

An Efficient Evaluation of XQuery with TGV

Nicolas Travers, Tuyet-Tram Dang-Ngoc, Tianxiao Liu

▶ **To cite this version:**

Nicolas Travers, Tuyet-Tram Dang-Ngoc, Tianxiao Liu. An Efficient Evaluation of XQuery with TGV. 3rd International Conference of WEB Information Systems and Technologies (Web-IST), 2007, Barcelona, Spain. hal-00733462

HAL Id: hal-00733462
https://hal.science/hal-00733462

Submitted on 18 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

XLIVE: INTEGRATING SOURCES WITH XQUERY
An efficient XQuery evaluation prototype with TGV

Nicolas Travers
PRiSM Laboratory, University of Versailles, 45 av des Etats-Unis, Versailles, France
Nicolas.Travers@prism.uvsq.fr

Tuyêt-Trâm Dang-Ngoc
ETIS Laboratory, University of Cergy-Pontoise, 2 rue A. Chauvin, Pontoise, France
Tuyet-Tram.Dang-Ngoc@u-cergy.fr

Tianxiao Liu
ETIS Laboratory, University of Cergy-Pontoise, 2 rue A. Chauvin, Pontoise, France and XCalia SA, Paris, France
Tianxiao.Liu@u-cergy.fr, Tianxiao.Liu@xcalia.com

Keywords: XQuery evaluation, Extensible Optimization

Abstract: This paper presents an efficient evaluation of XQuery in a heterogeneous distributed system.
XQuery(W3C, 2005) is a rich and so a complex language. Its syntax allows us to express a large scale of queries over XML documents. We have extended (Chen et al., 2003) proposal to rewrite XQuery expressions in ”canonical XQuery” in order to support the full XQuery specification.
The XQuery expressiveness makes difficulties to obtain an exclusive internal representation within a system. Models based on Tree Patterns have been proposed, and we have extended the tree pattern model to a model called TGV that (a) integrates the whole functionalities of XQuery (b) uses an intuitive representation that provides a global visualization of the request in a mediation context and (c) provides a support for optimization and for cost information.
Our paper is based on the XLive mediation system. XLive integrates sources in a uniform view. It is a running research vehicle designed at PRiSM Laboratory for assessing the integration system at every stage of the process starting from sources extraction to the user interface and is already used in several projects.

This paper presents XLive, an efficient XQuery processing in a heterogeneous distributed system.

1 XLIVE ARCHITECTURE

The XLive prototype is designed to be a light mediation system with high modularity and extension capabilities. It is a running research vehicle designed for assessing the integration system at every stage of the process starting from sources extraction to the user interface, including query parsing and modeling, optimization and evaluation, and also benchmarking.

As most mediation systems (Wiederhold, 1992), XLive is composed of three layers: *Presentation*, *Integration* and *Sources*.

Figure 1 shows the integration system. On the *Source Layer*, there are multiple heterogeneous data sources (relational/object/XML data sources, web-services, files, etc.) These sources are queryable by the XLive system *via wrappers*. The *Wrapper* is a component for accessing a specific *Source* for querying and retrieving result. As sources have specific access methods, the role of the wrapper is to translate the wrapper specificity to a common access method.

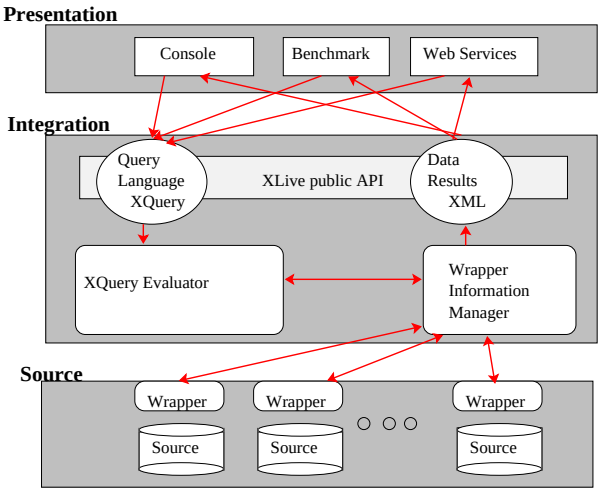


Figure 1: XLive Architecture

The *Integration Layer* is the heart of the medi-

ation system as it process XQuery requests according to sources and send by the result in XML form. The *Wrapper Information Manager* is for integrating information about wrapped sources. It provides to the mediator sources metadata, capabilities, and costs statistics on source data. The *XQuery Evaluator* parses the XQuery query, make logical plans then physical plans by using optimization rules, choose the best execution plan, and then evaluates the execution plan by querying relevant sources and merging results The XLive system is provide a public API that process an XQuery query and evaluates it on sources, and then return the result as an XML document.

Some clients using the XLive API have been implemented on the *Presentation Layer*: The *Console* is a graphical interface for managing and querying XLive. The *Web service* provides services for managing/querying XLive from the Web. And finally, the *Benchmark* is designed to test the XLive mediator and other sources in order to make comparisons.

More details on the XLive architecture can be found on (Dang-Ngoc and Gardarin, 2003; Dang-Ngoc et al., 2005). The TGV model used for XQuery evaluation has been described in (Travers and Dang-Ngoc, 2006a; Travers et al., 2007). And a paper describing the TGV annotation method and extensible optimization rules has been submitted to this conference in the paper session.

2 XQUERY EVALUATION

XQuery is a rich and complex language. Its powerful expression capabilities provide a large range of queries over XML documents. However the richness of the language makes the evaluation of XQueries inefficient in several cases.

Figure 2 describes the evaluation process: (1) XQuery is canonized into a canonical form of XQuery (2) then the canonized XQuery is modeled in the internal structure TGV which can (3) be restructured into equivalent structures using equivalence rules. (4) Then the TGV is annotated with information for evaluation such as the data sources location, cost models information, sources functional capabilities, etc. The optimal annotated TGV is then selected and (5) the logical TGV is transformed into an execution plan using a physical algebra. We have chosen the XAlgebra (Dang-Ngoc and Gardarin, 2003), an extension of the relational algebra to XML. (6) Finally, the execution plan is evaluated and produces an XML result.

The whole process is implemented in the XLive (Dang-Ngoc et al., 2005) system and validates all use-cases defined by the W3C that do not imply strong

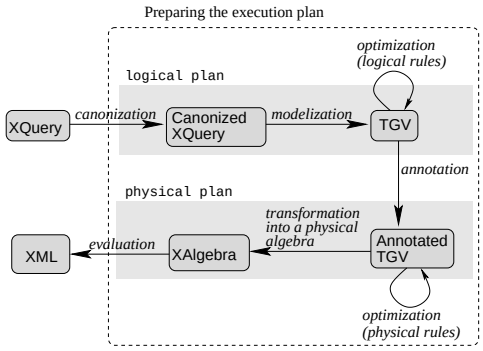


Figure 2: XQuery evaluation process

```
<bib>{
 for $b in doc("http://bstore1.example.com/bib.xml")//book
 where count ($b/author) > 0
 return
  <book>
   {$b/title}
   {$b/author[position() <= 2]}
   {if (count ($b/author) > 2)  then <et-al/>  else ()}
  </book>
}</bib>
```

Table 1: An XQuery query example

typing consideration (our system recognize 8 of the 9 categories of XQuery).

Table 1 gives an XQuery query example supported by our XQuery processor. It is the query XMP-6 defined in the W3C use-cases document. This query expresses the following request: *"For each book that has at least one author, list the title and first two authors, and an empty "et-al" element if the book has additional authors."*

2.1 XQuery Canonization

XQuery defines complex operations such as aggregation, ordering, nesting/unnesting, document construction, conditional cases (if-then-else), and XPath filter. To handle such functionalities, a canonical form using simple sequences of FOR...LET...WHERE...ORDER-BY...RETURN (FLWOR) expressions can be used (the demonstration begin in (Chen et al., 2003) and achieved in (Travers and Dang-Ngoc, 2006b)).

In order to optimize the evaluation of such queries, we propose to structure the language declaration by canonizing it using some transformation rules. These transformation rules keep the semantic of queries and make them more convenient to manipulate.

Table 2 gives the canonized XQuery form of query shown of the example presented in table 1. The following canonization rules have been used:

- Each FLWR-type queries nested in the *return* clause is redefined in a *let* clause and is assigned

```
let $r_1 :=
    for $b in doc("http://bstore1.example
                   .com/bib.xml")//book
    let $r_2 := for $a in $b/author
                where position ($a) <= 2
                return $a
    let $agg_1 := count ($b/author)
    where $agg_1 > 0
    return
     <book>
      {$b/title}
      {$r_2}
      {if ($agg_1 > 2) then <et-al/> else ()}
     </book>
return
 <bib> {$r_1} $</bib>
```

Table 2: Canonized XQuery query

to a variable. In the *return* clause, the nested queries are replaced by their assigned variable.

- Each aggregate function is redefined in a *let* clause and is assigned to a variable that replaces the aggregate function in the main expression.
- Each quantifier functions (*every and some*) is redefined in a *let* clause and is assigned to a variable that replaces the quantifier functions in the main expression.
- Nested expressions in conditional expressions *If/Then/Else* are declared in *let* clauses with new variables. The conditional expression is rewritten using those variables.

2.2 TGV

XQuery modeling is a difficult goal since it provides a large set of functionalities. Its canonization in a canonical form helps us defining a more simple model. Moreover, mediation purpose directs our reflection for modeling in specific Tree Patterns in which all XQuery operations are represented in a single expressive form. Each set of trees defined in canonized queries generates a particular representation.

Tree Pattern Queries (Sihem et al., 2002; Jagadish et al., 2001) are now well admitted for modeling parts of XML Queries. Works as GTP (Chen et al., 2003) use the Tree Pattern Query as a basis to model a part of the XQuery specification.

We propose the TGV (Tree Graph View) model which is implemented in the XLive mediator to process XQuery. The TGV model (Travers and Dang-Ngoc, 2006a; Travers et al., 2007) extends the Tree Pattern representation in order to make it intuitive and full non-typed XQuery support. XQuery queries are modeled in TGV, then optimized, translated to an execution plan and evaluated.

The TGV model allows us to cover 8 of 9 use cases of the W3C (unless strong typing). In fact, specific XQuery characteristics have been integrated into the TGV model by canonization or specific elements.

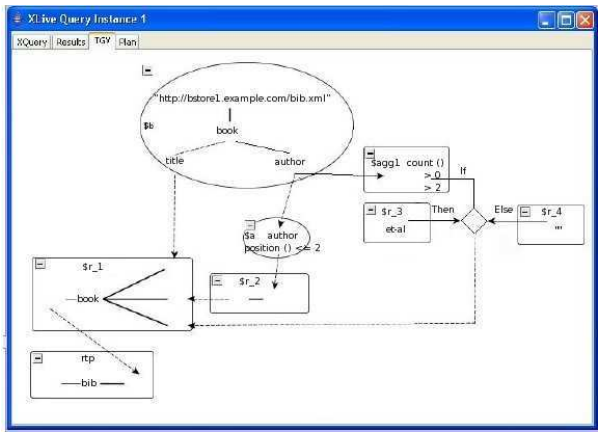


Figure 3: TGV

The TGV proposed in figure 3 illustrates the canonized query (table 2). We can identify tree patterns for source document, return document and intermediate structures. On tree patterns are represented by labels, constraints and ascendant-descendant links (simple line for descendant axis, double link otherwise), and a variable is bound. Between trees, hyperlinks represent aggregation, join, projection or condition. More description on TGV are given in (Travers and Dang-Ngoc, 2006a; Travers, 2006).

3 THE XLIVE SYSTEM

The whole XLive architecture with TGV support and full untyped-XQuery implementation has been released as an Open-Source software[1].

In the figure 4, we show a XLive screenshot where we see ① data sources management ② global schema management ③ an XQuery query ④ the associated logical TGV ⑤ the associated XAlgebra execution plan and ⑥ the XML result document.

A benchmark (Dragan and Gardarin, 2005) has been created in the context of distributed semi-structured model on heterogeneous sources. It has been specified and applied to XLive. It performs comparisons between queries evaluated on single sources and queries evaluated through the mediator. Mediator performance is shown in Figure 5 for a set of representative queries (Selection, projection, join, reconstruction). We use two kinds of XML data in our

[1]The XLive system is downloadable at `http://www.prism.uvsq.fr/~ntravers/xlive/`

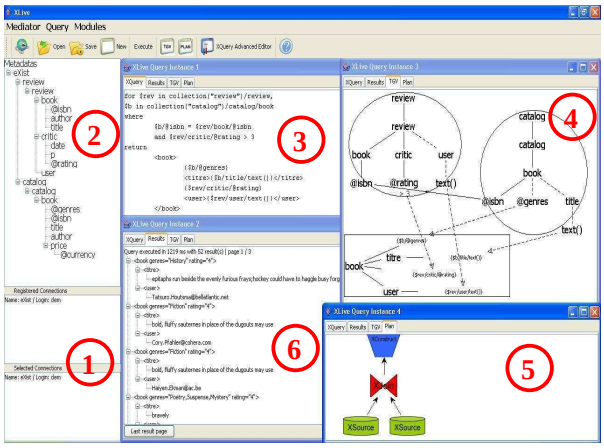


Figure 4: XLive execution

scenarios, data oriented and structure oriented. XLive stores those data in different systems: native XML repositories (XHive, Xyleme), relational systems (Oracle, MySQL) and web sources (Google, Amazon). There is an overhead due to the mediation process compared to single source evaluation. The execution is 1.4 to 2 times slower.

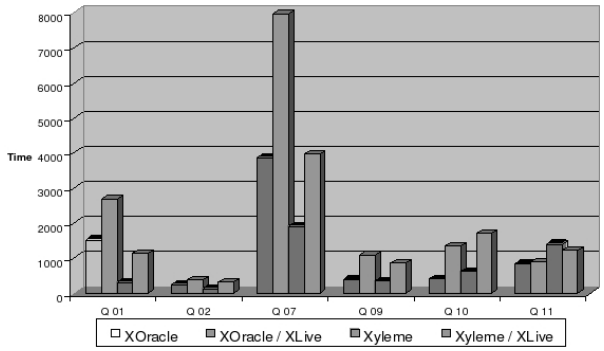


Figure 5: Benchmark

ACKNOWLEDGEMENTS

The XLive is supported by the ACI Semweb project. The cost model is supported by the ANR PADAWAN project and XCalia S.A.

REFERENCES

Chen, Z., Jagadish, H., Laksmanan, L. V., and Paparizos, S. (2003). From Tree Patterns to Generalized Tree Patterns: On efficient Evaluation of XQuery. In *VLDB*, pages 237–248, Germany.

Dang-Ngoc, T.-T. and Gardarin, G. (2003). Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*

Dang-Ngoc, T.-T., Jamard, C., and Travers, N. (2005). XLive: An XML Light Integration Virtual Engine. In *Proc. of BDA*.

Dragan, F. and Gardarin, G. (2005). Benchmarking an XML Mediator. In *ICEIS*, Miami USA.

Jagadish, H., Lakshmanan, L. V., Srivastava, D., and Thompson, K. (2001). TAX: A Tree Algebra for XML. In *DBPL*, pages 149–164.

Sihem, A.-Y., SungRan, C., Laks, L. V. S., and Divesh, S. (2002). Tree Pattern Query Minimization. *VLDB Journal*, 11(4)::315–331.

Travers, N. (2006). *Extensible Optimization in an XML Mediator*. PhD thesis, University of Versailles.

Travers, N. and Dang-Ngoc, T.-T. (2006a). Tree graph view (tgv). Technical report, PRiSM Laboratory.

Travers, N. and Dang-Ngoc, T.-T. (2006b). Xquery canonization. Technical report, PRiSM Laboratory.

Travers, N., Dang-Ngoc, T.-T., and Liu, T. (2007). Tgv: an efficient model for xquery evaluation within an interoperable system. *International Journal of Interoperability in Business Information Systems (IBIS)*, 2. ISSN: 1862-6378.

W3C (2005). An XML Query Language (XQuery 1.0).

Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49.