Now it's Obvious to The Eye—Visually Explaining XQuery Evaluation in a Native XML Database Management System

Andreas M. Weiner, Christian Mathis, Theo Härder, and Caesar Ralf Franz Hoppen
Databases and Information Systems Group
Department of Computer Science
University of Kaiserslautern
67653 Kaiserslautern, Germany
{weiner, mathis, haerder, hoppen}@cs.uni-kl.de

Abstract: As the evaluation of XQuery expressions in native XML database management systems is a complex task and offers several degrees of freedom, we propose a visual explanation tool—providing an easily understandable graphical representation of XQuery—for tracking the XQuery evaluation process from head to toe.

1 Introduction

In recent years, XML gained a lot of attention as a means for exchanging structured and semi-structured data. Native XML database management systems (XDBMSs) are a promising approach for storing and managing such documents in a transactional way. Having a closer look at XQuery—the dominant query language for XML—reveals that it is an extremely powerful, but at the same time, a very complex query language. In this work, we present the *XPlain* tool for visually explaining the evaluation of XQuery expressions in *XTC (XML Transaction Coordinator)* [HH07]—our prototype of a native XDBMS. Using our tool, we can track the complete XQuery evaluation process beginning at the translation of the query into an internal representation, ranging over the application of several rules for algebraic optimization, and ending in a query execution plan which is executed using the query evaluation engine of XTC.

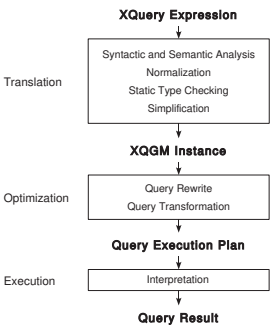

Figure 1: The XTC query evaluation process

We are not aware of any tool that allows to follow all stages of the XQuery evaluation process from the beginning to the end in a catchy way that is even easy to understand for XQuery novices and non-database experts. Our visual explanation tool supports different types of users in improving their work: (1) Developers of XML query optimizers can immediately see the impact of rewrite and optimization rules on subsequent query graphs, (2) Lecturers benefit from our self-explanatory graphical query representation and can use it to teach undergraduate XQuery classes, and (3) Database administrators can focus solely on the query execution plan and speed-up query evaluation by creating new indexes or by activating or deactivating different rewrite or optimization rules.

2 Related Work

Compared to the work of Rittinger et al. [RTG07], which empowers a relational query optimizer to evaluate XQuery expressions and visualizes only QEPs, we are able to illustrate every step in the query evaluation process. Furthermore, by sticking to a rule-based approach, we can re-configure our query optimizer even at runtime.

3 Architectural Issues

Figure 1 shows the three stages of the XTC query evaluation process: *translation*, *optimization*, and *execution*. During the translation stage, an XQuery statement is checked for syntactical and semantical correctness. These checks are followed by a normalization phase, where semantically equivalent queries are mapped to a common normal form expression according to the formal semantics of XQuery.

Before the normal form expression is mapped to the so-called *XML Query Graph Model (XQGM)* [WMH08][1], we perform static type checking and apply several simplification rules to remove redundant parts of the query. For example, Figure 2 shows a graphical representation of the XPath path expression doc("auction.xml")//site//mail which was exported using XPlain. Because an XQGM instance is equivalent to a logical algebra expression, it allows to perform algebraic optimization. Based on an XQGM graph provided as input for the optimization stage, several rewrite rules, e. g., query unnesting [Mat07] and join fusion [WMH08] are applied, resulting in a semantically equivalent structure which can be evaluated more efficiently than the initial one. In the query transformation step, a rewritten XQGM instance is mapped to a *Query Execution Plan (QEP)* (physical algebra expression). Finally, the QEP is executed by direct interpretation using the well-known open-next-close protocol [Gra93].

We developed our query optimizer following a strictly extensible rule-based approach, i. e., every modification of an XQGM instance (e. g., by algebraic rewrite) is specified by a rule consisting of a pattern and an action part. Patterns are identified by our generic pattern matching engine and the actions are applied by a transformation engine. Consequently, we can (1) easily extend our system by adding new rules and (2) switch on and off specific simplification, rewrite, and logical-to-physical mapping

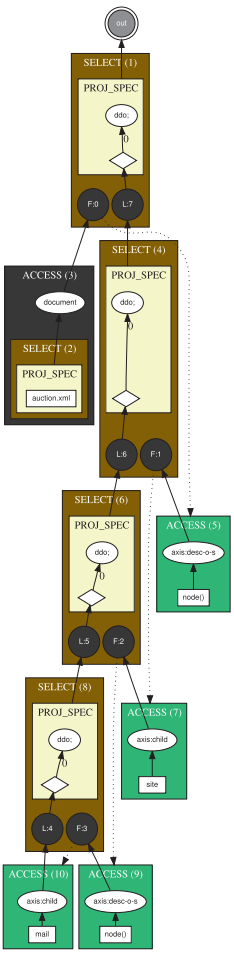

Figure 2: A sample XQGM instance

[1] Note, the XQGM is an extended version of Starburst's well-known *Query Graph Model (QGM)* [PHH92] which we made to measure for the XQuery language.

Figure 3: The XPlain GUI

rules according to our needs. Thus, we can play the role of a query optimizer and immediately see the impact of different optimization strategies even at runtime.

Whenever an action is performed by the transformation engine, a textual representation of the resulting XQGM graph—a so-called *dot graph*—is generated reflecting all changes performed. By doing so, we get a complete history of all transformations applied to the initial XQGM graph as well as a graphical representation of the final QEP.

The XPlain tool—implemented using Java 1.6—provides a sophisticated Swing-based GUI and connects to the XTC server as a client using Java RMI. It receives the query result, statistics on each phase of the query evaluation process, and all dot plans generated. Using the *GraphViz* visualization software [EGKW03]—a powerful framework for layouting huge graphs—all dot plans are converted into Scalable Vector Graphic (SVG) instances which are rendered in the XPlain GUI using the *Apache Batik SVG Toolkit*[2].

Figure 3 shows the XPlain GUI. At the left-hand side, you can see a list of all documents currently stored on the server (top-most box), the path synopsis—a kind of dynamic schema allowing to create XPath path expressions just by clicking on the node names (box in the middle), and meta data on currently available indexes for each document (bottom line). The main panel displays a rendered XQGM graph corresponding to the query entered in the text box atop of it. At the top-most right side, you can select a query from predefined query sets[3]. Furthermore, the right side shows the history of all dot plans generated during query evaluation, which can be rendered by just selecting the corresponding

[2] `http://xmlgraphics.apache.org/batik`

[3] For example, Figure 3 shows the query graph for query Q7 of the well-known XMark benchmark queries [SWK+02].

item. Moreover, by using the up-and-down buttons, you can linearly track each modification of the XQGM graph from beginning to the end. Finally, the menu bar provides three major menus (simplification, restructuring, and transformation) allowing to select all rules to be applied during query evaluation. Figure 3 shows the complete transformation menu. If there is more than one pattern finding a match in the graph, we can assign a priority to each rule, which may be used to give preferences over alternative ones. Because there are several dependencies between rules within and across the simplification, restructuring, and transformation rule sets, we provide predefined rule sets to choose from and support creating custom rule sets by experienced users.

4 Demonstration Setup

During the demonstration session, we come up with a predefined set of XMark benchmark queries [SWK+02] and provide different-sized XMark documents to run these queries on. Furthermore, we furnish different rule sets allowing to visually compare the impact of varying query evaluation strategies: Using the *node-at-a-time* configuration, we can explore how a query is evaluated according to XQuery's formal semantics. On the other hand, using different *set-at-a-time* configurations, we illustrate how exclusive or combined use of structural joins, holistic twig joins, and different index access operators can boost query execution tremendously.

References

[EGKW03] J. Ellson, E.R. Gansner, E. Koutsofios, and S.C. Northand G. Woodhull. Graphviz and Dynagraph—Static and Dynamic Graph Drawing Tools. In M. Junger and P. Mutzel, editors, *Graph Drawing Software*, pages 127–148. Springer, 2003.

[Gra93] Goetz Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[HH07] Michael Haustein and Theo Härder. An Efficient Infrastructure for Native Transactional XML Processing. *Data & Knowledge Engineering*, 61(3):500–523, 2007.

[Mat07] Christian Mathis. Extending a Tuple-Based XPath Algebra to Enhance Evaluation Flexibility. *Informatik – Forschung und Entwicklung*, 21(3–4):147–164, 2007.

[PHH92] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *Proc. SIGMOD Conference*, pages 39–48, 1992.

[RTG07] Jan Rittinger, Jens Teubner, and Torsten Grust. Pathfinder: A Relational Query Optimizer Explores XQuery Terrain. In *Proc. BTW Conference*, pages 617–620, 2007.

[SWK+02] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A Benchmark for XML Data Management. In *Proc. VLDB Conference*, pages 974–985, 2002.

[WMH08] Andreas M. Weiner, Christian Mathis, and Theo Härder. Rules for Query Rewrite in Native XML Databases. In *Proc. EDBT DataX Workshop*, pages 21–26, 2008.