TGV: a Tree Graph View for Modeling untyped XQuery

Nicolas TRAVERS[1], Tuyêt Trâm DANG NGOC[2], and Tianxiao LIU[3]

[1] PRiSM Laboratory - University of Versailles, France.
Nicolas.Travers@prism.uvsq.fr
[2] ETIS Laboratory - University of Cergy-Pontoise, France.
Tuyet-Tram.Dang-Ngoc@u-cergy.fr
[3] ETIS Laboratory - University of Cergy-Pontoise & XCalia S.A, France.
Tianxiao.Liu@u-cergy.fr

Abstract. Tree Pattern Queries [7, 6] are now well admitted for modeling parts of XML Queries. Actual works only focus on a small subpart of XQuery specifications and are not well adapted for evaluation in a distributed heterogeneous environment.
In this paper, we propose the TGV (Tree Graph View) model for XQuery processing. The TGV model extends the Tree Pattern representation in order to make it intuitive, has support for full untyped-XQuery queries, and for optimization and evaluation. Several types of Tree Pattern are manipulated to handle all XQuery requirements. Links between Tree Patterns are called hyperlinks in order to apply transformations on results. The TGV[4] has been implemented in a mediator system called XLive.

Keywords: XQuery evaluation, TGV, Extensible optimization, Cost model

1 Introduction

XQuery [9] has proved to be an expressive and powerful query language to query XML data both on structure and content, and to make transformation on data. In addition, its query functionalities come from both the database community (filtering, join, selection, aggregation), and the text community (supporting and defining function as text search). However, the complexity of the XQuery language makes its evaluation very difficult. To alleviate this problem, most of the systems support only a limited subset of the XQuery language.

XQuery expressions require a logical model to be manipulated, optimized and then evaluated. [1] introduced the TPQ model that expresses a single *FWR* query by a Pattern Tree and a formula. Then, [2] proposes GTPs that generalizes TPQs with several Pattern Trees, the formula contains all the operations. The representation is quite intuitive and acts as a template for the data source.

[4] The XLive system and TGV is supported by the ACI Semweb project.
TGV annotations and cost models are supported by the ANR PADAWAN project.

However, GTPs do not capture well all the expressiveness of XQuery, cannot handle mediation problems, and do not support extensible optimization.

We design a model called TGV which provides the following features: (a) It integrates the whole functionalities of XQuery (collection, XPath, predicate, aggregate, conditional part, etc.) (b) It uses an intuitive representation that provides a global visualization of the request in a mediation context. (c) It provides a support for optimization and a support for data evaluation.

In this paper we describe the TGV model for evaluating XQuery on heterogeneous distributed sources. This article is organized as follows. The next section introduces the TGV structure that we had defined for modeling XQuery in a practical way. Finally section 3 concludes with the TGV framework.

2 XQuery Modeling

XQuery modeling is a difficult goal since the language provides lots of functionalities. And it is all the more difficult as it needs to match mediation requirements (data localization on sources, heterogeneous sources capabilities).

Tree Pattern matching becomes usual in XQuery modeling, trees contain nodes and links, a formula constraints the tree pattern on tags, attributes and contents. Since GTP, it contains joins, nesting, aggregates and optionality.

However, GTP does not handle distributed queries requirements. In fact, data sources are not included, nor XML result constructor, nor views and query on views modeling, nor *Let* and functions and tags, relations and constraints are embedded in a boolean formula difficult to read. Moreover, there is no support for additional information useful for optimization. Thus, this model requires some extensions and adaptations to be the core of a distributed query-processing algorithm in a mediator. We propose the TGV (Tree Graph View) model.

Let see all characteristics of the TGV model. First, we introduce *TreePatterns* which are the XML document filters, and specific structures adapted to XQuery requirements. Then, *Constraints* are added to this model to integrate general filters, which can be attached to any type of the model. To complete this model, *Hyperlinks* are introduced to link together preceding structures. A Tree Graph View is composed of all this structure to model a complete XQuery query.

Tree Pattern A *Tree Pattern* is a tree with different tags an XML document must match with. This template is a set of *XPaths* extracted from the XQuery query. *TreePatterns* are composed of *Nodes* from a label, *NodeLinks* that represent axis between *Nodes* (child, descendant, etc.), and a mandatory/optional status. A Pattern Tree is illustrated on Figure 1. Specific Tree Patterns are integrated to model each characteristic of XQuery illustrated in figure 2:

- A *Source Tree Pattern* (STP) is defined by a targeted document and a root path. It corresponds to a `for` declaration on a targeted XML document with a specific root path, that defines the set of trees to work with.

– An *Intermediate Tree Pattern* (ITP) specializes a previous TreePattern on a specific Node. It corresponds to a `for` declaration with a new path from a previous variable that specializes an element by creating a new set of trees. Thus, it creates a Tree Pattern that defines a new domain.
– A *Return Tree Pattern* (RTP) defines the result construction of an XML document. It corresponds to the `return` clause of an XQuery query, which builds the main XML resulting document. Nodes is identified by tags, attributes with a "@", quoted texts, and required XPaths.
– An *Aggregate Tree Pattern* (ATP) builds a temporary result set. It corresponds to a `let` clause that defines a treatment on a set of trees. By canonization rules, nested queries and aggregate functions are defined in those clauses, so they build a temporary result set.

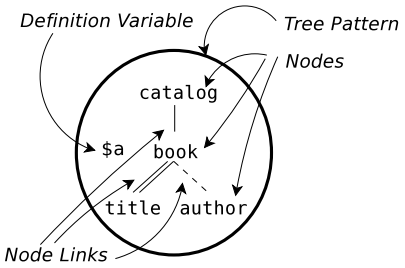

Fig. 1. Example of a Node and of a NodeLink in a TreePattern

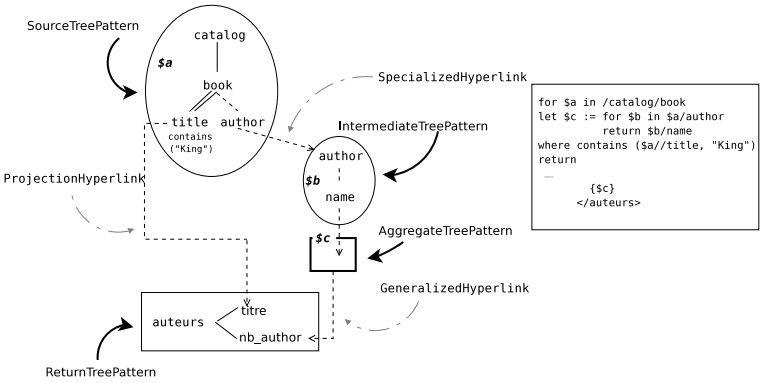

Fig. 2. Four types of Tree Patterns and three types of Hyperlinks

Constraints In XQuery queries, constraints are declared on XPaths to prune set of trees. This constrains may be a value predicate, function or different types of joins. Thus, we introduce the type `Constraint` for this purpose.

A *Constraint* is a restriction of the feasible solutions on sets of trees. It can be applied on *Nodes*, *Tree Patterns*, *Hyperlinks*, *Constraints* or *Constants*. It appears as *Predicates* or *Functions*:

– *Predicates* are constraints with a comparison operator between two element types. Linked types can be constants, nodes, tree patterns, hyperlinks or other constraints in order to compose constraints.
– *Functions* are constraints with a name and a set of links to different element types. The function name defines the type of operation to treat. Linked types can be constants, nodes, tree patterns, hyperlinks or other constraints for function composition.

Constraint representation depends of the linked element type. For a node, we put the constraint under the tag as we can see on figure 3. For a tree pattern, it is represented above it, as we saw the *count* function in figure 2 on the *AggregateTreePattern*. For hyperlinks, it depends of its type, as we will see on *JoinHyperlinks* in figure 3 a link between two nodes is annotated with a equality constraint. For constraint composition, we compose naturally at the position of the linked element (node, tree pattern and hyperlink).

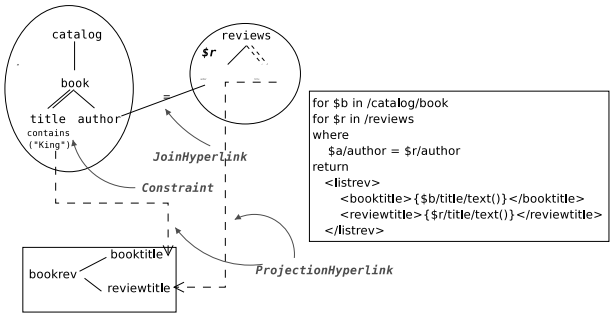

Fig. 3. Example of node constraint

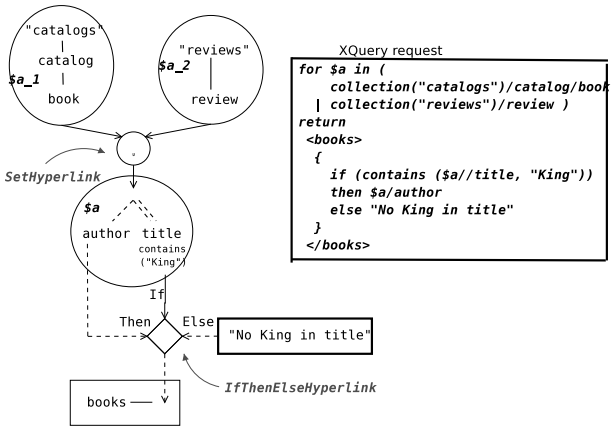

Fig. 4. Examples of Hyperlinks: A SetHyperLink (union) and a IfThenElseHyperlink

Hyperlinks *Hyperlinks* (see figure 3 and 4) have been defined to represent additional relations between elements of the Tree Graph View:

- *Hyperlinks* link elements in *Tree Graph Views*. It represents associations by *Association Hyperlinks* or transformations by *Directional Hyperlinks*.
- *Association Hyperlinks* are *Hyperlinks* that connects two elements of the same type to represent a specific association, in order to filter results by verifying this association. There are two types of *Association Hyperlink*:
  - *Join Hyperlinks* are associations between two *Nodes* under *Constraint* pruning non relevant trees on constraints (values or order operator).
  - *Constraint Hyperlinks* are associations between *Constraints* with a Boolean connector. It forms a tree, connected to a *ReturnTreePattern* in order to keep constraints declaration level. Relevant trees must verify the connected tree of constraints, at a given declaration level.
- *Directional Hyperlinks* are injected transformations between elements. It specifies a transformation from a set of elements to a single one. There are four types of *Directional Hyperlinks*:
  - *Projection Hyperlinks* are *Node* to *Node Directional Hyperlink* representing a value projection of the given node. It can be an optional hyperlink.
  - *Specialized Hyperlinks* are *Node* to *Tree Pattern Directional Hyperlink*. It contains a mandatory or optional status. It represents the specialization of a Node, by specifying a new TreePattern which root is the given node.
  - *Generalized Hyperlinks* are *Tree Pattern* to *Node Directional Hyperlink*. It contains a mandatory or optional status. It represents a TreePattern generalization result set, which result is projected into the given node.
  - *Set Hyperlinks* are set of *Tree Patterns* to *Node* under *Constraint Directional Hyperlink*. It represents a set operation (Union, Intersect or Difference) between few TreePatterns projected on a single Node.
  - *IfThenElse Hyperlinks* are set of *Elements* to *Node* under *Constraint Directional Hyperlink*. Elements can be a Node or an *AggregateTreePattern*, and the constraint is a Predicate or a Function. It represents a conditional expression which result is deduced by the constraint status.

Functions *Functions* take some parameters and give a single element in return. Into our model, we will treat only parameters with *element (), boolean and number* types. A function is represented by a TGV, and its parameters by an *Aggregate Tree Pattern* with its function name. Variables are linked to elements by *Projection Hyperlins*.

Tree Graph Views A *Tree Graph View* (TGV) is a representation of an XQuery query containing TreePatterns, Constraints and Hyperlinks. Input of the TGV is given by *SourceTreePatterns*, the output is defined by the *ReturnTreePattern* (not a AggregateTreePattern by inheritance).

Figures 1, 2, 3 and 4 are TGV examples. For more descriptions, see [3, 8].

Canonical XQuery to TGV Each queries in a canonical form can be translated to a tgv representation. All characteristics of XQuery queries correspond to an element in the tgv model. *For* clauses to STP and ITP, *where* clauses to *constraints*, *Constraint Hyperlinks* and *Join Hyperlinks*, *return* clauses to *ATP* and *RTP*, *let* clauses to *ATP*, and set and conditional operations to *Directional Hyperlinks*. All canonical XQuery queries can be translated in a TGV.

3 Conclusion

XQuery is an XML querying language that provides a rich expressiveness. By this way, an efficient query processing model is all the more difficult. In this paper, we describe our TGV model composed of Tree Patterns which are filters on XML documents. Thanks to this model, we are able to optimize TGV with transformation rules. Those rules rely on a mapping of *Rule Patterns* on a TGV (as a TGV on XML documents). In order to take into account physical information coming from the system, a generic annotation framework is designed on TGV. This annotation framework allows us to describe any type of information on TGVs (cost model, sources and traitments localization, evaluation algorithms, etc.). The cost model is annotated on TGV in order to estimate its execution cost. It allows the optimizer to choose an optimal TGV to evaluate the query. More information can be found in [8].

The whole XQuery evaluation process is implemented in the mediator XLive [4]. All XQuery queries of the W3C use-cases [5] except the typed use-cases (STRONG) are evaluated correctly by our system, using *Tree Graph Views*.

As the TGV model is not specifically bound to a specific language (first designed for XQuery), it can be applied to any untyped queries in any language (SQL, OQL, OEM-QL, etc.) on structured or semi-structured data.

References

1. S. Amer-Yahia, S. Cho, Laks V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD*, 2001.
2. Z. Chen, HV Jagadish, L. VS Laksmanan, and S. Paparizos. From Tree Patterns to Generalized Tree Patterns: On efficient Evaluation of XQuery. In *VLDB*, 2003.
3. T.T. Dang-Ngoc and G. Gardarin. Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*, 2003.
4. T.T. Dang-Ngoc, C. Jamard, and N. Travers. XLive: An XML Light Integration Virtual Engine. In *Proc. of BDA*, 2005.
5. D.Chamberlin, P.Fankhauser, D.Florescu, M.Marchiori, and J.Robie. XML Query Use Cases, september 2005. W3C. http://www.w3.org/TR/xquery-use-cases.
6. HV Jagadish, LVS Lakshmanan, D. Srivastava, and K. Thompson. TAX: A Tree Algebra for XML. In *DBPL*, pages 149–164, 2001.
7. A. Sihem, C. SungRan, V. S. Laks Lakshmanan, and D. Srivastava. Tree Pattern Query Minimization. *VLDB Journal*, 11(4)::315–331, 2002.
8. N. Travers. *Optimization Extensible dans un Médiateur de Données XML*. PhD thesis, University of Versailles, December 2006.
9. W3C. An XML Query Language (XQuery 1.0), 2005.