An Extensible Rule Transformation Model for XQuery Optimization

Nicolas Travers, Tuyet-Tram Dang-Ngoc

AN EXTENSIBLE RULE TRANSFORMATION MODEL FOR XQUERY OPTIMIZATION
Rules Pattern for XQuery Tree Graph View

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, Cergy, France
Tuyet-Tram.Dang-Ngoc@u-cergy.fr

Abstract: Efficient evaluation of XML Query Languages has become a crucial issue for XML exchanges and integration. Tree Pattern (Sihem et al., 2002; Jagadish et al., 2001; Chen et al., 2003) are now well admitted for representing XML Queries and a model -called TGV (Travers, 2006; Travers et al., 2006; Travers et al., 2007c)- has extended the Tree Pattern representation in order to make it more intuitive, respect full XQuery specification and got support to be manipulated, optimized and then evaluated.

For optimization, a search strategy is needed. It consists in generating equivalent execution plan using extensible rules and estimate cost of plan to find the better one. We propose the specification of extensible rules that can be used in heterogeneous environment, supporting XML and manipulating Tree Patterns.

1   INTRODUCTION

Efficient evaluation of XML Query Languages has become a crucial issue for XML exchanges and integration (Abiteboul, 1997). XQuery (W3C, 2005) has proved to be an expressive and powerful language to query XML data both on structure and content, and to make transformation on data. In addition, its query functionalities come from both database community (filtering, join, selection, aggregation), and text community (supporting and defining function as text search). However, such functionalities provided by the XQuery language imply complexity that makes its evaluation very difficult.

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.

In previous work (Dang-Ngoc et al., 2004; Travers et al., 2007c; Travers et al., 2007c), we have defined the TGV model that extends the Tree Pattern representation in order to make it intuitive, have full XQuery support and got support to be manipulated, optimized and then evaluated. XQuery queries are modeled with TGV to generate execution plan that

will be used to evaluate the XQuery query.

We are interested in providing an extensible framework to optimize XQuery queries' evaluation. Optimizing query evaluation can have different meaning depending on what the user or the application expects. The most popular and important aspect is efficiency or cost. Mainly, cost models referred to time cost (time for evaluating the whole request and provide the result), but also resource cost, energy cost, money cost, etc. But depending on the context, other aspects can be considered instead for an optimized query evaluation, as accuracy on the evaluation as introduced in weighted patterns (Damiani and Tanca, 2000), fuzzy (Damiani et al., 2000) or flexible (Calmès et al., 2003) queries or right access.

Extensible optimizer were studied in Exodus (Carey et al., 1990), Starburst (Widom, 1996), Volcano (Graefe and McKenna, 1993) and OPT++ (Kabra and DeWitt, 1999). An extensible optimizer aims at generating a query optimizer by integrating new transformation rules. These rules transform algebraic plans into alternative ones. Extensible optimizers need search strategies to order transformation rules for queries execution. It often relies on cost information as with introduced *expected cost factor* in (Carey et al., 1990) and (Graefe and McKenna, 1993).

However, these works are designed to relational or object contexts. And as far as we know, those solutions could not be applied on semi-structured data with tree pattern matching queries. So, we define a model for designing transformation rules that would be applied on TGV, to optimize XQuery evaluation.

In this article, we briefly recall the TGV model and annotations on which transformation rules rely on (section 2). In section 3, we present the goal of this article which defines a new model, the *rule patterns*, which would be integrated in our extensible optimizer. And, finally we conclude.

2 TREE GRAPH VIEW (TGV)

XQuery is a rich language on XML documents. It defines complex operations such as aggregation, ordering, nesting/unnesting, document construction, conditional cases, sets, sequences, quantifiers, and XPath filter. To handle such functionalities, a canonical form using simple sequences of FOR...LET...WHERE...RETURN (FLWR) expressions equivalent to any XQuery expression can be retrieved (the demonstration begin in (Chen et al., 2003) and achieved in (Travers et al., 2007b)).

The canonized XQuery evaluation process relies on our TGV model. Since we focus on TGV transformation rules, we briefly recall the TGV model. Details and Abstract Data Type formalization of the TGV model can be found in (Travers, 2006; Travers et al., 2007c). A TGV implementation is downloadable on `http://www.prism.uvsq.fr/~ntravers/xlive/`

2.1 The TGV model

To manipulate XQuery expressions, a tree logical structure is needed. (Amer-Yahia et al., 2001) has introduced the TPQ model that expresses a single *FOR-WHERE-RETURN (FWR)* query by a Pattern Tree and a formula. Then, (Chen et al., 2003) proposes the GTP model that are a generalization of the TPQ model. In this model, each *LET* clauses generates a Tree Pattern, and one predicate that contains all constraints. The representation acts as a pattern applicable on data sources. However, GTPs do not capture well all XQuery expressiveness and cannot be used in a mediation context. Moreover, it does not support extensible optimization.

We have proposed the TGV model (Dang-Ngoc et al., 2004; Travers, 2006; Travers et al., 2006) that provides the following features :

(a) It integrates all functionalities of non-typed XQuery queries (collection, XPath, predicate, aggregate, conditional and sequential parts, etc.)

(b) It uses an intuitive representation that provides a global visualization of the query for mediation.

(c) It provides an annotation support for optimization and evaluation purposes (e.g., cost, sources localization).

TGV are *Tree Patterns* sets corresponding to patterns applicable on XML documents. Each Tree Pattern is composed of *Nodes* linked together by axis and mandatory/optional information. Constraints could be attached to a node in order to restrict selected trees. There are four types of Tree Patterns :

- *Source Tree Pattern (STP)* : a simple Tree Pattern that filters XML documents. FOR...
- *Return Tree Pattern (RTP)* : this Tree Pattern builds new XML documents with different specified tags. RETURN...
- *Aggregate Tree Pattern (ATP)* : this Tree Pattern aggregates selected trees to provide new sets of trees. LET...
- *Intermediate Tree Pattern (ITP)* : this Tree Pattern filters selected trees to produce new sets of sub-trees. FOR $y in $x...

Tree Patterns are connected together by *Hyperlinks* allowing trees transformations. Among hyperlinks, we can recall :

- *Join Hyperlinks (JH)* : a join between two tree patterns.
- *Projection Hyperlinks (PH)* : a value projection.
- *Exploration Hyperlinks (EH)* : a filter on a Tree Pattern to generate a new one.
- *Set Hyperlinks (SH)* : a set operation between two tree patterns.

All these elements are connected together to form a TGV that represents an XQuery query. They are formalized in (Travers, 2006).

Table 1 illustrates an XQuery query which contains a FOR clauses ($x) with a constraint predicate defined in a filter (between "[]"). This constraint selects all documents containing the word "XML". Then, the book attribute "*isbn*" must not exceed "1526". Finally, the result includes the title of the book and authors if only there is more then two authors (aggregation and a join on "isbn" with another collection).

This query is represented by a TGV in figure 1. We can see Tree Patterns represented by ovals (STP) and rectangles (ATP, RTP). The first STP defines the collection filter for "*reviews*". It is associated to a "*contains*" function, the tree representation is composed of two branches with a *title* and an *isbn* (linked to a constraint). The second STP filters the collection
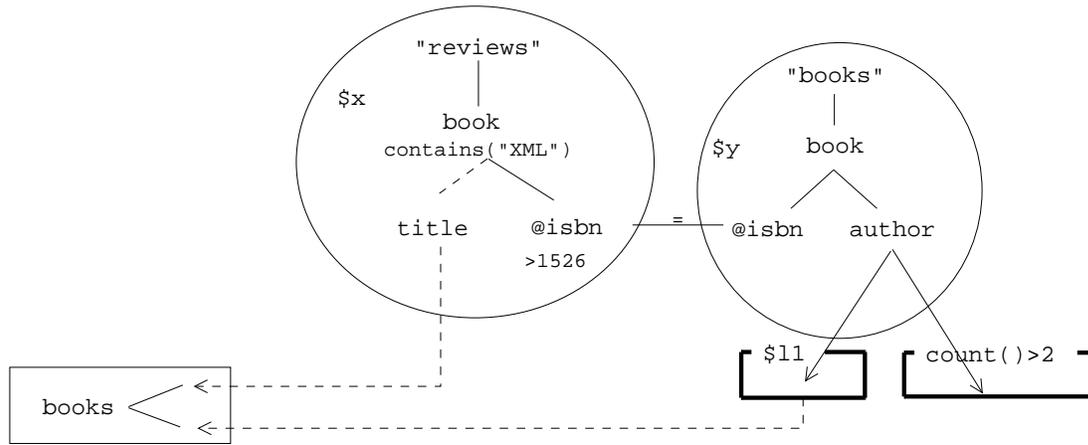
FIG. 1: TGV example

```
<!– For each book containing the word XML and
which ISBN greater than 1526, return the title and
each author information (only if there are at least 3
authors) –>
for $x in collection("reviews")/book[contains (.,
"XML")]
where $x/@isbn > 1526
return
    <books>
      {$x/title}
    {
      for $y in collection("books")/book
      where $x/@isbn = $y/@isbn
            and count($y/author) > 2
      return $y/author
    }
    </books>
```

TAB. 1: XQuery Example

"*books*". The *isbn* is linked to the previous *isbn* node
with a *Join Hyperlink*. The *author* node is linked to
two ATPs by *Projection Hyperlinks*. The ATP on the
right corresponds to the aggregate constraint that re-
quires more than two authors by book. The ATP on
the left nests results for each book (defined by its
isbn), this result is put in the RTP by a *Generalized
Hyperlink*. The RTP represents the XML document
creation, with tags (nodes relation) and values (hyper-
links).

The TGV logical structure recognizes the full
untyped-XQuery specification and covers 8 of 9 use
cases of the W3C. The use-cases category not cove-
red by our TGV model is the STRONG category that
includes query type information.

2.2 Annotations

Our TGV logical model can be extended by anno-
tating its elements with physical information, produ-
cing an annotated TGV (which is then a physical mo-
del). Annotations add new information to a TGV mo-
del in order to indicate for example, execution costs,
localization information, or some others constraints as
accuracy, access right, etc.

We introduce a generic annotation model, which
allows us annotating subsets of TGV elements with
information. The motivation for annotating a TGV is
to give, for each arbitrary granularity (i.e. subset size),
some additional information such as cost information,
system performance information, source localization,
etc. Our annotation model is generic and allows any
type of information. The set of annotation based on
the same annotation type is called an annotated view.
There can be several annotation view for the same
TGV, eg. time-cost annotated view, algorithm anno-
tated view, source-localization annotated view, etc.

For example, we can annotate the execution cost
on a hyperlink; it's also possible to annotate all ele-
ments of a source all together by adding an execution
cost and some localization information.

Figure 2 illustrates an annotated TGV with algo-
rithms annotations. This physical TGV represents a
view of the possible algebraic evaluation of the TGV
with given operators. We can see *XSource*s that selects
and filters data on sources *s1* and *s2* (we can notice
that aggregate functions are embedded in the source
s2). An *XJoin* will join data provided by *s1* and *s2*
with a left outer join. Then, an *XConstruct* will build
an XML document with specific projections on data.

Figure 3 shows a cost annotated TGV. Several
TGV elements are linked to cost annotations. This
view gives an estimate cost of each step of the eva-
luation. We can see the use of cardinality, selectivity,
projection, and aggregation in formulas. The final cost
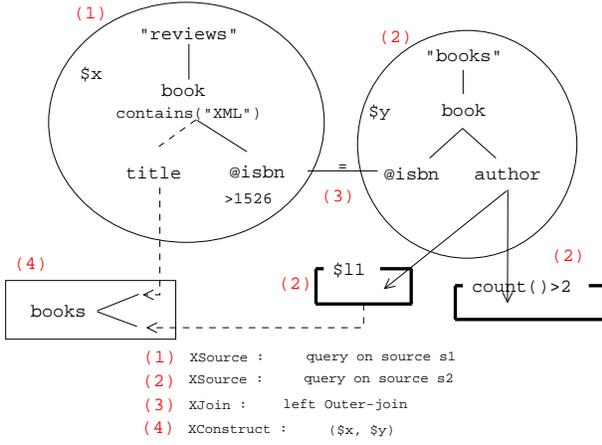formula (8) provides the evaluation global cost.

(1) XSource :	query on source s1
(2) XSource :	query on source s2
(3) XJoin :	left Outer-join
(4) XConstruct :	($x, $y)

FIG. 2: Algorithm-annotated TGV view

sel:selectivity
card:cardinality
proj:projection
agg:aggregation

(1) Cost_s1 = Cost_IO * card_s1 * (2)
(2) Cost_rest_isbn = card_s1 * sel
(3) Cost_s2 = Cost_IO * card_s2 * (4)
(4) Cost_rest_count = card_s2 * agg * sel
(5) Cost_nest_l1 = (3) * proj
(6) Cost_join_isbn = Cost_CPU * (card_s1 * card_s2)
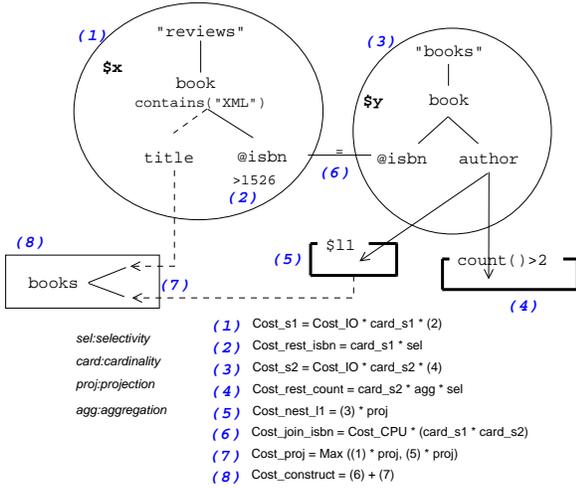(7) Cost_proj = Max ((1) * proj, (5) * proj)
(8) Cost_construct = (6) + (7)

FIG. 3: Cost-annotated TGV view

3 EXTENSIBLE OPTIMIZATION

Now we have recall TGV characteristics, we present the optimization framework. TGV are modified by the optimizer to generate more efficient evaluations.

The optimization framework takes several transformations to manipulate TGVs and generates new ones. We now introduced the optimization process, and then focus on transformations we call *Rule Patterns*. We present a model to create new transformation rules in order to extend our optimizer. We propose a model to represent and create rules ; we **do not** focus on which rule we add to our optimizer (or proving them). Some classical transformation rules can be found in (Cherniack and Zdonik, 1998; Lohman, 1988; Ali and Moerkotte, 2004).

3.1 Optimization framework

Since TGV is an intuitive representation model for XQuery, we aim at defining a representation model for transformation rules on TGV. Then, rules definition could be easier to create, according to existing equivalence rules on object-oriented queries (Cherniack and Zdonik, 1998; Lohman, 1988; Ali and Moerkotte, 2004).We present rules specification applicable on the TGV model.

Two TGV are said to be equivalent if they do have the same evaluation (i.e. same resulting document) independently of the XML documents sources : $eval(TGV_1, \tau) = eval(TGV_2, \tau)$. *eval* is the evaluation function for a TGV and τ an XML document set. Then, a transformation rule ϕ keeps equivalence if the TGV modified by ϕ is equivalent to TGV : $eval(\phi(TGV), \tau) = eval(TGV, \tau)$.

Equivalence Rules (or shortly *Rules*) define that under specified *conditions*, the *result* of the TGV after the *transformation* is equivalent (i.e. same results).

3.2 Rule Patterns

We propose a rule presentation model for this rule language : a *Rule Pattern Model* (RP Model). Like TGV mappings on XML documents with Tree Patterns, we represent rules in intuitive manners in order to build a pattern to map visually on TGV representations. Since TGV representation is a translation from ADT formalization, we can naturally use this transformation process and extract parts of the TGV.

The given rule pattern for the transformation rule is a part of a TGV in which is represented only concerned elements. Two rule patterns are necessary to represent a transformation rule. In fact, the first rule pattern is called *Condition Rule Pattern* and represents the *rule condition*. The second rule pattern is called *Conclusion Rule Pattern* and represents the *rule conclusion* (i.e. the final pattern after applying transformations using condition pattern's variables).

Then, the framework given in Figure 4 defines transformation rules. *Rx* is the transformation name identifier. The *condition rule pattern* represents which TGV elements a TGV must have to be applied. The *conclusion rule pattern* represents the pattern transformation when applied. Differences between the two rule patterns correspond to the transformation.

Transformation rules defined by the Rule Language allow us to improve optimizer's knowledge with new transformations which would modify TGV representations and reach an optimal representation.
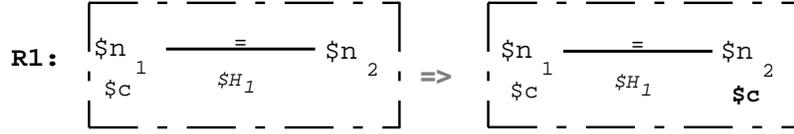
<Rx>: | Condition Rule Pattern | **=>** | Conclusion Rule Pattern |

FIG. 4: Transformation rules's framework

R1: $\$n_1$ ══ $=$ ══ $\$n_2$ / $\$c$ $\$H_1$ **=>** $\$n_1$ ══ $=$ ══ $\$n_2$ / $\$c$ $\$H_1$ **$\$c$**

FIG. 5: Logical Rule Pattern Example

R2:

$\$n_1$ ══ $=$ ══ $\$n_2$ $\$H_1$

(1)   *(3)*   *(2)*

(1) **<<** *(2)*

(1) Cardinality ()
(2) Cardinality ()
(3) Algorithm : : *left Outer-join*

=>

$\$n_1$ ══ $=$ ══ $\$n_2$ $\$H_1$

(1)   *(3)*   *(2)*

(1) Cardinality ()
(2) Cardinality ()
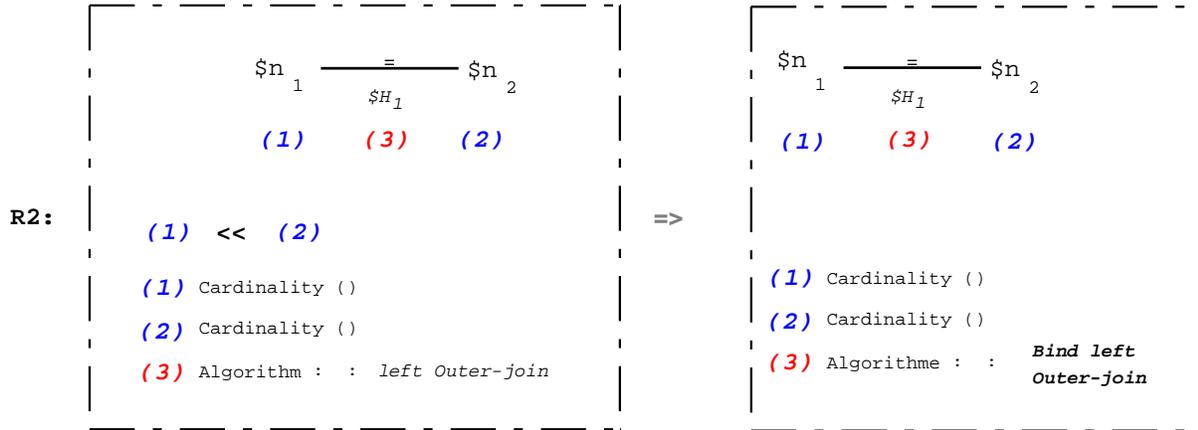*(3)* Algorithme : : ***Bind left Outer-join***

FIG. 6: Physical Rule Pattern Example

These transformations need to be classified in order to direct research strategy ; we can define two main classes : *Logical Equivalent Transformations* and *Physical Equivalent Transformations*. Those categorizations correspond to knowledge level on TGV and sources.

The two first categories include well-known equivalence rules : logical rules and physical rules. Those categories are detailed in following sub-sections with illustrating examples (figure 5 and 6).

3.2.1 Logical Pattern Rules

Thus, Logical Equivalent Transformations correspond to logical knowledge on TGV. Those rules deal with topology improvements on TGV without annotations requirements.

Figure 5 illustrates a transformation rule $R1$. This rule takes care of nodes which are linked to both a constraint and a join hyperlink. Since the equal operator is distributive, we can infer that the constraint will be cloned to the linked node. This figure shows a `Condition Rule Pattern` on the left, in which a join hyperlink $\$H_1$ links $\$n_1$ and $\$n_2$. $\$n_1$ is also linked to a constraint $\$c$. We notice that the join hyperlink is equality joint. Then, the `Conclusion Rule Pattern` represents the same pattern with a single modification. The constraint $\$c$ is cloned on $\$n_2$.

When we apply $R1$ on our example (figure 1), a join hyperlink is present with a linked constraint. Then the transformation is applied and we obtain a new TGV (figure 7) and the constraint is cloned to the other node *isbn*.

3.2.2 Physical Pattern Rules

Physical Equivalent Transformations use physical information that sources can provide like algorithms, cost information, function capabilities, systems statistics. These transformations are based on sources information annotated on TGV (see section 2.2).
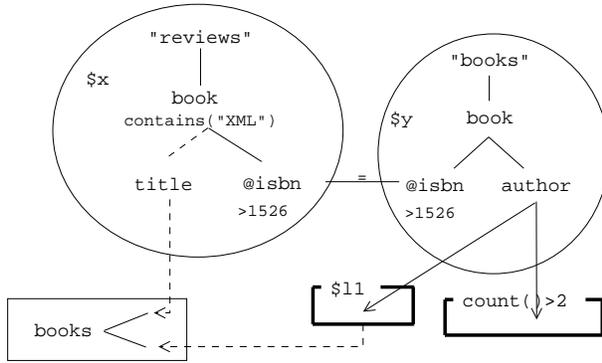
Figure 6 illustrates a physical transformation rule.

FIG. 7: Logical transformation on TGV

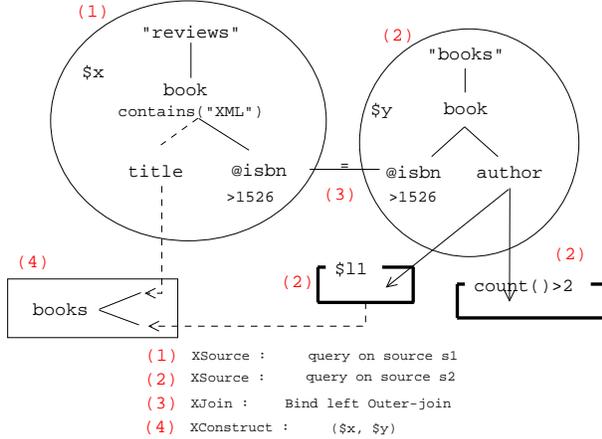| (1) | XSource : | query on source s1 |
(2)	XSource :	query on source s2
(3)	XJoin :	Bind left Outer-join
(4)	XConstruct :	($x, $y)

FIG. 8: Physical transformation on TGV

This physical rule uses two types of annotation. In fact, some annotations from the algorithm-annotated view showed in figure 2 and the cost-annotated view in figure 3 are used in Rule Patterns. We can see a join hyperlink on which annotations must be present (XJoin for the join hyperlink with a *left outer join*) and cardinality annotations for Tree Patterns (cardinality are given by formulas). The *Condition Rule Pattern* verifies if the cardinality of source *s1* is quite lower than source *s2*'s one. Then the transformation modifies the algorithm annotation by a *Bind left outer join*. The cardinality comparison is carried out by the optimizer (theoretical cardinalities are given by the cost model).

When the rule pattern is applied on the TGV, the join algorithm is modified has expected. The result is showed in figure 8.

3.3 Extensibility

User-define Transformations are rules provided by a user or an administrator that have knowledge on sources or data behavior. It could be used to create a specific rule.

Since we have categorized transformations, a search-strategy push the information into the optimizer proposed in (Travers, 2006). This strategy relies on rule valuation with a coefficient of improvement (Carey et al., 1990). This coefficient is directed by information provided by our cost model. Transformation rules, like cost model, need an annotation support on parts of the TGV model in order to manage external information on Tree Graph Views.

The extensibility of the optimizer is then given by new added rules. Each of these rules are added to the optimizer and can potentially improve performances.

4 EXPERIMENTAL EVALUATION

The purpose of this section is to validate the better performance of a query evaluation after having integrated previous defined optimization rules. Thus, we evaluate a query on the benchmark defined by (Dragan and Gardarin, 2005) within the mediator *XLive* (Travers et al., 2007a). We load the extensible optimizer with rule patterns that are applied if the rule condition matches. We use XML documents with various sizes and information distribution, and calculate their execution time after having applied optimization rules. Tests have been realized on an AMD Athlon 1.8GHz, with 1024Mo RAM under Windows XP SP2, sources were handled by a Pentium 2.65GHz, with 512Mo de RAM under Windows XP SP2 with a 10Mbps connection.

The transformation rules integrated into the optimizer are physical rules transformation as they use annotation information (functional capacities of the sources for the functions, cardinality for the algorithm of joint).

Thus, these three optimization rules are used successively :

– delegate function *contains* to sources ;
– delegate the aggregate operation to sources ;
– change the join algorithm (nested loop to hash).

On Figure 9, we report the time used by the XLive mediator to evaluate the XQuery shown in our previous example, applied on documents with different size. Of course, as the document data size increase, the execution time linearly increase too.

The objective is to validate the extensible optimizer and show that the more matching pattern rules there are, the faster is the evaluation.

Measurement are made with sets of rules that are loaded successively in the optimizer. In each additio-

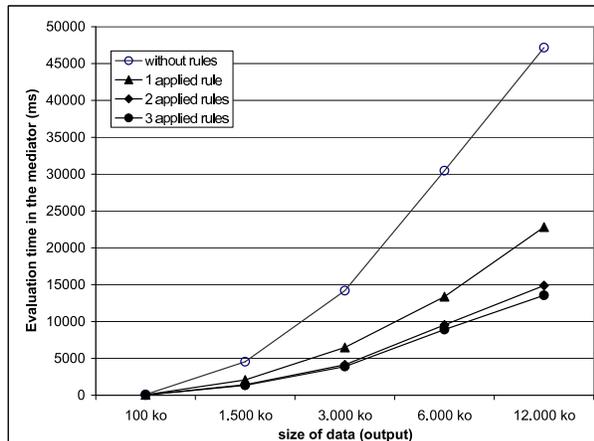nal set that is provided, one rule pattern match the request.

FIG. 9: Time execution depending on the data size

5   RELATED WORK

Using transformation rules, a given execution plan will be transformed into an equivalent execution plan (i.e. that give the same result when evaluated). Transformation rules can be logical (i.e. based on the algebraic properties of the operators), physical (i.e. based on the hardware, system and data statistics) or user-defined (i.e. defined arbitrarily by the user).

EXODUS (Carey et al., 1990) was the first system to include a query optimizer generator based on a rule language that specifies transformation of query trees. The EXODUS optimizer uses a best-first search strategy for rule-application based on the excepted benefit of applying a rule to a query. The Volcano (Graefe and McKenna, 1993) optimizer generator improves EXODUS generator through its use of heuristics and semantics to guide the search for transformations, its ability to learn optimization heuristics, its extensible support for physical properties of data, and its support for flexible costs models that can be used to generate plans for partially specified queries. Starburst (Pirahesh et al., 1992) optimizer is divided in two phases, each one having its own rules language. The query rewrite rules is written in C. Esprit EDS (Finance and Gardarin, 1994) defines an unified rules language for expressing query transformations in an extensible query optimizer.

However, all these works are designed to relational or object contexts. And as far as we know, those solutions could not be applied on a semi-structured context with tree pattern matching queries. So, our model defines transformation rules that would be ap-

plied on TGV, to optimize XQuery evaluation and bring adaptability to the mediator with specific rules.

6   CONCLUSION

In this paper, we propose a rule-based optimizer for XQuery. Rule-based optimizers are extensible as they consist in modifiable sets of rules. Sets of rules can be defined in two categories :

– Logical rules that rely on logical operators used by the query
– Physical rules that rely on physical information got from sources and from the system.

Rules are applied on the XQuery model called TGV designed for heterogeneous distributed sources, which supports full non-typed XQuery specification. A logical rules suite is provided using theorems deduced from the definition of the Abstract Data Type TGV. In order to define the physical rules suite, physical information on sources and system must be provided. User-defined rules provided by the user or the administrator based on her/his knowledge of data and systems. Moreover, the search strategy is all the more efficient since transformation rules are based on annotated information. This leads to a better execution plan. A generic annotation is defined to annotate the TGV and that can be used when defining rules. This generic annotation can support any type of information, mainly cost information, but also information as accuracy, access, preferences, etc.

Many systems relying on rules-based optimizer have been defined for relational sources (Pirahesh et al., 1992; Graefe and McKenna, 1993; Carey et al., 1990; Mitchell, 1993) and object-oriented sources (Kabra and DeWitt, 1999; Finance and Gardarin, 1994). But as far as we know, nothing has been done yet for semi-structured systems and *a fortiori* on heterogeneous distributed XQuery queries. For those reasons, our work on rules-based optimizer for XQuery is a new axis.

Our system based on the mediator/wrappers architecture is called XLive and already supports full XQuery evaluation on heterogeneous distributed sources, using TGV with annotation support (a poster presentation has also been submitted to the conference).

REFERENCES

Abiteboul, S. (1997). Querying Semistructured Data. In *Proceeding of the 6th International Conference on Database Theory*, Delphi, Greece.

Ali, R. and Moerkotte, G. (2004). Query Rewriting with Coko-Kola. Technical report, University of Mannheim.

Amer-Yahia, S., Cho, S., Lakshmanan, L. V. S., and Srivastava, D. (2001). Minimization of Tree Pattern Queries. In *SIGMOD Conference*.

Calmès, D., Prade, H., and Sedes, F. (2003). Requêtes flexibles et données semi-structurées - quelques éléments de discussion et d'implémentation. In *LFA*, pages 23–30.

Carey, M. J., DeWitt, D. J., Graefe, G., Haight, D. M., Richardson, J. E., Schuh, D. T., Shekita, E. J., and Vandenberg, S. (1990). The EXODUS Extensible DBMS Project : An Overview. In D. Maier and S. Zdonik, editor, *Readings on Object-Oriented Database Sys.* Morgan Kaufmann, San Mateo, CA.

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

Cherniack, M. and Zdonik, S. B. (1998). Changing the Rules : Transformations for Rule-Based Optimizers. In *SIGMOD Conference*, pages 61–72.

Damiani, E. and Tanca, L. (2000). Blind queries to xml data. In *Database and Expert System Applications*, pages 266–279.

Damiani, E., Tanca, L., and F.Arcelli (2000). Fuzzy xml queries via context-based choice of aggregation. *Kybernetika*, 36.

Dang-Ngoc, T., Gardarin, G., and Travers, N. (2004). Tree graph view : On efficient evaluation of xquery in an xml mediator. In *Actes de publication de la 20ème conférence Bases de Données Avancées (BDA 2004)*, pages 429–448, Montpellier, France.

Dragan, F. and Gardarin, G. (2005). Benchmarking an xml mediator. In *ICEIS (1)*, pages 191–196.

Finance, B. and Gardarin, G. (1994). A rule-based query optimizer with multiple search strategies. *Data Knowl. Eng.*, 13(1) :1–29.

Graefe, G. and McKenna, W. J. (1993). The Volcano Optimizer Generator : Extensibility and Efficient Search. In *ICDE*, pages 209–218.

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

Kabra, N. and DeWitt, D. J. (1999). OPT++ : an object-oriented implementation for extensible database query optimization. *VLDB Journal : Very Large Data Bases*, 8(1) :55–78.

Lohman, G. M. (1988). Grammar-like Functional Rules for representing Query Optimization Alternatives. Technical report, IBM Almaden Research Center.

Mitchell, G. (1993). Extensible query processing in an object-oriented database. *Ph.D. Thesis, Brown UNIV, CS TR 93-16*.

Pirahesh, H., Hellerstein, J. M., and Hasan, W. (1992). Extensible/rule based query rewrite optimization in Starburst. In *SIGMOD*, pages 39–48.

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. (12 December 2006). *Extensible Optimization in an XML Mediatior*. PhD thesis, University of Versailles.

Travers, N., Dang Ngoc, T.-T., and Liu, T. (2006). TGV : an Efficient Model for XQuery Evaluation within an Interoperable System. *Interoperability in Business Information Systems (IBIS)*.

Travers, N., Dang-Ngoc, T.-T., and Liu, T. (2007a). An Efficient Evaluation of XQuery with TGV. In *The 3rd International Conference of WEB Information Systems and Technologies (Web-IST)*, Barcelona, Spain.

Travers, N., Dang-Ngoc, T.-T., and Liu, T. (2007b). Full Untyped XQuery Canonization. In *the International workshop on Emerging Trends of Web Technologies and Applications (WebETrends)*, Huangshan, China.

Travers, N., Dang-Ngoc, T. T., and Liu, T. (2007c). TGV : a Tree Graph View for Modelling untyped XQuery. In *DASFAA*, Bangkok, Thaïland.

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

Widom, J. (1996). The Starburst Active Database Rule System. *Knowledge and Data Engineering*, 8(4) :583–595.