarded as the most popular XML query language, in 2001 [18].

TAX is one of the most popular XML tree algebras [10]. The TAX pattern tree represents the most basic pattern tree used in an algebraic context. It preserves *pc*-*ad* relationships from the input ordered data tree in output and it satisfies the formula associated to the pattern. The examples from Figure 1(a), (b) and (c) correspond to TAX data, pattern and witness trees, respectively.

Providing more matching options for edges connecting output nodes allows a more efficient extraction of these nodes when matching the relevant pattern tree. An important limitation of the TAX pattern tree comes in case of absence of one node in the subtrees matched with the pattern tree, which prevents them to appear in the result. Generalized Tree Patterns (GTPs) extend classical TAX pattern trees by creating groups of nodes to facilitate their manipulation and by enriching edges to be extracted by the *mandatory/optional* matching option [13].

An option more than a limitation of TAX pattern trees is that a set of similar nodes of the same subtree appears in the resulting tree. For example, a book written by more than one author results from a matching of a pattern containing a single author node. Edges of Annotated Pattern Trees (APTs) [19] solve this problem and present four matching specifications: one to many matches (+), one match only (-), zero to many matches (*) and zero or one match (?).

APTs, like TAX pattern trees and GTPs, preserve the order of nodes from the input XML data in the output (result), whatever the order of nodes in the pattern tree. To avoid this issue, it is necessary to specify node order in the pattern tree. APTs used in the TLC (Tree Logical Classes) Select and Join operators [19] are extended with an order parameter (*ord*) [20].

We recapitulate in Table I the characteristics of all pattern trees studied in this section.

IV. Complex Hierarchies

In this section, we define what we term complex hierarchies. To this aim, we first formalize the definitions of data warehousing concepts.

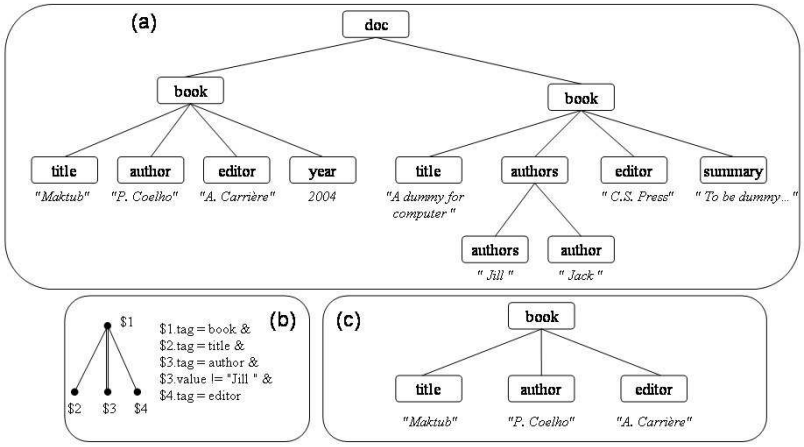

Fig. 1. XML data (a), pattern (b) and witness trees (c)

TABLE I
COMPARISON OF PATTERN TREES USED IN XML TREE ALGEBRAS

	Matching features	**Reordering**	**Hierarchies**
TAX PT [10]	Basic	No	No
GTP [13]	Mandatory/optional edges	No	No
APT [19]	Edge cardinality	No	No
Ordered APT [20]	Order specification	Yes	No

A. Data Warehouses

1) Data Warehouse: A data warehouse W modeled w.r.t. a snowflake schema (i.e., with dimension hierarchies) is defined as $W = (F, \mathcal{D})$, where:

- F is a set of facts to observe;
- $\mathcal{D}$ is a set of dimensions or analysis axes. Let $d = |\mathcal{D}|$.

2) Dimension and Hierarchy: $\forall i \in [1, d]$, a dimension $D_i \in \mathcal{D}$ is defined as a hierarchy made up of a set of n_i levels: $D_i = \{H_{ij}\}_{j=1,n_i}$. By convention, we denote H_{i1} as the lowest granularity level.

$\forall j \in [1, n_i]$, a hierarchy level H_{ij} is defined in intention as $H_{ij} = (ID_{ij}, \{A_{ijk}\}_{k=1,a_{ij}}, R_{ij})$, where:

- ID_{ij} is the identifier attribute of H_{ij};
- $\{A_{ijk}\}$ is a set of a_{ij} so-called member attributes of H_{ij};
- R_{ij} is an attribute that references a hierarchy level at a higher granularity than that of H_{ij} (notion of *rollup*).

Let $dom()$ be a function that associates to any attribute its definition domain. Let $h_{ij} = |H_{ij}|$. $\forall l \in [1, h_{ij}]$, instances of H_{ij} are tuples under form $(\sigma_{ijl}, \{\alpha_{ijkl}\}_{k=1,a_{ij}}, \rho_{ijl})$, where:

- $\sigma_{ijl} \in dom(ID_{ij})$;
- $\alpha_{ijkl} \in dom(A_{ijk})$ $\forall k \in [1, a_{ij}]$;
- $\rho_{ijl} \in dom(ID_{ij'})$ with $j' \in [1, n_i]$.

3) Fact: F is defined in intention as $F = (\{\Delta_i\}_{i=1,d}, \{M_j\}_{j=1,m})$, where:

- $\{\Delta_i\}$ is a set of d attributes that reference instances of hierarchy levels H_{i1} of each dimension $D_i \in \mathcal{D}$;
- $\{M_j\}$ is a set of m measure (or indicator) attributes that characterize facts.

Let $f = |F|$. $\forall k \in [1, f]$, instances of F are tuples under form $(\{\delta_{ik}\}_{i=1,d}, \{\mu_{jk}\}_{j=1,m})$, where:

- $\delta_{ik} \in dom(ID_{i1})$ $\forall i \in [1, d]$;
- $\mu_{jk} \in dom(M_j)$ $\forall j \in [1, m]$.

B. Complex Hierarchies

A dimension hierarchy D_i is termed complex if it is both non-strict and non-covering.

1) Non-Strict Hierarchy: A hierarchy is non-strict [21]–[23] or multiple-arc [24] when attribute R_{ij} is multivalued. In other terms, from a conceptual point of view, a hierarchy is non-strict if the relationship between two hierarchical levels is many-to-many instead of one-to-many. For example, in a dimension describing products, a product may belong to several categories instead of just one.

Similarly, a many-to-many relationship between facts and dimension instances may exist [24]. For instance, in a sale data warehouse, a fact may be related to a combination of promotional offers rather than just one. Formally, here, attributes Δ_i ($\forall i \in [1, d]$) may be multivalued.

2) Non-Covering Hierarchy: A hierarchy is non-covering [21]–[23] or ragged [24] if attribute R_{ij} allows linking a hierarchy level H_{ij} to another hierarchy level $H_{ij'}$ by "skipping" one or more intermediary levels, i.e., $\rho_{ij} = \sigma_{ij'}$ and $\exists H_{ij''} \in D_i$ / $\rho_{ij''} = \sigma_{ij'}$. This occurs, for instance, if in a dimension describing stores, the store-city-region-country hierarchy allows a store to be located in a given region without being related to a city (stores in rural areas).

Similarly, facts may be described at heterogeneous granularity levels. For example, still in our sale data warehouse,

sale volume may be known at the store level in one part of the world (e.g., Europe), but only at a more aggregate level (e.g., country) in other geographical areas. This means that $\forall i \in [1, d]$, $\delta_i \in dom(ID_{ij})$ with $j \in [1, n_i]$ (constraint $j = 1$ is forsaken).

3) Notes:

- The notion of ragged hierarchy has different meanings in the literature. For example, Beyer et al. define it as a hierarchy that is both non-strict and non-covering [3], while Rizzi defines it as non-covering only [24]. This is why we prefer and define the new terms of complex hierarchy. Malinowski and Zimànyi use similar switchable terms: generalized hierarchy [22] and complex generalized hierarchy [25]. However, these hierarchies include non-covering hierarchies, but not non-strict hierarchies.
- Taking complex hierarchies into account involves important summarizability issues [26]. However, taking them into account in an XOLAP context is relevant (real cases do exist) and necessary. Research devoted to normalizing conceptual models with summarizability problems [27] could be exploited for this sake.

C. Example

Let us expand the example from Figure 1 with Figure 2, where book sales are described by titles, categories and sale prices.

Each category is associated to a hierarchy level labeled C1 to C3, from the most detailed to the most general. Categories form a complex hierarchy (Figure 3). A category includes more than one book and a book is described by more than one category, thus making this hierarchy non-strict. Moreover, two books (title = "SQL" and "Manag. S.I") are described by complete hierarchies of categories (C3/C2/C1). While book entitled "PHP 5" is described by an incomplete hierarchy of categories (C3[SQL]//C1[Software]). Book entitled "SQL" is also described by two hierarchies (one complete and one incomplete). Hence, the hierarchy of categories is non-covering. Being also non-strict, it is thus complex.

V. Pattern Tree-based Rollup Operator

A. Motivation

In a previous work, we expressed classical OLAP operators with a succession of TAX operators of selection, grouping, join, aggregation and node update [9]. Multidimensional XML data introduced in this work were described by simple (strict with no overlap between levels) hierarchies.

The problem with complex hierarchies is that, when aggregating data, we handle facts described with respect to various levels of granularity. It is then be difficult, in this case, to aggregate measures. A second issue is that some data may not be taken into account because of missing levels in non-covering hierarchies (e.g., book entitled "PHP 5").

Choosing to extend the pattern tree of one or more TAX operators used to express the rollup operator (selection, grouping, join, aggregation and node update) from [9] is probably a good but not generic solution, since multiple possibilities of extension are possible. For example, we can employ a pattern tree adapted to complex hierarchies in the input of the TAX selection operator, but also join initial data with complex hierarchies using an adapted pattern tree. Furthermore, the TAX pattern tree and its extensions do not take hierarchies into account (Table I). Thus, we had to handle them in a separate representation [9].

Since we aim to define a formal framework for XOLAP, rather than extending one or multiple TAX operators, we propose a new rollup operator based on a pattern tree independent from TAX and respecting the definition of rollup [28].

In the following two sections, we detail our XOLAP rollup operator by presenting the proposed pattern tree and the algorithm allowing to aggregate multidimensional XML data expressed in complex hierarchies using this pattern. This rollup operator inputs a multidimensional XML data tree and outputs a second a multidimensional XML data tree where measures are aggregated. It is based on an algorithm allowing to match a pattern tree against a multidimensional XML data tree.

B. Pattern Tree for Rollup

We detail here the structure of our pattern tree (Figure 4(a)). The graph on the left-hand side represents the pattern tree, while the right-hand side of the figure features formula F.

Parent-child (*pc*) relationships are represented with single edges (/); ancestor-descendant (*ad*) relationships are represented with double edges (//); nodes with a white background ($1, $4, $6 and $7) do not appear in the result, unlike nodes with a black background; nodes connected to their parent by dotted edges ($4) are not used in the matching process since they do not have an equivalent in the data tree, unlike nodes connected to their parent nodes by solid edges.

In Formula F, $0 is the root of the fact document. $1 is a fact described by its dimensions and measures. $2 is the root of the complex hierarchy. $3 computes aggregation from measure $7 of each fact $1. $4 counts the number of matched facts. It is useful for aggregation operations such as average. $5 is the most detailed element of the hierarchy, child (direct descendant) of $2 in the data tree. $6 is any descendant of $5. $6 is used to browse through the hierarchy in the matched data tree.

C. Rollup Algorithm

Our rollup algorithm (Algoritm V-C) is based on the pattern tree from Figure 4(a). For each fact $1, it checks whether the highest hierarchical element (direct child $5 of the hierarchy root $2) corresponds to the input hierarchical element *Hel-agg*. If so, aggregation is computed from measure $7, $3 and $4 are updated (they are input-output parameters of function *AGGREGATE*), and the algorithm steps to the next fact. Otherwise, the algorithm continues to scan through the hierarchy $6 until finding the hierarchical element to be aggregated.

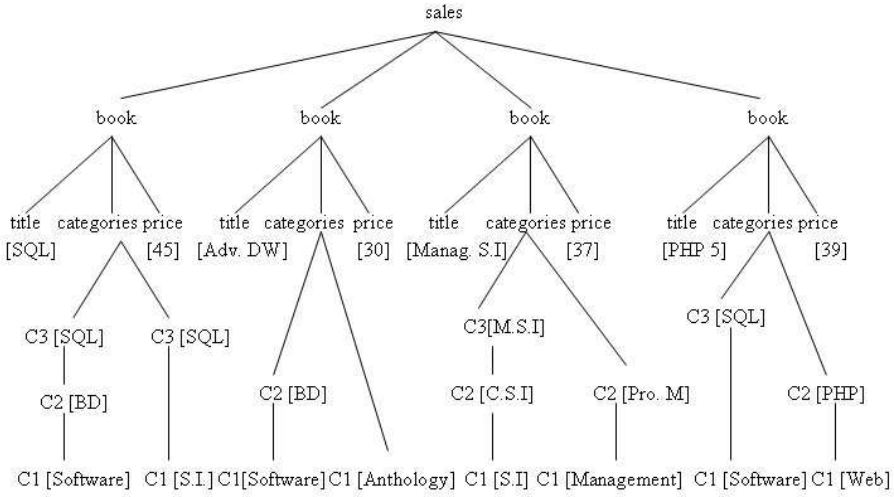

Fig. 2. Book sales expressed in complex hierarchy

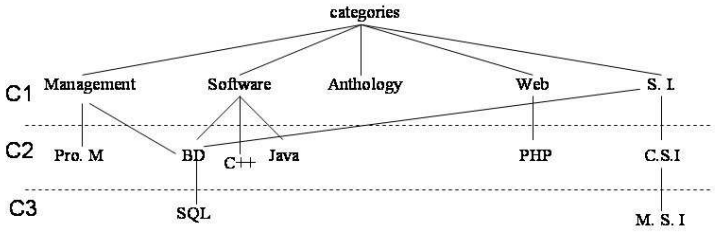

Fig. 3. Category complex hierarchy

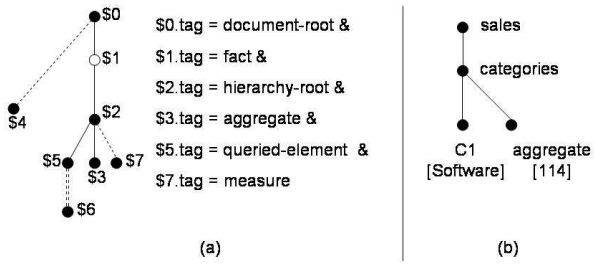

Fig. 4. Pattern tree for rollup (a) and witness tree (result) (b)

Algorithm 1 Rollup

```
Input: DT // Fact data tree
       H-el-agg // Hierarchical element to be aggregated
$3 ← 0
$4 ← 0
Stop ← FALSE
for all $1 in DT do
  while exists child $5 of $2 and not Stop do
    if $5.value = H-el-agg then
      AGGREGATE ($3, $4, $7)
      Stop ← TRUE
    else if $6.value = H-el-agg then
      AGGREGATE ($3, $4, $7)
      Stop ← TRUE
    end if
  end while
end for
```

D. Example

Let us consider query Q: “compute total of book sales for category Software”. Category Software means C1[Software] or any descendant category. The aggregate function used in *AGGREGATE* (Algorithm V-C) in this case is sum. When matching the pattern tree from Figure 4(a) against the data tree from Figure 2, pattern node $0 takes the value “sales” and $1 takes the value “book”. $2 must be equal to “categories” here. For each $2 = “categories” of every “book”, we check whether the value of $5 (the most detailed category of the book) is equal to the looked for hierarchical value (*H-el-agg* = Software). If it is true, we step to the next book (next $1). Otherwise, we continue to check whether one descendant of $5 ($6, an ancestor category) corresponds to category

Software. In case a book of category "Software" is found, $4 is incremented and $3 is incremented by measure value $7 of this book (to compute sale total). When Software is not found after searching for all categories of the current $1, we step to the next fact ($1 = book) and continue searching. After matching all the data tree with the pattern tree, the aggregate is computed, $5 takes the value of the searched category Software and $3 the computed total book sale, Aggregate, as shown in Figure 4(b), which represents the witness tree answering our initial query.

VI. Conclusion and Perspectives

In this paper, we propose to the best of our knowledge the first pattern tree for multidimensional data since the introduction of pattern trees in XML approaches [11]. Though it is simple, this pattern tree permits to aggregate data expressed in complex hierarchies, no matter their structure. We thus progressed toward the definition of a formal framework for XOLAP. It is important that XML multidimensional data are processed natively, which allows taking into account XML specifics such as complex hierarchies, which are intricate to handle in relational systems.

The perspectives of this work are twofold. We aim, in a first step, to adapt the principle of the pattern tree we introduce in this paper to other XOLAP operators (cube, drill down, etc.) in order to complete our algebra. More matching options (e.g., optional edges or edge cardinalities) might have to be added to the pattern tree model at this stage. Moreover, XOLAP operators performing aggregation raise summarizability problems. We aim to present solutions to detect and correct them in the algorithms (and patterns) associated to the different operators.

In a second stage, we plan to implement our algebra (as a proof of concept) and optimize its performance. Pattern tree-based XQuery optimization approaches may help optimize our operators under their physical form. For instance, we could use minimization techniques. Minimizing a pattern tree pt consists in constructing a minimal pattern that is equivalent to pt while bearing the minimum possible size [11].

References

[1] R. D. Hackathorn, *Web farming for the data warehouse*. San Francisco, USA: Morgan Kaufmann, 1999.

[2] J. Zhang, W. Wang, H. Liu, and S. Zhang, "X-Warehouse: Building Query Pattern-driven Data," in *14th international conference on World Wide Web (WWW 05), Chiba, Japan*, 2005, pp. 896–897.

[3] K. S. Beyer, D. D. Chamberlin, L. S. Colby, F. Özcan, H. Pirahesh, and Y. Xu, "Extending XQuery for Analytics," in *ACM SIGMOD 24th International Conference on Management of Data (SIGMOD 05), Baltimore, USA*, 2005, pp. 503–514.

[4] J. Darmont, O. Boussaïd, J.-C. Ralaivao, and K. Aouiche, "An Architecture Framework for Complex Data Warehouses," in *7th International Conference on Enterprise Information Systems (ICEIS 05), Miami, USA*, 2005, pp. 370–373.

[5] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language," http://www.w3.org/TR/xquery/, World Wide Web Consortium (W3C), 2007.

[6] V. R. Borkar and M. J. Carey, "Extending XQuery for Grouping, Duplicate Elimination, and Outer Joins," in *XML 2004 Conference, Washington DC, USA*.

[7] M. Kay, "Positional Grouping in XQuery," in *3rd International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 06), Chicago, USA*.

[8] H. Wang, J. Li, Z. He, and H. Gao, "OLAP for XML Data," in *1st International Conference on Computer and Information Technology (CIT 2005), Shanghai, China*, 2005, pp. 233–237.

[9] M. Hachicha, H. Mahboubi, and J. Darmont, "Expressing OLAP operators with the TAX XML algebra," in *3rd International Workshop on Database Technologies for Handling XML Information on the Web (EDBT-DataX 08), Nantes, France*.

[10] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson, "TAX: A Tree Algebra for XML," in *8th International Workshop on Database Programming Languages (DBPL 01), Frascati, Italy*, ser. LNCS, vol. 2397, 2001, pp. 149–164.

[11] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava, "Minimization of Tree Pattern Queries," in *ACM SIGMOD 20th International Conference on Management of Data (SIGMOD 01), Santa Barbara, USA*, 2001, pp. 497–508.

[12] S. Flesca, F. Furfaro, and E. Masciari, "On the minimization of Xpath queries," in *29th International Conference on Very Large Data Bases (VLDB 03), Berlin, Germany*, 2003, pp. 153–164.

[13] Z. Chen, H. V. Jagadish, L. V. S. Lakshmanan, and S. Paparizos, "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery," in *29th International Conference on Very Large Data Bases (VLDB 03), Berlin, Germany*, 2003, pp. 237–248.

[14] P. Michiels, G. A. Mihaila, and J. Siméon, "Put a Tree Pattern in Your Algebra," in *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey*, 2007, pp. 246–255.

[15] L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. J. Zhao, "On Testing Satisfiability of Tree Pattern Queries," in *30th International Conference on Very Large Data Bases (VLDB 04), Toronto, Canada*, 2004, pp. 120–131.

[16] D. Beech, A. Malhotra, and M. Rys, "A formal data model and algebra for XML," W3C XML Query Working Group Note, Tech. Rep., 1999.

[17] D. D. Chamberlin, J. Robie, and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources," in *3rd International Workshop on The World Wide Web and Databases (WebDB 00), Dallas, USA*.

[18] D. D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu, "XQuery 1.0: An XML Query Language," http://www.w3.org/TR/xquery/, World Wide Web Consortium (W3C), 2001.

[19] S. Paparizos, Y. Wu, L. V. S. Lakshmanan, and H. V. Jagadish, "Tree Logical Classes for Efficient Evaluation of XQuery," in *ACM SIGMOD 23rd International Conference on Management of Data (SIGMOD 04), Paris, France*, 2004, pp. 71–82.

[20] S. Paparizos and H. V. Jagadish, "Pattern Tree Algebras: Sets or Sequences?" in *31st International Conference on Very Large Data Bases (VLDB 05), Trondheim, Norway*, 2005, pp. 349–360.

[21] A. Abellò, J. Samos, and F. Saltor, "YAM^2: a multidimensional conceptual model extending UML," *Information Systems*, vol. 31, no. 6, pp. 541–567, 2006.

[22] E. Malinowski and E. Zimànyi, "Hierarchies in a multidimensional model: from conceptual modeling to logical representation," *Data & Knowledge Engineering*, vol. 59, no. 2, pp. 348–377, 2006.

[23] R. Torlone, "Conceptual Multidimensional Models," in *Multidimensional Databases: Problems and Solutions*, E. Maurizio Rafanelli, Ed. Hershey, USA: IDEA Group Publishing, pp. 69–90.

[24] S. Rizzi, "Conceptual Modeling Solutions for the Data Warehouse," in *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, R. Wrembel and E. Christian Koncilia, Eds. Hershey, USA: IRM Press, pp. 1–26.

[25] E. Malinowski and E. Zimànyi, *Advanced Data Warehouse Design*. Berlin, Heidelberg, Germany: Springer, 2008.

[26] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo, "A survey on summarizability issues in multidimensional modeling," *Data & Knowledge Engineering*, vol. 68, no. 12, pp. 1452–1469, 2009.

[27] ——, "Solving summarizability problems in fact-dimension relationships for multidimensional models," in *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP 08), Napa Valley, USA*, 2008, pp. 57–64.

[28] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *SIGMOD Record*, vol. 26, no. 1, pp. 65–74, 1997.